

Indice

1 FASE DI AVVIO DEL PROGETTO	4
1.1 Documento di visione	4
1.2 Requisiti FURPS+	4
1.2.1 Functionality	4
1.2.2 Usability	5
1.2.3 Reliability	5
1.2.4 Performance	5
1.2.5 Supportability	5
1.3 Glossario	6
1.4 Diagramma dei casi d'uso	6
2 SPECIFICA E ANALISI DEI REQUISITI	8
2.1 Requisiti funzionali	8
2.1.1 Formato breve	8
2.1.2 Formato informale	9
2.1.3 Formato dettagliato	10
2.1.4 Diagrammi dei requisiti	13
2.2 Modello di contesto	14
2.3 Modello di dominio	15
2.4 Altri diagrammi	16
2.4.1 Diagrammi di attività di alto livello	17
2.4.2 Diagramma di stato	18
2.4.3 Site Map	19
3 DOCUMENTAZIONE DI PROGETTO	21
3.1 Scelte di progetto	21
3.1.1 Pattern Architetturali utilizzati	21
3.1.1.1 Model-View-Controller	21
3.1.1.2 Model-View-ViewModel	21
3.1.2 Stili Architetturali utilizzati	21
3.1.3 Design Pattern utilizzati	22
3.1.3.1 Repository e Validation ¹	23
3.1.3.2 Observer	23
3.1.3.3 Dependency Injection	24
3.2 Diagrammi di alto livello	24
3.2.1 Diagramma delle classi	24
3.2.2 Diagrammi di sequenza	25
3.3 Diagrammi di Basso Livello	28
3.3.1 Server e Web App	28
3.3.1.1 VueJs	29
3.3.1.2 Middleware	31
3.3.1.3 Policies	31

¹Validation non è considerato proprio un design pattern, piuttosto una best practice

3.3.1.4	Repositories e Validators	32
3.3.1.5	Rules	32
3.3.1.6	Config	32
3.3.1.7	Seeders	32
3.3.1.8	Migrations	32
3.3.1.9	Models	33
3.3.1.10	Controllers	34
3.3.2	App Android	35
3.3.2.1	Activity	35
3.3.2.2	Componenti di Vincolo	36
3.3.2.3	Entity	37
3.3.2.4	Api	37
3.4	Database	40
3.5	Diagrammi Funzionalità Implementate	41
3.5.1	Gestione delle Eccezioni	41
3.5.1.1	Lato Server	41
3.5.1.2	Lato Frontend (Vue)	43
3.5.1.3	App Android	44
3.5.2	Visualizza Prenotazioni	44
3.5.3	Gestisci Prenotazione	45
3.5.4	Prenota Richiamo	46
3.5.5	Effettua Prenotazione	46
3.5.5.1	Lato Android	46
3.5.5.2	Lato Server	47
3.6	Diagrammi Globali	47
3.6.1	Web Application Component Diagram	47
3.6.2	Deployment diagram	48
3.6.3	Rest API	50
3.6.3.1	Diagramma UML	50
3.6.3.2	SWAGGER	50
4	DOCUMENTAZIONE DI IMPLEMENTAZIONE	52
4.1	FRAMEWORK E LIBRERIE	52
4.1.0.1	<u>Laravel</u> 8	52
4.1.0.2	<u>Vue</u> 2	56
4.1.0.3	<u>InertiaJs</u>	59
4.2	SOFTWARE UTILIZZATI	61
4.2.1	PhpStorm	61
4.2.2	Android Studio	61
4.2.3	Composer	62
4.2.4	XAMPP	62
4.2.5	NodeJs	63
4.2.6	Vue Devtool	63
4.2.7	Visual Paradigm	64
4.3	SETUP APPLICAZIONE	65
4.3.1	Setup Server	65
4.3.1.1	Php e Database SQL	65
4.3.1.2	Composer	65
4.3.1.3	Download e Installazione Package	65
4.3.1.4	Configurazione File d'Ambiente	65
4.3.1.5	Configurazione Database	65
4.3.1.6	Generazione Swagger	66
4.3.2	Setup app Android	66

Capitolo 1

FASE DI AVVIO DEL PROGETTO

1.1 Documento di vision

Il progetto consiste nella realizzazione di una web application per la gestione delle prenotazioni di vaccini anti-Covid. L'obiettivo del sistema è quello di facilitare i pazienti, nella prenotazione del vaccino tramite una comoda applicazione o semplicemente da un web browser.

Il servizio della prenotazione dovrà essere reso indipendente dalla tecnologia utilizzata dal client mettendo a disposizione endpoint contattabili da qualsiasi device, in modo da rendere l'applicazione facilmente estensibile in futuro. Per dimostrare il funzionamento del servizio, l'idea è quella di sviluppare un prototipo di app Android che consentirà ai pazienti di prenotarsi tramite i servizi offerti dall'applicazione centrale lato server scritta con il linguaggio PHP. I pazienti non saranno gli unici a trarre vantaggio dalla piattaforma, i responsabili sanitari, infatti, potranno utilizzare la piattaforma (solo tramite web browser) per gestire le richieste di prenotazioni, ovvero confermarle, rifiutarle o modificarle. Potranno inoltre confermare la somministrazione del vaccino e prenotare il richiamo. La piattaforma deve garantire anche la gestione delle scorte dei vaccini presenti in un centro vaccinale.

I dettagli di come il software soddisfa queste esigenze sono dettagliate nei casi d'uso e nelle specifiche supplementari.

1.2 Requisiti FURPS+

Si svilupperà un sistema molto robusto, che dovrà gestire gli input scorretti e limita (dove possibile) le risposte aperte sulla compilazione del profilo in modo tale da ridurre il più possibile informazioni errate. Abbiamo preferito la sicurezza e la modificabilità a discapito delle performance, facendo controllare manualmente dal responsabile sanitario le informazioni immesse dal paziente invece di progettare un sistema di controllo automatico, supponendo che questi dispongano di servizi aggiuntivi per sincerarsi della correttezza dei dati immessi. E' ovviamente necessaria un'alta usabilità: le interfacce dell'applicazione dovranno esser user-friendly per permettere una buona esperienza di utilizzo. E' possibile riassumere i seguenti requisiti non funzionali (e le soluzioni adottate per realizzarli) utilizzando uno dei più noti modelli per la definizione di requisiti di qualità del software: il modello FURPS+.

1.2.1 Functionality

Il sistema tratta dati sensibili relativi agli utenti che non devono essere in alcun modo accessibili a chi non è autorizzato. In particolare i dati sensibili riguardano:

- Utenti: utenti non autorizzati non devono essere in grado di accedere ai dati sensibili. Per minimizzare l'errore umano, verranno inseriti dei requisiti minimi sulla password. E' inoltre necessario gestire correttamente i possibili errori negli input utente, rendendo robusto il sistema ed impedendo blocchi irreparabili
- Pazienti: un paziente sarà in grado di vedere solo i dati relativi al suo account e alle sue prenotazioni/somministrazioni.
- Personale sanitario: ciascun membro del personale sanitario sarà responsabile solamente delle prenotazioni della sua struttura. Un membro esterno non dovrà assolutamente essere in grado di interagire con le vaccinazioni assegnate ad un altro centro vaccinale.

- Vaccinazioni: Si tratta di un dato molto sensibile, bisognerà prestare particolare attenzione al modo in cui verrà gestita la conferma della vaccinazione.

Quando qualcuno proverà ad accedere a dati che non è autorizzato a vedere, dovrà essere reindirizzato altrove prima di accedere a tali informazioni.

Per verificare l'idoneità a visualizzare/modificare dei dati, verrà usato l'account associato agli utenti che quindi dovranno effettuare il login per accedere all'applicazione.

Un altro aspetto importante della sicurezza sarà la validazione, bisognerà inserire dei controlli molto restrittivi nella validazione degli input per evitare ogni tipo di **exploit**.

La **portabilità** dell'applicazione permetterà a questa di girare su diversi tipi di device, così che anche le strutture con i sistemi informatici più disparati potranno eseguire il software senza cambiare l'hardware.

1.2.2 Usability

L'interfaccia sarà diversificata per gli utenti. Personale Sanitario e Dirigenti utilizzeranno una web app per gestire e verificare le prenotazioni del paziente, mentre a quest'ultimo verrà fornita anche la possibilità di scaricare un'app sul telefono in modo da poter prenotare comodamente senza aver bisogno di un computer o accedere via sito web per farlo (anche se in ultimo verrà implementata un'interfaccia web anche per i pazienti).

Le interfacce grafiche dovranno possedere le seguenti caratteristiche:

- Intuitività: per permettere anche ai meno avvezzi alla tecnologia di operare
- Responsività: l'applicazione dovrà essere responsive, vale a dire che si adatterà al dispositivo in uso che sia un PC, un telefono o un tablet
- Feedback: i componenti dell'interfaccia grafica dovranno fornire feedback ben visibili agli utenti in maniera tale che si abbia conferma visibile dell'avvenuta/fallita esecuzione di un comando, ad esempio non si deve verificare che il responsabile annulli una prenotazione, non compaia nessun messaggio di conferma e questi sia costretto, nel dubbio, a riaprirla per verificarne lo stato
- Estetica: per quanto non sia prettamente necessaria ai fini dello svolgimento delle operazioni, un'interfaccia gradevole alla vista potrebbe rendere più stimolante l'attività lavorativa

Data la rapida evoluzione riguardante le norme anti Covid-19, l'applicazione dovrà essere ampiamente documentata per facilitare eventuali future modifiche anche a distanza di molto tempo.

1.2.3 Reliability

Dal punto di vista del sistema centrale, questo dovrà essere **affidabile** e **disponibile** per tutta la durata dell'emergenza da Covid-19.

Per quanto riguarda invece i client, la portabilità dell'applicazione, descritta prima, permette, in caso di guasti alle attrezzature o ad esempio mancanza di elettricità, di operare con device di emergenza come tablet e cellulari.

Le eccezioni e gli errori dovranno essere gestiti in maniera tale che il funzionamento del sistema non ne venga compromesso, dovranno inoltre produrre report dettagliati che consentano agli sviluppatori di risalire alla radice del problema. Infine non dovranno mostrare agli utenti delle informazioni sensibili riguardanti il codice.

1.2.4 Performance

Il sistema permette in breve tempo a qualsiasi paziente di completare la propria prenotazione. Una volta effettuato questo passaggio sarà compito del responsabile sanitario aggiornare lo stato di prenotazione tramite un controllo manuale dei dati inseriti. Anche in questo caso l'operatore non deve subire rallentamenti da eventuali lunghi tempi di caricamento.

1.2.5 Supportability

L'applicazione dovrà essere molto **flessibile** ad eventuali cambiamenti, come ad esempio l'aggiunta di nuovi stock, lotti, vaccini, strutture, etc... consentendo agli utenti autorizzati di configurare il sistema.

