AUTORITÀ NAZIONALE ANTICORRUZIONE

LINEE GUIDA

Nuove Linee Guida di sviluppo

SOMMARIO

INTRODUZIONE 8

Obiettivo del documento 8

Processo di Consolidamento del documento 8

Flusso collaborativo 8

Struttura del documento 9

Acronimi e glossario 9

Convenzioni di carattere generale 10

Contesto normativo 10

Riferimenti interni 11

Riferimenti Esterni 11

Versionamento del documento 13

ARCHITETTURA DI RIFERIMENTO 13

Caratteristiche generali 14

Tipologia dei servizi erogati 14

Tipologie di utenti 15

Architettura 15

User Interface 16

Tecnologie 18

Api Gateway 18

Tecnologie 19

Microservizi 19

Service Code 20

Scheletro del microservizio 21

Piattaforma di riferimento 22

Tecnologie 23

Servizi dati 23

Tecnologie 23

Componenti trasversali Comuni 23

BPM 24

Enterprise Integrator 24
Message Broker 24
Motore di regole 25
Search Engine 25
Profiling 25
Logging & Monitoring 26
Tecnologie 26

PROCESSO DI RILASCIO 27

Premessa 27
Ambienti di rilascio 27
Metodologie e prodotti utilizzati 28
Requirement Manager 29
Knowledge Manager 29
Source Manager 30
Dependency & build Management 30
Support ticket 31
Bug tracking 31
Source Control 31
Sistema di build automation e Testing 31
Sistema di continuous integration 32
Sistema di continuous deployment 32
Sistema di Knowledge management 32
Sistema di misurazione della qualità del software 32
Fasi del processo di CI/CD 32
Fasi del processo 33
Source Control Management 34
GIT FLOW Ambienti ANAC 35
Ambienti 36
Continuous integration 36
Continuous deployment 37
Continuous Code Inspection 37
Scenari implementativi 37
Scenario 1 – Implementazione nuovo componente 37

Scenario 2 – Modifica componente esistente 39
Scenario 3 – Bug Fixing 40
Scenario 4 - Liferay User Interface - Workspace 40

REQUISITI GENERALI DI SVILUPPO 43
Requisiti di carattere generale 43
Requisiti di versionamento 44
Requisiti per il logging 44
Requisiti per l'audit 44
Requisiti per il testing 44
Requisiti di sicurezza 46
Requisiti non funzionali 48
[RNF-GNRCROSS.1] Attori Activiti 48
[RNF-GNRCROSS.2] Integrazione Protocollo 48
[RNF-GNRCROSS.3] Integrazione IAM 49
[RNF-GNRCROSS.4] Standard AGID 49
[RNF-GNRCROSS.5] Integrazione con servizi ejb 3.0 Legacy 49
[RNF-GNRCROSS.6] Integrazione con AMP 49
[RNF-GNRCROSS.7] Gestione dei file marcati con Firma Digitale 49
[RNF-GNRCROSS.8] Modello di interoperabilità 49
[RNF-GNRCROSS.10] Firma digitale Remota 49
[RNF-GNRCROSS.11] Interfaccia Web 49
[RNF-GNRCROSS.12] Integrazione con Alfresco 50
[RNF-GNRCROSS.14] Monitoraggio applicativo 50
[RNF-GNRCROSS.15] Pulizia disco 50
[RNF-GNRCROSS.16] Utilizzo dello storage 50
[RNF-GNRCROSS.18] Scalabilità 50
[RNF-GNRCROSS.19] Tempo medio di risposta per elaborazioni sincrone 50
[RNF-GNRCROSS.20] Timeout per chiamate a servizi 51
[RNF-GNRCROSS.21] Tempo medio di completamento per elaborazioni asincrone 51
[RNF-GNRCROSS.22] Capacità di accesso utenti 51
[RNF-GNRCROSS.23] Adeguatezza dei tempi di risposta all'aumento degli accessi 51
[RNF-GNRCROSS.24] Adeguatezza del consumo di risorse e consumo di risorse di aumento degli accessi 51
[RNF-GNRCROSS.25] Completezza di descrizione 51

[RNF-GNRCROSS.26] Completezza della guida utente 52
[RNF-GNRCROSS.27] Valori già noti al sistema nei campi di input 52
[RNF-GNRCROSS.28] Comprensibilità dei messaggi d'errore 52
[RNF-GNRCROSS.29] Opzione Annullamento 52
[RNF-GNRCROSS.30] Salvataggi parziali 52
[RNF-GNRCROSS.31] Evitare errori nelle operazioni dell'utente 52
[RNF-GNRCROSS.32] Copertura dei test 52
[RNF-GNRCROSS.33] Robustezza agli errori e gestione delle eccezioni 52
[RNF-GNRCROSS.34] Completezza della tracciabilità utente 53
[RNF-GNRCROSS.35] Installabilità 53
[RNF-GNRCROSS.36] Configurazione delle applicazioni 53
[RNF-GNRCROSS.37] Connessioni ai DBMS - Utenti 53
[RNF-GNRCROSS.38] Riusabilità degli asset 53
[RNF-GNRCROSS.39] Accoppiamento di componenti 53
[RNF-GNRCROSS.40] Migrazione/Riusabilità dei dati 53
[RNF-GNRCROSS.42] Configurazione secondo gli standard di internazionalizzazione e localizzazione 54
[RNF-GNRCROSS.43] conformità ai principi del secure/privacy by design - AGID 54
[RNF-GNRCROSS.44] Monitoraggio accessi ai servizi web 54

LINEE GUIDA DI SVILUPPO 54

User Interface **54**

Requisiti di sviluppo User Interface 55
Requisiti di accessibilità 55
Requisiti di usabilità 55
Requisiti di testing 55
Struttura del progetto 56
Ambiente di runtime 57
Predisposizione ambiente di sviluppo 58
Componenti di Sviluppo - Gitlab 59
Creazione modulo Angular 61
Come sviluppare 62
Tema grafico 62
Portlet application 64
Portlet specification 65

SPA Portlet 65
Service builder 71
Microservizi 72
Requisiti di sviluppo Microservizi 72
Requisiti per Business service layer 73
Requisiti di versionamento 75
Requisiti di nomenclatura 75
Struttura del progetto 76
Come sviluppare 77
Dto 77
Controller 78
Service 78
Mapper 79
Accesso ai dati 80
Configurazione 80
Model 81
Repository 82
Aspetti trasversali 82
Health check, Metrics 82
Logging 83
Aggiunta delle dipendenze 83
Configurazione di logback 84
Utilizzo delle API di SLF4J 85
Tracing distribuito 86
Specifica OpenAPI 86
Sicurezza 87
Circuit Breaker 88
SERVIZI TRASVERALI A SUPPORTO 88
APPENDICE 88
Tipologie di test 88
Test unitari 89
Test di integrazione 89
Contract Test 90

Component Test 91
Test end-to-end 91

Introduzione

Obiettivo del documento

Il presente documento ha lo scopo di fornire una guida per la progettazione e lo sviluppo di applicazioni in linea con gli scenari architetturali ANAC (**RIF.DOCINT.001**). Verranno presentati gli standard sui cui si basano le scelte architetturali in essere e gli ambiti di utilizzo e di applicazione di tali standard. Sono presentati, inoltre, i principi generali, le raccomandazioni e i criteri nell'ambito dello sviluppo delle applicazioni all'interno dell'architettura.

Viene, inoltre, presentato il modello di riferimento per l'intero ciclo di vita dei prodotti software realizzati in Autorità al fine di semplificare le fasi di analisi e progettazione standardizzando sia le tecniche progettuali e metodologiche, sia le fasi operative di esercizio.

Processo di Consolidamento del documento

Anac ha avviato un piano di trasformazione digitale che prevede la revisione del sistema informativo e la riscrittura dei servizi in ottica Cloud native per consentire di scalare la capacità elaborativa in coerenza con le richieste ricevute. Nell'implementazione di questo modello verranno applicate nuove logiche applicative per la conservazione dei dati. In questo modo, completata la fase trasmissiva, il sistema lavorerà su dati ottimizzati per mantenere le applicazioni leggere ed efficaci.

Nell'ambito del progetto di trasformazione digitale è stato definito un percorso "collaborativo" atto a definire i nuovi standard di riferimento per lo sviluppo dei servizi applicativi. Di seguito viene rappresentato lo schema che evidenzia gli step identificati all'interno del piano evolutivo della piattaforma e del documento di linee guida.

image-20200521161124449

Flusso collaborativo

Per consentire un'evoluzione snella del documento, ANAC ha predisposto una strategia collaborativa utile ad intercettare, nel corso degli sviluppi applicativi, tutte le esigenze dei gruppi di

lavoro e comunicare tempestivamente eventuali aggiornamenti del documento previsti nel percorso di trasformazione digitale. Di seguito viene rappresentata la strategia identificata.

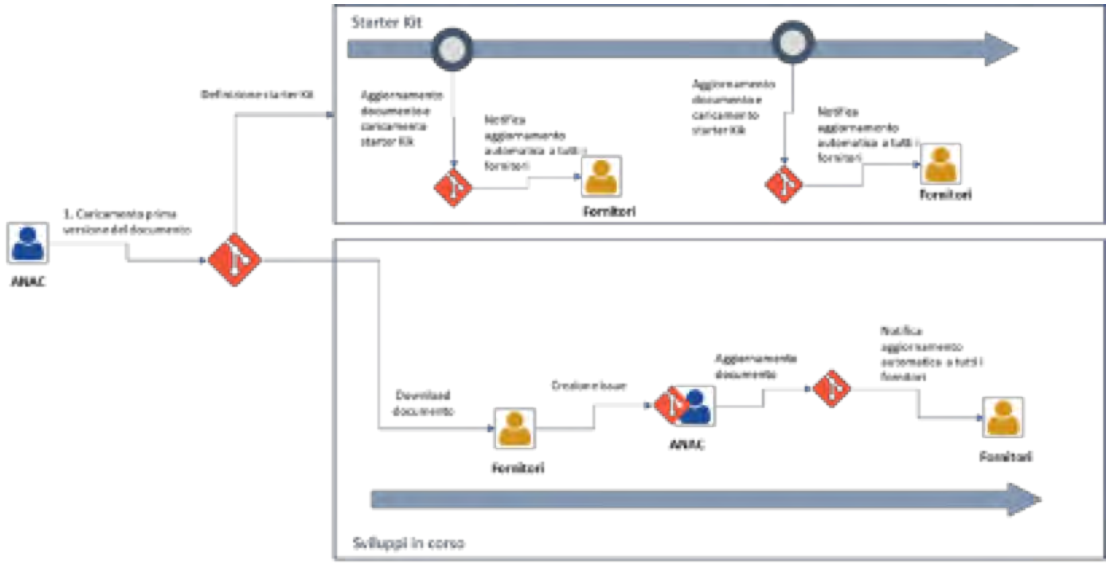


image-20200521161149452

È possibile scaricare il documento di linee guida di sviluppo al seguente indirizzo: *http://git.lab.anticorruzione.local/starterkit/linee-guida-sviluppo*

Struttura del documento

Il presente documento è strutturato in 6 capitoli:

- Architettura di riferimento dell'Autorità
- Processo di rilascio
- Requisiti generali di sviluppo
- Linee guida per lo sviluppo delle applicazioni
- Servizi trasversali a supporto
- Appendice, Capitolo

Acronimi e glossario

Definizione	**Descrizione**
ANAC	Autorità Nazionale Anticorruzione
BDNCP	Banca Dati Nazionale Contratti Pubblici
SPA	Single Page Application
BFF	Backend for Front-end
AVCP	Autorità per la Vigilanza Contratti Pubblici Organo collegiale che vigila sul rispetto delle regole che disciplinano la materia dei contratti pubblici, dotata di indipendenza funzionale, di giudizio, di valutazione e di autonomia organizzativa.
Autorità	Si riferisce all'ANAC Struttura Committente (SC)
UAFI	Ufficio Analisi Flussi Informativi

UPSI	Ufficio Progettazione e sviluppo, Servizi Informatici e Gestione del Portale dell'ANAC
UESI	Ufficio Esercizio Sistemi
BDNCP	Banca Dati Nazionale dei Contratti Pubblici Anagrafe unica dei contratti pubblici. È la banca dati di riferimento di ANAC per utenze e soggetti rappresentati.
Portale Internet	Portale Internet Punto di erogazione dei servizi web per gli utenti esterni ANAC
Servizi ANAC	Servizi ANAC Nucleo centralizzato di servizi che contengono la logica di Business
SPC	Sistema Pubblico di Connettività L'insieme di infrastrutture tecnologiche e di regole tecniche per lo sviluppo, la condivisione, l'integrazione e la diffusione del patrimonio informativo e dei dati della pubblica amministrazione.
PDD	Porta di dominio
FWXML	Firewall XML
B2C	Business to Consumer
B2B	Business to Business
SOA	Service Oriented Architecture
LDAP	Lightweight Directory Access Protocol
KMS	knowledge management system
EIS	Enterprise Information System
RPC	Responsabile della prevenzione della corruzione
RT	Responsabile della Trasparenza
RUP	Responsabile Unico del Procedimento
RASA	Responsabile anagrafica stazione appaltante
OE	Operatore economico

Convenzioni di carattere generale

Identificativo	Descrizione
RIF. {short_desc}.nnn	Identificativo univoco di un riferimento dove {short_desc} è una descrizione breve e nnn è un progressivo numerico
REQ.{short_desc}.nnn	Identificativo univoco di un requisito dove {short_desc} è una descrizione breve e nnn è un progressivo numerico

Contesto normativo

Codice riferimento	Riferimento normativo	Descrizione
RIF.NORM.01	Legge 9 Gennaio 2004, n.4	Disposizioni per favorire l'accesso dei soggetti disabili agli strumenti informatici" (c.d. "Legge Stanca")

RIF.NORM.02	d.P.R. 1 marzo 2005, n.75	Regolamento di attuazione della Legge 9 gennaio 2004, n. 4, per favorire l'accesso dei soggetti disabili agli strumenti informatici
RIF.NORM.03	D.M. 8 luglio 2005	Requisiti tecnici e i diversi livelli per l'accessibilità agli strumenti informatici
RIF.NORM.04	D.M. 20 marzo 2013	Aggiornato l'allegato A del D.M. 8 luglio 2005 in merito ai requisiti tecnici e i diversi livelli per l'accessibilità agli strumenti informatici. Si tratta di un aggiornamento dei requisiti di accessibilità informatica dalle WCAG 1.0 alle WCAG 2.0
RIF.NORM.05	Circolare n.61/2013 del 29 marzo 2013 dell'Agenzia per l'Italia Digitale	Accessibilità dei siti web delle pubbliche amministrazioni
RIF.NORM.06	D.Lgs. 7 marzo 2005, n.82 smi	Codice dell'Amministrazione digitale
RIF.NORM.07	Direttiva 27 luglio 2005	Qualità dei servizi on line e misurazione della soddisfazione degli utenti emanata dal Ministro per l'innovazione e le tecnologie e dal Ministro per la funzione pubblica

Riferimenti interni

Codice riferimento	Descrizione del documento	Versione	Codice del documento
RIF.DOCINT.001	Scenari architetturali	1.0	SPC-ARCT-Scenari Architetturali v.1.0
RIF.DOCINT.002	Architettura tecnica API management – Linee guida	1.0	SPCL4-ARCT-Linee Guida Servizi v1.0
RIF.DOCINT.003	Linee Guida Liferay	1.2	SMC_ANAC_Linee_Guida_v1.2
RIF.DOCINT.004	ANAC – Processo di sviluppo	1.0	ANAC-ProcessoSviluppo_v1.0

Riferimenti Esterni

Identificativo	Codice	Descrizione
RIF.EXT.001	ISO/IEC 17799	Codice di condotta per la gestione della sicurezza dell'informazione
RIF.EXT.002	ISO/IEC 25010 (ISO/IEC 9126).	Standard di qualità software
RIF.EXT.003	ISO/IEC 12207	Gestione del ciclo di vita del software
RIF.EXT.004	"Enterprise Integration Pattern"	2011 - Gregor Hohpe e Bobby Woolf. Designing, Building, and Deploying Messaging Solutions

RIF.EXT.005	“SOA design pattern”	2009 - Tomas Erl
RIF.EXT.006	Java Code style	Google - https://google.github.io/styleguide/javaguide.html
RIF.EXT.07	Istio – Circuit Breaker	https://istio.io/docs/tasks/traffic-management/circuit-breaking/
RIF.EXT.008	Linee guida design PA	“Linee guida di design per i servizi digitali della PA” fornite da AgID
RIF.EXT.009	Piano triennale AgID	Piano triennale per l’informatica nella Pubblica Amministrazione (2017-2019)
RIF.EXT.010	Linee Guida Sviluppo Software Sicuro - AgID	Linee guida di sicurezza nello sviluppo delle applicazioni
RIF.EXT.011	Circolare AgID n. 3/2017	Raccomandazioni e precisazioni sull’accessibilità digitale dei servizi pubblici erogati a sportello dalla Pubblica Amministrazione, in sintonia con i requisiti dei servizi online e dei servizi interni
RIF.EXT.012	Linee guida per la modellazione delle minacce ed Individuazione delle azioni di Mitigazione conformi ai Principi del secure/privacy by design - AgID	Linee guida di conformità ai principi del secure/privacy by design fornite da AgID
RIF.LGS.01	Linee guida per i siti web della PA	Emanate da DigitPA (ora Agenzia per l’Italia Digitale) nel 2011, in riferimento all’art. 4 della Direttiva n. 8/2009 del Ministro per la pubblica amministrazione e l’innovazione
RIF.LGS.02	SPCoop – Servizio pubblico di cooperazione applicativa	Riferimento tecnico per la cooperazione applicativa fra le amministrazioni pubbliche
RIF.LGS.03	Linee guida nazionali per la valorizzazione del patrimonio	Documento di LINEE GUIDA per l’individuazione degli standard tecnici, compresa la determinazione delle ontologie dei servizi e dei dati, le procedure e le modalità di attuazione delle disposizioni del Capo V del Codice dell’Amministrazione Digitale

	informativo pubblico	con l'obiettivo di rendere il processo omogeneo a livello nazionale, efficiente ed efficace
RIF.LGS.04	Linee guida sviluppo API Rest Liferay	https://portal.liferay.dev/docs/7-1/frameworks/-/knowledge_base/f/headless-rest-apis https://portal.liferay.dev/docs/7-1/appdev/-/knowledge_base/a/generating-apis-with-rest-builder https://portal.liferay.dev/docs/7-1/tutorials/-/knowledge_base/t/creating-remote-services#creating-remote-services
RIF.LGS.05	Mongo University Realizzare microservizi con mongo e sping boot	https://www.mongodb.com/blog/post/rest-apis-with-java-spring-boot-and-mongodb
RIF.LGS.06	Strumenti di amministrazione Liferay	https://portal.liferay.dev/docs/7-2/user/
RIF.LGS.07	Creare Workspace Liferay	https://portal.liferay.dev/docs/7-1/tutorials/-/knowledge_base/t/creating-a-liferay-workspace-with-liferay-ide
RIF.LGS.08	Liferay	https://sourceforge.net/projects/lportal/files/Liferay%20IDE/3.7.1/

Versionamento del documento

L'aggiornamento delle linee guida è di specifica competenza dell'Autorità, la quale si riserva di cambiare in qualsiasi momento il contenuto del presente documento. Ogni variazione viene tracciata nel presente paragrafo. Nella tabella seguente vengono riportate le versioni del documento; la data di emissione, coincidente con la pubblicazione del documento e con la comunicazione ai fornitori interessati; la data di entrata in vigore delle nuove linee guida, che di norma avviene 30 giorni dopo la loro emissione, in casi particolari è possibile specificare una data di entrata in vigore differente. Dalla data di entrata in vigore il fornitore è tenuto a rispettare la nuova versione delle linee guida

Versione	Data emissione	Data di entrata in vigore	Descrizione delle modifiche
1.0	05/05/2020	05/05/2020	Prima stesura

Architettura di riferimento

In questo capitolo viene presentata una visione d'insieme del disegno logico e fisico dell'architettura 'api-centric' adottata da parte dell'Autorità per garantire l'erogazione di:

- servizi applicativi da fornire agli stakeholder esterni all'Autorità come stazioni appaltanti, operatori economici, enti pubblici e privati;

- servizi di cooperazione applicativa;
- insieme di applicazioni aziendali interne all'Autorità;
- insieme di servizi infrastrutturali;
- insieme di servizi trasversali.

Caratteristiche generali

L'architettura generale di riferimento dell'Autorità è stata realizzata secondo il pattern architetturale orientato ai servizi (API Rest) presentato in precedenza, con lo scopo di sfruttarne i vantaggi offerti. Al fine di garantire l'aderenza ai requisiti fondamentali di un'architettura multi-layer si è proceduto a:

- realizzare un layer di presentation standardizzando l'interfaccia utente per le applicazioni web-based;
- separare e incapsulare la logica di business in specifici layer con compiti di carattere generale o specifico;
- isolare il livello di accesso ai dati.

Tipologia dei servizi erogati

I servizi erogati dall' Autorità sono sostanzialmente di quattro tipologie:

- Servizi B2C: sono i servizi che consentono di interagire con l'utente finale, sia interno sia esterno all'Autorità, attraverso un interfaccia web fruibile tramite un browser.
- Servizi B2B: sono i servizi che consentono l'interoperabilità e la cooperazione (puntuale o massiva) tra i sistemi previo accordo di servizio o protocollo d'intesa.
- Servizi Trasversali: sono i servizi di supporto all'erogazione dei servizi dell'Autorità come servizi di gestione documentale, sicurezza applicativa, service registry e catalogo dei servizi.
- Servizi Infrastrutturali: sono i servizi che compongono l'infrastruttura dell'Autorità come servizi di rete, sistema, data center, sicurezza, posta elettronica, firma digitale e monitoraggio della qualità dei servizi.

Nella figura seguente è mostrato il modello concettuale e gli stakeholder dei servizi erogati dall'Autorità:

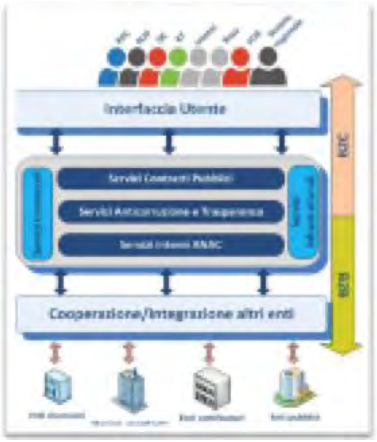


image-20200521165420733

Tipologie di utenti

I servizi sono accessibili da varie tipologie di utenti:

- Utenti interni dell'Autorità che possono:
  - Consultare i dati attraverso un portale web oppure attraverso funzioni ad-hoc;
  - Utilizzare i servizi gestionali dell'Autorità;
  - Accedere alla intranet aziendale.
- Utenti esterni dell'Autorità registrati e profilati per l'utilizzo dei servizi ad accesso riservato come SIMOG, RISCOSSIONE, SMARTCIG, etc;
- Utenti esterni dell'Autorità, per l'accesso pubblico ai servizi come Catalogo dei dati aperti, Portale trasparenza, casellario delle imprese qualificate, etc;
- Enti esterni, per l'accesso ai servizi in cooperazione applicativa o mediante integrazione di sistemi.

Architettura

In questa sezione viene descritto lo standard architetturale previsto per lo sviluppo di applicazioni a microservizi e delle componenti di BI relative ai dati acquisiti.