1.3 Glossario

Struttura Sanitaria	Struttura formata da personale sanitario qualificata a gestire e somministrare vaccini (sinonimo: centro vaccinale)
Responsabile Sanitario	Personale qualificato che deve controllare se i dati del paziente sono corretti, inoltre deve gestire i vaccini della relativa struttura
Paziente	Soggetto che prima di poter accedere ai servizi che il sistema offre deve effettuare la registrazione compilando l'apposito profilo
Paziente Loggato	Soggetto che si è già registrato e può accedere al sistema richiedendo i servizi che offre
Vaccino	Prodotto che viene inoculato al paziente previa richiesta
Profilo	L'insieme di dati anagrafici e sanitari sul paziente specifico
Dirigente	A capo dell'intera struttura sanitaria ha il compito di gestire l'intero centro vaccinale
Lotto	Partita di un determinato vaccino. Un lotto è un quantitativo di vaccini di caratteristiche uniformi. I vaccini di un determinato lotto possono essere smistati anche in centri vaccinali differenti.
Stock	Quantità di vaccini relativi ad un singolo lotto presenti nella struttura sanitaria (associazione molti a molti tra Struttura Sanitaria e Lotto)
Richiamo	Inoculazione di una seconda (o più) dose di vaccino determinata dal personale sanitario
Admin	Amministratore dell'intero sistema che ha il controllo su dirigenti, centri e vaccini

Tabella 1.1: Glossario

1.4 Diagramma dei casi d'uso

Per valutare se assegnare o meno un caso d'uso ad un singolo requisito, sono stati utilizzati i seguenti criteri:

- **Test del capo:** il capo sarebbe felice di avere un caso d'uso ad esempio per il login?
- **Test EBP (Elementary Business Process):** il caso d'uso corrisponde ad un EBP?
- **Test dimensione:** i casi d'uso dovrebbero essere corposi e non corrispondere ad una singola azione elementare (ad esempio alcune sono state raggruppate con le CRUD)

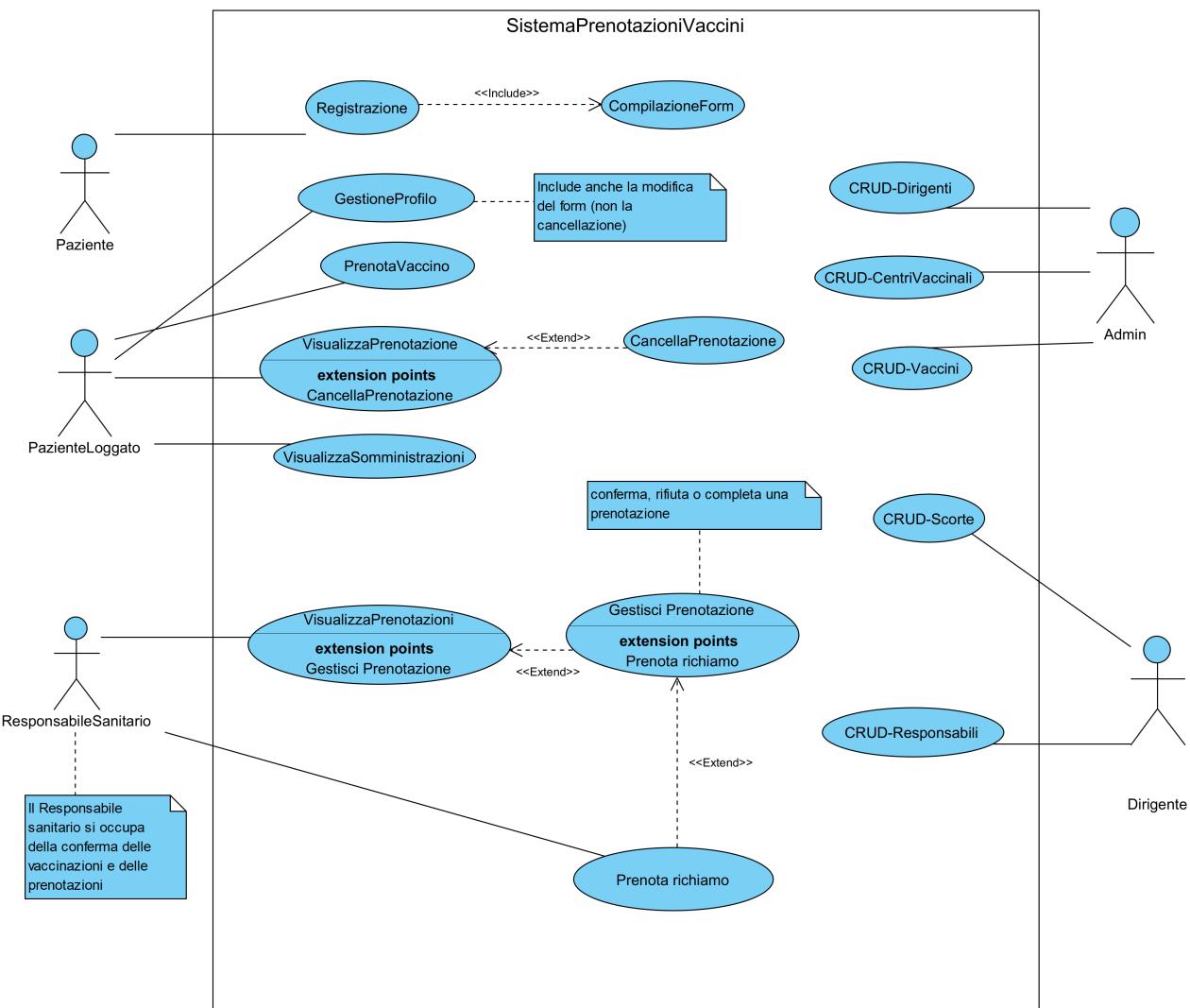


Figura 1.1: Diagramma dei casi d'uso

Capitolo 2

SPECIFICA E ANALISI DEI REQUISITI

2.1 Requisiti funzionali

2.1.1 Formato breve

Attore	Caso d'Uso	Descrizione	Priorità
Paziente	Registrazione	Il paziente prima del login deve registrarsi alla piattaforma	media
	Compilazione Profilo	Il paziente per potersi registrare deve compilare il profilo con tutte le informazioni richieste	media
Paziente Loggato	Gestione Profilo	Il paziente deve poter modificare le proprie informazioni precedentemente immesse all'atto della registrazione	media
	Prenota Vaccino	Una volta completato il login, il paziente può effettuare la prenotazione di un vaccino richiedendola dall'app.	alta
	Visualizza Prenotazione	Consente la visualizzazione della prenotazione che ha effettuato (se esiste)	media
	Cancella Prenotazione	Consente al paziente loggato di poter cancellare la prenotazione	bassa
	Visualizza Somministrazioni	Il paziente deve poter controllare le informazioni di tutti i vaccini che si è fatto somministrare	bassa
Responsabile Sanitario	Visualizza Prenotazioni	Il responsabile sanitario, tramite interfaccia web, può visualizzare tutte le informazioni relative alle prenotazioni della struttura sanitari	alta
	Gestisci Prenotazione	Il responsabile sanitario deve poter confermare, rifiutare o completare la prenotazione di un paziente	alta
	Prenota Richiamo	Se previsto, il responsabile sanitario gestisce la prenotazione di una seconda dose di un determinato paziente	alta
Dirigente	CRUD Scorte	Il dirigente deve poter gestire tutte le scorte relative alla struttura sanitaria di sua competenza	bassa
	CRUD Responsabili	Il dirigente deve poter gestire i responsabili sanitari relativi al centro vaccinale di sua competenza	bassa
Admin	CRUD Dirigenti	L'admin gestisce i dirigenti, associandolo o cancellandolo da una determinata struttura	bassa
	CRUD Centri Vaccinali	L'Admin deve poter aggiungere, modificare o eliminare un determinato centro vaccinale	bassa
	Crud Vaccini	L'admin può inserire o rendere non disponibile una tipologia di vaccino	bassa

Tabella 2.1: Casi d'uso in formato breve

2.1.2 Formato informale

- **Registrazione:** Procedura di registrazione all'app da parte del paziente, obbligatoria quando accede per la prima volta alla piattaforma. Il paziente dovrà compilare correttamente tutti i campi richiesti (e-mail, password, nome, cognome, data, sesso, codice fiscale, città, cap, telefono e malattie) affinché la registrazione vada a buon fine.
- **Login:** Procedura di login da parte del paziente, del personale sanitario o del dirigente per accedere alla piattaforma.
- **Modifica profilo:** Dalla homepage consente al paziente di modificare le informazioni del profilo immessi all'atto della registrazione compreso il form per le eventuali malattie. Se il paziente immette dati non validi o non compila campi obbligatori il sistema non salva le modifiche e rindirizza il paziente sulla homepage. L'indirizzo e-mail non può essere modificato.
- **Prenota vaccino:** Permette di prenotare un vaccino scelto dal sistema. La regione e la struttura sanitaria viene scelta dal paziente (verranno mostrate tutte le strutture relative a quella regione). Il paziente può scegliere la data dell'appuntamento scegliendo tra quelle disponibili mostrate dal sistema, l'ora invece verrà assegnata in base al numero di prenotazioni totali della struttura in quella giornata. Il responsabile dovrà poi confermare o meno la prenotazione.
- **Visualizza prenotazione (Paziente):** Il paziente una volta che effettua il login entra automaticamente nella home dove gli verranno mostrate tutte le informazioni (struttura, data, ora, lotto-vaccino, stato) relative alla prenotazione, se esiste.
- **Cancella prenotazione:** Consente al paziente di cancellare la propria prenotazione. La cancellazione si può effettuare al massimo entro le 72 ore dalla data dell'appuntamento. In seguito, il paziente potrà effettuare una nuova prenotazione.
- **Visualizza somministrazioni:** Consente al paziente di visualizzare le vaccinazioni che ha già effettuato. Cliccando sulla pagina di visualizza somministrazioni, il sistema mostra al paziente le prenotazioni già complete (ovvero con la conferma della somministrazione). Se non esistono prenotazioni complete il sistema mostra un messaggio di somministrazioni non presenti. Se il paziente clicca su una somministrazione verrà rindirizzato sulla pagina di dettaglio della somministrazione in cui può decidere di scaricare un attestato di vaccinazione.
- **Visualizza Prenotazioni (Responsabile sanitario):** Permette all'utente di visualizzare tutte le prenotazioni oppure un gruppo di prenotazioni in base ad un filtro (da confermare, confermata, cancellata, completata) relative al centro vaccinale di cui fa parte. È possibile ricercare anche le prenotazioni relative ad un singolo paziente digitando il suo nome e cognome nella barra di ricerca.
- **Prenota richiamo:** Consente al responsabile sanitario di prenotare un eventuale richiamo per il paziente dopo la somministrazione della prima dose di un vaccino. Il responsabile dalla pagina di gestione prenotazioni può decidere di prenotare un richiamo cliccando su un pulsante dalla pagina di gestione prenotazione, solo se la prenotazione precedente risulta nello stato completata. A questo punto il responsabile viene rindirizzato sulla pagina di prenotazione di un richiamo relativo allo stesso paziente, in cui dovrà compilare i campi come data, orario, vaccino, e cliccare su conferma. La nuova prenotazione verrà salvata con lo stato confermata e sarà di tipo richiamo.
- **Gestisci prenotazione:** Consente al responsabile sanitario di confermare, rifiutare, completare o modificare una prenotazione (ovvero confermare l'avvenuta somministrazione) dopo aver selezionato una prenotazione dalla pagina di visualizza prenotazioni. Se la prenotazione è nello stato "da confermare" allora saranno disponibili solo i pulsanti per confermarla o rifiutarla. Se la prenotazione è nello stato "confermata" o "cancellata" allora è disponibile solo il pulsante per completarla. Se il responsabile modifica una prenotazione attraverso i campi del form, questa verrà salvata solo in concomitanza della pressione del pulsante per confermare.

2.1.3 Formato dettagliato

Si riporta la specifica dettagliata dei requisiti relativi ai casi d'uso implementati nelle due iterazioni del processo di sviluppo.

Caso d'uso: Gestisci prenotazione

Attori: Responsabile sanitario

Priorità: alta

Dipendenze: extend di Visualizza Prenotazioni

Precondizioni:

Il responsabile ha effettuato il login per accedere alla piattaforma.

Il responsabile si trova nella pagina di visualizza prenotazioni.

Sequenza degli eventi:

1. Il caso d'uso ha inizio quando il responsabile seleziona una prenotazione dalla pagina di visualizza prenotazioni.
2. Il sistema mostra i dettagli della prenotazione (stato, data, orario, vaccino) e i dettagli del paziente prenotato (dati anagrafici e form compilato).
3. Il responsabile modifica opzionalmente i campi modificabili della prenotazione e clicca su conferma per confermare la prenotazione.
4. Il sistema salva la prenotazione salvata con i nuovi campi e con lo stato “confermata”.
5. Il sistema reindirizza il responsabile sulla pagina di visualizza prenotazioni.

Scenari alternativi

3.1.1 Il responsabile non modifica i campi della prenotazione ma si limita a confermarla.

3.1.2 Il sistema salva la prenotazione con lo stato “confermata”

3.1.3 Il sistema reindirizza il responsabile sulla pagina di visualizza prenotazioni.

3.2.1 Il responsabile decide di rifiutare la prenotazione.

3.2.2 Il sistema salva la prenotazione con lo stato “cancellata”

3.2.3 Il sistema reindirizza il responsabile sulla pagina di visualizza prenotazioni.

3.3.1 Il responsabile decide di completare la prenotazione.

3.3.2 Il sistema salva la prenotazione con lo stato “completata”

3.3.3 Il sistema aggiorna la pagina della gestione della prenotazione con il nuovo stato

Postcondizioni:

Il sistema salva la prenotazione con il nuovo stato

Caso d'uso: VisualizzaPrenotazioni

Attori: Responsabile sanitario, Dirigente

Priorità: media

Dipendenze: extend di Visualizza Prenotazioni

Precondizioni:

Il responsabile ha effettuato il login per accedere alla piattaforma.

Sequenza degli eventi:

1. Il caso d'uso inizia quando il responsabile sanitario richiede di visualizzare le prenotazioni
2. Il sistema mostra tutte prenotazioni della struttura
3. Il responsabile sceglie di filtrare le prenotazioni in base ad un parametro
4. Il sistema mostra le prenotazioni filtrate

Scenari alternativi

3.1 Se il parametro inserito non è valido il sistema restituisce una lista vuota

Postcondizioni:

Caso d'uso: Prenota richiamo

Attori: Responsabile sanitario

Priorità: alta

Dipendenze: extend del caso d'uso Gestisci Prenotazione

Precondizioni:

- Il responsabile ha effettuato il login per accedere alla piattaforma
- Il responsabile si trova nella pagina di gestione di una prenotazione
- La prenotazione della prima dose risulta completata

Sequenza degli eventi:

1. Il responsabile decide di prenotare un richiamo cliccando sul relativo pulsante
2. Il sistema rindirizza il responsabile sulla pagina per prenotare un richiamo
3. Il responsabile compila i campi (data, orario, vaccino) e conferma
4. Il sistema salva la prenotazione con lo stato di confermata
5. Il sistema rindirizza il responsabile sulla pagina di visualizza prenotazioni

Scenari alternativi

1.1 Se la prenotazione relativa all prima dose non risulta nello stato di completata, il pulsante relativo alla prenotazione di un richiamo è disabilitato e la prenotazione non può essere effettuata.

3.1 Se i campi da compilare sono sbagliati (inserimento data passata, inserimento data occupata, inserimento orario di chiusura della struttura , dati mancanti nella compilazione del modulo, scelta di vaccino senza scorte) il sistema chiede di reinserire i dati.

Postcondizioni:

Il sistema crea una nuova prenotazione relativa al paziente della precedente prenotazione e con i dati compilati dal responsabile

Caso d'uso: Prenota Vaccino

Attori: PazienteLoggato

Priorità: alta

Dipendenze:

Precondizioni:

- Il paziente ha effettuato il login
- Il paziente non ha già prenotazioni attive
- Il paziente non ha completato già la vaccinazione per questa campagna vaccinale

Sequenza degli eventi:

1. Il caso d'uso inizia quando il paziente richiede di prenotare un vaccino
2. Il sistema mostra a video i campi che il paziente deve compilare

3. Il paziente sceglie una regione
4. Una volta scelta la regione, sarà sbloccata al paziente la possibilità di scegliere la struttura sanitaria desiderata
5. Il sistema fornisce un calendario con le date e il paziente sceglie in base alle sue esigenze
6. Il sistema assegna un vaccino in base alla quantità disponibile in quella struttura tra i vaccini idonei.
7. Il sistema assegna un orario in base alle prenotazioni della struttura in quel giorno
8. Il sistema mostra il riepilogo della prenotazione nella home dell'applicazione

Scenari alternativi

- Se l'utente nasbaglia la compilazione dei campi richiesti (data passata, data occupata, dati mancanti) allora il sistema segnala l'errore e chiede di reinserire i dati.
- Se l'utente, in qualunque momento abbandona la pagina prima di confermare, il sistema non salva le informazioni e l'utente deve di nuovo richiedere una prenotazione

Postcondizioni:

Il paziente non potrà effettuare nuove prenotazioni se non completa oppure cancella la prenotazione

Caso d'uso: Registrazione

Attori: Paziente

Priorità: alta

Dipendenze:

Precondizioni:

Il paziente non è registrato già al sistema

Sequenza degli eventi:

1. Il caso d'uso inizia quando il Paziente installerà e apre l'app "Project Sad".
2. Il paziente compila i dati richiesti
3. Il paziente viene autenticato dal sistema, e il caso d'uso termina

Scenari alternativi

Scenari:

2.1 Se il Paziente immette i dati correttamente, la registrazione va a buon fine

2.2 Se il Paziente non compila tutti i campi richiesti, il sistema mostra un messaggio di errore e chiede di ricompilare i campi

2.3 Se il Paziente si è già registrato precedentemente verrà mostrato il messaggio di "account già esistente"

2.4 Se il Paziente immette campi non corretti (es. Data di Nascita) il sistema mostra un messaggio di errore "incorrect details"

Postcondizioni:

Verrà creato un nuovo record relativo all'utente nel database

Caso d'uso: Login

Attori: Paziente, Responsabile Sanitario, Dirigente

Priorità: alta

Dipendenze:

Precondizioni:

Gli attori che avviano il caso uso devono essere registrati al sistema (quindi presenti nel database) compilando gli appositi campi correttamente (login e password)

Sequenza degli eventi:

1. Il caso d'uso inizia quando un paziente apre l'applicazione.
2. L'utente inserisce i dati necessari all'accesso e clicca su login.
3. Il sistema fa accedere l'utente alla piattaforma

Scenari alternativi

Scenari:

2.1 Se l'utente non ha effettuato la registrazione oppure immette username e/o password sbagliata allora il sistema mostra un messaggio di errore “incorrect details” e chiede di reinserire i dati.

2.2 Se l'utente prova ad entrare nell'area riservata ad un responsabile con un account di un paziente il sistema non consente l'accesso.

Postcondizioni:

L'utente è autenticato dal sistema

Caso d'uso: VisualizzaPrenotazione

Attori: PazienteLoggato

Priorità: alta

Dipendenze:

Precondizioni:

- Il paziente ha effettuato il login

Sequenza degli eventi:

1. Il caso d'uso inizia quando il Paziente richiede di visualizzare la schermata home.
2. Una volta nella home il sistema mostrerà la prenotazione effettuata con tutte le informazioni.

Scenari alternativi

- 2.1 Se non ci sono prenotazioni attive il sistema non mostra alcun tipo di informazione.
-

2.1.4 Diagrammi dei requisiti

Di seguito è riportato il diagramma dei requisiti realizzato in Visual Paradigm. Per ulteriori dettagli consultare i sub diagram di tale diagramma all'interno del progetto.

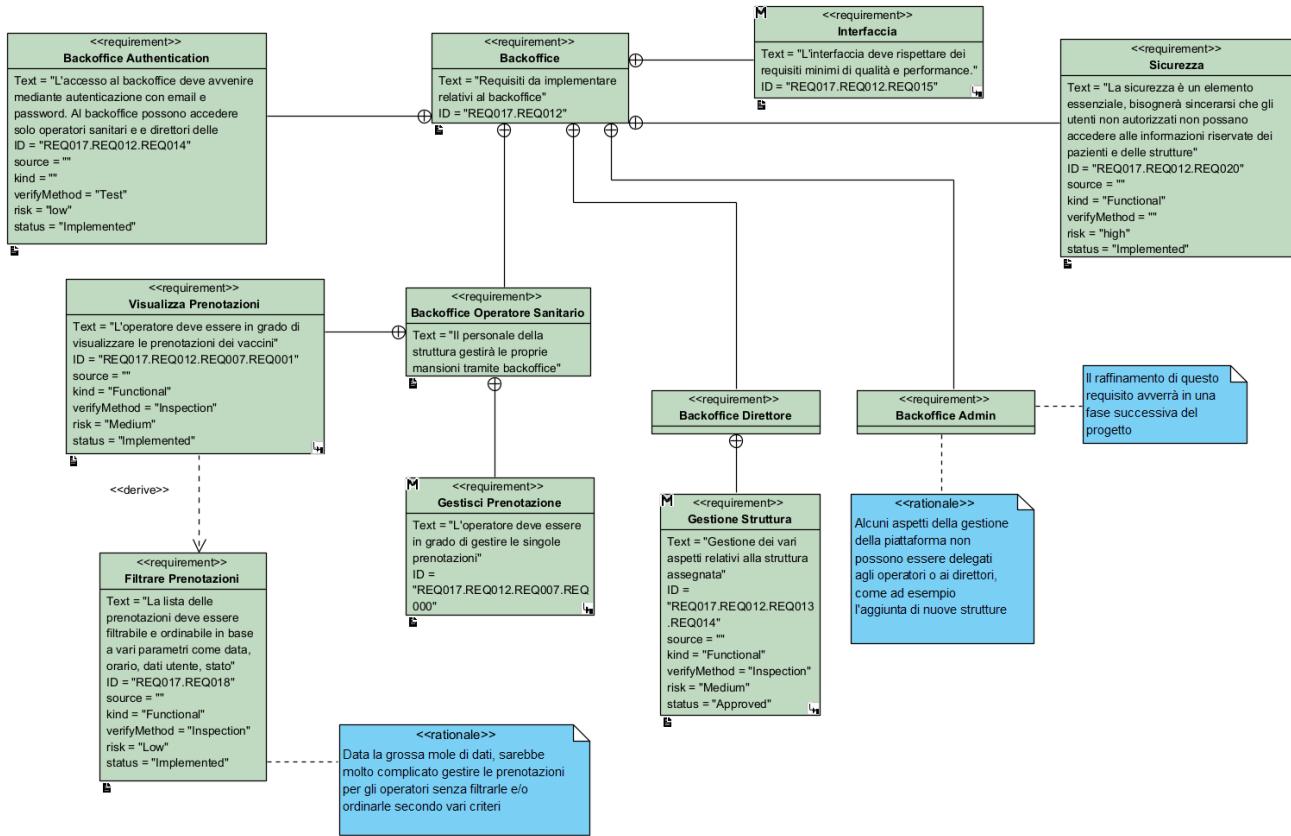


Figura 2.1: Diagramma dei Requisiti back office

2.2 Modello di contesto

Rappresenta il confine tra il sistema (inteso come hardware e software) e il mondo esterno. Come si vede dal diagramma il paziente può interagire con il sistema di prenotazioni ed effettuare le stesse operazioni sia attraverso uno Smartphone Android, sia attraverso un web Browser, mentre il responsabile sanitario può interagire col sistema solo attraverso web browser.

Il sistema quindi presenta 3 interfacce, di cui 2 per interagire con un web browser e una per interagire con uno smartphone Android.

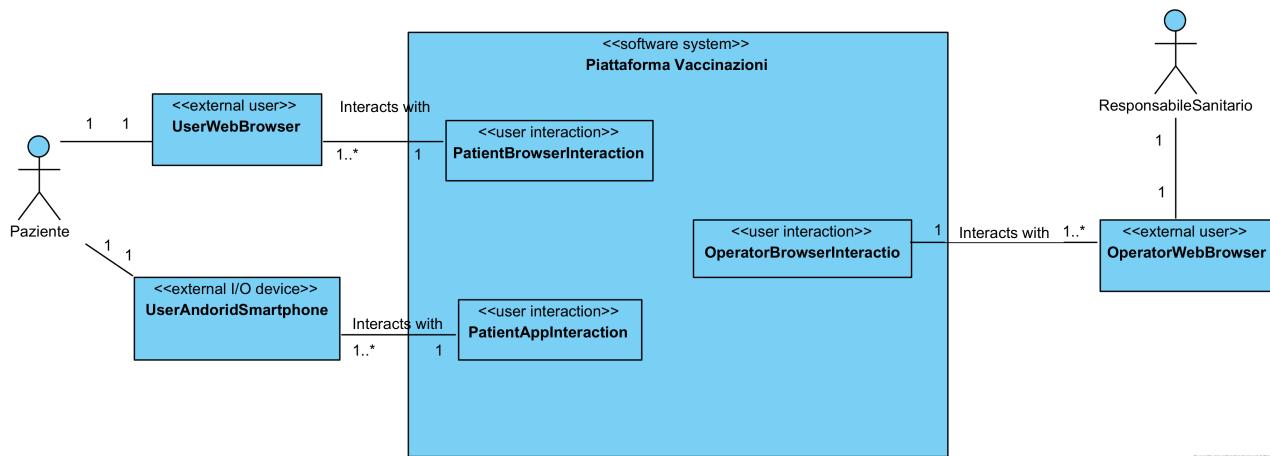


Figura 2.2: Software system context class diagram

2.3 Modello di dominio

In fase di analisi serve a stabilire le entità del dominio applicativo che hanno lo scopo di memorizzare e fornire accesso ai dati persistenti. Sono classi con attributi e relazioni, ma senza le operazioni, che saranno stabilite nella fase di progettazione.