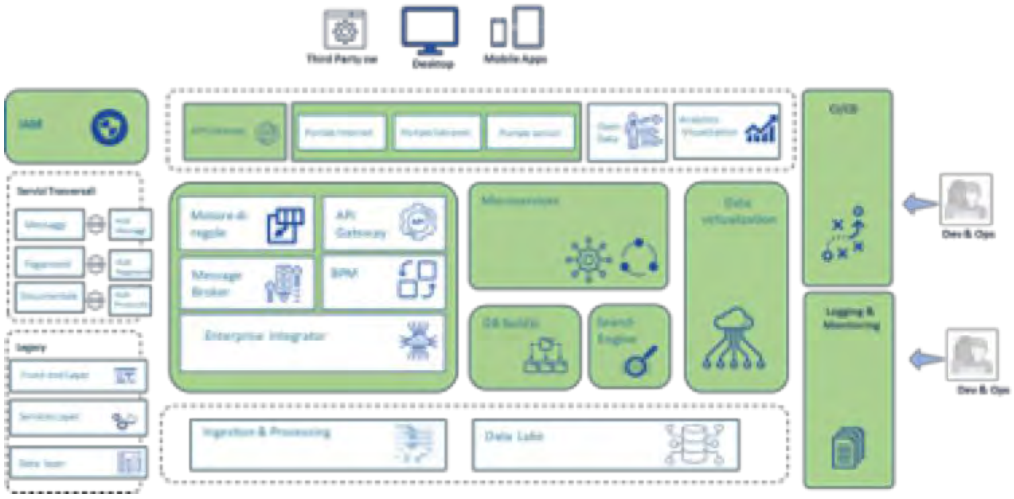


image-20200521165635501

User Interface

Il nuovo **Portale ANAC (Servizi ed Intranet),** è la componente che ricopre il ruolo di punto unico d'accesso web per tutti gli utenti abilitati (interni ed esterni). Per lo sviluppo delle interfacce utente devono essere realizzate portlet "**Single Page Application**" erogate attraverso il "**Portale Liferay**".

image-20200521165722788

Le sezioni della pagina e le risorse vengono caricate solamente in fase iniziale. Durante la navigazione l'applicazione interagisce con i microservizi applicativi, tramite chiamate http/rest, per recuperare i dati e modificare il proprio aspetto e contenuto, senza dover ricaricare l'intera pagina.

Per consentire la visualizzazione delle pagine di front-end da diversi dispositivi le Single Page Application devono essere progettate per essere Responsive.

Per la realizzazione delle Single Page Application deve essere utilizzato il framework "**Bootstrap"** per rendere responsive la fruizione delle funzionalità di front-end.

Per separare livello logico dall'interfaccia utente il pattern architetturale da utilizzare è MVVM (Model View ViewModel)

image-20200521170222027

Model Il Model rappresenta modello di dominio (domain model) dell'applicazione che include il modello dei dati, la logica di business e la validazione dei dati stessi. **View** la View consente di gestire eventi, eseguire operazioni ed effettuare il data-binding. **ViewModel** Il ViewModel è un Model esteso con funzionalità per la manipolazione dei dati e per l'interazione con la View.

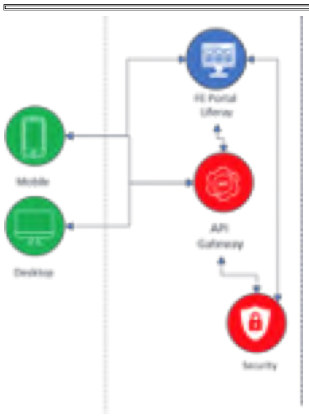


image-20200521170354503

Di seguito viene rappresentato il flusso standard per l'accesso ai servizi di front-end - L'utente accede alla pagina del portale Liferay contenente la SPA del servizio applicativo - La SPA richiama le API esposte: - dall'Api gateway - dal portale Liferay implementando il pattern BFF (Back-end for Front-end)

Nota informativa

Alla data di redazione del documento sono in corso le attività di integrazione della piattaforma di API Management con la componente IAM. Pertanto, per la pubblicazione e l'erogazione delle API rest, richiamate dalle SPA (Single Page application) è necessario sviluppare API rest sul portale Liferay (pattern BFF).Per i servizi di cooperazione applicativa con enti cooperanti (B2B) è possibile utilizzare la piattaforma di API Management. (RIF.DOCINT.002)

ANAC utilizza più istanze di portale in una singola installazione di prodotto. Come pure la possibilità di avere siti con diversi "virtualhost" nella stessa istanza di portale. Il codice sviluppato deve rispettare le best-practices di prodotto in modo da non inserire limiti o difformità nell'utilizzo di queste funzionalità.

Non è possibile implementare servizi (REST, SOAP, ecc...) che siano vincolati a una particolare istanza del portale, o a un sito, quando queste trattano informazioni cross-site o cross-istanza. Viceversa non bisogna implementare servizi che accedono a entità funzionali che dovrebbero rimanere confinati a un site o a un'istanza, ma vengono implementati con "visibilità" globale.

Di seguito viene rappresentata la suddivisione logica delle virtual instance predisposte in ANAC

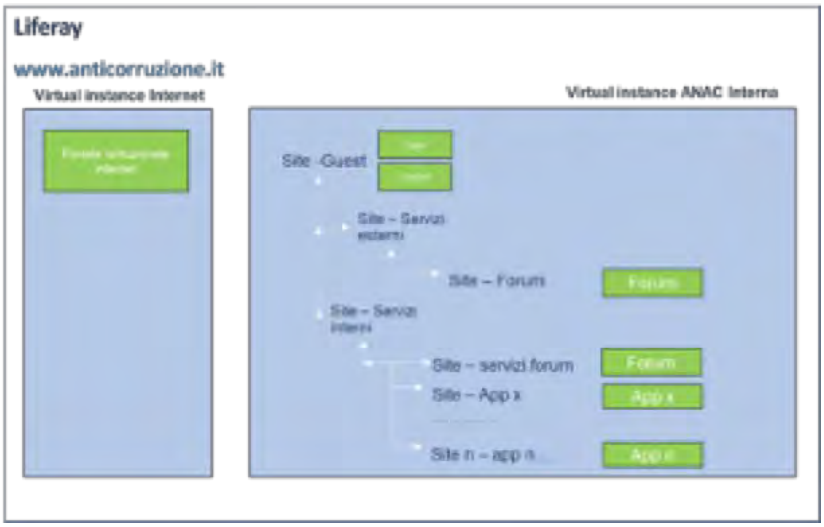


image-20200521171432763

Tecnologie

Di seguito sono elencate le tecnologie per lo sviluppo delle user-Interface

Tecnologia	Logo	Funzionalità	Versione
Liferay		Digital Experience Platform	7.1 SP3
Angular		Single Page Application	7.2.0
Bootstrap		Responsive Design	4.3.1
CSS3/HTML5		Design	nd
React		Single Page Application	15.3.1

Api Gateway

L'API Gateway è l'entry point per l'accesso ai servizi. Espone le api rest dei microservizi applicativi sia agli end-user che agli enti cooperanti. l'API Gateway:

- accetta le chiamate alle API e le indirizza verso i servizi di backend;
- verifica che l'utente sia autenticato e autorizzato all'utilizzo della risorsa richiesta (JWT, certificati,etc)
- Effettuare l'orchestrazione ed eventualmente la trasformazione delle api
- Produce, ove richiesto, risposte salvate in cache;

- Effettua il logging dei metadati delle chiamate per scopi di analisi

image-20200521171934292

Utilizzare l'API Gateway per:

1. Pubblicare le api rest dei microservizi applicativi da esporre agli enti cooperanti
2. Pubblicare le api rest dei microservizi applicativi da esporre alle Single Page application di Front-en
3. Pubblicare api rest dei microservizi applicati da esporre ad altri microservizi interni alla piattaforma

Nota informativa

Alla data di redazione del documento l'api gateway può essere utilizzato per la cooperazione con enti cooperanti esterni (B2B). È possibile utilizzare l'api gateway anche per la comunicazione B2B tra microservizi

Tecnologie

Tecnologia	Logo	Funzionalità	Versione
WSO2		Api Gateway	2.6

Per le linee guida di sviluppo fare riferimento al documento (**RIF.DOCINT.002**).

Microservizi

I microservizi comunicano utilizzando meccanismi basati sul protocollo **http/rest** o protocollo **grpc**. Di seguito viene rappresentata la struttura di riferimento per la realizzazione dei microservizi:

image-20200521172422643

- **Service Code**: contiene l'insieme delle componenti (Classi) che implementano la logica di business del microservizio.
- **Scheletro del microservizio**: "artifact" (scheletro) di riferimento, che gestisce i problemi trasversali secondo il Pattern "**Microservice chassis**"
- **Database**: layer di persistenza per la memorizzazione delle informazioni inerenti il dominio del microservizio
- **Microservizi Esterni**: i microservizi esterni che comunicano tramite protocollo http/rest. È possibile inoltre contattare tramite api i sistemi trasversali comuni (vedi par. 2.4.5 Componenti trasversali comuni).

Service Code

Di seguito viene rappresentata la struttura logica identificata per lo sviluppo del Service Code

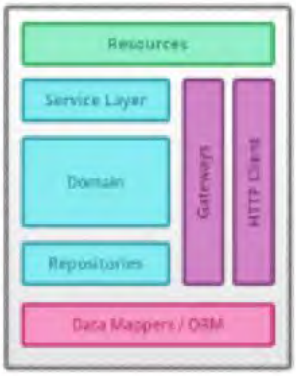


image-20200521172506343

Resources: Le risorse fungono da **mapping** tra il protocollo di integrazione del servizio e gli oggetti che rappresentano il dominio di business. Hanno l'obiettivo di validare le richieste e costruire la risposta in base all'esito della transazione di business.

Business Logic: tutta la logica del servizio deve risiedere in un modello di dominio che rappresenta il dominio di business del microservizio.

- **Il service Layer:** definisce l'insieme di operazioni disponibili e coordina la risposta dell'applicazione per ciascuna operazione. Incapsula la logica di business dell'applicazione, controllando le transazioni e coordinando le risposte.
- **Dominio**: contiene le entità di business
- **Repository:** ha l'obiettivo di separare il dominio del microservizio dalle logiche di accesso ai dati.

Gateway: Se un servizio necessita di contattare altri microservizi, è necessaria una logica per comunicare con il servizio esterno. Il gateway incapsula il body del messaggio che transita tra il

microservizio ed un servizio remoto, effettuando il marshalling di richieste e risposte da e verso gli oggetti del dominio.

HttpClient: protocollo sottostante per gestire il ciclo richiesta-risposta.

Data Mappers/ORM: effettua il mapping tra gli oggetti di business ed il database. In genere per il mapping vengono utilizzati framework dedicati (ORM) richiamati dalle classi "**Repository**".

Scheletro del microservizio

Ogni microservizio, oltre ad implementare la logica di business, deve gestire in modo appropriato tutti gli aspetti trasversali per la corretta erogazione del servizio in produzione.

image-20200521172544273

Di seguito sono elencati gli aspetti trasversali che devono essere presi in considerazione durante lo sviluppo dei microservizi:

- **Controlli di integrità (Health check)**: URL che un servizio di monitoraggio può eseguire per verificare lo stato del servizio
- **Metriche**: misurazioni che forniscono informazioni su ciò che l'applicazione sta facendo e su come sta andando
- **Gestione dei log**: configurazione del framework candidato alla gestione dei log
- **Trace distribuito**: generare un codice identificativo univoco ad ogni richiesta esterna che viene passato tra i servizi.
- **Security**: gestisce le policy di sicurezza per l'accesso al microservizio. Questo layer ha il compito a fronte di una richiesta di recuperare il JWT e verificare che l'utente o il servizio chiamante sia autorizzato.
- **Audit Logging:** contiene le logiche per la gestione dei log di Audit
- **Circuit Breaker:** in caso di irraggiungibilità o di problemi di accesso a servizi esterni, consente di isolare temporaneamente le chiamate ai servizi non raggiungibili evitando la propagazione del problema a tutto il sistema
- **Configurazione esterna:** include credenziali e riferimenti ai servizi esterni come database e broker di messaggi

Di seguito sono elencati i microservizi identificati che possono essere realizzati nell'ambito degli sviluppi applicativi

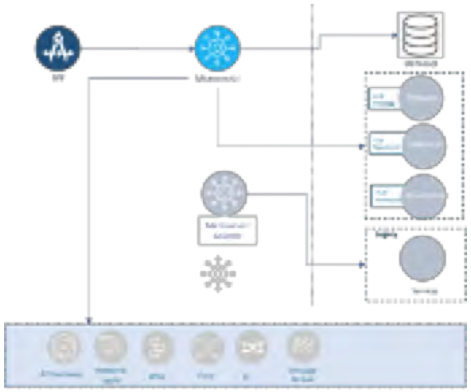


image-20200521172630689

- **BFF (Backend For Front-end)**: microservizi che espongono api rest compatibili al client di riferimento.
- **Microservizi**: servizi applicativi che implementano la business logic e che accedono al database NoSQL. I microservizi possono richiamare anche le api dei servizi trasversali (messaggi, pagamenti, protocollo,etc)
- **Microservizi Adapter**: microservizi dedicati all'integrazione con i servizi legacy

Nota informativa

*Alla data di redazione del documento è possibile implementare tale pattern tramite l'esposizione di api rest sviluppate sul portale **Liferay**.*

Piattaforma di riferimento

La piattaforma target di riferimento per il deploy dei microservizi è **OpenShift**, un platform as a service (PaaS) che eroga servizi basati su tecnologia Containers (*Containers Orchestration*), la loro gestione (*Delivery Management*) e la connessione con funzioni per lo sviluppo basati su incapsulamento a Containers (*CI/CD pipeline instance*s). L'unità di rilascio da utilizzare per i microservizi è "l'application container".

image-20200521173233803

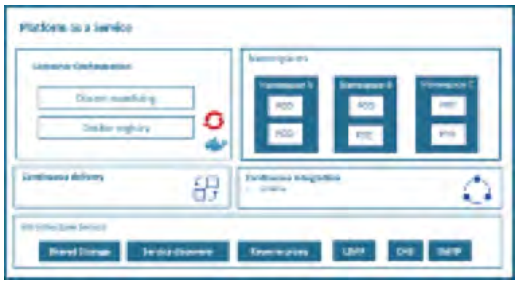


image-20200521173244855

Tecnologie

Framework/Linguaggio	Logo	Versioni
NodeJS		12.x
Spring Boot		2.2.x
Spring Cloud		2.1.x
Spring Data		2.2.5
J2EE		1.8

Servizi dati

Rappresenta lo strato incaricato della gestione e la persistenza dei dati. Tutti i microservizi devono essere progettati per mantenere **privati** i propri dati e rendere accessibile le informazioni solo tramite l'esposizione di **API REST** o protocollo **grpc**.

Di seguito sono elencate le tecniche da adottare per ottenere la separazione dei dati tra microservizi

- *Database Redazionale - Schema per service*: ogni servizio ha uno schema privato di un database
- *Database NoSQL – Database per service*: ogni servizio ha un database dedicato contenente una o più collection specifiche del suo dominio di business. Saranno valutate con L'Amministrazione eventuali esigenze che richiedono la creazione di più collection per il singolo microservizio.

Tecnologie

Tecnologia	Logo	Funzionalità	Versione
MongoDB	mongoDB.	DB NoSQL	4.2.x
SQL Server		Database relazionale	2018

Componenti trasversali Comuni

Le componenti trasversali sono costituite da servizi e librerie ad uso comune della nuova piattaforma architetturale:

BPM

Il Business Process Management deve essere utilizzato nei seguenti casi:

- gestire il ciclo di vita di un processo complesso o long-running, composto da task di sistema o da un'azione dell'utente;
- quando i processi di business possono variare nel tempo;

- quando è necessario automatizzare e monitora i processi di business;
- quando i processi di business prevedono diversi stati e richiedono azioni approvative da parte dell'utente finale

L'integrazione dei processi BPM con i microservizi ANAC deve essere realizzata mediante l'utilizzo delle API Rest/JSON offerte dalla piattaforma

image-20200525151758836

Enterprise Integrator

Abilita la comunicazione tra le diverse componenti della piattaforma e i sistemi esterni attraverso una moltitudine di protocolli e formati di messaggio.

L'enterprise integrator deve essere utilizzato:

· Necessità di effettuare operazioni di message transformation quali ad esempio da SOAP a REST

· Necessità effettuare operazioni di protocol transformation

· Necessità di integrarsi con enti cooperanti esterni

L'integrazione dell'enterprise integrator con i microservizi ANAC deve essere realizzata attraverso API **Rest/JSON**

image-20200525152141671

Message Broker

Permette lo scambio di eventi tra i micro-servizi in near-real-time secondo il pattern Publish/Subscribe

Il message broker deve essere utilizzato:

- Necessità di notificare una variazione ad altri microservizi o sistemi trasversali della piattaforma.

image-20200525152840865

Motore di regole

Il motore di regole consente di definire e modificare esternamente le regole di business senza dover modificare necessariamente l'applicazione o dover interrompere l'esecuzione della stessa.

Il Rule Engine sarà utilizzato nei seguenti casi:

- modifiche frequenti dei sistemi di business che richiedono risposte e attenzione immediate;
- decisioni estremamente variabili che richiedono l'automazione per supportare il modello di business aziendale e le offerte di prodotti/servizi.

image-20200525153049058

Search Engine

Il Search Engine permette di eseguire ricerche complesse (aggregazioni, filtri, where condition articolate) su un dominio di dati complesso, superando le limitazioni dovute al contesto limitato imposte dall'architettura a microservizi.

Il Search Engine sarà utilizzato nei casi in cui è necessario effettuare l'indicizzazione e ricerche complesse sui dati di business

image-20200525153158077

Profiling

I servizi ad accesso riservato prevedono che l'utente abbia superato preventivamente la fase di autenticazione e autorizzazione. Per consentire ciò, l'Autorità dispone di una propria infrastruttura di autenticazione e autorizzazione denominata IAM (Identity e Access Management), il quale gestisce, in modo centralizzato, gli accessi alle applicazioni, attraverso un servizio di autenticazione centralizzato LDAP, esposto tramite interfaccia REST, per la gestione delle credenziali di accesso degli utenti ,e mediante un servizio centralizzato di gestione dei permessi di accesso alle funzionalità di ogni singola applicazione, esposti tramite interfaccia REST .

Logging & Monitoring

La componente di Logging & Monitoring è dedicata alla raccolta, conservazione e storicizzazione dei log applicativi prodotti dalla piattaforma allo scopo di diagnosticare eventuali malfunzionamenti e prevenire fault di sistema.

image-20200525153220525

Le informazioni di interesse estratte dai log sono accessibili al personale tecnico abilitato alla consultazione dei Log tramite un'unica interfaccia grafica centralizzata. Inoltre, consente di configurare servizi di alerting per consentire di pre-allertare il personale al verificarsi di un evento critico (ad es. l'indisponibilità di un determinato servizio).

I singoli servizi forniscono i dati relativi alle metriche di monitoraggio tramite meccanismi di push o di pull. Il sistema di monitoraggio aggrega ed elabora questi dati per fornire una vista complessiva dello stato del sistema.

Tecnologie

Tecnologia	Logo	Funzionalità	Versione
Activity	Immagine che contiene disegnando, orologio, luce, segnale Descrizione generata automaticamente	BPM	1.6.4
WSO2		Enterprise Integrator	
ELK		Logging	
RedHat AMQ Stream RedHat AMQ	img	Message Broker	
Drools	img	Motore di regole	
ElasticSearch	img	Search Engine	
Sonarqube Git Nexsus Jenkins	Immagine che contiene disegnando, cibo Descrizione generata automaticamente Immagine che contiene disegnando Descrizione generata automaticamente Immagine che contiene disegnando Descrizione generata automaticamente img	CI/CD	nd

Processo di rilascio

Nel capitolo viene descritto il processo di rilascio da utilizzate nell'ambito di ogni singolo intervento realizzato in Autorità. Verranno descritte, oltre alle metodologie di sviluppo, anche le infrastrutture tecnologiche, che possono variare da sistema a sistema a seconda delle specifiche esigenze e caratteristiche, in modo da avere una visione globale dell'intero processo che il fornitore è tenuto a seguire.

Premessa

L'applicazione va intesa come l'insieme di tutti i componenti hardware e software, e le relative configurazioni, necessari per rendere disponibili all'utente le funzionalità per lo svolgimento della propria attività. Il ciclo di vita dell'Applicazione nasce con il Business Case e, una volta approvato, continua con lo sviluppo ed il rilascio, ma non termina con la conclusione del progetto, l'applicazione viene gestita come un prodotto/servizio fino a quando non viene dismessa. Ciascun obiettivo è assimilabile ad un "progetto", la cui esecuzione è suddivisa nelle fasi, indicate dal ciclo di vita applicato, che richiedono la realizzazione di specifici prodotti. Si riassumono di seguito le fasi che caratterizzano l'intero ciclo di vita dell'applicazione:

- Gestione progetto: definizione dell'obiettivo e caratteristiche dell'intervento (ad es. manutenzione evolutiva), vincoli di progetto (tempo, costi, normative, etc.), descrizione delle esigenze.
- Ciclo di sviluppo software e l'integrazione del sistema: raccolta dei requisiti, analisi, progettazione, sviluppo e integrazione del software e dell'infrastruttura, test per la verifica continua della qualità.
- Collaudo e Rilascio: validazione dell'applicazione con il coinvolgimento dell'utente e successivo rilascio in ambiente di esercizio.
- Esercizio: monitoraggio e manutenzione dell'applicazione, gestione delle anomalie durante l'esercizio, gestione delle configurazioni.

Per quanto riguarda il processo di sviluppo si può fare riferimento al relativo documento (**RIF.DOCINT.004**) tenendo conto che ANAC sta revisionando la sua organizzazione ed il processo di sviluppo con particolare attenzione all'inserimento delle metodologie di sviluppo Agile.

Ambienti di rilascio

Per poter gestire correttamente l'intero ciclo di vita del software prodotto e le diverse modalità di erogazione dei servizi sono presenti in Autorità degli ambienti logici, ognuno con un compito ben definito. Di seguito sono descritti tali ambienti con particolare riferimento ai ruoli che ricoprono nel processo di rilascio.

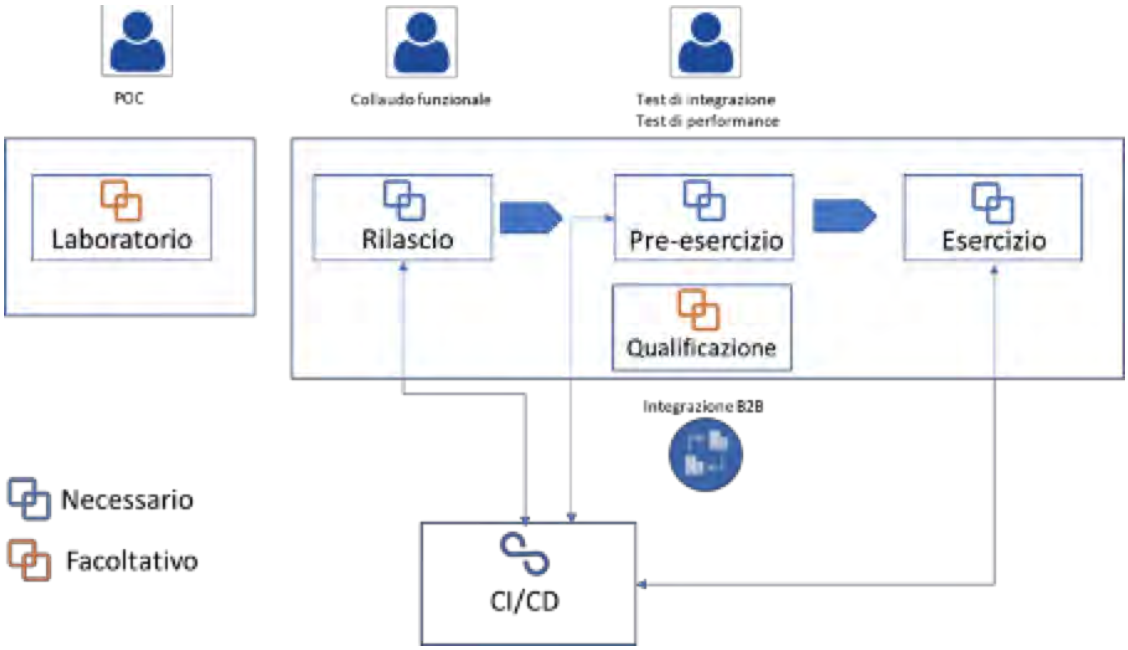


image-20200525153700727

- **Ambiente di laboratorio**: è un ambiente utilizzato per gli sviluppi interni dell'Autorità, le POC e le sperimentazioni di nuovi prodotti e tecnologie.
- **Ambiente di rilascio**: Viene utilizzato per istallare il software, ottenuto attraverso il processo di build interno dell'Autorità a partire dai sorgenti rilasciati dal fornitore, sul quale vengono eseguite le attività di testing delle funzionalità implementate.
- **Ambiente di pre-esercizio**: viene utilizzato prima dell'effettivo rilascio in esercizio per consolidare il pacchetto di rilascio, eseguire i test di integrazione e di carico/prestazionali ove necessario e previsto.
- **Ambiente di esercizio**: È l'ambiente principale dove vengono istallate le componenti software che devono essere esercite, ottenute a valle della fase di realizzazione e il cui esito delle fasi di testing è positivo.
- **Qualificazione**: ambiente completo finalizzato a verificare l'integrazione B2B con gli enti cooperanti Comprensivo, ove previsto, di servizi applicativi per la verifica tramite Front-end dei dati scambiati, durante il test di integrazione, con l'ente cooperante.