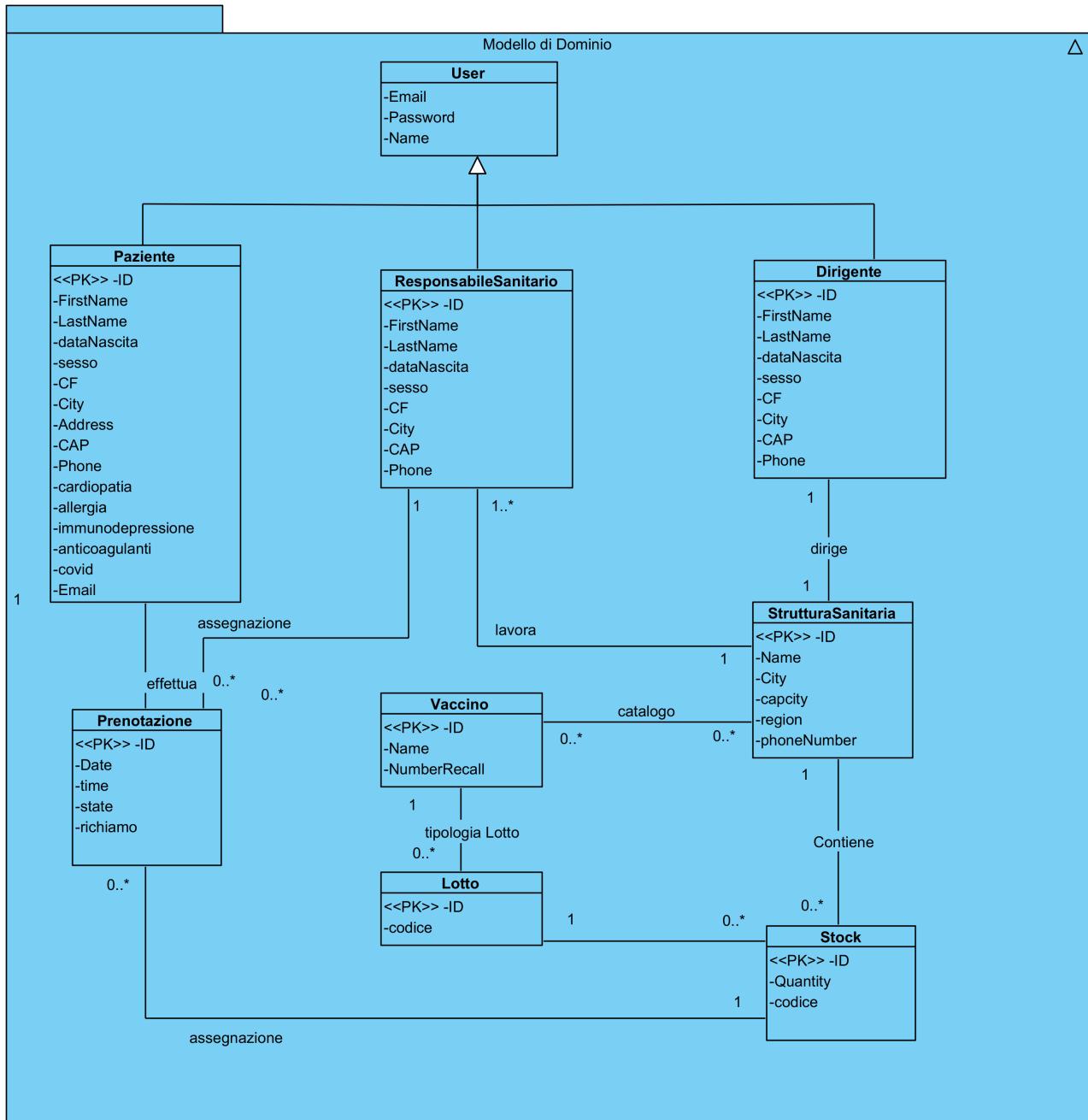


Figura 2.3: Modello di dominio

2.4 Altri diagrammi

In questa sezione si specificano altri diagrammi di alto livello che permettono di comprendere meglio i requisiti funzionali.

2.4.1 Diagrammi di attività di alto livello

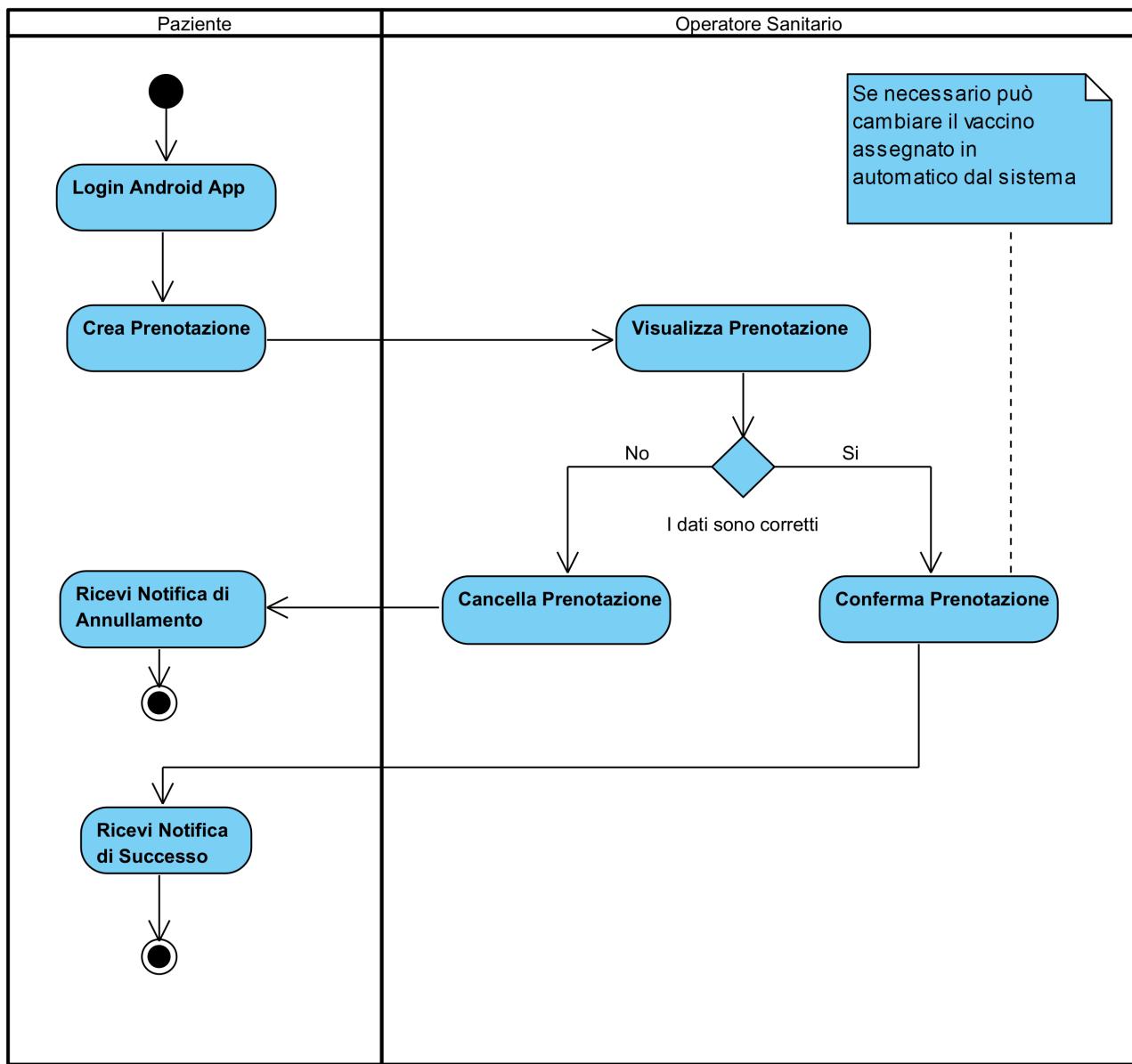


Figura 2.4: Activity prenotazione

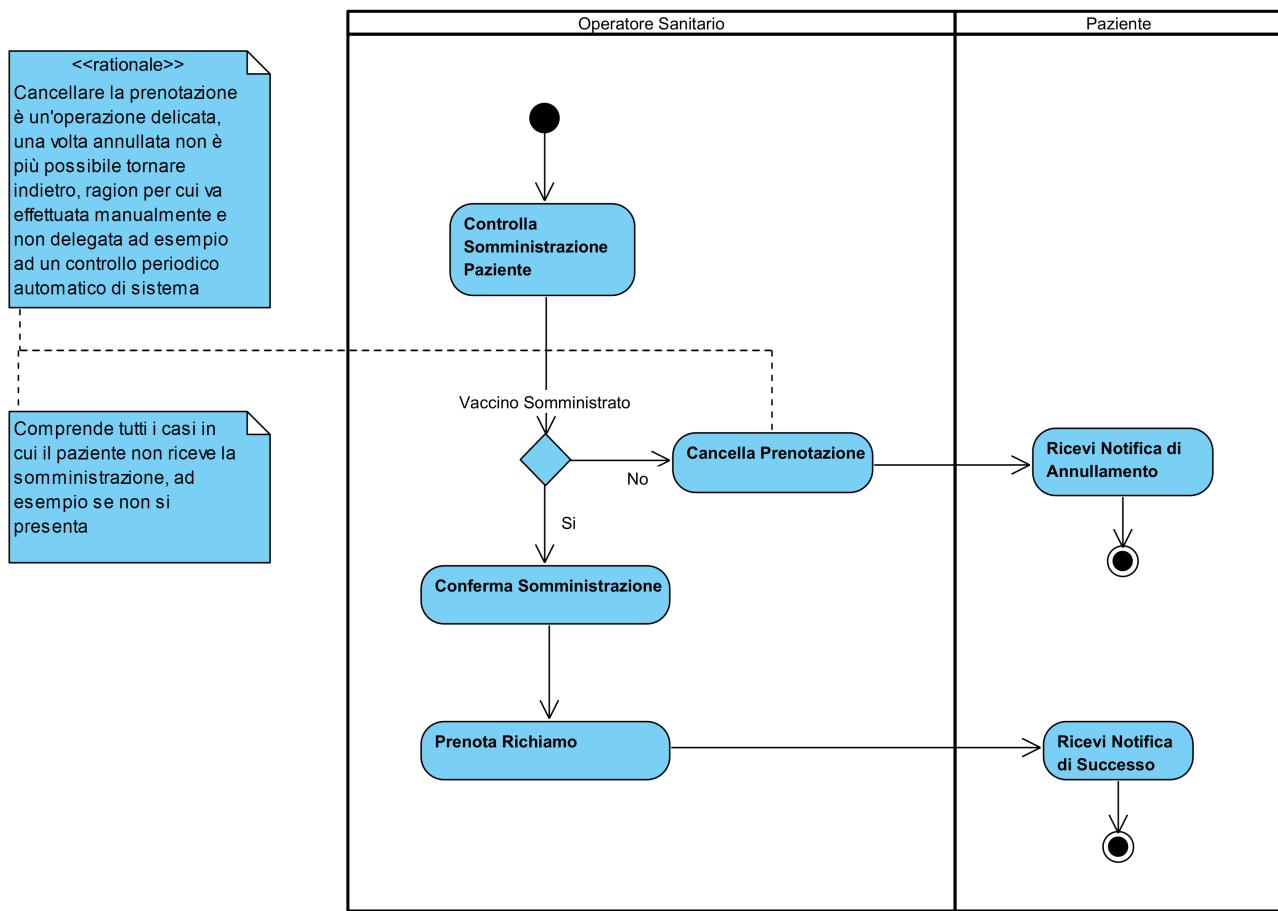


Figura 2.5: Activity somministrazione

2.4.2 Diagramma di stato

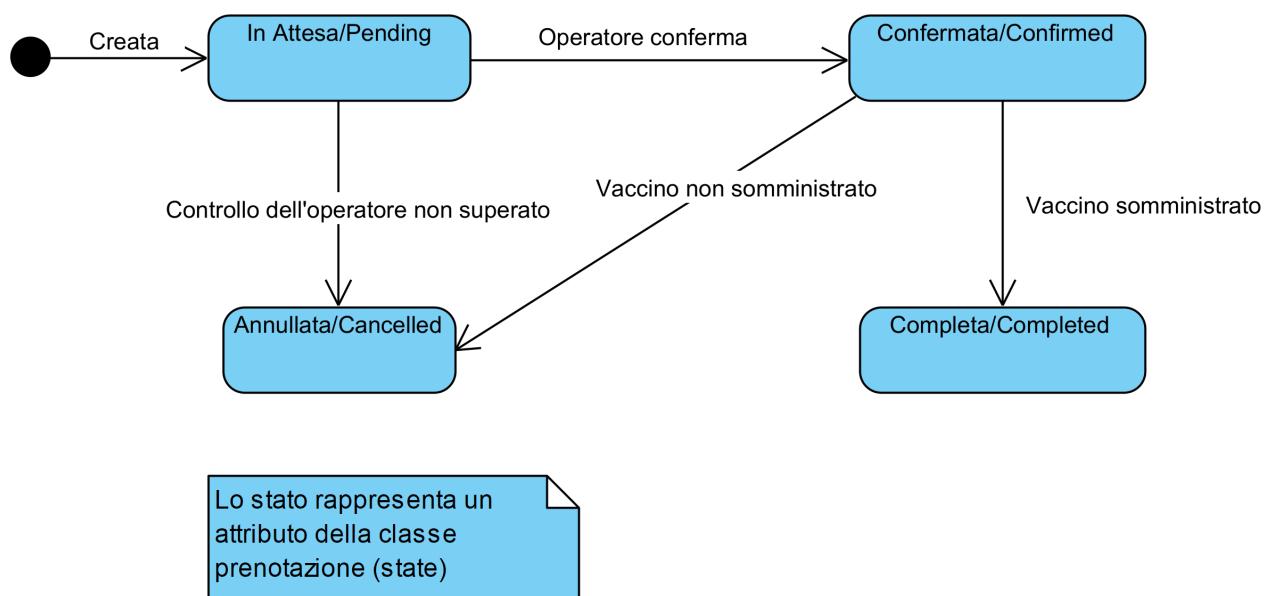


Figura 2.6: diagramma di stato Prenotazione

2.4.3 Site Map

Utili a comprendere meglio come deve essere la navigabilità delle pagine dell'applicazione.

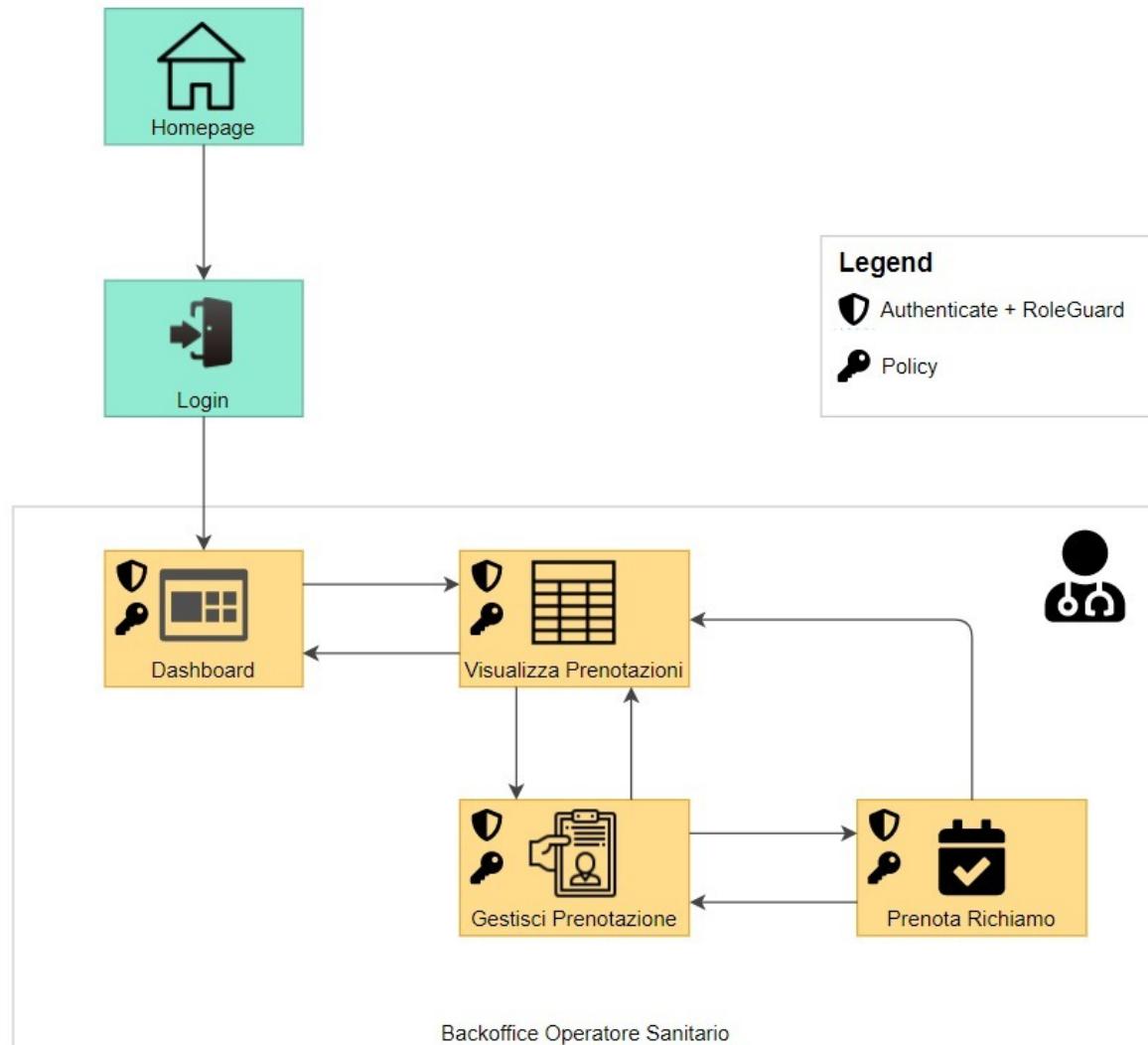


Figura 2.7: Site map lato backoffice

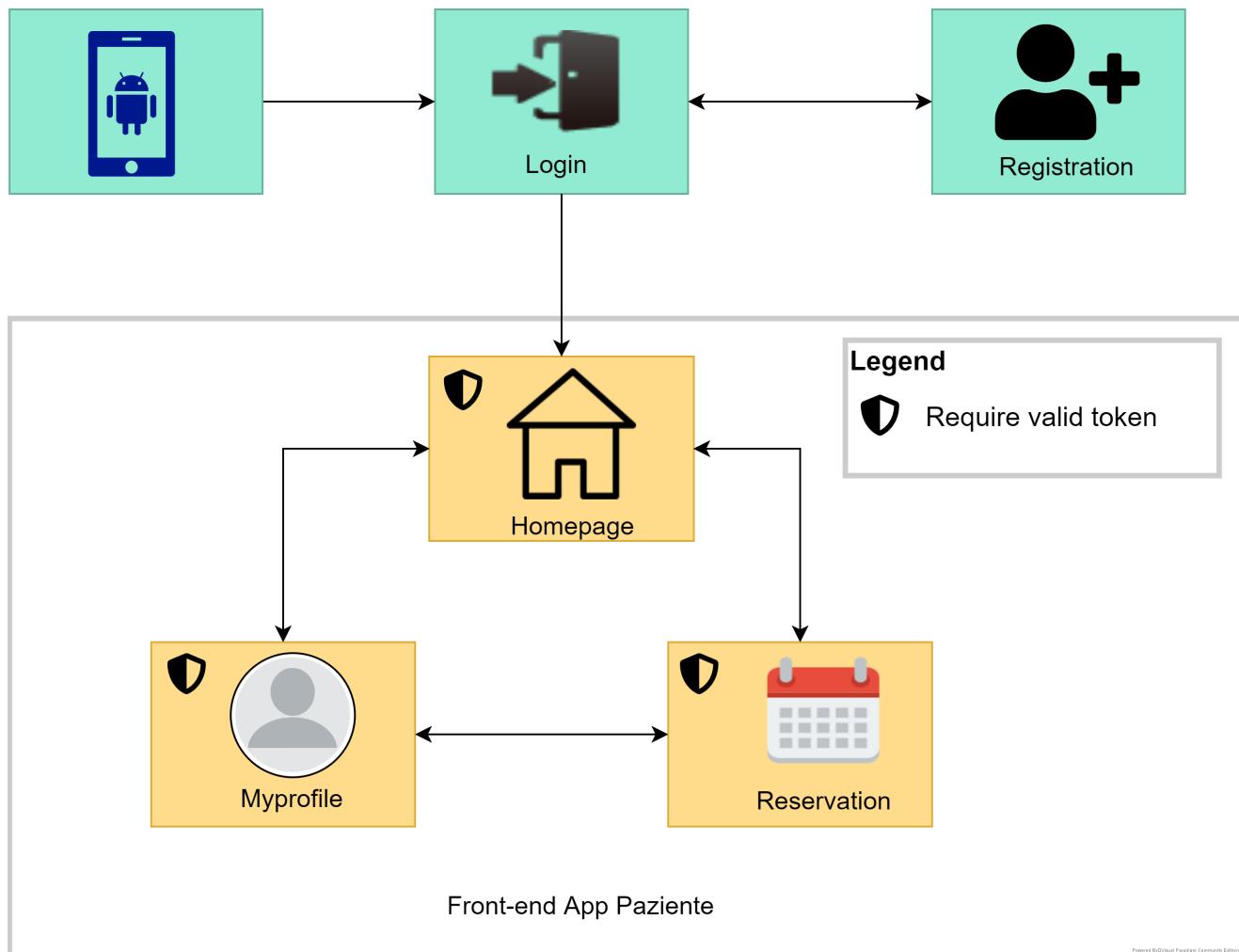


Figura 2.8: Site map Android

Capitolo 3

DOCUMENTAZIONE DI PROGETTO

3.1 Scelte di progetto

Di seguito sono riportate le scelte di implementazione effettuate.

3.1.1 Pattern Architetturali utilizzati

3.1.1.1 Model-View-Controller

Il pattern architettonico Model-View-Controller possiede diverse varianti, per ulteriori informazioni riguardo a quella utilizzata dal framework Laravel si riporta al capitolo 5.

L'MVC è stato scelto in quanto particolarmente adatto in quelle applicazioni dove assume molta importanza il layer di presentazione, grazie alla struttura di tipo triangolare che sussiste tra model-view-controller.

L'MVC ha contribuito inoltre a migliorare la manutenibilità del codice.

3.1.1.2 Model-View-ViewModel

Pattern architettonico adottato da Vue, per ulteriori informazioni si riporta al capitolo 5.

Il Model-View-ViewModel si è rivelato particolarmente utile nello sviluppo del frontend, dato che ci ha permesso di separare la logica interna alle view riguardante i dati, e come questi vengono rappresentati e delle view. Ciò ha contribuito ad aumentare la manutenibilità e la testabilità del codice.

3.1.2 Stili Architetturali utilizzati

- **Stile Object Oriented:** i componenti sono oggetti, con dati e operazioni annesse; i connettori sono dati scambiati tramite messaggi ed invocazione di metodi. Ogni oggetto è responsabile dell'integrità della loro rappresentazione interna e questa è nascosta ad altri oggetti. L'adozione di tale stile ci ha consentito di avere interfacce ben definite e di utilizzare principi di progettazione come l'information hiding e l'incapsulazione che aumentano la modificabilità. Inoltre, ogni interfaccia verso un dispositivo esterno, un sistema, o un utente è stata incapsulata in una classe boundary, ciò contribuisce ad aumentare la modificabilità.
- **Stile Client-Server:** è una variante dello stile a livelli e conta due soli livelli. I componenti sono il client e il server. I primi conoscono l'identità del server e per la nostra applicazione sono di tipo Thin, ovvero il client ha solo la logica di visualizzazione e l'elaborazione è demandata al server. Il server non conosce né il numero né l'identità dei client che si connetteranno ad esso. I connettori sono i protocolli di rete. Gli svantaggi sono dati dalla banda e dalle prestazioni dei server che possono calare o ridursi per eccessive connessioni dei client. I vantaggi sono un elevato livello di astrazione, **facilità nell'evolvere il sistema**, disaccoppiamento tra client e server, **semplicità nel riuso, indipendenza dall'implementazione** (basta che l'interfaccia sia rispettata).