Metodologie e prodotti utilizzati

Per gestire l'intero ciclo di vita del progetto l'Autorità utilizza una piattaforma Application Lifecycle Management (ALM), costituita da un insieme di software opensource, finalizzata a gestire ed automatizzare l'intero processo di sviluppo e di rilascio in ottica *DevOps*. La piattaforma ALM rappresenta l'unione delle attività di gestione con le attività di ingegneria del software, resa possibile dall'utilizzo di strumenti che facilitano la gestione delle fasi di: analisi dei requisiti, progetto architetturale, sviluppo, testing, gestione delle release, del change e del deployment. Un processo efficace di Application Lifecycle Management, supportato da strumenti adeguati può:

- Aumentare la produttività, poiché il gruppo di lavoro condivide le pratiche di sviluppo e deployment, e gli sviluppatori possono focalizzarsi sui requisiti di business correnti.
- Migliorare la qualità, in modo che il prodotto applicativo soddisfi i bisogni e le aspettative degli utenti
- Eliminare le barriere di comunicazione attraverso l'efficace collaborazione ed il facile flusso di informazioni
- Accelerare lo sviluppo semplificando l'integrazione
- Tagliare i tempi di manutenzione sincronizzando lo sviluppo e la progettazione
- Massimizzare l'investimento sulle capacità, sui processi e sulle tecnologie
- Aumentare la flessibilità, riducendo il tempo ed i costi di integrazione con nuove applicazioni che soddisfano nuovi requisiti di business

image-20200525153854203

Di seguito verranno descritte le singole componenti della piattaforma adottata da ANAC e le metodologie utilizzate.

Requirement Manager

Il sistema di requirement management, integrato nella piattaforma **Easy Redmine**, consente l'inserimento dei requisiti utente come issue di progetto. Su ogni item il committente ed il team di progetto dell'Autorità hanno la possibilità di aggiungere commenti e aprire una discussione finalizzata al consolidamento del requisito stesso. Il sistema consente anche la gestione dei casi d'uso e dei relativi casi di test.

Knowledge Manager

Il sistema di Knowledge management ha come finalità la gestione e la condivisione della conoscenza attraverso un insieme di strategie e metodi per identificare, raccogliere, sviluppare, conservare e rendere accessibile la conoscenza delle persone che fanno parte di una organizzazione. Attraverso

la piattaforma **Easy Redmine** l'Autorità per ogni singolo progetto raccoglie la documentazione ne gestisce un wiki ed un forum.

Source Manager

Il controllo di versione del codice è un sistema che registra, nel tempo, i cambiamenti ad un file o ad una serie di file, così da poter richiamare una specifica versione in un secondo momento. Questo approccio consente allo sviluppatore di ripristinare i file ad una versione precedente, ripristinare l'intero progetto a uno stato precedente, revisionare le modifiche fatte nel tempo, vedere chi ha cambiato qualcosa che può aver causato un problema, l'Autorità dispone nella sua piattaforma ALM dei seguenti sistemi di Versionamento:

- GIT: Repository GIT basato su **GitLab Comunity Edition**
- SVN: Repository SVN in via di dismissione utilizzato per gestire i progetti legacy

Tali strumenti dovranno essere utilizzati per tracciare tutte le versioni consolidate e candidate all'installazione dei vari ambienti.

Dependency & build Management

Attraverso la piattaforma è possibile:

- Gestire i servizi: progetti e attività possono essere facilmente inserite, assegnate alle risorse e monitorate.
- Gestire le risorse: comunicare ed assegnare i lavori alle risorse, sapere sempre chi fa cosa e quando.
- Gestire budget e costi: ogni risorsa che viene impiegata su una commessa apporta il proprio costo.
- Osserva lo stato di salute delle tue commesse in tempo reale.

L'Autorità dispone di una piattaforma basata su **Easy Redmine**. Nella piattaforma dovranno essere inseriti e gestiti:

- I requisiti del progetto raccolti e consolidati dall'Autorità
- La documentazione prodotta
- il wiki di progetto
- Forum di progetto
- Le versioni del servizio rilasciate ed in corso di rilascio con il riferimento ai requisiti presenti ed alle eventuali correttive.
- il collegamento al repository dei sorgenti

Support ticket

Il sistema gestisce, registra e traccia le richieste di assistenza o di problemi. I sistemi di registrazione delle richieste vengono usati spesso dai centri di assistenza telefonica, per registrare, aggiornare e risolvere i problemi segnalati dai clienti o dagli stessi impiegati dell'azienda. Rientra nel novero degli strumenti utilizzati per interagire con il cliente. Per questa funzionalità l'Autorità utilizza la piattaforma **OTRS**. Se il ticket viene identificato come anomalia software viene migrato attraverso un apposito connettore sulla piattaforma **Easy Redmine** come ticket di SUP che il fornitore deve valutare se si tratta di un evolutiva (MEV) o di una correttiva (MAC)

Bug tracking

Un sistema di bug tracking è un applicativo che consente di tracciare le segnalazioni di bug all'interno dei software, in modo che tali errori siano mantenuti sotto controllo, con una descrizione della riproducibilità e dei dettagli ad essi correlati, e dunque più facilmente risolvibili. A tal fine l'autorità utilizza le funzionalità disponibile nel prodotto **Easy Redmine**

Source Control

Il controllo di versione del codice è un sistema che registra, nel tempo, i cambiamenti ad un file o ad una serie di file, così da poter richiamare una specifica versione in un secondo momento. In questo modo lo sviluppatore potrà ripristinare i file ad una versione precedente, ripristinare l'intero progetto a uno stato precedente, revisionare le modifiche fatte nel tempo, vedere chi ha cambiato qualcosa che può aver causato un problema, l'Autorità dispone nella sua piattaforma ALM dei seguenti sistemi di Versionamento:

- Repository GIT basato su GitLab Comunity Edition
- Repository SVN basato su Collabnet

Tali strumenti dovranno essere utilizzati per tracciare tutte le versioni consolidate e candidate all'installazione dei vari ambienti.

Sistema di build automation e Testing

L'Automazione dello sviluppo è l'atto di scrivere o automatizzare un'ampia varietà di compiti che gli sviluppatori software fanno nelle loro attività quotidiane di sviluppo come:

- Compilazione del codice sorgente in codice binario;
- Gestione delle dipendenze
- Pacchettizzazione del codice binario;
- Esecuzione di test unitari;
- Deployment di sistemi di produzione;
- Creazione di documentazione e/o note di rilascio.

Ogni progetto java deve utilizzare il framewotk **MAVEN** per il build automation dei progetti e per la gestione delle dipendenze. Maven usa un costrutto conosciuto come Project Object Model (POM);

un file XML che descrive le dipendenze fra il progetto e le varie versioni di librerie necessarie nonché le dipendenze fra di esse. In questo modo si separano le librerie dalla directory di progetto utilizzando questo file descrittivo per definirne le relazioni. Maven effettua automaticamente il download di librerie Java e plug-in Maven dai vari repository definiti scaricandoli in locale o in un repository centralizzato lato sviluppo.

L'Autorità dispone di un repository interno basato su **Nexus** dove sono memorizzate tutte le librerie presenti in produzione. Tramite MAVEN è possibile eseguire i test unitari realizzati a copertura dei casi d'uso previsti nel progetto.

Sistema di continuous integration

L'integrazione continua (continuous integration in inglese, spesso abbreviato in CI) è una pratica che si applica in contesti in cui lo sviluppo del software avviene attraverso un sistema di versioning. Consiste nell'allineamento frequente dagli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso (mainline). L'autorità utilizza **Jenkins** come piattaforma di CI integrata con i repository di codice sorgente GIT e con il repository di artefatti **Nexus**. Ad ogni Commit eseguito sulla piattaforma il sistema di CI esegue l'analisi statica del codice la build tramite Maven ed i test unitari (JUnit).

Sistema di continuous deployment

I sistemi di continuous deployment nascono come estensione del processo di continuous integration con l'obbiettivo di minimizzare il tempo di rilascio in produzione di una nuova versione del software. Per raggiungere l'obbiettivo atteso il team di progetto utilizza un'infrastruttura che automatizza i vari passaggi che portano al rilascio in produzione.

Sistema di Knowledge management

Il sistema di Knowledge management ha come finalità la gestione e la condivisione della conoscenza attraverso un insieme di strategie e metodi per identificare, raccogliere, sviluppare, conservare e rendere accessibile la conoscenza delle persone che fanno parte di una organizzazione. A tal fine l'autorità utilizza le funzionalità disponibile nel prodotto **Easy Redmine**

Sistema di misurazione della qualità del software

Il sistema consente di verificare in maniera automatica la qualità del software rilasciato dal fornitore sul repository secondo gli standard (RIF.EXT.002) La piattaforma utilizzata dell'Autorità è **Sonar QUBE**.

Fasi del processo di CI/CD

Di seguito sono descritte le fasi previste per la Continuous Integration/Continuous Delivery predisposte in ANAC.

image-20200525155310521

Il codice sorgente viene inserito nel sistema di controllo della versione del codice (git) dal team di sviluppatori quindi tramite strumenti di CI viene compilato automaticamente. Una volta terminata la fase di creazione, si passa alla fase di test. In questa fase, vengono eseguiti diverse tipologie di test (test di qualità del codice sorgente, Test unitari, etc.). Al termine del test, si passa alla fase di deploy , in cui viene distribuito sull'ambiente di Rilascio per i test funzionali. Quindi viene effettuato il deploy sull'ambiente di pre-esercizio per effettuare i test di integrazione e di performance. Se tutto è accettato, può essere distribuito in esercizio. In ogni passaggio, se si verifica un errore, è possibile inviare un feedback al team di sviluppo in modo che possano risolverlo. Questo ciclo di vita continua fino a quando tutti i test previsti nel ciclo di CI/CD vengano superati per consentire la distribuzione in esercizio.

Fasi del processo

Per ogni nuovo sviluppo deve essere richiesta all'Autorità la creazione di un progetto all'interno del Respository git.

L' URL da utilizzare per l'accesso al repository è il seguente: http://git.lab.anticorruzione.local/. Per accedere al è necessario essere in possesso di un'utenza di Dominio da richiedere all'Autorità.

Ogni nuovo progetto, in base alla tipologia di sviluppo da realizzare, deve essere associato ad uno dei seguenti Gruppi preimpostati:

- Gruppo microservizi
- Gruppo front-end
- Gruppo Database

  per specifiche esigenze di progetto posso essere richiesti e creati altri gruppi da concordare con l'Autorità

Di seguito viene rappresentato il flusso di rilascio previsto per lo sviluppo ed il deploy dei servizi applicativi nell'ambito della piattaforma di riferimento.

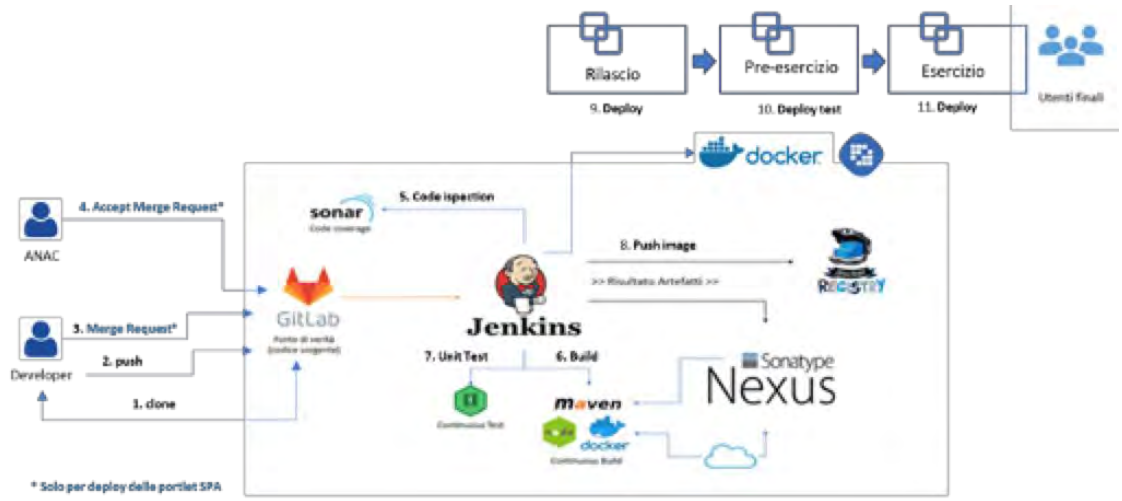


image-20200525155329180

Prima di effettuare il deploy sugli ambienti target è necessario effettuare le configurazioni di tutte le componenti trasversali comuni identificate in fase di progettazione dei servizi applicativi. Il fornitore dovrà documentare tutte le configurazioni preliminari previste in fase di deploy.

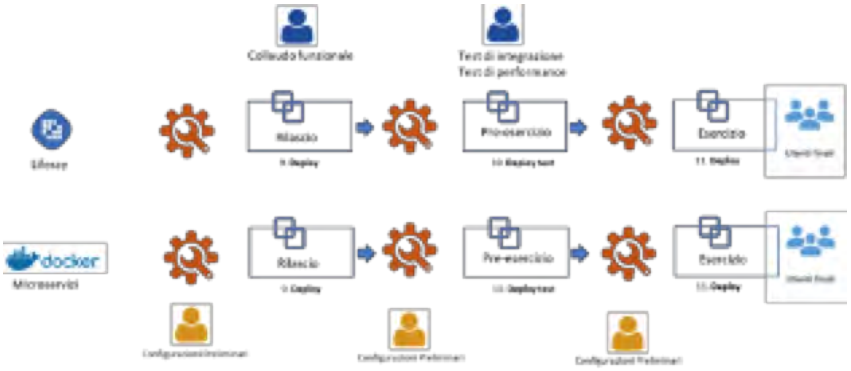


image-20200525155346236

Source Control Management

Ogni repository GitLab presente nell'infrastruttura ANAC è strutturato in tre branch:

- Master (Esercizio)
- Release (Pre-esercizio)
- Develop (Rilascio)

Ogni implementazione software relativa ad un repository prevede la creazione/allineamento di un repository "**GitLab Fork**" (nel seguito semplicemente Fork) sul quale il branch di **master** e quello di **release** sono bloccati. Sul Fork è previsto l'allineamento del branch di **develop** con quello di partenza attraverso una pull request con richiesta di approvazione da parte dell'Autorità.

Il branch di **develop** del Fork viene allineato con il codice del fornitore, all'interno del quale partiranno i processi di sviluppo del software.

In fase di rilascio, il fornitore colleziona tutte le modifiche (feature) da portare nel rilascio mediante allineamento con il "**Fork**". In questa fase è possibile verificare eventuali problemi di integrazione al momento delle commit dovuti a eventuali sviluppi contemporanei. Nella commit deve essere riportata la issueID relativa alle feauters di riferimento (es. #123 Closed).

Attraverso una pull request sul branch di **develop** del repository di ANAC si attivano le pipeline di CI e CD che procedono alla verifica del codice e, in caso di verifica positiva, al deploy nell'ambiente di riferimento.

Il modello del Source Control Management è rappresentato dalla figura seguente

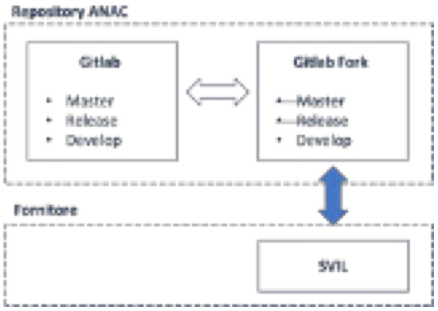


image-20200525155417542

GIT FLOW Ambienti ANAC

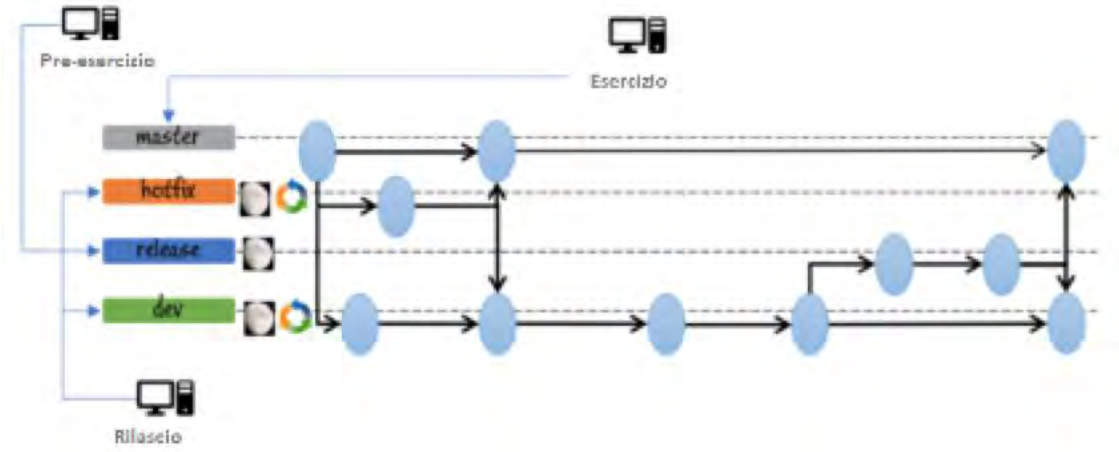


image-20200525155440561

Obbiettivo dell'utilizzo del GIT FLOW è il puntuale riconoscimento di quale versione del codice sorgente stia girando in ciascun ambiente predisposto ad ospitarlo, garantendo l'allineamento del codice Sorgente agli ambienti di **rilascio**, **pre-esercizio** ed **esercizio**.

Attraverso il controllo dell'aggiornamento del codice sui branch sul repository di ANAC e gli automatismi di deploy, possiamo così dire che a ciascun ambiente corrisponde il codice sul corrispettivo branch.

Nel particolare al branch di "**develop**" corrisponde l'ambiente di "**rilascio**", al branch di "**release**" quello di **pre-esercizio**, mentre al "**master**" fa riferimento l'ambiente di "**esercizio**".

Per quanto riguarda gli **hotfix**, faranno riferimento all'ambiente di "**pre-esercizio**".

Bloccando lo sviluppo diretto sui rispettivi branch inoltre abbiamo la certezza dell'allineamento di quanto in esecuzione sui vari ambienti, a meno ovviamente di configurazioni specifiche per ciascun ambiente e diversità di versioni.

Ambienti

La figura seguente illustra lo schema architetturale per la CI/CD in ambiente ANAC.

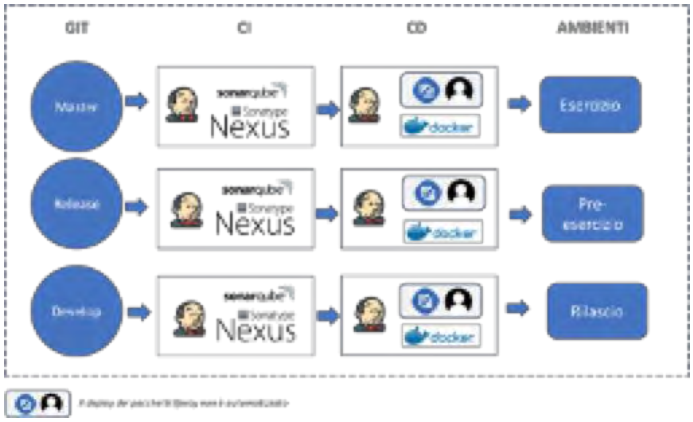


image-20200525155458779

Continuous integration

Il processo di CI comprende strumenti automatici che verificano la correttezza del codice sorgente attraverso strumenti di Continuous Inspection e di integrazione. Ogni integrazione è verificata da una build automatizzata per rilevare gli errori di integrazione. La Continuous Integration implementa una pipeline che ha inizio dal commit del codice sorgente nel sistema.

I passi della pipeline sono i seguenti:

- checkout del codice e analisi statica
- build e test unitari e contract test
- impacchettamento e distribuzione
- test di integrazione

Se una delle fasi fallisce, si riparte dalla modifica del codice.

A valle del rilascio del codice su ciascun branch (**develop**, **release**, **master**) è prevista l'esecuzione di una prima pipeline su Jenkins, di integrazione, che prevede:

- la compilazione e creazione dell'artefatto: Gitlab si preoccupa di innescare jenkins avvisandolo della disponibilità di una nuova versione del codice. Jenkins una volta attivato si occupa di recuperare il codice dal **branch** aggiornato ed avvia la parte di compilazione del codice e creazione dell'artefatto;

- la gestione delle dipendenze ed il versionamento dell'artefatto prodotto: Nexus consente di recuperare le dipendenze alla corretta versione e gestire eventuali vulnerabilità di tali dipendenze, consentendone l'esclusione dallo sviluppo. Inoltre, permette il versionamento dell'artefatto creato.
- l'esecuzione del controllo di qualità del codice: SonarQube riconosce il linguaggio di programmazione utilizzato, ne esegue l'analisi qualitativa e restituisce un report con l'elenco dei problemi riscontrati.
- Test unitari
- generazione dell'immagine docker con id identificativo uguale a quello dell'artefatto per poter avere una coerenza tra artefatto contenuto ed immagine. A valle si eseguirà un push sul docker registry.
- Test funzionali su interfaccia di servizio
- Test sull'interfaccia grafica tramite lo strumento Selenium

Ad ogni rilascio in esercizio verrà creato un tag del branch master sul repository git che ne identifica la versione rilasciata.

Continuous deployment

Il CD utilizza la CI come base per la generazione degli artefatti, in modo da poter a valle automatizzare il suo rilascio fino alla produzione assicurandosi e raggiungendo la condizione che qualsiasi artefatto sia pronto per il suo rilascio in produzione in qualsiasi momento. Con questo automatismo ad ogni commit del codice si avvia la pipeline jenkins portando rapidamente in esercizio le modifiche testate. Alla conclusione della fase di integrazione avvenuta sui branch: **develop**, **release** e **master**, viene avviata la pipeline di deploy che, agganciando Jenkins a OpenShift permette di eseguire il deploy delle risorse utilizzando l'immagine precedentemente creata.

Continuous Code Inspection

La Code Inspection è abilitata dal corretto utilizzo degli strumenti e dei processi messi a disposizione e descritti nei paragrafi precedenti; durante lo sviluppo il programmatore è quindi in grado di rendersi conto continuamente, attraverso i report generati nelle pipeline da SonarQube e Jenkins, della qualità del codice prodotto prima che questo arrivi in esercizio. Inoltre, attraverso il test continuo si evita il verificarsi di regressione nello sviluppo, l'esecuzione di regression test nella pipeline evita che lo sviluppo di nuove feature comprometta il funzionamento di feature già consolidate.

Scenari implementativi

Nel seguito si illustrano gli scenari implementativi del processo descritto in precedenza.