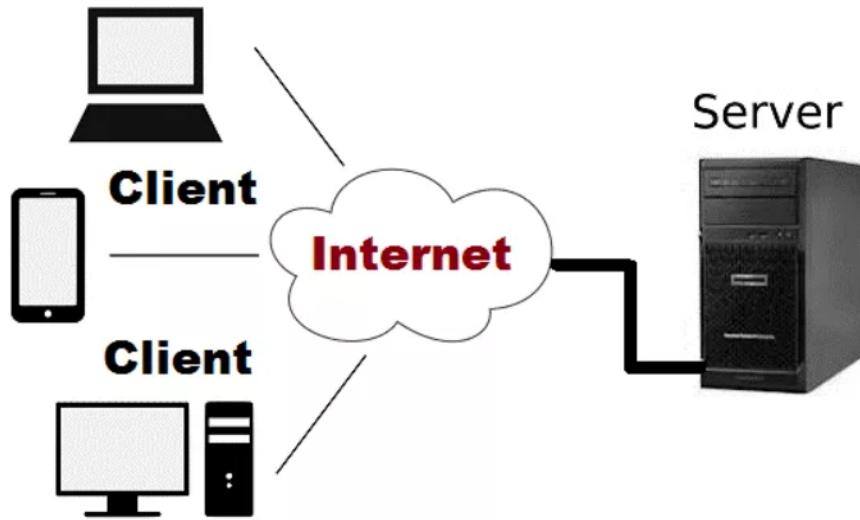


Figura 3.1: Stile Client-Server

- **Representational state transfer (REST):** è uno stile architetturale (di architettura software) per i sistemi distribuiti. Un'API REST, nota anche come API RESTful, è un'interfaccia di programmazione delle applicazioni (API o API web) conforme ai vincoli dello stile di architettura REST, che consente inoltre l'interazione con servizi web RESTful. L'API funge quindi da elemento di intermediazione tra gli utenti o i clienti e le risorse o servizi web che questi intendono ottenere. È anche un mezzo con il quale un'organizzazione può condividere risorse e informazioni assicurando al contempo sicurezza, controllo e autenticazione, poiché stabilisce i criteri di accesso. L'utilizzo dell'API inoltre non impone all'utente di conoscere le specifiche con cui le risorse vengono recuperate o la loro provenienza.

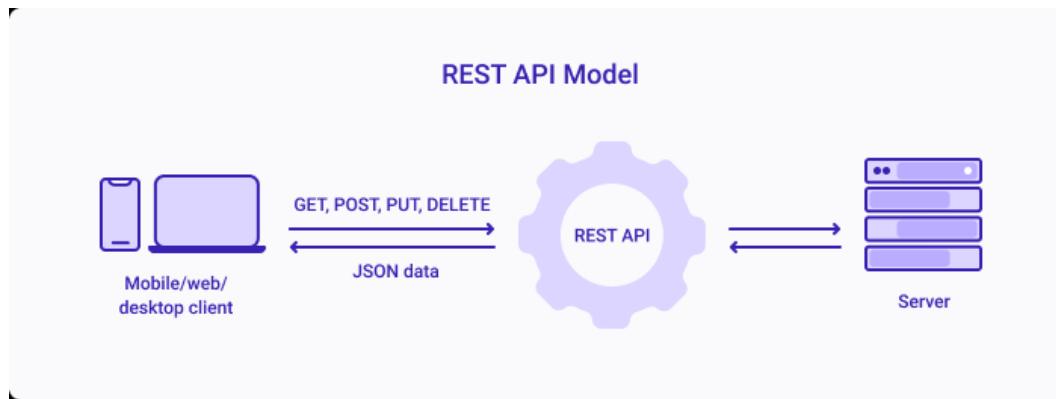


Figura 3.2: Protocollo REST

3.1.3 Design Pattern utilizzati

I design pattern forniscono una soluzione ad un problema ricorrente di basso livello. In particolare sono stati adottati i seguenti design pattern:

3.1.3.1 Repository e Validation¹

I repository costituiscono un ulteriore livello di astrazione tra i dati usati dall'applicazione e quelli contenuti nel database, mentre le classi di validation incapsulano la logica di validazione dei dati che devono essere salvati. Le operazioni effettuate su una model, al momento della creazione o del salvataggio, potrebbero comprendere degli step aggiuntivi che seguono un determinato pattern comune che bisognerebbe ripetere in ogni punto del programma. Ad esempio, nel caso degli stock, deve essere generato un codice alfanumerico casuale ogni qualvolta ne viene creato uno nuovo, o più in generale tutti i salvataggi nel database, per salvaguardare la consistenza e la correttezza dei dati, dovrebbero essere validati prima di effettuare la richiesta al DB.

Invece di ripetere ogni volta queste operazioni, abbiamo creato un repository e una classe di validation per ciascun business object che incapsulano tali operazioni.

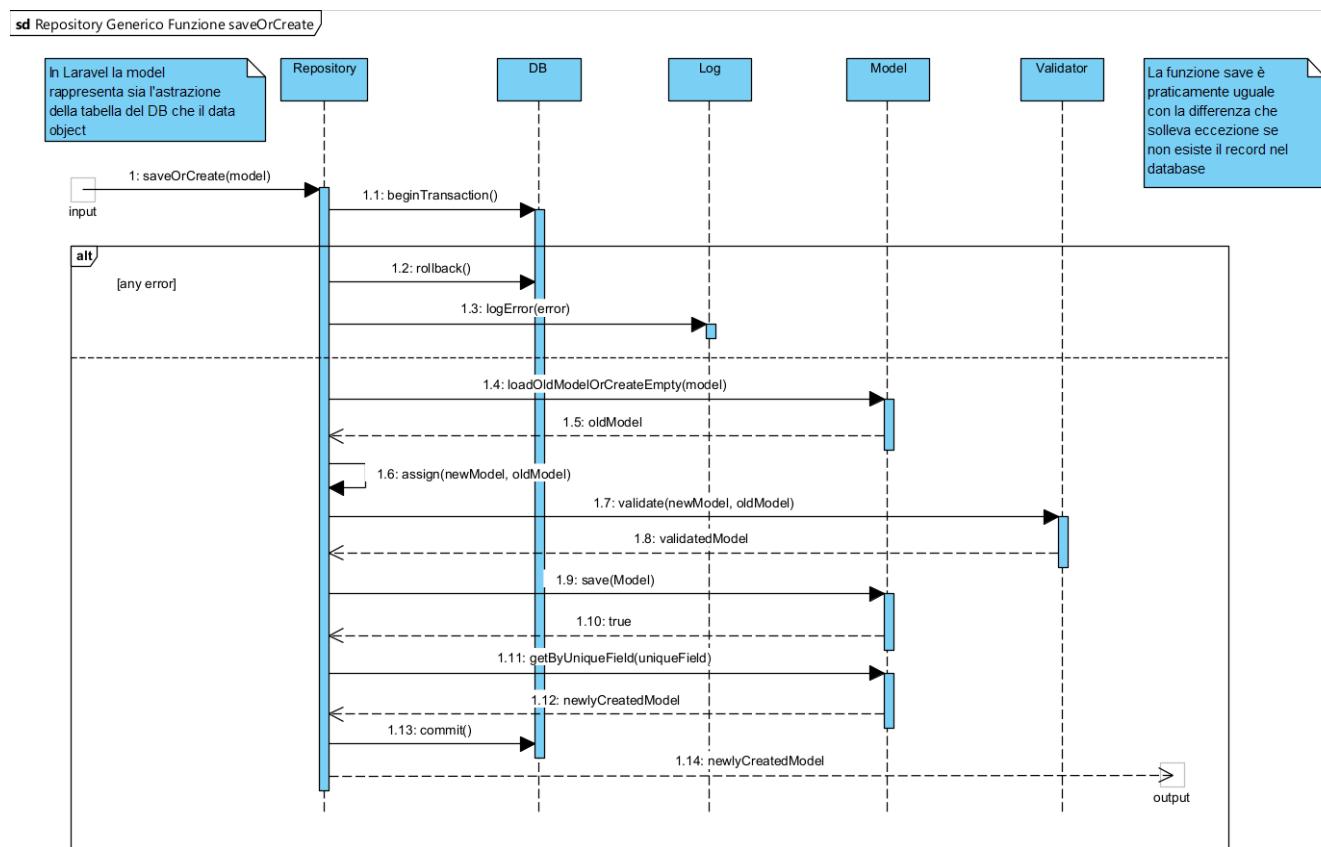


Figura 3.3: Funzionamento di un Repository

3.1.3.2 Observer

Il pattern observer definisce una dipendenza uno a molti tra oggetti diversi, in maniera tale che se l'oggetto cambia il suo stato tutti gli altri vengono notificati. E' stato utilizzato per aggiornare le strutture con il timestamp relativo all'ultima modifica effettuata sulle prenotazioni in maniera tale da notificare e aggiornare le view dell'operatore sanitario con le nuove modifiche.

¹ Validation non è considerato proprio un design pattern, piuttosto una best practice

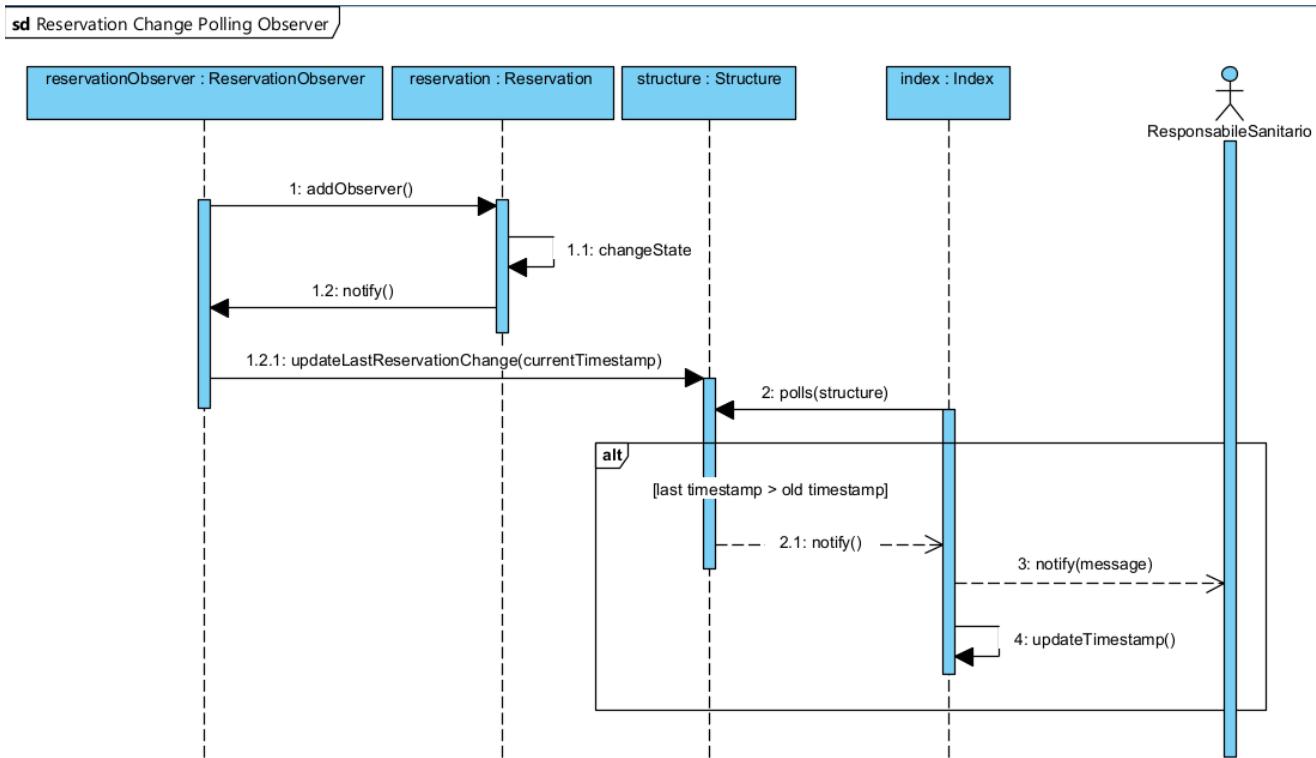


Figura 3.4: Funzionamento di un Repository

3.1.3.3 Dependency Injection

Dependency injection (DI) è un design pattern della Programmazione orientata agli oggetti il cui scopo è quello di semplificare lo sviluppo e migliorare la testabilità di software di grandi dimensioni.

Per utilizzare tale design pattern è sufficiente dichiarare le dipendenze di cui un componente necessita (dette anche interface contracts). Quando il componente verrà istanziato, un iniettore si prenderà carico di risolvere le dipendenze (attuando dunque l'inversione del controllo). Se è la prima volta che si tenta di risolvere una dipendenza l'injector istanzierà il componente dipendente, lo salverà in un contenitore di istanze e lo restituirà. Se non è la prima volta, allora restituirà la copia salvata nel contenitore. Una volta risolte tutte le dipendenze, il controllo può tornare al componente applicativo.

Come descritto in precedenza, l'utilizzo di questo pattern è stato semplificato da Laravel.

3.2 Diagrammi di alto livello

3.2.1 Diagramma delle classi

Si riporta di seguito il diagramma delle classi di progettazione. Questo diagramma è un modello intermedio, ancora indipendente dalla tecnologia implementativa che verrà utilizzata, ed è molto utile per la definizione delle classi e delle operazioni principali che dovranno essere poi implementate a basso livello. E' stato realizzato partendo dal modello di dominio realizzato in una fase precedente. Inoltre è stato raffinato di pari passo con la produzione dei diagrammi di sequenza, poiché analizzando i diagrammi di sequenza si sono potute dedurre alcune operazioni che il sistema deve offrire e che quindi vanno inserite nelle classi. Si è utilizzata inoltre una progettazione guidata dalle responsabilità con i principi GRASP.

Nel diagramma sono presenti le classi necessarie alla realizzazione dei casi d'uso delle iterazioni eseguite.

L'applicazione del pattern GRASP «information expert» ha portato alla definizione di una nuova classe (ContenitoreStrutture), dato che non era già presente una classe che avesse tutte le informazioni della classe StrutturaSanitaria.

L'applicazione del pattern GRASP «information expert» ha portato alla definizione di una nuova classe (ContenitorePazienti), dato che non era già presente una classe che avesse tutte le informazioni della classe Paziente.

L'applicazione del pattern GRASP «controller» ha portato alla definizione di una classe di tipo session controller (PrenotazioneHandler) che ha lo scopo di ricevere e controllare le richieste provenienti dalle classi UI.

Le classi UI rappresentano delle interfacce utente che trasmettono l'input dell'utente al controller. In questo caso ogni schermata utente avrà un propria classe associata. Inoltre non contengono logica applicativa nei propri metodi.

Questo diagramma verrà raffinato ulteriormente nelle fasi successive, dove si è scelto il pattern architettonale da utilizzare (MVC) e la tecnologia per l'implementazione. Nel diagramma delle classi di implementazione ogni classe UI sarà mappata con una classe View, la classe PrenotazioneHandler sarà proprio una classe di tipo Controller del pattern MVC e le classi entity saranno mappate con delle classi Model. Inoltre, in fase di implementazione non ci sarà bisogno di utilizzare le classi information expert poiché il framework permette di avere i riferimenti alle entità senza ricorrere a classi aggiuntive.

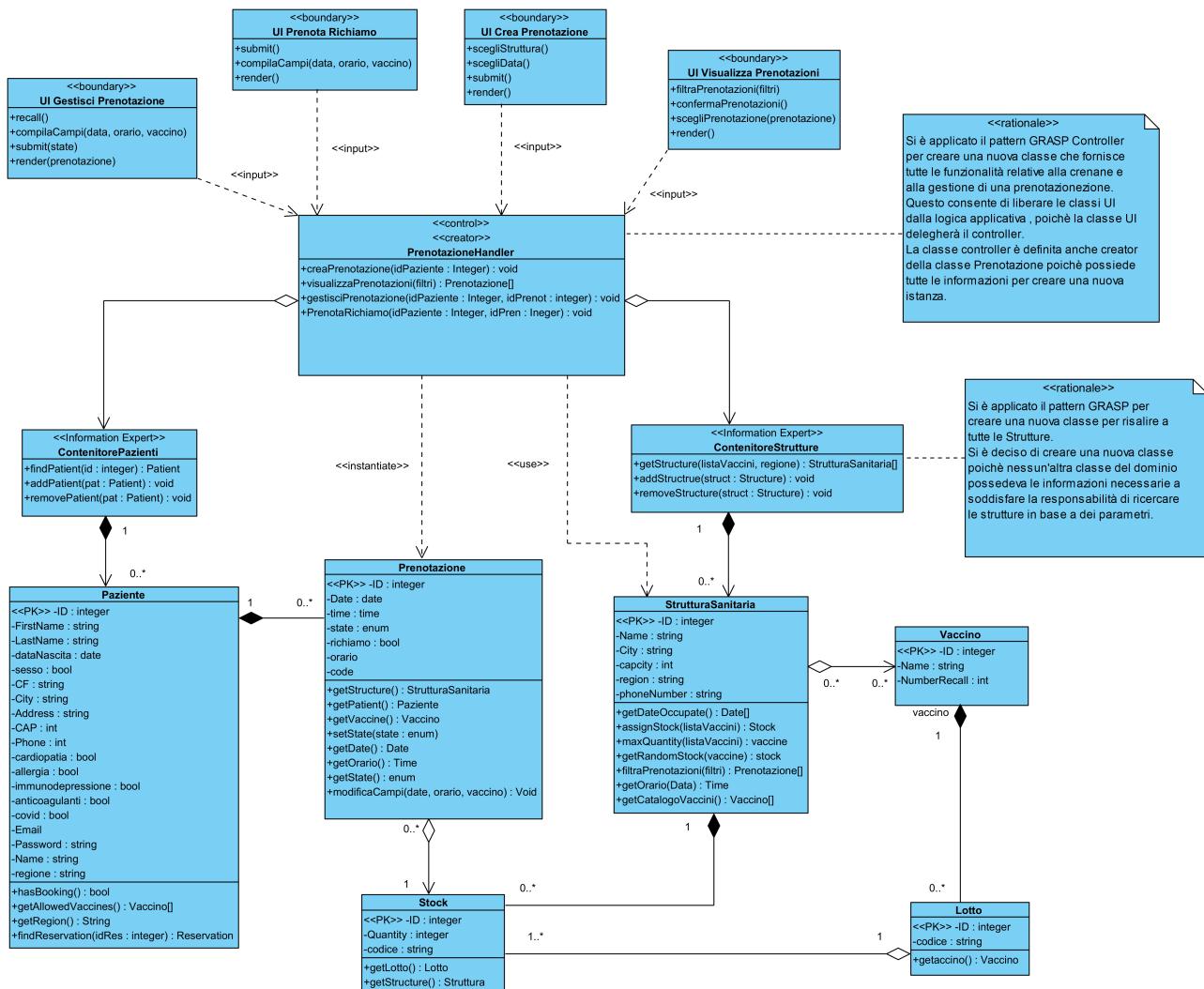


Figura 3.5: Diagramma della classi di alto livello

3.2.2 Diagrammi di sequenza

I sequence diagram mostrano gli scambi di messaggi e le interazioni tra le varie classi al fine di realizzare un caso d'uso. Sono stati realizzati utilizzando esclusivamente classi e operazioni presenti nel diagramma delle classi. Per

una corretta progettazione delle operazioni conviene partire da questi diagrammi, cosicché analizzando i messaggi scambiati fra gli oggetti, si possano cogliere tutte le operazioni effettuate, i dati adoperati e il loro tipo. Inoltre ci hanno permesso di capire quali fossero le funzioni principali della logica di business da incorporare nelle classi controller e model.

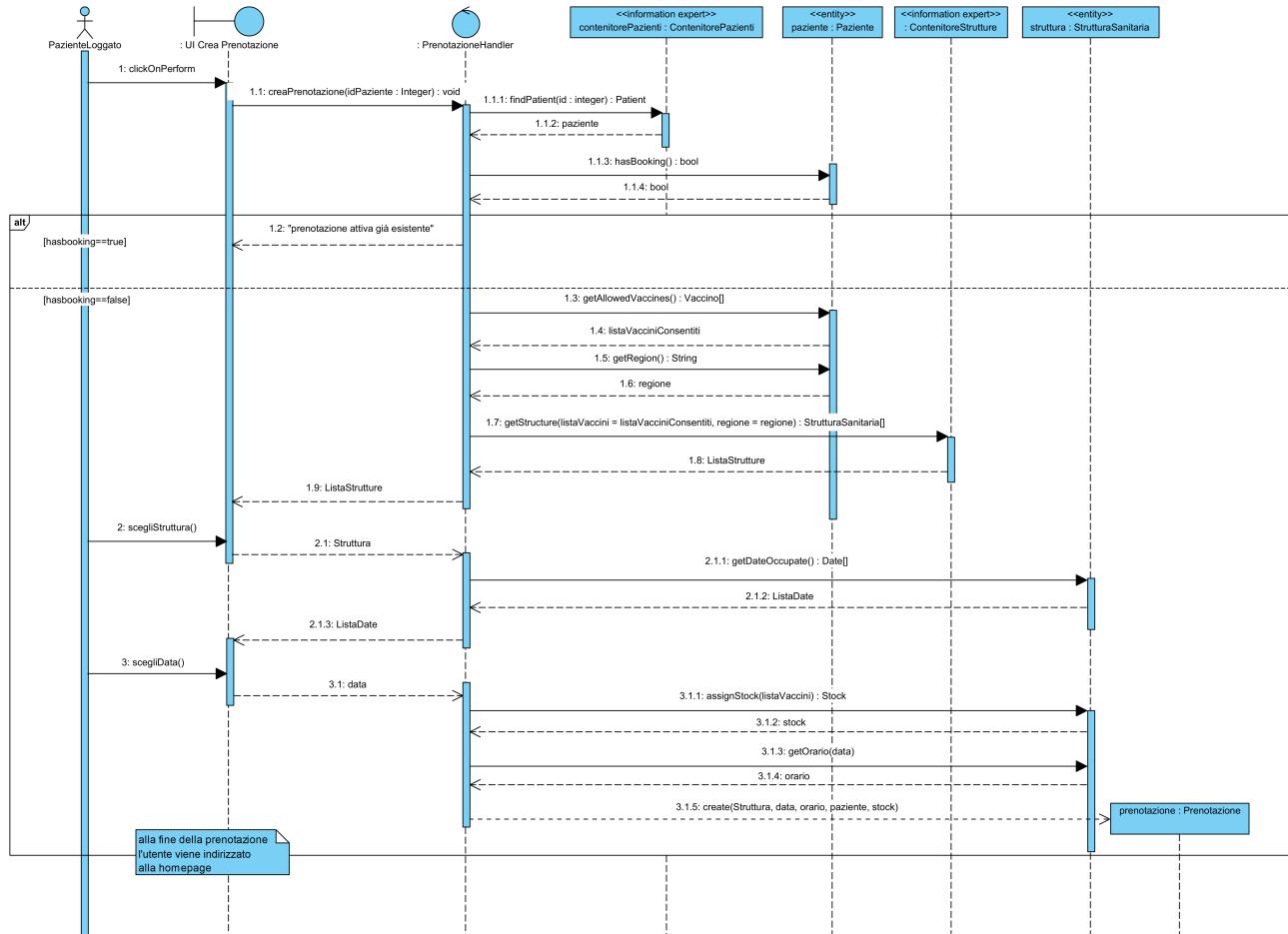


Figura 3.6: Diagramma di sequenza crea prenotazione (progettazione)