Scenario 1 – Implementazione nuovo componente

Ad ogni nuovo componente software corrisponde un solo repository GitLab e l'implementazione di tale componente avviene in un unico Fork

In questo scenario gli attori coinvolti sono:

- Release Manager per l'ambiente ANAC
- Team leader per l'ambiente di sviluppo del fornitore

Predisposizione Sviluppo

- Step predisposizione sviluppo a carico del Release Manager (**ANAC**):
  - Creazione del repository in GitLab ANAC
  - Creazione dei branch Master – Release – Develop
  - Creazione del Fork a partire dal repository ANAC
  - Assegnazione dei permessi sul repository del Fork
  - Blocco branch Master - Release sul Fork
- Step predisposizione sviluppo a carico del Team Leader (**Fornitore**):
  - Download del Fork creato a partire dal branch Develop

Rilascio

- Step rilascio a carico del Team Leader (**Fornitore**):
  - Allineamento sorgente fornitore con branch Develop Fork; l'allineamento del branch Develop Fork scatena la Continuous Integration con l'esecuzione di:
    - Build del codice
    - Unit/regression test
    - Code Analisys
  - Verifica report continuos integration
  - Consegna check list di rilascio
  - Allineamento branch Fork con branch Develop ANAC mediante pull request
- Step rilascio a carico del Release Manager (**ANAC**):
  - Approvazione pull request; passaggio in ambiente di collaudo del codice sul branch di develop mediante continuous integration e continuos deploy
  - Passaggio dal branch develop al branch release; scatena la continuous integration e richiesta di approvazione per passaggio in ambiente di pre-esecizio
  - Passaggio dal branch release al branch master; scatena la continuous integration e richiesta di approvazione per passaggio in ambiente di esercizio

Scenario 2 – Modifica componente esistente

La modifica di un componente software esistente prevede la presenza di un Fork di sviluppo da allineare con la versione del software presente sul branch Develop di ANAC.

In questo scenario gli attori coinvolti sono:

- Release Manager per l'ambiente ANAC (**ANAC**)
- Team leader per l'ambiente di sviluppo del fornitore (**Fornitore**)

Predisposizione Sviluppo

- Step predisposizione sviluppo a carico del Release Manager (**ANAC**):
  - Allineamento del Fork a partire dal branch di Develop del repository ANAC
  - Verifica i permessi sul repository del Fork
- Step predisposizione sviluppo a carico del Team Leader (**Fornitore**):
  - Download del Fork creato a partire dal branch Develop

Rilasci

- Step rilascio a carico del Team Leader (**Fornitore**):
  - Allineamento sorgente fornitore con branch Develop Fork; l'allineamento del branch Develop Fork scatena la Continuous Integration con l'esecuzione di:
    - Build del codice
    - Unit/regression test
    - Code Analisys
  - Verifica report continuos integration
  - Consegna check list di rilascio
  - Allineamento branch Fork con branch Develop ANAC mediante pull request
- Step rilascio a carico del Release Manager (**ANAC**):
  - Approvazione pull request; passaggio in ambiente di "**rilascio**" del codice sul branch di "**develop**" mediante continuous integration e continuos deploy
  - Passaggio dal branch **develop** al branch **release**; scatena la continuous integration e richiesta di approvazione per passaggio in ambiente di **pre-esercizio**
  - Passaggio dal branch **release** al branch **master**; scatena la continuous integration e richiesta di approvazione per passaggio in ambiente di **esercizio**

Scenario 3 – Bug Fixing

Il workflow relativo al bug fixing prevede la creazione di un HotFix sul Gitlab di ANAC a partire dal branch su cui si è verificato il bug (Release – Master) da parte del Release Manager (ANAC). Il processo che segue è analogo a quanto descritto nel paragrafo precedente. Il riallineamento del branch di **HotFix** in ambiente ANAC puo' generare conflitti sul branch di **Develop** dovuti ad eventuali sviluppi contemporanei. È a carico del Release Manager la gestione di tali conflitti

Scenario 4 - Liferay User Interface - Workspace

Per lo sviluppo dei moduli che dovranno essere installati sul Liferay è necessario utilizzare il "**Liferay Workspace**" (o Liferay Module SDK) per agevolare la gestione coerente degli artefatti.

Il Liferay Workspace sarà gestito all'interno di un singolo repository git. Sarà un singolo repository perché deve contenere tutti gli artefatti che saranno installati all'interno della medesima installazione di Liferay è che quindi devono essere sviluppati da subito come un gruppo omogeneo.

L'URL per l'accesso al repository è il seguente:http://git.lab.anticorruzione.local/frontend/liferay-workspace

Ogni fornitore lavorerà solo sui moduli inerenti il proprio progetto di riferimento.

In fase di Merge Request ANAC verificherà che le modifiche apportate riguardino solo i moduli del progetto in carico al fornitore. Ogni modulo presente nel repository dispone di un proprio numero di versione composto da tre parti: major, minor e micro. Ogni modulo nasce in versione "1.0.0" ed evolve in modo autonomo secondo una logica di "Versionamento Semantico"

- nella parte micro in caso di fix o cambiamenti ad elementi interni
- nella parte minor in caso di cambiamenti importanti sulle logiche implementative che non impattano sui moduli dipendenti
- nella parte major in caso di cambiamenti che impattano su moduli dipendenti

Come da best-practices il numero di versione viene esposto nel nome dell'artefatto quando questo viene costruito a partire dai sorgenti. Ogni modulo deve avere una nomenclatura ben definita che identifica il progetto di riferimento.

Issue e Branch

Qualsiasi sviluppo deve essere realizzato a partire da una issue censita nel sistema di ticketing di ANAC. Se tale issue coinvolge anche Liferay allora dovrà essere creata una issue specifica anche nel relativo repository gitlab. Anche più di una se necessario. Qualsiasi intervento sul software dovrà essere realizzato attraverso un branch, un feature-branch, che deve referenziare la issue. Ovvero avere un nome nel formato "<id_issue><short_descr>" dove:

- id_issue è il numero della issue
- short_descr è una breve descrizione dell'obiettivo della issue (per breve si intende circa 20 caratteri)

Qualsiasi commit realizzato su tale branch deve iniziare per "Issue #numero:", ovvero deve referenziare la issue che lo ha reso necessario. Eccezionalmente è ammesso l'accorpamento di sviluppi che coprono più issue se questo si rende assolutamente necessario in corso d'opera. In ogni caso il tutto deve essere documentato e motivato nei commenti delle issue coinvolte.

Milestone

Poiché i feature-branch dovrebbero avere una vita breve è importante utilizzare il concetto di milestone offerto da gitlab per raggruppare le issue temporalmente; anche per agevolare l'attenzione su quelle necessarie per il prossimo rilascio. Le milestone infatti consentiranno di gestire anche la cadenza dei rilasci.

Approccio Trunk Based Developmen

Il repository Liferay ed i suoi sviluppi seguiranno un approccio TBD (Trunk Based Development). Dichiarare un approccio non garantisce automaticamente stabilità e qualità del progetto. Come qualsiasi altro approccio, anche TBD richiede la presenza di un "maintainer" ovvero di una figura esperta che monitori l'attività dei gruppi di sviluppo e ne intercetti le deviazioni. L'approccio TBD prevede che:

- il branch master, pur producendo artefatti snapshot, deve contenere sviluppi stabili.
- un rilascio comporta la creazione di un branch dedicato al rilascio, ovvero
  - alla sua fase preparatoria
  - al rilascio vero e proprio o alle correttive di manutenzione

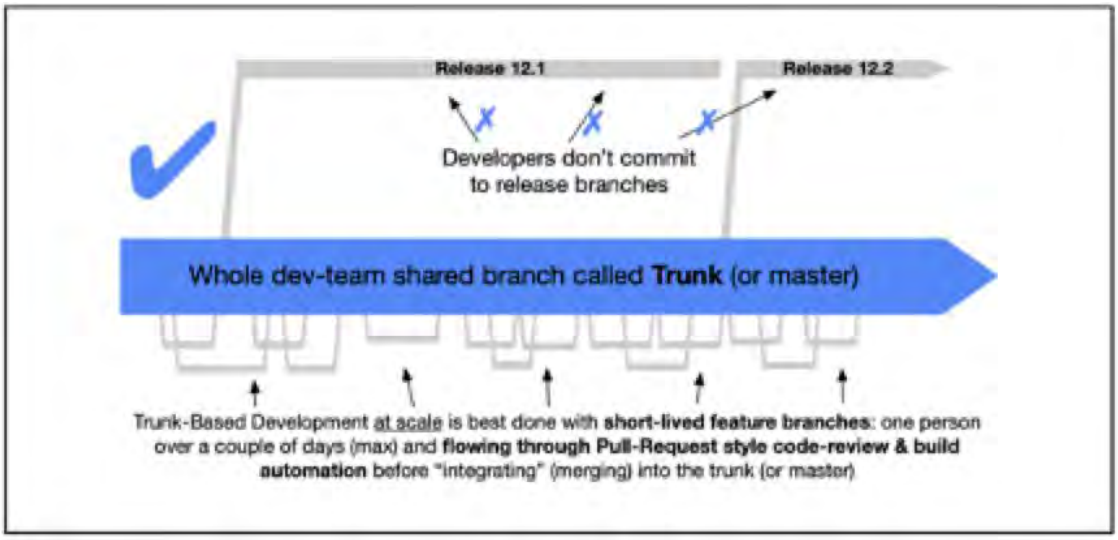


image-20200525155734590

I feature-branch saranno principalmente staccati da master ed i loro sviluppi saranno ricondotti nel master attraverso una merge-request. Uno dei compiti del maintainer è proprio quello di verificare la merge-request e rifiutarla se questa non rispetta le logiche di progetto, come:

- contiene errori di compilazione
- contiene file che non devono essere versionati (es: .project, .classpath)
- contiene conflitti rispetto al branch master
- non rispetta il livello minimo di qualità del codice, e nel caso di sviluppi su Liferay non rispetta le best-practices Liferay

I feature-branch correttivi saranno staccati dal branch di release e serviranno a realizzare correzioni di emergenza. Correzioni che dovranno essere replicate anche nel master, se necessario, per essere così incluse nella successiva release.

DevOps

Alcuni flussi ripetitivi possono essere automatizzati attraverso strumenti come Jenkins. Le pipeline realizzate per replicare le azioni che il maintainer potrà comunque svolgere in modo manuale; ovvero non devono impedire l'operatività manuale se questa si rende necessaria, ovvero per gestire tempestivamente casi non previsti in attesa dell'evoluzione delle pipeline stesse. Le attività automatizzabili sono:

- creazione del branch di rilascio (es: release/v1.0.x).
- creazione effettiva del rilascio base e manutentivo (es: 1.0.0, 1.0.1, 1.0.2, ...). Con conseguente pubblicazione degli artefatti di Artifactory.
- controllo delle merge request. In questa fase è utile inserire strumenti come SonarQube per il controllo del codice.
- creazione del pacchetto di rilascio (es: lo zip con tutti gli artefatti di uno specifico rilascio).

Sebbene l'approccio TBD consenta al singolo sviluppatore di introdurre i suoi sviluppi nel master, questo privilegio rappresenta una criticità se lo skill dello sviluppatore non è in linea con i requisiti richiesti. Infatti, sebbene tutto sia versionato, eventuali instabilità possono non essere di immediata rilevazione, creando criticità più o meno gravi. Come indicato precedentemente è importante definire delle figure esperte (i maintainer) che oltre a sovrintendere le fasi delicate (la costruzione di un rilascio) effettueranno dei controlli a campione sulle merge-request.

Sviluppo distribuito

In presenza di diversi Team di sviluppo distribuiti o eterogenei è consigliabile che questi eleggano un leader che avendo la responsabilità di quello che produce il suo team, avrà cura di verificare e controllare quanto prodotto prima di effettuare la merge-request. In base all'organizzazione interna ANAC, solo loro potranno essere autorizzati confermare una merge-request. ANAC metterà a disposizione dei gruppi di sviluppo un account per poter interagire con il repository gitlab e gli altri strumenti devops coinvolti. Per convenienza un Team esterno può effettuare un fork locale del repository per agevolare il suo team. Ad esempio per sopperire a problematiche di connettività. In questo caso però:

- nei commit la issue riferita sarà quella del git di ANAC,

- deve essere mantenuta la logica "una issue = un feature branch",
- la merge request va effettuata sul repository ANAC. E' altamente sconsigliato effettuare la merge-reguest nel master del fork e poi fare una merge-request del master “forkato” nel master ANAC,
- preferire la merge-request al "replica per diff". La storia dei commit può essere fonte di crescita per gli sviluppatori e strumento di verifica per i maintainer.

Open Api

Per agevolare l'integrazione tra le applicazioni è importante che le API esposte o da consumare siano descritte in una modalità standard. Per le API REST tale standard è OpenAPI 2.0 (oppure OpenAPI 3.0). Standard che Liferay 7.1 supporta attraverso diversi pattern di sviluppo.

Test, Mock, README

Il team di sviluppo deve essere messo in condizione di verificare i propri sviluppi. Significa che per le funzionalità che prevedono di accedere a servizi esterni, il team che sviluppa l'integrazione deve anche realizzare un mock che consente di simulare il servizio esterno e testare l'integrazione senza di esso. Ovviamente il mock deve essere evoluto e manutenuto per garantire un sufficiente grado di affidabilità dei test effettuati con esso. L'utilizzo di Unit Test è importante soprattutto per gli aspetti software backend che esprimono logiche complesse o non facilmente intuibili leggendo il codice. Come altri aspetti gli Unit Test sono utili se non abusati. Per gli aspetti funzionali delicati è sicuramente utile avere uno Unit Test; ma è molto più utile avere un README.md (formato markdown) che ne descriva le diverse sfaccettature, le attenzioni e le criticità.

Requisiti generali di sviluppo

Si indicano i seguenti requisiti di sviluppo che devono essere “**rispettati**” nello sviluppo delle applicazioni e dei servizi:

Requisiti di carattere generale

Codice requisito	Descrizione
REQ.CG.01	Ogni componente realizzato (classe, metodo, variabile, vista, function, ecc) dovrà essere commentata in modo da rendere facilmente individuabile il compito svolto.
REQ.CG.02	Il nome di ogni componente realizzato (classe, metodo, variabile, tabella, vista, function, ecc) dovrà utilizzare la notazione italiana.
REQ.CG.03	Ogni applicazione o servizio deve gestire una KMS degli errori assegnandogli un codice univoco ed una descrizione che identifica la possibile causa e l’eventuale soluzione. IL messaggio restituito dovrà contenere il codice transazione ed il codice identificativo dell’errore.
REQ.CG.04	Al momento della consegna del software, quanto realizzato deve essere compatibile con la release attiva negli ambienti in cui è previsto il deployment.

Requisiti di versionamento

Ogni applicazione, servizio o componente realizzata/o deve essere identificata/o da un numero di versione secondo le seguenti regole:

Codice requisito	Descrizione
REQ.CGVERS.01	Le regole per il versionamento suddividono il numero di versione in major number, minor number e revision number con le seguenti regole per l'identificazione della versione: ● Il "major number" deve essere variato se vengono aggiunte nuove funzionalità core; ● Il "minor number" deve essere variato se vengono aggiunte funzionalità minori e correzioni significative; ● Il "revision number" deve essere variato se vengono corretti bug.
REQ.CGVERS.02	Deve essere sempre presente nelle pagine web delle applicazioni il numero di versione attualmente esercita
REQ.CGVERS.03	Ogni componente applicativa facente parte di un servizio deve stampare al suo avvio sui log il proprio numero di versione

Requisiti per il logging

Codice requisito	Descrizione
REQ.CGLOG.01	Tutte le applicazioni ed i servizi realizzati devono loggare in maniera esaustiva le operazioni effettuate in modo che sia possibile ricostruire le attività dell'utente
REQ.CGLOG.02	Ogni scrittura applicativa nel log deve avere il seguente formato: --{messaggio di testo} dove: ● Il codice transazione identifica la singola request dell'utente ● Il codice sessione identifica la sessione applicativa dell'utente Se il codici transazione o il codice sessione non sono presenti devono essere generati dalle applicazioni
REQ.CGLOG.03	Per ciascun errore/eccezione tracciare a livello error il messaggio di errore e gli oggetti rilevanti che hanno causato l'anomalia
REQ.CGLOG.04	Devono essere tracciati a livello di debug le query, i parametri di invocazione di un servizio, i parametri di apertura di una connessione
REQ.CGLOG.05	Nei log non devono essere presenti informazioni sensibili quali nominativo dell'utente, dati carta di credito ecc.

Requisiti per l'audit

Codice requisito	Descrizione
REQ.CGAUDIT.01	Ogni operazione di inserimento, modifica e cancellazione deve essere tracciata secondo le specifiche definite dall' Autorità

Requisiti per il testing

È necessario garantire che i componenti abbiano un comportamento interno coerente, e che ogni singolo componente si integri correttamente con altri componenti del sistema. Affinché i test siano

efficaci, devono essere parte integrante della pipeline di sviluppo e rilascio (CI/CD). L'esito dei test deve necessariamente abilitare o meno il rilascio del codice sorgente sottoposto a verifica.

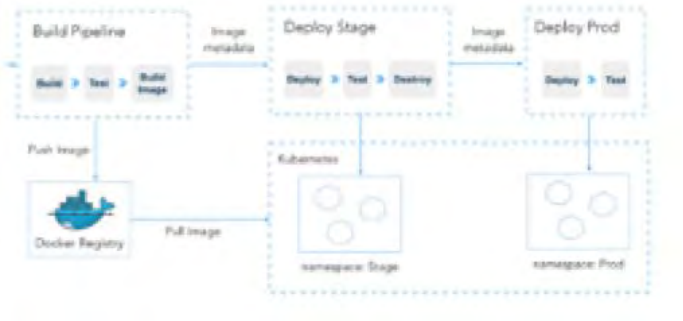


image-20200525160401023

Ogni microservizio prima del deploy viene sottoposto a diverse tipologie di test (automatici) atti a verificare sia la qualità del codice sorgente, tramite strumenti di continuous inspection, sia il corretto funzionamento delle funzionalità implementate. I test automatici sono integrati nel processo di merge del codice sorgente all'interno del repository centralizzato. Quando arriva il momento di effettuare il merge sul ramo master sono effettuati i controlli di stato che prendono il codice ed eseguono i test su di esso. Se i controlli di stato hanno esito positivo, l'unione è abilitata, ma se i controlli di stato falliscono non consentirà il merge e la successiva build dell'immagine docker. Superati i test si procede al deploy automatico sull'ambiente di Rilascio per procedere con i test.

Per ogni applicazione o servizio devono essere rispettati i seguenti requisiti per il testing:

Codice requisito	Descrizione
REQ.CGTEST.01	Per ogni caso d'uso, relativo ad applicazioni o servizi, deve essere realizzato un test automatizzato che consenta di verificare la sua corretta implementazione.
REQ.CGTEST.02	Per ogni modulo facente parte di una applicazione o servizio devono essere realizzati dei test di unità che consentano di verificare la sua corretta implementazione.
REQ.CGTEST.03	Devono essere realizzati servizi mock, ove non disponibili, per simulare il comportamento di un altro oggetto che durante il test sarebbe difficile o oneroso da istanziare
REQ.CGTEST.04	Devono essere predisposti i "**Test di integrazione**" che verificano il comportamento esterno di un singolo servizio. Ad esempio, un servizio che espone API REST deve essere testato inviando richieste HTTP al servizio esterno. ● **Gateway integration test**: per eliminare tutti gli errori a livello di protocollo come intestazioni HTTP mancanti, gestione SSL errata o mancate corrispondenze del body della richiesta e della risposta alla massima granularità possibile. ● **Persistence integration test**: assicurano che lo schema assunto dal codice corrisponde a quello disponibile nell'archivio dati. Nel caso in cui sia in

	uso un ORM, questi test forniscono anche la certezza che eventuali mapping configurati sono compatibili con i set di risultati che ritornano.
REQ.CGTEST.05	Devono essere predisposti i "**Contract Test**" per garantire che i messaggi inviati o ricevuti siano conformi a una specifica condivisa documentata in un "contratto" (swagger).

Requisiti di sicurezza

Di seguito viene fornito l'elenco dei requisiti di sicurezza da mettere in atto. La figura rappresenta uno scenario di riferimento che evidenzia le contromisure (in verde) da adottare per mettere in sicurezza i microservizi.

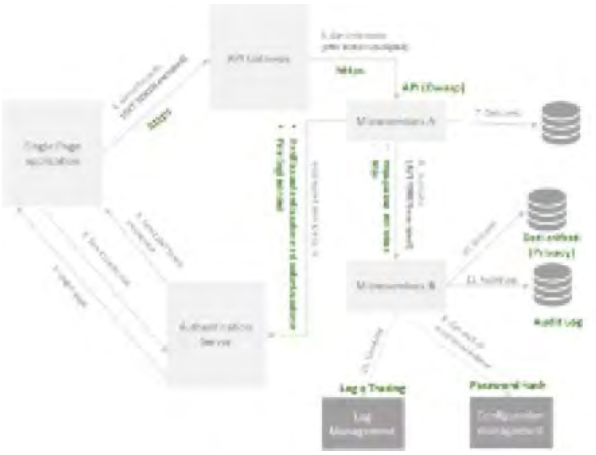


image-20200525173240774

Codice requisito	Descrizione
REQ.CGSEC.01	Le applicazioni ed i servizi realizzati dovranno rispettare i requisiti di sicurezza dettati dal framework OWASP (Open Web Application Security Project)
REQ.CGSEC.02	Sebbene la rete dei microservizi sia protetta dal traffico esterno, qualcuno all'interno della rete potrebbe accedere al traffico tra i servizi. Ove opportuno, occorre utilizzare protocolli di comunicazione cifrati (encrypted), anche all'interno della rete dei microserivizi. Il protocollo TLS fornisce una buona qualità di encryption. Per essere sicuri che una richiesta ricevuta da un utente autenticato non sia stata manomessa, è necessario utilizzare protocolli di comunicazione, come TLS, che facciano uso di meccanismi di integrità, anche all'interno della rete dei microservizi.
REQ.CGSEC.03	I microservizi devono effettuare dei controlli per verificare l'autenticazione e l'autorizzazione delle chiamate che ricevono. Quelli che svolgono azioni particolarmente delicate non dovrebbero accettare chiamate non verificate.
REQ.CGSEC.04	Ciascun microservizio deve verificare sempre l'autorizzazione del chiamante, attraverso il controllo del token di identificazione. Se i microservizi ne chiamano altri, un attaccante riuscirebbe a controllare a cascata gli altri microservizi, nel

caso in cui l'identità del chiamante non venisse verificata ad ogni chiamata ad un API.

REQ.CGSEC.05 In fase di design occorre prevedere di utilizzare una funzione di hash per memorizzare le password

REQ.CGSEC.06 I dati personali devono preventivamente essere anonimizzati, di modo che le informazioni trafugate non siano utili a identificare i soggetti interessati (GDPR).

REQ.CGSEC.07 Per rilevare e analizzare eventuali violazioni della sicurezza, è necessario garantire che i log siano generati in un formato che può essere facilmente utilizzato da una soluzione di gestione dei log centralizzata.

REQ.CGSEC.08 Se i microservizi hanno accesso a database, middleware, sistemi di messaggistica, occorre delimitare gli accessi solo a quei servizi che necessitano di effettuare le operazioni. Evitare di utilizzare un singolo login a questi componenti per tutti i microservizi. I privilegi concessi non possono essere gli stessi per tutti i microservizi.

REQ.CGSEC.09 Al fine di gestire in maniera adeguata gli eventi di "sicurezza" significativi, è obbligatorio includere almeno le seguenti informazioni all'interno dei messaggi di log: ● Timestamp: questo elemento deve essere coerente con la data e ora di riferimento dell'infrastruttura. In caso di timestamp non corretto, le analisi di sicurezza potrebbero essere compromesse al punto da rendere ingestibile un incidente di sicurezza; ● Gravità dell'evento di sicurezza: la gravità di un evento di sicurezza può essere espressa in una scala da 1 a 3 (dove uno rappresenta il valore meno significativo e tre quello più significativo), in cui l'indice di gravità è dato dalle ripercussioni sul sistema e sull'account dell'utente. Ad esempio, un log relativo al blocco dell'utenza potrebbe avere valore 3 (ALTO), un log relativo alla rilevazione di un XSRF bloccato dalla corretta implementazione delle mitigazioni potrebbe avere un valore 1 (BASSO); ● Tagging dei messaggi di sicurezza: i messaggi di log relativi ad eventi di sicurezza devono contenere il tag "" al fine di consentire una rapida individuazione di tali eventi; Informazioni di contesto relative all'origine del log: i messaggi di log devono includere le informazioni relative all'utente che ha generato la violazione di sicurezza, l'indirizzo ip da cui è stata effettuata la richiesta, il browser che è stato utilizzato, gli header della richiesta e gli eventuali parametri che sono stati passati all'applicazione; ● Esito dell'evento di sicurezza: l'esito di un evento di sicurezza può essere positivo o negativo. Ad esempio, il blocco di un utente a seguito di troppi tentativi errati può andare a buon fine o per qualsiasi ragione può fallire, ed in tal caso devono essere intraprese le azioni utili a gestire quanto accaduto; ● Descrizione dell'evento: una descrizione testuale ed esplicativa dell'evento che si sta loggando. È obbligatorio che le applicazioni generino dei log almeno i seguenti eventi di sicurezza: ● Tutte le validazioni dell'input non andate a buon fine; ● Tutti gli accessi al sistema, in particolar modo quelli non andati a buon fine; ● Tutti i tentativi di violazione delle costanti autorizzative; ● Tutti i tentativi apparenti di tampering, inclusi i cambiamenti non attesi dello stato (viewstate, csrf_tokens, ecc); ● Tutti i tentativi di accesso al sistema con identificativi di sessione non validi o scaduti; ● Tutte le eccezioni nelle funzionalità specifiche di

sicurezza inclusi gli errori che si verificano all'interno dei moduli per la gestione della crittografia.

Requisiti non funzionali

Di seguito sono riportati i requisiti non funzionali applicabili a tutti gli interventi previsti nell'ambito de progetto.

Sulla base degli ambiti di progetto e delle aree funzionali, la convenzione per l'identificazione dei requisiti è riportata di seguito.

Ciascun requisito è individuato da un identificativo univoco nella forma [RF/AM_AR.nn] o [RNF/AM_AR.nn], dove:

- **RF** Requisito Funzionale;
- **RNF** Requisito Non Funzionale;
- **AM** identifica l'**ambito;**
- **AR** identifica l'**area funzionale;**
- **nn** è un progressivo numerico all'interno dell'area funzionale.

In particolare, per il presente progetto verranno utilizzati i codici riportati nella tabella sottostante.

Codice Ambito	Descrizione Ambito	Codice Area	Descrizione Area
GNRCROSS	Requisiti non funzionali applicabili a tutti i sistemi realizzati	N.a.	N.a.

Per ogni requisito è previsto un codice di versionamento ed uno stato. Lo stato può assumerei i valori:

- In definizione: il requisito non è ancora approvato
- Approvato: il requisito impegna il fornitore e deve essere realizzato
- Sostituito: il requisito è stato sostituito da una nuova versione approvata
- Annullato: il requisito non è più in vigore.

[RNF-GNRCROSS.1] Attori Activiti

Codice Requisito	Descrizione
RNF.GNRCROSS.01	Gli utenti esterni non sono attori del processo di Activiti ma devono essere "ricondotti" ad unico utente che li rappresenti.

[RNF-GNRCROSS.2] Integrazione Protocollo

Codice Requisito	Descrizione

RNF.GNRCROSS.02	Al fine di garantire l'integrazione con Protocollo, saranno utilizzati i Servizi per il recupero dei Protocolli in ingresso ed in uscita messi a disposizione da ANAC.

[RNF-GNRCROSS.3] Integrazione IAM

Codice Requisito	Descrizione
RNF.GNRCROSS.03	Al fine di garantire l'integrazione con IAM, saranno utilizzati i Servizi per il recupero delle informazioni anagrafiche degli utenti interni ed esterni messi a disposizione da ANAC (RIF.INT.001).

[RNF-GNRCROSS.4] Standard AGID

Codice Requisito	Descrizione
RNF.GNRCROSS.04	Gli sviluppi sono effettuati secondo quanto previsto dalle "Linee guida di design per i servizi digitali della PA" fornite da AgID (RIF.EXT.001)

[RNF-GNRCROSS.5] Integrazione con servizi ejb 3.0 Legacy

Codice Requisito	Descrizione
RNF.GNRCROSS.05	Al fine di garantire l'integrazione con i servizi ejb 3.0 legacy, è necessario sviluppare microservizi Adapter come wrapper ejb 3.1

[RNF-GNRCROSS.6] Integrazione con AMP

Codice Requisito	Descrizione
RNF.GNRCROSS.06	È prevista l'integrazione con la piattaforma AMP per l'invio di email standard ed email PEC non protocollate tramite il microservizio Adapter predisposto per l'integrazione con la piattaforma.

[RNF-GNRCROSS.7] Gestione dei file marcati con Firma Digitale

Codice Requisito	Descrizione
RNF.GNRCROSS.07	Il sistema deve gestire i file firmati digitalmente in formato PAdES e CAdES. In fase di acquisizione di un documento firmato con firma digitale il sistema, se previsto, deve verificare: ● che la firma digitale sia valida mediante l'utilizzo di servizi di verifica remoti; ● qualora il file da firmare sia stato generato dal sistema, che il contenuto del pdf non abbia subito variazioni, oltre la firma

[RNF-GNRCROSS.8] Modello di interoperabilità

Codice Requisito	Descrizione
RNF.GNRCROSS.08	I nuovi servizi devono essere implementati secondo il paradigma API first e devono essere erogabili in modalità B2B via API REST. Il livello di sicurezza è demandato a un sistema esterno.

[RNF-GNRCROSS.10] Firma digitale Remota

Codice Requisito	Descrizione
RNF.GNRCROSS.10	Il sistema, se previsto, deve interfacciarsi con il sistema di firma remota messo a disposizione da ANAC.

[RNF-GNRCROSS.11] Interfaccia Web

Codice Requisito	Descrizione

RNF.GNRCROSS.11	Le funzionalità devono essere messe a disposizione degli utenti esterni ed interni attraverso un'interfaccia web responsive. Le funzionalità previste dal sistema possono essere ad accesso libero o riservato (tramite autenticazione). In quest'ultimo caso, tali funzionalità devono dipendere dai profili dell'utente, ove prevista la profilazione utente.

[RNF-GNRCROSS.12] Integrazione con Alfresco

Codice Requisito	Descrizione
RNF.GNRCROSS.12	Per la gestione dei documenti prodotti o acquisiti dal sistema è utilizzato il document management Alfresco.

[RNF-GNRCROSS.14] Monitoraggio applicativo

Codice Requisito	Descrizione
RNF.GNRCROSS.14	Il servizio/applicazione da integrare nell'ambiente elaborativo ANAC deve essere sviluppato tenendo conto della necessità di monitoraggio dello stesso.

[RNF-GNRCROSS.15] Pulizia disco

Codice Requisito	Descrizione
RNF.GNRCROSS.15	Il servizio/applicazione da integrare nell'ambiente elaborativo ANAC, qualora utilizzi cartelle per il salvataggio temporaneo dei dati, deve provvedere alla loro pulizia periodica. Tali cartelle se previste dovranno comunque essere indicate nel manuale di configurazione e gestione

[RNF-GNRCROSS.16] Utilizzo dello storage

Codice Requisito	Descrizione
RNF.GNRCROSS.16	Il servizio/applicazione da integrare nell'ambiente elaborativo Anac, qualora debba utilizzare cartelle per il salvataggio definitivo dei dati al di fuori dei sistemi documentali e basi dati, deve utilizzare apposite cartelle di rete, al fine di facilitare il backup degli stessi da parte di Anac. I file non dovranno essere salvati in modalità flat in una singola cartella, ma l'applicazione dovrà prevedere la gerarchizzazione in sottocartelle

[RNF-GNRCROSS.18] Scalabilità

Codice Requisito	Descrizione
RNF.GNRCROSS.18	L'applicazione deve essere scalabile per mantenere le prestazioni attese e verificate in fase di test all'aumento del carico nel momento della messa in esercizio e per la durata dello stesso nel tempo anche all'aumento dei dati e al passare del tempo. Al momento della progettazione dev'essere prevista una platea di utenti e un numero di transazioni contemporanee pari a quelle in esercizio, il dimensionamento di progetto e quanto descritto nel documento di installazione e gestione dev'essere idoneo a gestire tali requisiti.

[RNF-GNRCROSS.19] Tempo medio di risposta per elaborazioni sincrone

Codice Requisito	Descrizione

RNF.GNRCROSS.19	Il tempo medio impiegato dal software per rispondere a una richiesta utente o svolgere un'attività di sistema deve essere inferiore a 20 secondi. Tale tempo va calcolato come media tramite test di carico opportunamente definiti nei piani di test e non contempla le tempistiche dipendenti dall'attivazione di altri sistemi. A tale scopo, sarà utilizzata una soluzione tecnologica in via di definizione da parte di ANAC (ad es. ISTIO RedHat/ circuit breaker).

[RNF-GNRCROSS.20] Timeout per chiamate a servizi

Codice Requisito	Descrizione
RNF.GNRCROSS.20	Il timeout per le chiamate a servizi interni deve essere impostato a 30 secondi, al termine dei quali il sistema deve gestire l'errore secondo la strategia definita nei requisiti del singolo progetto.

[RNF-GNRCROSS.21] Tempo medio di completamento per elaborazioni asincrone

Codice Requisito	Descrizione
RNF.GNRCROSS.21	Il tempo medio impiegato da un software per completare l'esecuzione di una transazione o di un processo asincrono che prevede risposta nei confronti dell'utente deve essere inferiore a un valore medio definito come requisito generale da valorizzare contestualmente ai requisiti funzionali di una specifica applicazione. Tale tempo va calcolato come media tramite test di carico opportunamente definiti nei piani di test.

[RNF-GNRCROSS.22] Capacità di accesso utenti

Codice Requisito	Descrizione
RNF.GNRCROSS.22	Il numero di utenti concorrenti accettati dal sistema per ogni osservazione deve essere pari ad almeno 1000 in ambiente di pre-esercizio. Tale valore va verificato tramite test di carico opportunamente definito nei piani di test.

[RNF-GNRCROSS.23] Adeguatezza dei tempi di risposta all'aumento degli accessi

Codice Requisito	Descrizione
RNF.GNRCROSS.23	Il tempo di risposta del sistema deve presentare un andamento al più lineare in funzione del numero di utenti presenti nel sistema. Tale andamento va verificato tramite test di carico opportunamente definito nei piani di test.

[RNF-GNRCROSS.24] Adeguatezza del consumo di risorse e consumo di risorse di aumento degli accessi

Codice Requisito	Descrizione
RNF.GNRCROSS.24	Il consumo di risorse del sistema (potenza elaborativa, consumo medio di memoria, banda utilizzata, consumo medio di I/O) deve presentare un andamento al più lineare in funzione del numero di utenti presenti nel sistema. Tali andamenti vanno verificati tramite test di carico opportunamente definiti nei piani di test.

[RNF-GNRCROSS.25] Completezza di descrizione

Codice Requisito	Descrizione

RNF.GNRCROSS.25	La percentuale degli scenari d'uso descritta nella documentazione deve essere pari al 100%

[RNF-GNRCROSS.26] Completezza della guida utente

Codice Requisito	Descrizione
RNF.GNRCROSS.26	La percentuale delle funzioni che è descritta (nella documentazione o nell'help) con un dettaglio tale da consentire all'utente di utilizzarle deve essere pari al 100%

[RNF-GNRCROSS.27] Valori già noti al sistema nei campi di input

Codice Requisito	Descrizione
RNF.GNRCROSS.27	Qualora il sistema preveda maschere di input per dati già disponibili il sistema deve presentarli valorizzati di default. La possibilità o meno di modificare il valore da parte dell'utente dipende dal singolo sistema ed è declinata nei relativi requisiti funzionali.

[RNF-GNRCROSS.28] Comprensibilità dei messaggi d'errore

Codice Requisito	Descrizione
RNF.GNRCROSS.28	I tipi di errore gestiti ed i relativi messaggi devono essere definiti e codificati; nel caso i messaggi siano esposti agli utenti, devono essere esplicativi.

[RNF-GNRCROSS.29] Opzione Annullamento

Codice Requisito	Descrizione
RNF.GNRCROSS.29	La percentuale di attività che dispongono di opzione di conferma o di annullamento deve essere pari al 100%.

[RNF-GNRCROSS.30] Salvataggi parziali

Codice Requisito	Descrizione
RNF.GNRCROSS.30	In sistemi web, nel caso di compilazione di moduli utente con un numero di campi maggiore di un valore da definire di volta in volta in funzione della operatività specifica, deve essere previsto l'uso di tab e funzionalità di salvataggio parziale dei dati inseriti.

[RNF-GNRCROSS.31] Evitare errori nelle operazioni dell'utente

Codice Requisito	Descrizione
RNF.GNRCROSS.31	La percentuale delle azioni e input degli utenti che sono protette dal causare malfunzionamenti deve essere pari al 100%.

[RNF-GNRCROSS.32] Copertura dei test

Codice Requisito	Descrizione
RNF.GNRCROSS.32	La percentuale delle funzionalità e casi d'uso del software coperte da test effettivamente svolti e documentati in ambiente di pre-esercizio deve essere pari al 100%.

[RNF-GNRCROSS.33] Robustezza agli errori e gestione delle eccezioni

Codice Requisito	Descrizione

RNF.GNRCROSS.33	La percentuale di errori critici messa sotto controllo deve essere pari ad almeno il 90%. Tale dato sarà rilevato dalla Analisi statica del codice sorgente.

[RNF-GNRCROSS.34] Completezza della tracciabilità utente

Codice Requisito	Descrizione
RNF.GNRCROSS.34	La percentuale di tracciabilità dell'accesso ai sistemi o alle informazioni da parte degli utenti deve essere pari al 100%. Tutte le applicazioni ed i servizi realizzati devono loggare in maniera esaustiva le operazioni effettuate in modo che sia possibile ricostruire le attività dell'utente.

[RNF-GNRCROSS.35] Installabilità

Codice Requisito	Descrizione
RNF.GNRCROSS.35	Gli oggetti software prodotti devono poter essere installati sugli ambienti predisposti dall'Autorità secondo le modalità previste dalla nota di installazione. Devono essere definiti i vincoli di installabilità all'interno della documentazione prodotta.

[RNF-GNRCROSS.36] Configurazione delle applicazioni

Codice Requisito	Descrizione
RNF.GNRCROSS.36	Tutti i punti di configurazione (verso servizi esterni, DB, filesystem, etc.) devono essere chiaramente indicati all'interno delle note di installazione. I file contenenti configurazioni devono essere esternalizzati rispetto agli eventuali pacchetti software utilizzati, in modo da facilitare eventuali aggiornamenti del software.

[RNF-GNRCROSS.37] Connessioni ai DBMS - Utenti

Codice Requisito	Descrizione
RNF.GNRCROSS.37	Le connessioni ai DBMS devono avvenire tramite utenze applicative create appositamente ed utilizzate nei datasource. Tali utenze devono avere il minimo dei permessi necessari per il corretto funzionamento degli applicativi.

[RNF-GNRCROSS.38] Riusabilità degli asset

Codice Requisito	Descrizione
RNF.GNRCROSS.38	I moduli sviluppati devono essere riusabili

[RNF-GNRCROSS.39] Accoppiamento di componenti

Codice Requisito	Descrizione
RNF.GNRCROSS.39	I componenti sviluppati devono essere totalmente disaccoppiati.

[RNF-GNRCROSS.40] Migrazione/Riusabilità dei dati

Codice Requisito	Descrizione
RNF.GNRCROSS.40	In caso di evoluzione/modifica di un prodotto software, la percentuale dati che possono essere riutilizzati / importati, rispetto al totale dei dati che si richiede siano riutilizzabili / importabili dal nuovo prodotto software deve essere pari al 100% (retrocompatibilità).

[RNF-GNRCROSS.42] Configurazione secondo gli standard di internazionalizzazione e localizzazione

Codice Requisito	Descrizione
RNF.GNRCROSS.42	I messaggi applicativi (errore, successo, warning, etc), il testo delle e-mail, le label e tutte le altre informazioni di cui si prevede una potenziale evoluzione devono essere configurabili. I file contenenti tali configurazioni devono essere esternalizzati rispetto agli eventuali pacchetti software utilizzati, in modo da facilitare eventuali aggiornamenti del software. Il sistema dovrà essere realizzato utilizzando lo standard di internazionalizzazione i18n e localizzazione i10n.

[RNF-GNRCROSS.43] conformità ai principi del secure/privacy by design - AGID

Codice Requisito	Descrizione
RNF.GNRCROSS.43	Gli sviluppi sono effettuati secondo quanto previsto dalle " Linee guida di conformità ai principi del secure/privacy by design " fornite da AgID (RIF.EXT.005) in rispetto dei principi e le best practice di secure e privacy by design.

[RNF-GNRCROSS.44] Monitoraggio accessi ai servizi web

Codice Requisito	Descrizione
RNF.GNRCROSS.44	L'Autorità utilizza google analytics per l'analisi delle statistiche degli accessi ai servizi web. L'applicazione dovrà integrare nella costruzione di tutte le proprie pagine web il cosiddetto codice di tracciamento di Google (GATC, Google Analytics Tracking Code), che sarà fornito dall'Autorità.

Linee guida di sviluppo

User Interface

Di seguito sono descritti gli standard da adottare per lo sviluppo dei servizi di Front.-end. Per lo sviluppo delle interfacce utente devono essere realizzate portlet "**Single Page Application**" erogate attraverso il "**Portale Liferay**". Devono essere realizzati inoltre servizi REST per l'accesso ai microservizi secondo il pattern architetturale BFF (Back-end for Front-end). Di seguito viene rappresentato il flusso logico da realizzare per lo sviluppo dei servizi di Front-end

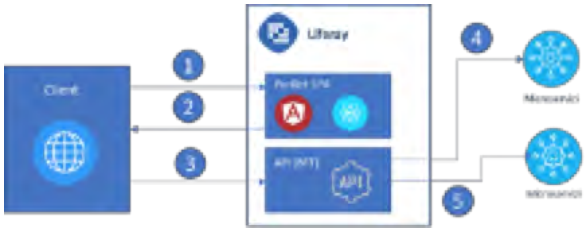


image-20200527174325498

1. Il client accede alla portlet SPA erogata tramite il portale Liferay
2. Il portal Liferay, effettuati i controlli di sicurezza, restituisce la pagina elaborata al Client
3. Il Client Browser, tramite la SPA, invia una richiesta rest al portale Liferay (pattern BFF)

4. Il portale Liferay chiama la api del microservizio applicativo

5. È possibile richiamare più microservizi a fronte di una richiesta del client

Per lo sviluppo delle api da esporre tramite Liferay, si rimanda alla guida ufficiale liferay (vedi **RIF.LGS.04**)

Requisiti di sviluppo User Interface

Di seguito sono elencati i requisiti previsti per lo sviluppo delle user Interface

Codice requisito	Descrizione
REQ.GPL.01	Deve essere garantita, laddove possibile, l'aderenza alle raccomandazioni del World Wide Web Consortium (W3C) [HTTP 1.1, HTML 5.0, XHTML (eXtended Hypertext Markup Language) 5, e CSS 3.0]
REQ.GPL.02	Deve essere garantita la compatibilità con i browser: Internet Explorer,Edge, Safari, Firefox; Opera; Google Chrome, nelle ultime 3 major releases.

Requisiti di accessibilità

Codice requisito	Descrizione
REQ.APL.01	Deve essere garantito, laddove possibile, il pieno rispetto della normativa italiana in materia di accessibilità come definito nei RIF.NORM_01,02,03,04,05
REQ.APL.02	Deve essere garantito, laddove possibile, il pieno rispetto delle raccomandazioni W3C (WCAG 2.0- Web Content Accessibility Guidelines) e standard internazionali ISO in materia di accessibilità Tabella

Requisiti di usabilità

Codice requisito	Descrizione
REQ.UPL.01	Laddove possibile, devono essere realizzate funzioni di help contestuale on-line sulle diverse funzionalità offerte dall'interfaccia. Le funzioni di help potranno fare riferimento ad un "Manuale utente" nei casi previsti o richiesti
REQ.UPL.02	I campi presenti nelle varie interfacce hanno tutti un'etichetta verbale, chiara e non ambigua, che spiega il dato che l'utente deve inserire o selezionare. In alcuni casi è possibile che venga richiesto di inserire ulteriori testi al fine di spiegare meglio il dato richiesto come punto fisso oppure a comparsa attivabile dall'utente
REQ.UPL.03	Per ogni form deve essere evidenziato quali informazioni (campi) sono obbligatorie
REQ.UPL.04	Per quanto non già specificato dai requisiti precedenti devono essere garantiti, laddove possibile, il pieno rispetto degli standard internazionali ISO per l'usabilità

Requisiti di testing

Codice requisito	Descrizione

REQ.TPL.01 I test dovranno essere realizzati attraverso il prodotto selenium HQ al fine di consentirne l'esecuzione automatizzata come test JUnit

Struttura del progetto

Per gestire lo sviluppo di componenti da installare su Liferay, sarà utilizzato il Liferay Module SDK con alcune estensioni migliorative. La tabella seguente ne mostra la struttura e gli elementi principali:

configs/		
	common/ dev/ local/ prod/ uat/	Ogni cartella permette di versionare i file di configurazione (standard e osgi) dei moduli realizzati per i diversi ambienti di Deployment
gradle/		Contiene le indicazioni sulla versione di Gradle da usare per questo workspace
modules/		Conterrà i moduli custom realizzati, raggruppati per applicazione o ambito funzionale. Eventuali moduli che intervengono sul comportamento standard di Liferay saranno inseriti nelle cartelle "portal-hooks" o "portalovverrides"
themes/		Conterrà i moduli che implementano il Tema grafico o i diversi temi grafici previsti. Conterrà anche eventuali LayoutTemplate
wars/		Contiene i moduli (diversi dai Temi) che per motivazioni specifiche devono essere realizzati come "old-style" WAR e non come modulo OSGi
.gitignore		Concentra le regole che identificano i file da non versionare
.gitattribute		Aiuta git a gestire i file di testo nei diversi sistemi operativi che gli sviluppatori, o i tool automatici, possono usare. La regola è usare Unix come default.
build-dist.gradle		In questo file vengono utilizzare le configurazioni Gradle per censire i diversi Profili di Distribuzione che sono usati per indicare quali moduli il progetto rilascia.
build-nexus.gradle		Sono presenti le logiche di accesso ad Artifactory o a nexus (per recuperare o pubblicare artefatti). Sono inoltre presenti le logiche per comprendere se ci troviamo in un branch che gestisce artefatti SNAPSHOT oppure RELEASE
build.gradle		Contiene le logiche di gestione del workspace con i task Gradle aggiuntivi per consentire una

	gestione manuale della vita del progetto. Contiene l'indicazione del gruppo di pubblicazione (in Artifactory) di tutti i moduli del workspace
gradle.properties	Contiene le configurazioni di default del workspace. In particolare: · la versione base di Liferay da utilizzare per risolvere le dipendenze · le url per raggiungere Artifactory · la versione del progetto
gradlelocal.properties.sample	Ogni sviluppatore utilizza un file chiamato "gradlelocal.properties", non versionato, per configurare gli aspetti locali del workspace. Questo file può essere usato come punto di partenza
settings.gradle	Contiene elementi utili a Gradle
source-formatter.properties	File di configurazione usato dal tool "SourceFormatter" di Liferay per aiutare a riformattare i codice sorgente in modo che sia più leggibile e conforme allo standard Liferay

Ambiente di runtime

Per lo sviluppo delle portlet è necessario utilizzare il Liferay Developer Studio; ovvero un Eclipse con diversi add-on realizzati da Liferay. Questo strumento è scaricabile dal sito di Liferay; Installando il Developer Studio viene reso disponibile anche il tool "**blade**".Deve essere predisposto un ambiente di Runtime con Liferay. L'unico pre-requisito è la presenza di Oracle Java Development Kit 8. Questa specifica versione è obbligatoria per la versione Liferay 7.1 DXP utilizzata da ANAC.

ANAC fornirà uno zip contenente un bundle Liferay 7.1 DXP al medesimo fixpack presente nell'ambiente di Produzione. Trattandosi di una versione DXP ogni sviluppatore sarà dotato di una licenza development.

Per l'installazione della licenza è necessario copiare il file xml nella cartella "${liferay.home}/deploy" prima di avviare il portale agendo su "${liferay.home}/wildfly-11.0.0/bin/standalone.bat".

Nel caso sia stata installata sia la componente JDK che la component JRE è necessario modificare il file "${liferay.home}/wildfly-11.0.0/bin/standalone.conf" per indicare a JBoss di usare il JDK. Oppure aggiungere quest'ultima all'environment PATH di Windows.

Dopo la fase di setup iniziale ci troveremo un bundle operativo su database Hypersonic.

image-20200527175145757

È sempre preferibile avviare Liferay in maniera autonoma. Per migliorare l'esperienza di sviluppo è necessario abilitare le impostazioni di debugging remoto di JBoss in modo da potersi collegare "in debug" dall'IDE.

Predisposizione ambiente di sviluppo

La predisposizione di un ambiente minimale per lo sviluppo è descritta nella guida di liferay (vedi **RIF.LGS.07**)

I passi principali sono riassunti di seguito

- **Scelta dell'IDE per lo sviluppo:**
    1. Liferay Developer Studio versione di Eclipse personalizzata da Liferay (vedi **RIF.LGS.08**)
    2. Eclipse con integrazione traite siteUpdate dei plugin liferay
- **Creare il workspace:** avviato l'IDE deve essere creato il workspace Liferay (comprensivo di bundle ed SDK) come indicato all'indirizzo (vedi **RIF.LGS.07**)
- **Importazione workspace:** Anac fornisce un workspace che permette di uniformare lo sviluppo. Effettuato il download da GIT è necessario impostare il workspace all'interno del proprio IDE. Avviato l'IDE andiamo a selezionare "Import Liferay Workspace"

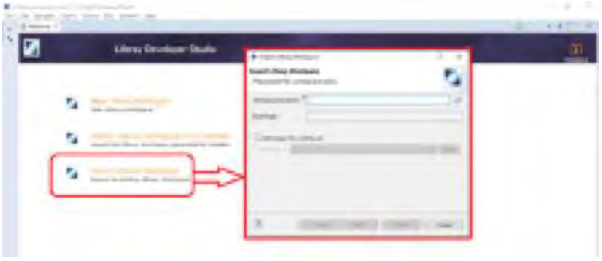


image-20200527175309785

Si aprirà un wizard che permette di selezionare i file scaricati in precedenza sul file system

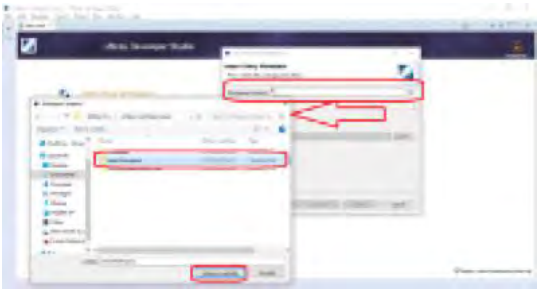


image-20200527175530613

Selezionare la cartella contenente il workspace di Liferay

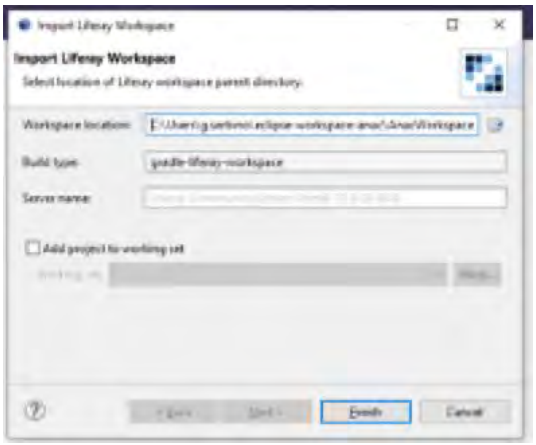


image-20200527175620733

Cliccando su “Finish” sarà avviato il processo di import del workspace

Componenti di Sviluppo - Gitlab

Poiché il progetto è gestito in Gitlab il primo pre-requisito è la presenza del tool git.

Procediamo quindi all’installazione di “git for windows” recuperandolo dal sito ufficiale “https://github.com/git-for-windows/git/releases/latest”.

Nel nostro esempio installiamo la versione 2.25.0 accettando diversi default proposti in quanto rappresentano la soluzione migliore

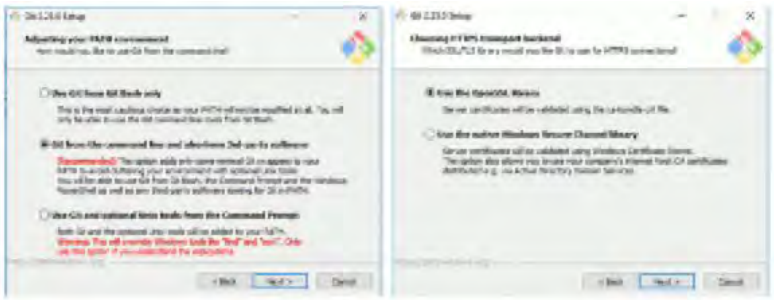


image-20200528092654703

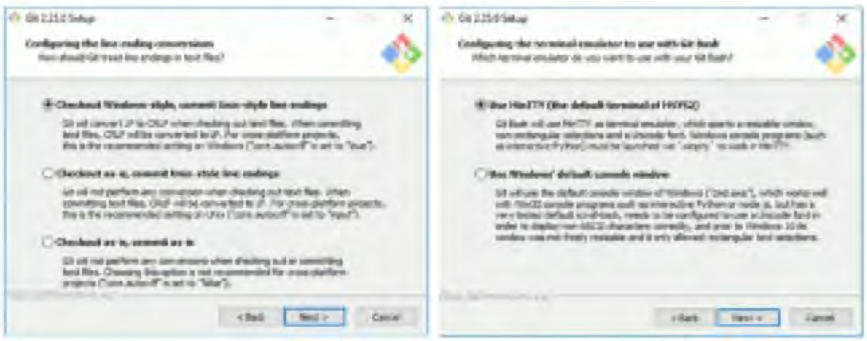


image-20200528092712905

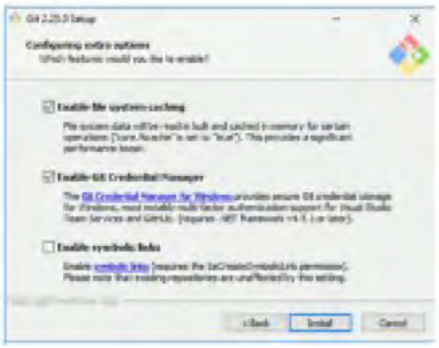


image-20200528092727596

Posizionandoci sulla PROJECT_HOME (C:\java\anac71dxp) attiviamo una Git Bash per poter effettuare il clone del repository contenente il progetto Liferay (il nome utilizzato è solo di esempio).

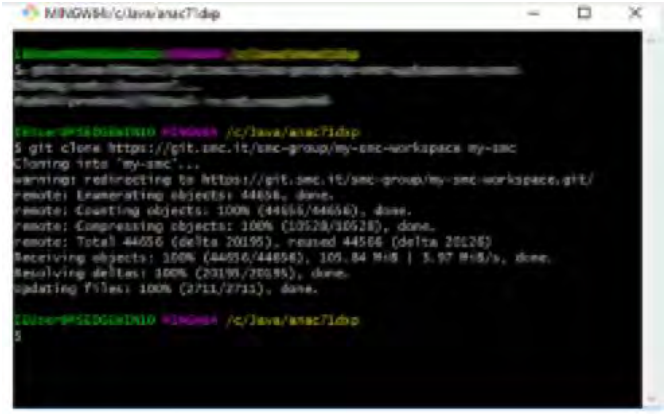


image-20200528092800647

Utilizziamo il file “gradle-local.properties.sample” per creare il file “gradle-local.properties”

```
# Full path of Liferay Bundle in you local environment # liferay.workspace.home.
dir=c:/Java/anac71dxp/liferay-dxp-7.1.10.1-sp1

# SMC Nexus registry. Please add your credential to be able to download artifact
s # #nexus.public.url=https://registry.lab.anticorruzione.local /repository/mave
n-public/ #nexus.releases.url=https://registry.lab.anticorruzione.local /reposit
```

```
ory/maven-releases/ #nexus.snapshots.url=https://registry.lab.anticorruzione.loc
al /maven-snapshots/ nexus.username=my-username nexus.password=my-password
```

Queste indicazioni sono sufficienti per poter eseguire il task “listDistProfiles” per conoscere i Profili di Distribuzione censiti

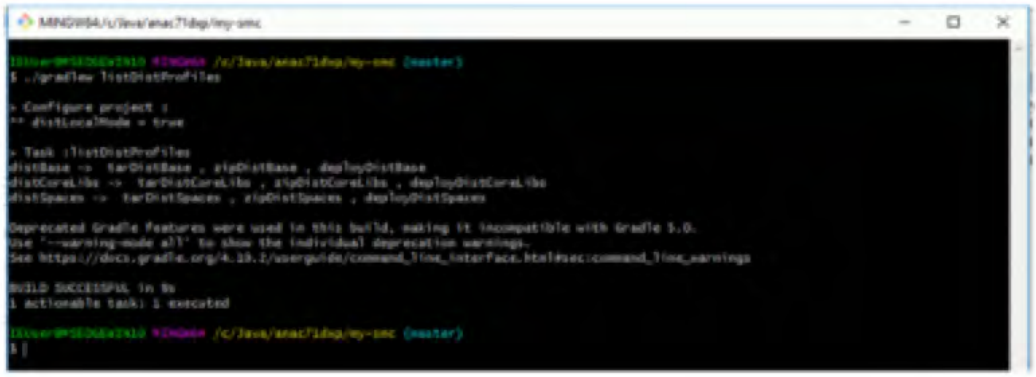


image-20200528092935599

E quindi anche il task “deployDistBase” per deployare nel Liferay locale tutti i moduli che rappresentano la distribuzione base del progetto. Nel caso di moduli locali questi saranno compilati; mentre i moduli di progetti esterni saranno recuperati da Artifactory usando le credenziali inserite in “gradle-local.properties”

Creazione modulo Angular

Utilizzando l’IDE di sviluppo andiamo a creare un nuovo modulo per il portale. Nella fattispecie andremo a creare un “npm-angular-portlet”

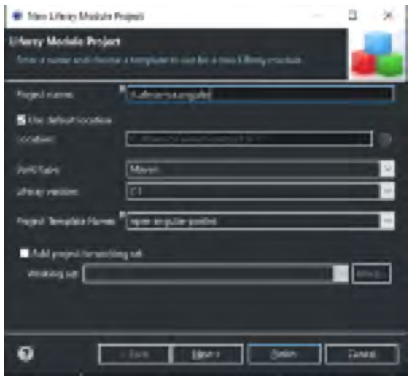


image-20200528092957590

Oltre alla normale struttura di una portlet vengono aggiunti gli elementi essenziali di un progetto Angular. Che rappresentano il core del nostro sviluppo

image-20200528093010508

Nella forlder ${NOME_PROGETTO}/src/main/resorces/…./app/sono contenuti i file da utilizzare per la reazione del progetto. Di seguito illustriamo i file:

- App.module.ts : modulo che permette di strutturare e organizzare le entità e le relazioni tra esse all'interno della nostra applicazione
- App.component.ts : rappresenta il componente View dell'applicazione
- App.html : file HTML di template richiamato all'interno del component

Come sviluppare

Nei paragrafi successivi vengono descritte le nozioni ritenute basilari per l'implementazione dei moduli di front-end:

- Temi grafici
- Portlet application
- Portlet Specification
- SPA
- Service Builder
- Strumenti di amministrazione

Tema grafico

ANAC dispone già di un tema grafico che detta le linee guida per quanto concerne la grafica. Se ci dovesse essere la necessità di ampliare e/o modificare le proprietà grafiche è necessario agire andando a modificare specifici elementi. Di seguito viene illustrata la struttura base di un THEME liferay e le modalità da adottare per integrare eventuali customizzazioni.

L'alberatura dei sorgenti e delle risorse di un tema classico Liferay è rappresentata in figura.

In figura sono evidenziati tre rami dell'alberatura.

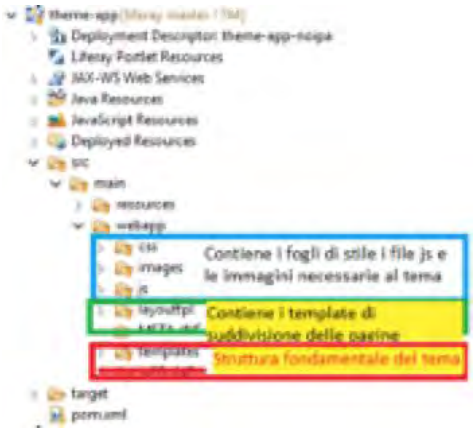


image-20200528093138116

In **azzurro** le risorse statiche (immagini, fogli di stile, snippet javascript) comuni al tema grafico.

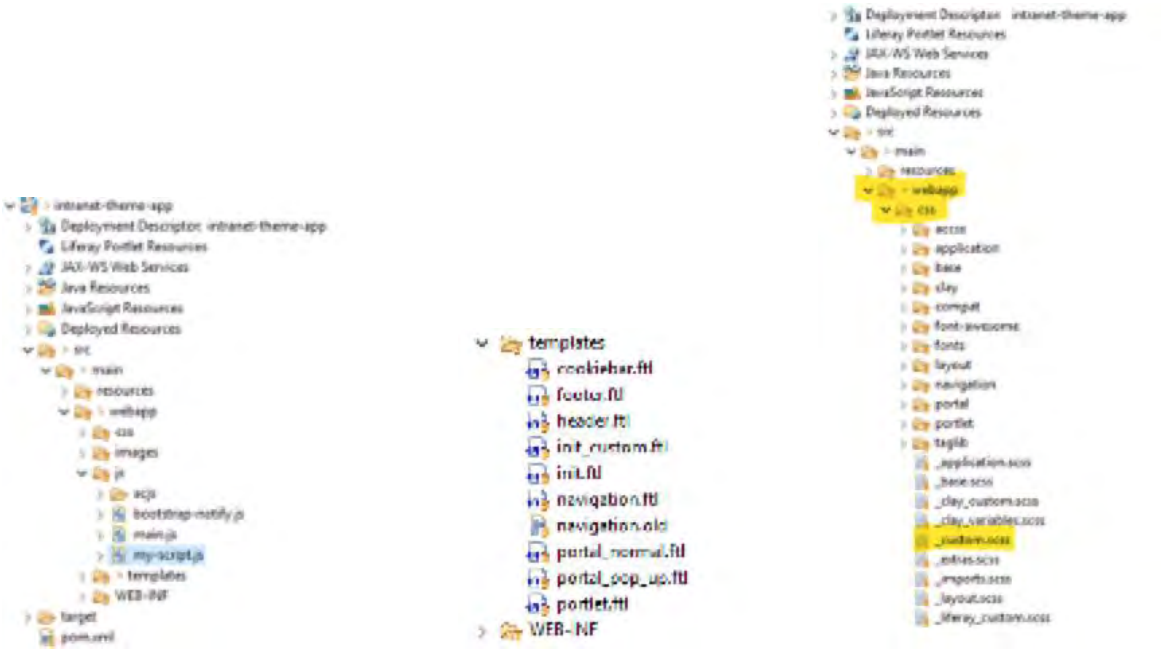
In **verde** i layout che descrivono la struttura della pagina, come cioè saranno disposti le componenti istanziate su di essa. Ne layout possono anche essere definiti dei *placeholder* per aggiungere elementi grafici particolari. Di seguito viene riportato un esempio di layout.

In **rosso** la cartella "Template" che contiene gli elementi comuni alla grafica del portale, come: header, footer, menù ecc.

image-20200528093228892

Se il developer dovesse avere necessità di ulteriori componenti come CSS e/o JavaScript le estensioni, da un punto di vista dei file devono essere aggiunti o modificati in:

1. ${NOME_THEME}/webapp/css/<nomemodulo>_custom.css
2. ${NOME_THEME}/webapp/js/<nomemodulo>_my-script.js



È devono essere rese disponibile all'interno del portale andando ad aggiungere sul portal_normal.ftl il riferimento ai file

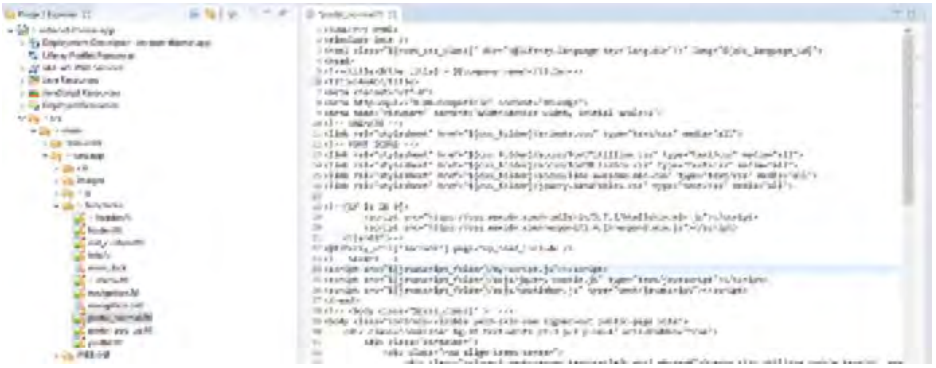


image-20200528093529839

Portlet application

Ogni pagina di frontend è una composizione di singole interfacce utente (micro frontend), ciascuna con un compito e delle responsabilità ben precise (es. menu e barra di navigazione, contenuti specifici, buca di ricerca, ecc.).

Una *portlet* è un modulo riusabile che, secondo quanto definito dalla specifica JSR-168 (e successive) implementa un set di funzioni specifiche (visualizzazione di un contenuto, visualizzazione del ramo, nell'alberatura del portale, attualmente acceduto, CRUD di entità specifiche, visualizzazione di dati secondo un'interrogazione specifica, ecc.). Tecnicamente, un portlet è una componente Java, *web based*, con un ciclo di vita ben specifico, che eroga un set di funzionalità, e che, opportunamente configurata può essere istanziata (portlet instance) come micro frontend. Il portlet container (es. Liferay Portal) è responsabile della gestione del ciclo di vita di ciascuna portlet instance, della visibilità, decorazione e disposizione all'interno di un layout precedentemente stabilito.

Ogni fornitore lavorerà sono sulle portlet inerenti il proprio progetto di riferimento.

Portlet specification

Portal Server e portlet sono regolati da standard specifici:

- Java Portlet Specification 1.0 (JSR-168) del 2003, definisce il comportamento standard secondo JAVA.
- Java Portlet Specification 2.0 (JSR-286) del 2008 è un'evoluzione della JSR-168, che, pur mantenendo la compatibilità con le versioni precedenti, aggiunge specifiche come la comunicazione tra portlet (IPC)

SPA Portlet

Una "Single Page Application" (SPA) riduce drasticamente i tempi di caricamento introducendo il concetto di elaborazione client side. L'utente è agevolato perché una parte della logica elaborativa (prevalentemente navigazione, semplici controlli formali, costrutti orientati alla User Experience) è presa in carico dal browser dell'utente e non necessità di scambio continuo con il portal server. Tutto questo fornisce un'esperienza più fluida creando "l'illusione" di un portale più veloce e reattivo. Dalla versione 7.0, Liferay ha introdotto un motore per la gestione di SPA: SennaJS.

Introduzione a SennaJS

SennaJS gestisce i dati lato client mentre tramite funzioni AJAX vengono caricati dinamicamente i contenuti della pagina. SennaJS è orientato nativamente ad un modello architetturale SPA, e gestisce:

- SEO/Bookmarking: generarando un url che mostra sempre lo stesso contenuto una volta richiesto (bookmarkability) e questo permetta la possibilità di condividere questo url ed aiuta l'indicizzazione su motori di ricerca.
- State Retention: lo scorrimento, il ricaricamento o la navigazione nella cronologia della pagina viene memorizzato e riporta al punto in cui ci si trovava nelle suddette pagine della cronologia.
- UI feedback: È possibile customizzare ed avvisare un utente quando un contenuto è stato richiesto ed è in fase di caricamento.
- Pending navigations: il rendering dell'interfaccia utente viene bloccato fino al caricamento dei dati e il contenuto viene visualizzato tutto in una volta.
- Timeout detection: se il caricamento della richiesta impiega troppo tempo o l'utente tenta di accedere a un collegamento diverso mentre è in sospeso un'altra richiesta, la richiesta scade.
- History navigation: la cronologia del browser viene manipolata tramite l'History API, quindi è possibile utilizzare i pulsanti cronologia avanti e indietro per navigare nella cronologia della pagina.
- Cacheable screens: una volta caricata una surface, il contenuto viene memorizzato nella cache e recuperato in seguito senza alcuna richiesta aggiuntiva, velocizzando l'applicazione.

- Page resources management: script e fogli di stile vengono caricati ed interpretati in maniera dinamica. È possibile poi aggiungere altri contenuti al DOM tramite XMLHttpRequest.
- Hybrid rendering: È un metodo che permette di renderizzare in maniera dinamica oggetti e componenti web che hanno parti che andrebbero richiesti per intero lato server (ad esempio le view delle JSP o i template Freemaker).

Il motore SennaJS è presente e attivo di default su un'istanza Liferay.

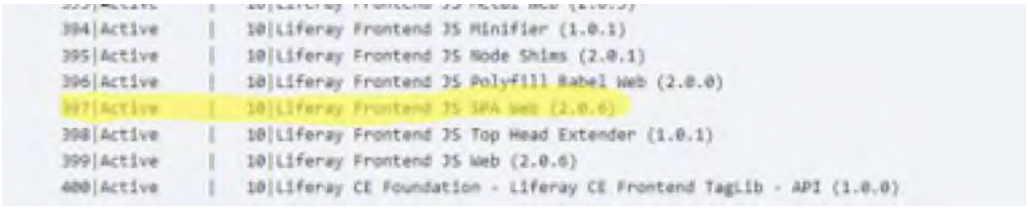
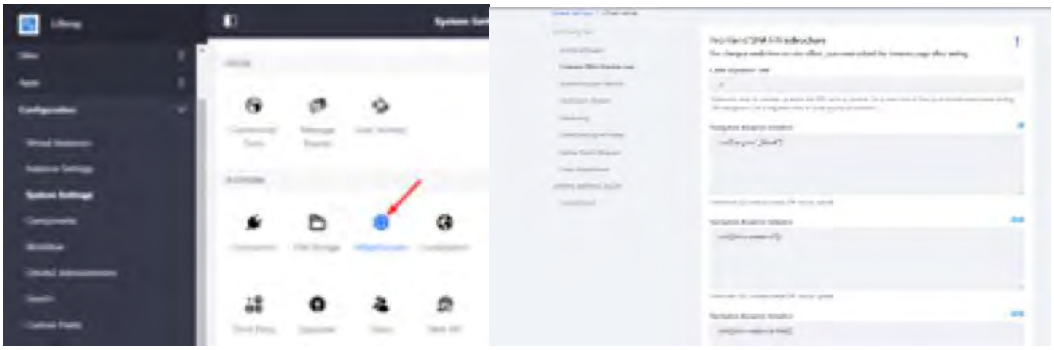


image-20200528093604812

Possono essere configurati parametri specifici



La differenza per quanto riguarda l'esperienza di navigazione è tangibile anche su una semplice portlet come la seguente, dove faremo una semplice navigazione tra due viste:

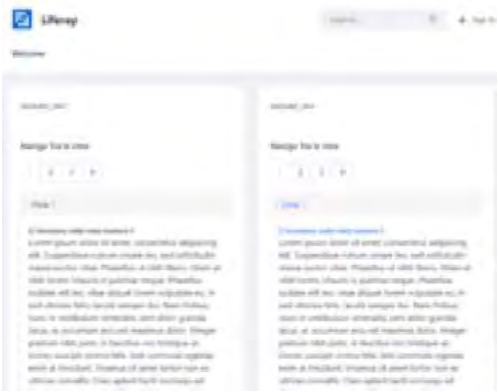


image-20200528093707145

È possibile far convivere portlet che hanno la funzionalità SPA in due stati diversi (true/false) sulla stessa pagina web, questo può essere utile nel caso in cui vogliamo forzare il caricamento di determinate risorse.

Una portlet non potrà mai avere il completo controllo degli URL (ci saranno sempre quantomeno gli elementi basi del portale). Quindi il concetto di SPP (Single Page Portlets) diventa peggiore all'aumentare delle portlet sulla pagina, in quanto ogni portlet vorrà manipolare e controllare gli URL.

Nel portale le singole portlet non devono salvare il loro stato negli URL.

Portlet SPA in Angular

Di seguito viene descritto un esempio di come realizzare una portlet SPA (Single Page Application) tramite il framework Angular. Nell'esempio, la portlet viene utilizzata per consumare dei servizi REST esposti nativamente dal portale.

All'interno di app.module.ts è necessario dichiarare gli import utili al corretto funzionamento della stessa

```
import { NgModule }   from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { AppComponent } from './app.component';

@NgModule({
    imports: [
                        BrowserModule,
                        FormsModule,
                        HttpClientModule
                    ],
    declarations: [AppComponent],
    entryComponents: [AppComponent],
    bootstrap: [], // Do not bootstrap anything (see ngDoBootstrap() below)*
    providers: []
})

export class AppModule {

    // Avoid bootstraping any component statically because we need to attach to*
    // the portlet's DOM, which is different for each portlet instance and,*
    // thus, cannot be determined until the page is rendered (during runtime).*

    ngDoBootstrap() {}

}
```

Viene dichiarato quindi l'import di HttpClientModule, presente nel pacchetto "angular/common/http", e aggiunto agli import dell' @NgModule.

All'interno di app.component.ts andremo a dichiarare gli import necessari al funzionamento della portlet, nello specifico: HttpClient, HttpParams, HttpHeaders necessari per l'invio della chiamata REST;

classi del pacchetto rxjs per utilità: map, catch, retry, of.

```
import { HttpClient, HttpParams, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/retry';
import 'rxjs/add/observable/of';
```

Viene realizzato un semplice template in HTML composto da un bottone che provvede alla chiamata REST tramite il metodo presente nell' app.component.ts e un tag html per visualizzare i risultati della chiamata.

```
<h1>Liferay Angular Portlet HTTP</h1>
<div>
    <label>email: </label>
    <input [(ngModel)]="user.email" placeholder="email">
    <button (click)="getUserByEmail()">Get User by email</button>*
    <div>
        <pre> {{ user | json }} </pre>
    </div>
</div>
```

Attraverso **[(NgModel)]** è possibile effettuare un recupero a runtime del valore inserito nell'input.

Dopo aver importato i pacchetti necessari all'interno dell'app.component.ts vediamo gli ulteriori passaggi necessari.

```
export class User {
    email:string
}

@Component({
    selector: 'my-app',
    templateUrl: '/o/it.almaviva.angular.http/lib/app/app.html'
})
```

Viene creata una classe User che ha come attributo un unico campo string "email" e verrà utilizzata come entità all'interno del component. All'interno del decoratore @Component vengono dichiarate alcune proprietà del componente, nello specifico:

- **selector** : nominativo della parte view a cui è agganciato il component.
- **templateUrl** : indica il path dove è presente il file html da visualizzare. Deve essere indicato tramite questa sintassi: '//<portlet-name>/<project-folders>/file.html'

```
export class AppComponent {
    user: User={
                    email:''
                }

    constructor(private http: HttpClient ) {}
```

```
    readonly ROOT_URL = 'http://localhost:8080/api/jsonws';
    readonly EMAIL = 'test@liferay.com';
    readonly PASSWORD = 'test';
    readonly COMPANYID = '20099';
    authorizationData = 'Basic ' + btoa(this.EMAIL + ':' + this.PASSWORD);

    getUserByEmail(){
        let headers = new HttpHeaders().set('Authorization', this.authorizationD
ata);
        this.http.get(this.ROOT_URL+'/user/get-user-by-email-address/company-id/
'+this.COMPANYID+'/email-address/'+this.user.email, { headers }).subscribe((data
: any)=>{
                console.log(data);
                this.user=data;
        });

    }
```

Arriviamo quindi all'ultima parte dell'app.component.ts che verrà descritta nel dettaglio di seguito. Viene creata una classe AppComponent che al suo interno contiene i seguenti attributi:

- Un' oggetto user della classe User, utilizzato per il recupero del parametro email;
- Un' oggetto http della classe HttpClient con il suo costruttore (*constructor(private http: HttpClient) {}*);
- Attributi richiesti per la Basic Authentication;
- Attributi di sola lettura.

All'interno della classe AppComponent è presente il metodo getUserByEmail() che si occupa di effettuare la chiamata REST e di rispondere con l'utente trovato per l'email che gli è stata passata. Vediamolo nel dettaglio:

```
getUserByEmail(){
    let headers = new HttpHeaders().set('Authorization', this.authorizationData)
;
    this.http.get(this.ROOT_URL+'/user/get-user-by-email-address/company-id/'+th
is.COMPANYID+'/email-address/'+this.user.email, { headers }).subscribe((data : a
ny)=>{
        console.log(data);
        this.user=data;
    });
}
```

- **headers** : oggetto della classe HttpHeaders al quale viene impostata la Basic Authorization.
- **this.http.get(this.ROOT_URL+'/user/get-user-by-email-address/company-id/'+this.COMPANYID+'/email-address/'+this.user.email, { headers })** : attraverso l' oggetto http precedentemente creato inviamo al portale Liferay una chiamata GET esplicitando l'URL e

richiamando i parametri. Il parametro “this.user.email” sarà quello recuperato a runtime dal form html. Viene poi passato l’header per la Basic Authentication.

- **.subscribe((data : any)=>{console.log(data) ;this.user=data; });** : il metodo subscribe ci consente di recuperare i dati contenuti all’interno della response e di assegnarli all’oggetto user.

Per la corretta compilazione della precedente applicazione illustriamo una seri di consigli:

“npm install” e finita la procedura utilizzare “mvn install”.

Se vengono effettuate modifiche aggiuntive basta effettuare la build attraverso il comando “mvn install”. Questo serve ad evitare errori nelle directory target. Una volta ottenuto il .jar dal build del pacchetto, effettuare il deploy tramite app manager dal portale Liferay.

Compilato ed installato il pacchetto otterremo quanto segue:

image-20200601120920115

Nel nostro esempio abbiamo reso possibile il recupero a runtime del valore inserito nell’input che andrà ad indicare l’e-mail dell’utente da recuperare.

Cliccando su “Get User by email” andremo a consumare un servizio esposto da liferay ${IP_NOME_HOST:PORT}/api/jsonws che recupera le informazioni utente dalla mail inserita

image-20200601120935829

Service builder

Il Service Builder astrae lo strato di persistenza facilitando l'integrazione tra quest'ultimo, generalmente basato su paradigmi relazionali, e lo strato di business aderente a logiche di progettazione ad oggetti. Di fatto Service Builder fa uso di strumenti di Object Relationa Mapping (ORM) per collegare Entità di business con elementi del modello dati.

I principali vantaggi del Service Builder sono:

- Integrazione con Liferay
- Modello, persistenza e livelli di servizio generati automaticamente
- Servizi locali e remoti generati automaticamente
- Configurazioni di sospensione e sospensione generate automaticamente
- Supporto per la generazione di metodi finder per entità e metodi finder che tengono conto delle autorizzazioni
- Supporto di memorizzazione nella cache delle entità incorporato
- Supporto per query SQL personalizzate e query dinamiche
- Tempo di sviluppo

Microservizi

Lo sviluppo dei microservizi avverrà tramite il framework **OSS Spring Boot**, che permette di creare applicazioni stand alone basate su Spring, occupandosi di tutta la parte di struttura e configurazione e focalizzando il lavoro dello sviluppatore nella creazione della logica di business.

Di seguito viene rappresentata una guida per realizzare microservizi con tecnologia Spring-boot. L'obiettivo è di fornire una guida di riferimento per strutturare il repository del codice sorgente e gli oggetti funzionali alla realizzazione dei microservizi.

Requisiti di sviluppo Microservizi

Di seguito sono elencati i requisiti previsti per lo sviluppo dei microservizi

Codice requisito	Descrizione
REQ.JAVA.01	Il codice java realizzato per le applicazioni ed i servizi dovrà rispettare gli stili definiti nel documento “Google Java Style” (RIF.EXT.011) ove non diversamente specificato dal presente documento.
REQ.JAVA.02	**Nomenclatura** Per la nomenclatura e la naming convention si utilizzerà il prefisso “it.anticorruzione.” come base package name specializzando poi il package in base al dominio applicativo/funzionale, esempio: ● Namespace package: it.anticorruzione.ms.{dominio-business}.{nome-microservizio} ● Maven GroupId: it.anticorruzione.ms.{dominio-business} ● ArtifactId: {nome-microservizio}-ms (progetto microservizi) Dove dominio-business può essere (anagrafica, appalti, anticorruzione, trasparenza,casellario) Ad esempio per il microservizio di Persona Giuridica del dominio anagrafica sarà: ● Namespace package: it.anticorruzione.ms.anagrafica.personaGiuridica ● Maven GroupId: it.anticorruzione.ms.anagrafica ● ArtifactId: persona-giuridica-ms
REQ.JAVA.03	Ogni applicazione o servizio dovrà essere realizzata come progetto Maven al fine di descriverne in modo dichiarativo come è composto il progetto, quali sono le sue dipendenze, come deve essere compilato e distribuito.
REQ.JAVA.04	Devono essere implementati per tutte le classi che contengono la logica di business dell’applicazione o del servizio i relativi test unitari utilizzando il framework junit. Per testare il corretto funzionamento dei CDI bean si richiede l’utilizzo del framework Arquilian.
REQ.JAVA.05	Utilizzare per il logging la libreria logback e non la System.out
REQ.JAVA.06	Sostituire sempre toString () con la concatenazione degli attributi per dare una buona rappresentazione dell’oggetto.
REQ.JAVA.07	Il codice deve contenere i commenti scritti secondo le regole definte in RIF.EXT.011, utili alla produzione automatica di documentazione javadoc
REQ.JAVA.08	Scegliere in maniera appropriata il livello di log di ogni messaggio, dipendentemente dalla natura del messaggio stesso. Si raccomandano le seguenti linee guida: ● **ERROR**: Usato per loggare eccezioni, errori a runtime o comunque serie situazioni inattese. Questi messaggi dovrebbero essere immediatamente

visibili sulla console. ● **WARN**: Usato per loggare API deprecated, un utilizzo improprio o inatteso delle API, potenziali errori, oppure generiche situazioni a runtime che sono inattese ma non necessariamente sbagliate. Questi messaggi dovrebbero essere immediatamente visibili sulla console. ● **INFO**: Eventi di runtime che sono particolarmente interessanti, come ad esempio lo startup e lo shutdown dell'applicazione. Questi messaggi dovrebbero essere immediatamente visibili sulla console, quindi è bene generarne un numero limitato. ● **DEBUG**: Informazioni dettagliate sul flusso dell'applicazione. Questi messaggi, vista la numerosità e il livello di dettaglio, andrebbero scritti solo su file di log, non sulla console. ● **TRACE**: Informazioni ancora più dettagliate. Questi messaggi, vista la numerosità e il livello di dettaglio, andrebbero scritti solo su file di log, non sulla console.

Requisiti per Business service layer

Codice requisito	Descrizione
REQ.GBL.01	Un servizio esiste solo se implementa almeno un caso d'uso individuato
REQ. GBL.02	Per ogni operation di scrittura deve esistere il corrispettivo servizio di validazione che ha il compito di verificare la correttezza dei dati ricevuti. Questo servizio può essere richiamato direttamente per verificare il set dei dati ricevuti in input e viene chiamato dal servizio stesso prima di procedere con le operazioni di business
REQ. GBL.03	Ad ogni operation del servizio deve essere passato in input un oggetto che contiene: ● Il codice transazione relativo all'operazione utente necessario per le attività di logging (richiesto) ● L'ID di sessione utente necessario per le attività di logging (opzionale) ● La busta SAML necessaria per le attività di audit (opzionale)
REQ. GBL.04	I servizi devono gestire la paginazione dei dati restituiti
REQ. GBL.05	I servizi devono implementare tecniche di caching per l'accesso ai dati più frequenti e con minor frequenza di aggiornamento.
REQ. GBL.06	**Versionamento delle API** La gestione del versionamento delle API ha lo scopo di garantire che il rilascio di nuove versioni dello stesso servizio non generino malfunzionamenti ai client che lo utilizzano. Il versionamento deve essere effettuato utilizzando la tecnica dell'URL versioning, la quale prevede che le diverse versioni dello stesso servizio fanno parte della URI, come per esempio: ● api/v1/servizio1 ● api/v2/servizio1 ● api/v1/servizio2 Il versionamento sarà applicato nei casi di modifiche che generino un impatto lato client, come nel caso di cambio dell'interfaccia del servizio o di variazioni alle strutture dati scambiate. Con l'utilizzo dell'URL versioning si avrà pieno controllo e visibilità dell'utilizzo delle varie versioni del servizio grazie alle metriche di accesso alle singole URL che permetteranno per esempio di decidere quando rendere una versione deprecata.
REQ. GBL.07	**Status Code API** Le API devono utilizzare i seguenti status code: ● 2xx per situazioni di successo in cui la request del client è stata ricevuta, compresa e accettata: o 200 OK: Generico status code usato per indicare che la richiesta è andata a buon fine. o 201 Created: La richiesta è andata a buon fine e la nuova risorsa è stata creata (a fronte di una PUT o POST). La response DOVREBBE

includere un header Location contenente un link alla risorsa creata. o 204 No Content: Il server ha processato la request con successo, ma non ritorna alcun contenuto nella response (tipicamente restituito a fronte di una DELETE). ● 4xx per situazioni di errore dovute al client: o 400 Bad Request: La request non può essere compresa dal server a causa di una sintassi non riconosciuta. Il client NON DOVREBBE ripetere la riquest senza prima modificarla. o 401 Unauthorized: La request richiede l'autenticazione da parte dell'utente. La response DEVE includere l'header WWW-Authenticate contenente la challange adatta alla risorsa richiesta. o 403 Forbidden: Il server ha compreso la request, ma si rifiuta di servirla per qualche ragione, magari perché l'utente autenticato non ha i permessi per accedere alla risorsa. o 404 Not Found: quando le richieste dei consumer hanno come oggetto una risorsa singola o una collezione di risorse che non esiste. o 405 Method not allowed: Il metodo specificato dalla request non è adatto alla risorsa identificata dall'URL. La response DEVE includere header Allow contenente la lista dei metodi validi per la risorsa in questione. o 415 Unsupported Media Type: il server si rifiuta di servire la re quest perché in un formato non supportato dalla risorsa. ● 5xx per situazioni di errore dovute al server

REQ. GBL.08 **Playload degli errori** ● in caso di errore, le risposte di un'API devono contenere un payload esplicativo che non dipende dalla tecnologia di implementazione. ● il payload delle risposte di errore deve contenere una lista di elementi ("error objects"), ciascuno dei quali descrive la/le situazione/i di errore occorse. In caso di errore singolo la lista conterrà un solo elemento, mentre nel caso di errori multipli la lista conterrà più elementi. ● un error object deve rappresentare una situazione di errore applicativo o tecnologico e deve contenere i seguenti campi: o "status": l'HTTP status code (stringa) relativo allo specifico errore. Questa sezione è mandatoria. o "code": codice di errore applicativo custom (stringa). Questa sezione è mandatoria. o "title": stringa che descrive in maniera succinta l'errore. Questa sezione è opzionale. o "detail": oggetto che descrive più in dettaglio il motivo dell'errore (non deve contenere la strack trace dell'errore); è un oggetto a gerarchia zero, ossia composto solo da una serie di coppie chiave:valore e senza innestamenti. Questa sezione è opzionale. o "links": lista di URI (stringhe) che possono essere di utilità al client per la comprensions/gestione degli errori – ad es. link alla documentazione dell'API stessa. Questa sezione è opzionale. ● a fronte di errori multipli, un'API deve assegnare alla risposta HTTP d'errore lo status code più generalmente applicabile al contesto. Esempio: se si sono verificati errori multipli del server (5xx), lo status code usato globalmente per la risposta deve essere 500. ● un'API deve opportunamente documentare tutti i codici di errore ritornabili ai consumer, contestualizzandone le cause e specificando in occasione di quali status code HTTP essi figurano. ● un'API deve documentare tutte le tipologie di error object ritornate. ● un'API deve adottare una delle seguenti strategie di error-handling e documentarla: o Interrompere l'elaborazione di una richiesta al primo errore che accade (sia esso causato dal client o dal server) e ritornare una risposta di errore con un solo error object nel payload o Proseguire fin dove possibile l'elaborazione della richiesta e ritornare una risposta di errore con error object

multipli nel payload ● gli error object ritornati da un'API devono essere espressi in uno dei media-type richiesti dal consumer nella richiesta originale.

Requisiti di versionamento

Attraverso l'adozione dei design pattern Canonical Versioning di tipo Strict (RIF.EXT.010) si adotta un'unica convenzione di versionamento per i business service, dove ogni cambiamento compatibile o incompatibile implica una nuova versione del contratto di servizio con la mancanza di retro compatibilità o compatibilità in avanti.

Codice requisito	Descrizione
REQ.VBL.01	Le regole per il versionamento suddividono il numero di versione in major number, minor number e revision number con le seguenti regole per l'identificazione della versione: ● I cambi della signature delle operation, l'aggiunta di nuove operation e la modifica della semantica di operation pre-esistenti comportano l'aggiornamento del numero di versione major; ● L'aggiunta di funzionalità minori e correzioni significative comportano l'aggiornamento del minor number; ● La correzione di bug comporta l'aggiornamento del revision number.

Inoltre, sono definite delle politiche di dismissione dei servizi non più utilizzati per evitare il proliferare degli stessi.

Requisiti di nomenclatura

Le convenzioni di nomenclatura sono un insieme di regole con cui identificare, attraverso nomi logici, tutti i servizi ANAC. Tutti i servizi ANAC devono avere un nome quanto più possibile auto-esplicativo della funzionalità di business che offre all'interno del dominio ANAC. Di seguito i requisiti di massima, relativi alle convenzioni di nomenclatura, che devono essere soddisfatti nel momento della creazione di un servizio:

Codice requisito	Descrizione
REQ.NBL.01	Nel nome di un servizio/operation devono essere omessi proposizioni, articoli, caratteri speciali
REQ.NOM.02	Nel nome di un servizio/operation non devono essere usati numeri come elementi di distinzione o per distinguere le diverse versioni ad eccezione di quanto richiesto dal pattern di versionamento.
REQ. NBL.02	Il nome di un servizio: a. deve iniziare con la lettera maiuscola; b. deve essere composto solo da lettere minuscole e la separazione tra parole deve essere ottenuta attraverso la lettera maiuscola della parola che segue (notazione CamelCase); c. non sono ammessi blank o, di norma, caratteri underscore. d. Il nome del servizio deve essere significativo per la funzionalità che eroga e. Il nome di un servizio deve essere composto dalla coppia dove: 1. indica lo scopo del servizio ad esempio Gestione 2. rappresenta una delle entità di business

	presenti nel modello dei dati del dominio ANAC. (es: Gara, Lotto, StazioneAppaltante)
REQ. NBL.03	Il nome delle operation del servizio: a. deve iniziare con la lettera minuscola b. deve essere composto solo da lettere minuscole e la separazione tra parole deve essere ottenuta attraverso la lettera maiuscola della parola che segue (notazione camelCase); c. non sono ammessi blank o, di norma, caratteri underscore. d. Il nome dell'operation deve essere significativo per la funzionalità che eroga e. deve essere composto dalla coppia dove: a. 1. può essere un verbo CRUD (Create, Read, Update, Delete) nella notazione italiana (Crea, Consulta, Modifica, Cancella etc etc) 2. un verbo specifico del dominio ANAC (es: Perfeziona) 3. un verbo indicativo di una verifica (es: Verifica) b. un sostantivo che rappresenta una delle entità di business presenti nel modello dei dati del dominio ANAC. (es: Gara, Lotto, StazioneAppaltante)
REQ. NBL.04	Le operation possono essere distinte in due insiemi: a. Operation che lavorano su di un singolo elemento (sia lettura che scrittura) b. Operation che lavorano su di una lista di elementi (sia lettura che scrittura) e nel qual caso al nome viene aggiunta la stringa 'Lista' Esempi di nomi di operation: consultaGara (tipo a), inserisciListaStazioniAppaltanti (tipo b)

Struttura del progetto

Di seguito viene rappresentata la struttura del progetto che deve essere rispettato per lo sviluppo dei microservizi:

image-20200601123602263

***src/main/java*:**

A partire dalla struttura logica, descritta nel capitolo dell'architettura, che rappresenta le componenti interne ad un microservizio sono identificati i package di riferimento contenuti nel Namespace package

- **domain** (Domain): sono le classi che implementano le entità di business
- **repository** (Repositories): rappresentano le classi che effettuano gli accessi al dato
- **service** (Services).: è lo strato che implementa la logica di business del servizio
- **controller** (Resources): rappresenta lo strato applicativo che espone i service come api REST su protocollo HTTP.

- **dto**: sono le classi java che rappresentano i DTO, Data Transfer Object, oggetti passati alla vista per la visualizzazione in pagina.
- **mapper**: Consentono di convertire le entità “Model” con le entità “ViewModel” e viceversa
- **client** (Gateways): contiene le classi che implementano l’accesso ai servizi esterni
- **common:** package di funzioni comuni cross dominio (ad esempio per le component di log)

---

src/main/resource:

Contiene i file di properties del microservizio

---

src/main/test:

Contiene gli unit test e gli integration test

---

pom.xml:

il file di configurazione che contiene le informazioni per la build del servizio

---

Dockerfile.yaml:

Il docker file contiene le informazioni per la build dell’immagine docker. Di seguito un esempio di Dockerfile.yaml

```
# Start with a base image containing Java runtime

FROM openjdk:8-jdk-alpine
RUN apk add --no-cache bash \
                           nano \
                           curl \
                           tzdata

ENV TZ=Europe/Rome
ADD target/servizio.jar servizio.jar
ARG CONFIG_FILE='cfg/application.yml'*
ENV CONFIG_FILE=${CONFIG_FILE}

EXPOSE 8080
```

Come sviluppare

Dto

Le classi create all’interno del package “dto” rappresentano i DTO, Data Transfer Object, oggetti passati alla vista per la visualizzazione in pagina. La sua struttura deve mappare quella del **model** della pagina di front-end.

```
public class UserDTO{

  private String firstname;
    private String lastname;
    private String country;
    private int age;

}
```

Controller

Rappresenta lo strato applicativo che espone i servizi come api REST su protocollo http. In Spring i "Controller" (Resources) sono utilizzati per la creazione di servizi web RESTful. Questi componenti sono facilmente identificabili dall'annotazione @RestController. Di seguito lo snippet di esempio per la creazione di una classe Controller per l'esposizione dei servizi da pubblicare:

```
@RestController
public class UserController {

  @Autowired
    UserService userService;

  @RequestMapping(value = "/users", method = { RequestMethod.GET })
    public List<UserDto> listAllUsers() {
        List<UserDto> userList = userService.getAllUsers();
        return userList;
    }

}
```

Service

È lo strato che implementa la logica di business del servizio. È il layer che integra l'accesso al database applicativo od ai servizi esterni all'architettura.

```
@Service
public class UserService {

  @Autowired
    UserRepository userRepo;

    @Autowired
    UserMapper userConv;

  public List<UserDto> getAllUsers() {
        List<User> userList = userRepo.findAll();
        List< UserDto > users = userConv.toDtoList(userList);
        return users;
    }

}
```

I Repository sono delle astrazioni che ci permettono di minimizzare la quantità di codice da scrivere nel nostro Data Access Layer, offrendoci delle interfacce che implementano per noi out-of-the-box i metodi per il recupero e il salvataggio delle istanze delle nostre entità. Il Repository, essendo il nostro Data Access Layer, ci restituisce oggetti che rappresentano le entità di business della nostra applicazione.

Nel **Service layer** è necessario trasformare gli oggetti passati dal repository in oggetti che il controller dovrà restituire alle viste, ovvero i "Dto". All'interno della classe è definito un oggetto UserConverter. I **Converter** consentono di convertire le entità model, che mappano le tabelle tramite JPA, e le entità "ViewModel" che rappresentano i DTO passati alla vista per la visualizzazione dei dati all'interno della pagina.

Mapper

I mapper consentono di convertire le entità "domain" con le entità "dto" e viceversa.

```
public interface IModelDtoMapper<M, VM> {

  public M fromDto(VM vm);
  public List<M> fromDtolList(List<VM> vmList);
  public VM toDto(M m);
  public List<VM> toDtoList(List<M> mList);

}

public class UserMapper implements IModelDtoMapper<User, UserDto> {

  @Override
    public User fromDto(UserDto vm) {
        // TODO Auto-generated method stub
        return null;
    }

  @Override
  public List<User> fromDtoList(List<UserDto> vmList) {
        // TODO Auto-generated method stub
        return null;
    }

  @Override
  public UserDto toDto(User m) {

        UserDto uvm = new UserDto();
        uvm.setFirstname(m.getFirstname());
        uvm.setLastname(m.getLastname());
        uvm.setCountry(m.getCountry());
        uvm.setAge(m.getAge());
        return uvm;

    }
```

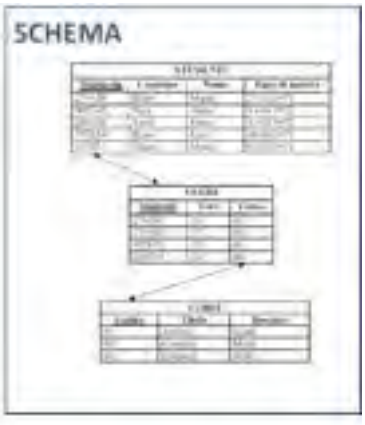
```
  @Override
    public List<UserDto> toDtoList(List<User> mList) {
        // TODO Auto-generated method stub
        List<UserDto> users = new ArrayList<>();
        for( User u : mList)
            users.add(this.toDto (u));
            return users;
    }

}
```

Accesso ai dati

Ogni microservizio dovrà gestire esclusivamente la porzione di dati necessaria all'adempimento delle funzionalità che il microservizio deve implementare. Un microservizio non potrà mai accedere direttamente ai dati gestisti da altri microservizi, ma potrà farlo esclusivamente comunicando, in maniera sincrona o asincrona, con il microservizio proprietario di tali dati.

Di seguito sono elencate le tecniche identificate per ottenere la separazione dei dati tra microservizi

- *Database Redazionale - Schema per service*: ogni servizio ha uno schema privato di un database
- *Database NoSQL – Database per service*: ogni servizio ha un database dedicato contenente una o più collection specifiche del suo dominio di business. Saranno valutate con L'Amministrazione eventuali esigenze che richiedono la creazione di più collection per il singolo microservizio.

Database NoSQL Mongo DB (Collection)	**Database Redazionale SQL Server (Schema)**
img	

Per lo sviluppo di un microservizio spring boot che accede a database NoSQL vedi la guida **RIF.LGS.05**

Configurazione

Aggiungere quanto segue all'elemento file pom.xml dependencies:

```
<dependencies>

 <!-- other dependency elements omitted -->

 <dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-mongodb</artifactId>
 </dependency>

</dependencies>
```

All'interno del file di properties application.properties inserire le informazioni necessarie a spring per la connessione al database

```
spring.data.mongodb.host=[host]
spring.data.mongodb.port=[port]
spring.data.mongodb.authentication-database=[authentication_database]
spring.data.mongodb.username=[username]
spring.data.mongodb.password=[password]
spring.data.mongodb.database=[database]
```

Model

Sono le classi java che implementano le entità di business previste dal contesto applicativo (Es. Cittadino)

```
package org.spring.mongodb.example;

public class Customer {

    private String id;
    private String name;
    private int age;

    public Customer (String name, int age) {
    this.name = name;
        this.age = age;
    }

    public String getId() {
        return id;
    }

  public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    @Override
```

```
    public String toString() {
        return " Customer [id=" + id + ", name=" + name + ", age=" + age + "]";
    }

}
```

Repository

Spring mette a disposizione un meccanismo per le operazioni di interazione con il database sottostante tramite i Repository. I Repository sono delle astrazioni che ci permettono di minimizzare la quantità di codice da scrivere nel nostro Data Access Layer, offrendoci delle interfacce che implementano per noi out-of-the-box i metodi per il recupero e il salvataggio delle informazioni.

```
package com.example.accessingdatamongodb;

import java.util.List;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface CustomerRepository extends MongoRepository<Customer, String> {

  public Customer findByName(String name);
    public List<Customer> findByName(String name);

}
```

Aspetti trasversali

Health check, Metrics

Per consentire il monitoraggio dell'intero sistema è necessario che ogni microservizio sia progettato per esporre le metriche sullo stato delle risorse interne al microservizio stesso. Il modulo "**Spring Boot Actuator**" consente di monitorare e gestire l'applicazione Spring Boot fornendo funzionalità di l'Health check (controllo dello stato), Auditing, raccolta delle metriche, Tracing HTTP, ecc. Tutte queste funzionalità sono accessibili tramite endpoint JMX o HTTP. Actuator consente l'integrazione con la piattaforma di monitoraggio (es. Prometheus) che ha il compito di raccogliere le metriche di tutti i microservizi e fornire un unico punto di accesso per verificare lo stato dell'intero sistema.

È necessario aggiungere il modulo di "**spring-boot-actuator**" aggiungendo la seguente dipendenza al pom maven.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Actuator, in automatico, creerà diversi **endpoint (Endpoint ID)** che possono essere esposti su HTTP o JMX per consentire il monitoraggio dell'intero microservizio. Di seguito l'elenco degli endpoint che vengono esposti da "Actuator"

Endpoint ID	Description
auditevents	Exposes audit events (e.g. auth_success, order_failed) for your application

info	Displays information about your application.
health	Displays your application's health status.
metrics	Shows various metrics information of your application.
loggers	Displays and modifies the configured loggers.
logfile	Returns the contents of the log file (if logging.file or logging.path properties are set.)
httptrace	Displays HTTP trace info for the last 100 HTTP request/response.
env	Displays current environment properties.
flyway	Shows details of Flyway database migrations.
liquidbase	Shows details of Liquibase database migrations.
shutdown	Lets you shut down the application gracefully.
mappings	Displays a list of all @RequestMapping paths.
scheduledtasks	Displays the scheduled tasks in your application.
threaddump	Performs a thread dump.
heapdump	Returns a GZip compressed JVM heap dump.

Logging

Ogni microservizi deve necessariamente loggare le informazioni di tutti gli eventi previsti dai requisiti di sviluppo. Il framework da utilizzare per la gestione dei log è Logback.

Aggiunta delle dipendenze

Di seguito l'estratto del pom maven con evidenza delle dipendenze da aggiungere

```
<properties>
    <slf4j.version>1.7.25</slf4j.version>
    <logback.version>1.2.3</logback.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${slf4j.version}</version>
    </dependency>
    <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${logback.version}</version>
    </dependency>
    <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-core</artifactId>
    <version>${logback.version}</version>
```

```
    </dependency>
</dependencies>
```

Configurazione di logback

Creiamo un file chiamato logback.xml nella root del classpath. Il nome del file è prefissato ma anche logback-test.xml è un nome valido. Nel caso in cui siano entrambe presenti nel classpath verrà utilizzato quello col suffisso -test. Segue un tipico file di configurazione di logback che mostra alcune delle funzionalità di utilizzo più frequente:

```
<?xml version="1.0" encoding="UTF-8"?>*

<configuration scan="true" scanPeriod="60 seconds" >
  <!-- Define variables that will be used for substitution further down this fil
e by enclosing their name           between an opening ${ and a closing } -->
  <variable name="FILE_PREFIX" value="/path/to/my/log/file" />
  <variable name="PATTERN_MSG_ONLY" value="%msg%n" />
  <variable name="PATTERN_COMPLETE" value="% d{yyyy-MM-dd HH:mm:ss.SSS} [%thread
] %-5level %logger{5} - %msg%n" />
  <variable name="MAX_HISTORY" value="30" />

  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
      <level>info</level>
    </filter>
    <!-- encoders are assigned the type ch.qos.logback.classic.encoder.PatternLa
youtEncoder by default -->
    <encoder>
      <pattern>${PATTERN_COMPLETE}</pattern>
    </encoder>
  </appender>*

  <appender name=" FILE" class="ch.qos.logback.core.rolling.RollingFileAppender"
>
    <file>${FILE_PREFIX}.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <!-- Daily rollover -->*
      <fileNamePattern>${FILE_PREFIX}_%d{yyyy-MM-dd}.log</fileNamePattern>
      <!-- Keep ${MAX_HISTORY} days' worth of history -->
      <maxHistory>${MAX_HISTORY}</maxHistory>
    </rollingPolicy>*
    <encoder>
      <pattern>${PATTERN_COMPLETE}</pattern>
    </encoder>
  </appender>
  <logger name="it.foo" level="INFO"></logger>
  <!-- Strictly speaking, the level attribute is not necessary since
       the level of the root logger is set to DEBUG by default -->
  <root level="DEBUG">
    <appender-ref ref="STDOUT" />
    <appender-ref ref="FILE" />
```

```
  </root>
</configuration>
```

Utilizzo delle API di SLF4J

Ora creiamo le classi che useranno le API di SLF4J. La prima classe, chiamata BarComponent, fa parte del package it.bar:

```
package it.bar;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class BarComponent {

  private static final Logger logger = LoggerFactory.getLogger(BarComponent.clas
s);

  public void bar() {
        String name = "Davide";
    logger.info("Hello from Bar.");
        logger.debug("In bar my name is {}.", name);
    }
}
```

Si noti la coppia di caratteri '{}'. Solo dopo aver deciso se loggare o no, logback formatterà il messaggio rimpiazzando '{}' col valore dato.

L'altra classe, chiamata FooComponent, appartiene al package it.foo:

```
package it.foo;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class FooComponent {

    private static final Logger logger = LoggerFactory.getLogger(FooComponent.cl
ass);

    public void foo () {
        String name = "Francesco";
        logger.info("Hello from Foo.");
        logger.debug("In foo my name is {}.", name);
    }

}
```

Ora eseguendo i metodi foo e bar con la configurazione precedente otterremo gli output seguenti in base alla destinazione:

- Su console i messaggi a livello DEBUG provenienti da tutti i package non vengono loggati, in quanto dall'appender STDOUT abbiamo filtrato tutti i messaggi con livello di dettaglio superiore a INFO

```
13:49:59.586 [main] INFO i.a.b.BarComponent - Hello from Bar.
13:49:59.618 [main] INFO i.a.f.FooComponent - Hello from Foo.
```

- Su file di log i messaggi a livello DEBUG provenienti dal package it.foo non sono stati loggati, in quanto al logger associato abbiamo impostato il livello INFO, tuttavia sono presenti i messaggi a livello DEBUG provenienti dal package it.bar

```
13:49:59.586 [main] INFO i.a.b.BarComponent - Hello from Bar.
13:49:59.617 [main] DEBUG i.a.b.BarComponent - In bar my name is Davide
13:49:59.618 [main] INFO i.a.f.FooComponent - Hello from Foo.
```

Tracing distribuito

Per il tracing ogni microservizio deve generare un **TraceID** che dovrà essere propagato lungo l'intero flusso applicativo. La propagazione del TraceID consentirà in fase di consultazione dei log, sulla piattaforma centralizzata di Log Management, di identificare in caso di errore il set di log che appartengono al flusso applicativo. **Spring Cloud Sleuth** e la tecnologia da utilizzare per aggiungere ai log un ID traccia e un ID intervallo.

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
```

Una volta inclusa la dipendenza, Sleuth viene attivato di default. Esso crea un nuovo trace ID per ogni nuova richiesta ricevuta da un particolare servizio. Se l'header "**X-B3-TRACEID**" della richiesta contiene già un trace ID, allora Sleuth continuerà ad utilizzare il trace id ricevuto. Nelle richieste in uscita dal microservizio, il trace ID viene automaticamente inserito nell'header della richiesta così che il prossimo microservizio possa leggerlo e riutilizzarlo.

Sleuth mette quindi a disposizione di ogni microservizio il trace ID scrivendolo nel Mapped Diagnostic Context (MDC) di SLF4J.

Ogni microservizio dovrà predisporre un proprio log applicativo all'interno del quale riportare anche il trace ID di ogni richiesta.

Specifica OpenAPI

Al fine di semplificare e standardizzare i processi per documentare le API dei microservizi è necessario utilizzare la specifica **OpenAPI**. La specifica OpenAPI (swagger) consiste nel documentare in un file testuale (in formato sia YAML che JSON) tutte le funzionalità di un'applicazione web e i dettagli di input e output in un formato interpretabile. In particolare, dovrà essere utilizzata la libreria **Springfox**, la quale implementa le specifiche Swagger 2.0.

Per importare Springfox all'interno di ogni microservizio basterà includere la seguente dipendenza all'interno del file pom.xml.

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>*
```

Successivamente è necessario effettuare la configurazione abilitando Swagger2 tramite l'annotation *@EnableSwagger2*

```
@Configuration
@EnableSwagger2
public class SpringFoxConfig {

  @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
                .select()
          .apis(RequestHandlerSelectors.any())
          .paths(PathSelectors.any())
          .build();
   }

}
```

Per rendere la consultazione della documentazione prodotta di più facile lettura per un essere umano, ogni microservizio dovrà anche esporre la Swagger UI. Per utilizzare Swagger UI dovrà essere aggiunta la seguente dipendendenza:

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
```

L'url da richiamare per visualizzare la console ui: http://localhost:8080/your-app-root/swagger-ui.html

Sicurezza

I micorservizi applicativi devono necessariamente essere predisposti per mettere in sicurezza l'accesso alle API. Di seguito non elencate le linee di intervento previste per la sicurezza dei servizi:

- **Owasp**: le api devono essere sviluppate per prevenire gli attacchi previsti dalle specifiche owasp.
- **JWT**: tutte le chiamate alle api devono prevedere la propagazione del JWT token cifrato. Pertanto, i servizi applicativi devono implementare le opportune logiche utili a decifrare il JWT token ed applicare i controlli di autorizzazione per l'accesso alle API. Per i servizi realizzati in Spring Boot si consiglia l'utilizzo di "**Spring Cloud Security**" (https://spring.io/projects/spring-cloud-security)

Circuit Breaker

Per migliorare la resilienza del sistema in esecuzione nelle chiamate sincrone quando le richieste ad un servizio falliscono è necessario isolare temporaneamente il servizio stesso evitando la propagazione del problema a tutto il sistema. Per l'implementazione del Circuit Breaker si prevede l'utilizzo "**Istio**" (**RIF.EXT.007**)

Servizi Trasverali a supporto

Nome	Tipologia di integrazione	Descrizione	Stato
Profiling	http/rest	Servizi per il recupero del profilo autorizzativo dell'utente loggato	Disponibile
Hub Pagamenti	http/rest	Servizi per i pagamenti dei contributi all'ANAC	Disponibile
Hub notifiche	http/rest	Servizi per l'invio di notifiche tramite e-mail	In corso di sviluppo*
Hub Documentale	Code di messaggi	Servizi per la gestione del ciclo di vita di un documento	In corso di sviluppo*
Anagrafiche	http/rest	Servizio per il recupero delle anagrafiche	In corso di sviluppo*

Appendice

Tipologie di test

Al fine di costruire una strategia di test efficiente, è opportuno partire dal concetto di "piramide di test" che identifica diverse tipologie di test necessarie per verificare che l'intero sistema sia coerente.

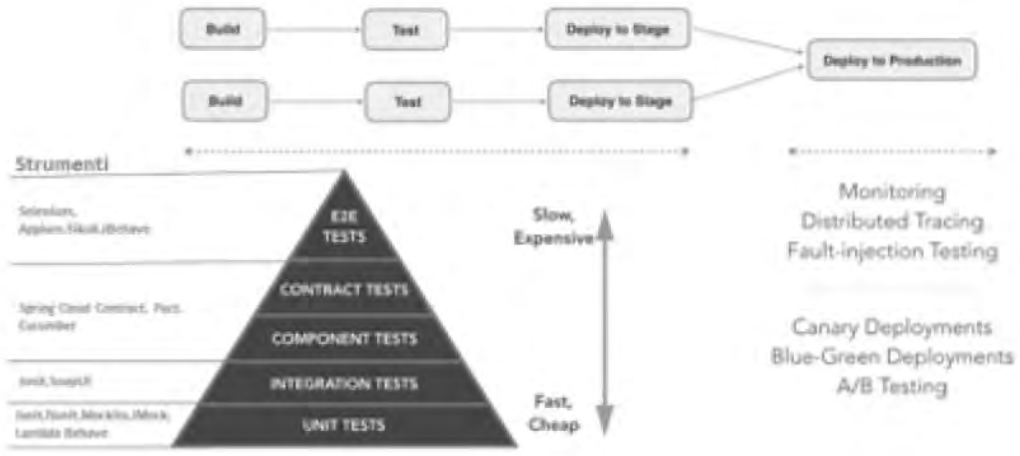


image-20200601150600080

Test unitari

Test che verificano il comportamento dei metodi o delle funzioni di classe all'interno di un servizio. Questi test a partire dal codice del servizio eseguono le sue funzioni o metodi di classe usando input diversi, verificando al contempo l'output delle funzioni per ciascun input. Al fine di riprodurre le condizioni di input che provengono da sistemi esterni è necessario implementare gli **stub** di funzione, chiamati anche **mock**. Questo può essere fatto usando "**dependency injection**" oppure "**method swizzling**". Entrambe le tecniche consentono a un framework di test di eseguire una funzione testata garantendo al contempo che qualsiasi chiamata a una dipendenza di funzione sottostante venga reindirizzata a una funzione diversa che è uno **stub** utilizzato per il test

Test di integrazione

Test che verificano il comportamento esterno di un singolo servizio. Il framework di test avvia un'istanza del servizio ed utilizza l'interfaccia esterna per eseguire la logica di business del servizio nello stesso modo in cui un servizio esterno si integrerebbe con il microservizio testato. Ad esempio, un servizio API REST verrebbe testato inviando richieste HTTP al servizio esterno.

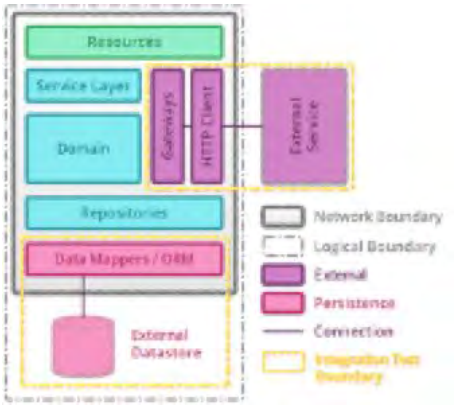


image-20200601150639395

- **Gateway integration test**: I test di integrazione del modulo gateway consentono di eliminare tutti gli errori a livello di protocollo come intestazioni HTTP mancanti, gestione SSL errata o mancate corrispondenze del body della richiesta e della risposta alla massima granularità possibile. Eventuali casi particolari di gestione degli errori devono essere testati per garantire che il client del servizio e protocollo impiegato risponda come previsto in circostanze eccezionali. A volte è difficile innescare comportamenti anomali come timeout o risposte lente dal componente esterno. In questo caso può essere utile utilizzare uno **stub** del componente esterno che può essere configurato per fallire in modi predeterminati. È possibile utilizzare strumenti di **service virtualization** quali "**mountebank**" in grado di simulare il microservizio esterno.

- **Persistence integration test**: I test di integrazione della persistenza assicurano che lo schema assunto dal codice corrisponde a quello disponibile nell'archivio dati. Nel caso in cui sia in uso un ORM, questi test forniscono anche la certezza che eventuali mapping configurati sono compatibili con i set di risultati che ritornano. È importante strutturare i test in modo tale che le transazioni si chiudano tra precondizioni, azioni e asserzioni per essere sicuri che i dati

facciano il giro completo. Poiché la maggior parte degli archivi di dati esiste in una partizione di rete, sono anche soggetti a timeout e guasti di rete. I test di integrazione dovrebbero tentare di verificare che i moduli di integrazione gestiscano questi errori.

Contract Test

Il Contract Test garantisce che una coppia di servizi funzionerà correttamente insieme testando ciascuna applicazione separatamente per garantire che i messaggi inviati o ricevuti siano conformi a una specifica condivisa documentata in un "contratto". Per le applicazioni che comunicano tramite HTTP, questi "messaggi" corrispondono al formato della richiesta e della risposta HTTP, mentre per un'applicazione che utilizzava le code, corrisponde al messaggio che viene inviato alla coda.

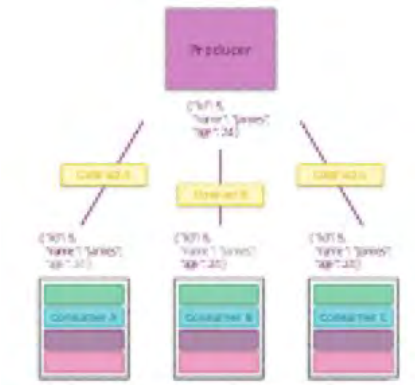


image-20200601150818043

Ci sono due tipi principali di attori nel test di contratto, consumatore e provider.

- **Consumatore**: un client che desidera ricevere alcuni dati (ad esempio un front-end Web o un endpoint di ricezione messaggi).
- **Provider**: un servizio o un server che fornisce i dati (ad esempio, un'API su un server che fornisce i dati richiesti dal client o il servizio che invia messaggi).

Un contratto tra un consumatore e un fornitore si chiama patto. Ogni patto è una raccolta di interazioni. Ogni interazione descrive:

- Una richiesta prevista: che descrive ciò che il consumatore dovrebbe inviare al provider (questo è sempre presente per interazioni sincrone come richieste HTTP, ma non richiesto per interazioni asincrone come code di messaggi)
- una risposta minima prevista: che descrive le parti della risposta che il consumatore desidera che il fornitore restituisca.

Questo contratto o patto garantisce che la comunicazione tra i servizi avvenga come previsto. Esistono numerosi strumenti per facilitare la stesura di **Contract Test** come **Pact** , **Pacto**, **Spring Cloud Contract** e **Janus** .

Component Test

Il test di componente limita l'ambito del software esercitato a una parte del sistema in prova, manipolando il sistema attraverso interfacce di codice interne e utilizzando “double test” per isolare il codice in prova da altri componenti.

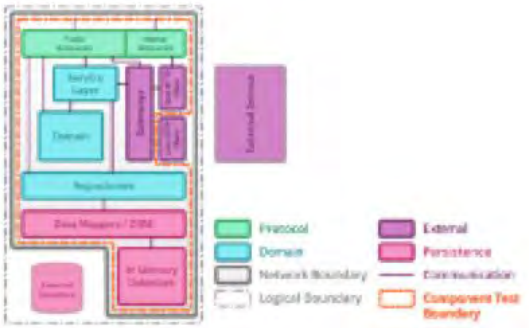


image-20200601150838127

Creando un'istanza dell'intero microservizio in memoria utilizzando le tecniche di “Double test” e Datasources in memory è possibile scrivere test di componenti che non necessitano della rete per simulare il caso di test permettendo di effettuare il test anche in locale. Ciò può portare a tempi di esecuzione dei test più rapidi e ridurre al minimo il numero di parti mobili riducendo la complessità della costruzione. I test comunicano con il microservizio attraverso un'interfaccia interna che consente di inviare richieste e recuperare risposte. Per isolare il microservizio dai servizi esterni, i “**gateway**” possono essere configurati per utilizzare le tecniche di “double test” (Stub http Client) anziché le classi “Client” che si occupano dell’integrazione a livello di protocollo.

I double test possono anche essere utilizzati per simulare i casi di fault, ad esempio quando i servizi esterni sono offline o rispondono lentamente o con risposte non valide. Ciò consente di testare le condizioni di errore in modo controllato e ripetibile. Per testare l’accesso al database è possibile sostituire un archivio dati esterno con un'implementazione in memoria . Sebbene questa tecnica esclude il reale archivio dati dal perimetro del test, qualsiasi test di integrazione della persistenza fornirà una copertura sufficiente. È possibile utilizzare anche strumenti che emulano datastore esterni in memoria, come il motore di database H2 .

Test end-to-end

Test che verificano il comportamento di più servizi che comunicano tra loro. Questi test vengono eseguiti eseguendo più microservizi in un ambiente isolato in cui possono effettivamente comunicare tra loro. I test end-to-end avviano una richiesta di rete a un servizio, come un'API REST, quindi verificano che altri servizi di backend dipendenti siano stati aggiornati in risposta all'input del primo servizio.