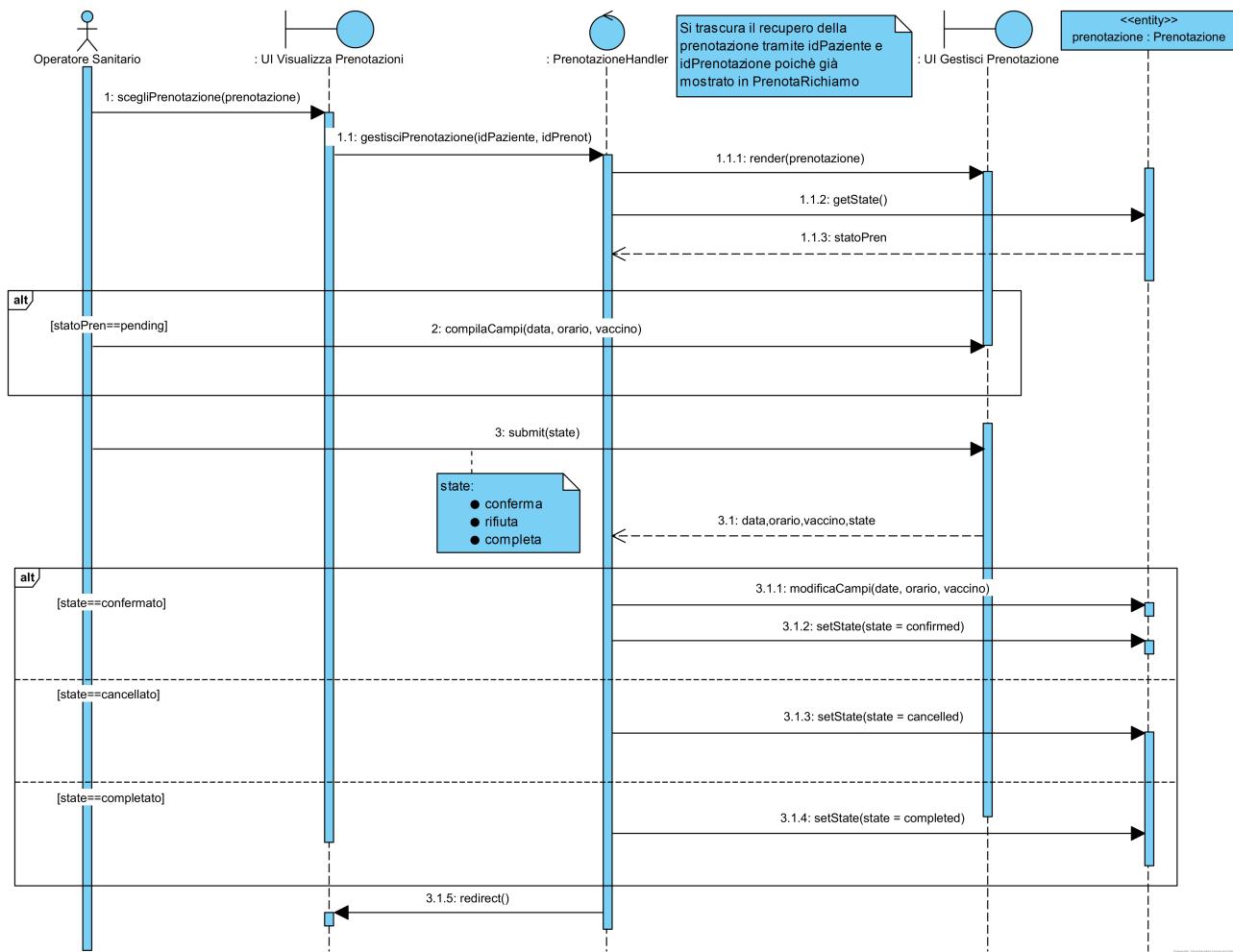


Figura 3.7: Diagramma di sequenza Gestisci prenotazione (progettazione)

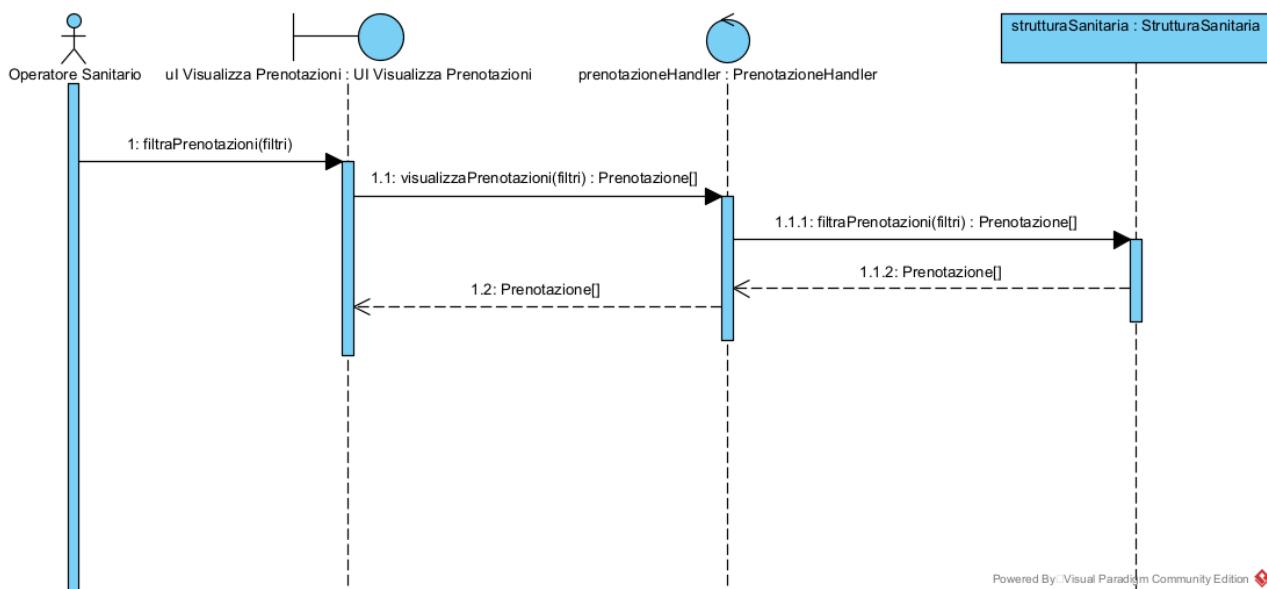


Figura 3.8: Diagramma di sequenza Visualizza prenotazioni (progettazione)

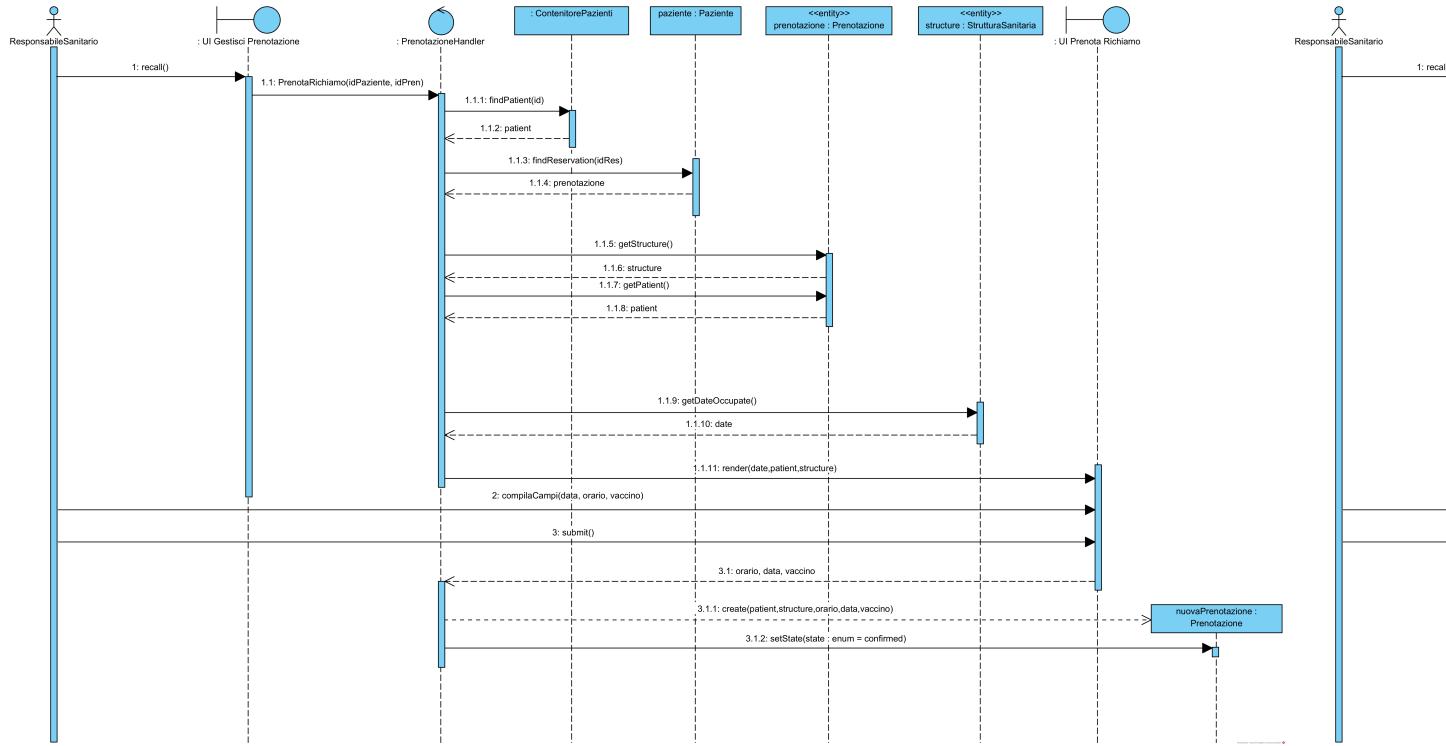


Figura 3.9: Diagramma di sequenza Prenota richiamo (progettazione)

3.3 Diagrammi di Basso Livello

Di seguito si riportano i diagrammi di basso livello realizzati per documentare le applicazioni implementate. Per ulteriori dettagli si rimanda al file di progetto di Visual Paradigm.

3.3.1 Server e Web App

Il server costituisce il cuore dell'applicazione, di seguito sono riportati e descritti i package utilizzati per l'implementazione del progetto:

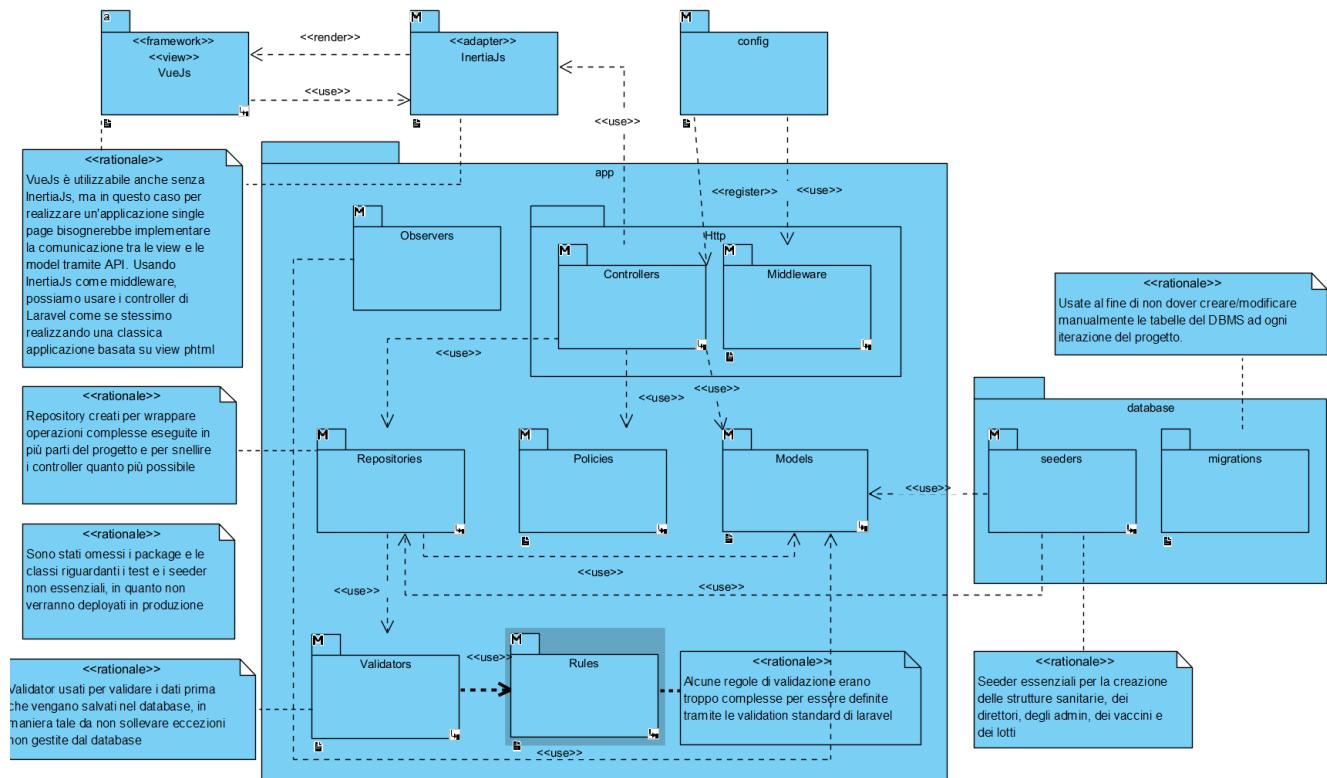


Figura 3.10: Web App Package

3.3.1.1 VueJs

Tale package contiene tutti i file relativi al framework del frontend ampiamente descritto prima. Esaminandolo più nel dettaglio, questo contiene ulteriori sub-packages.

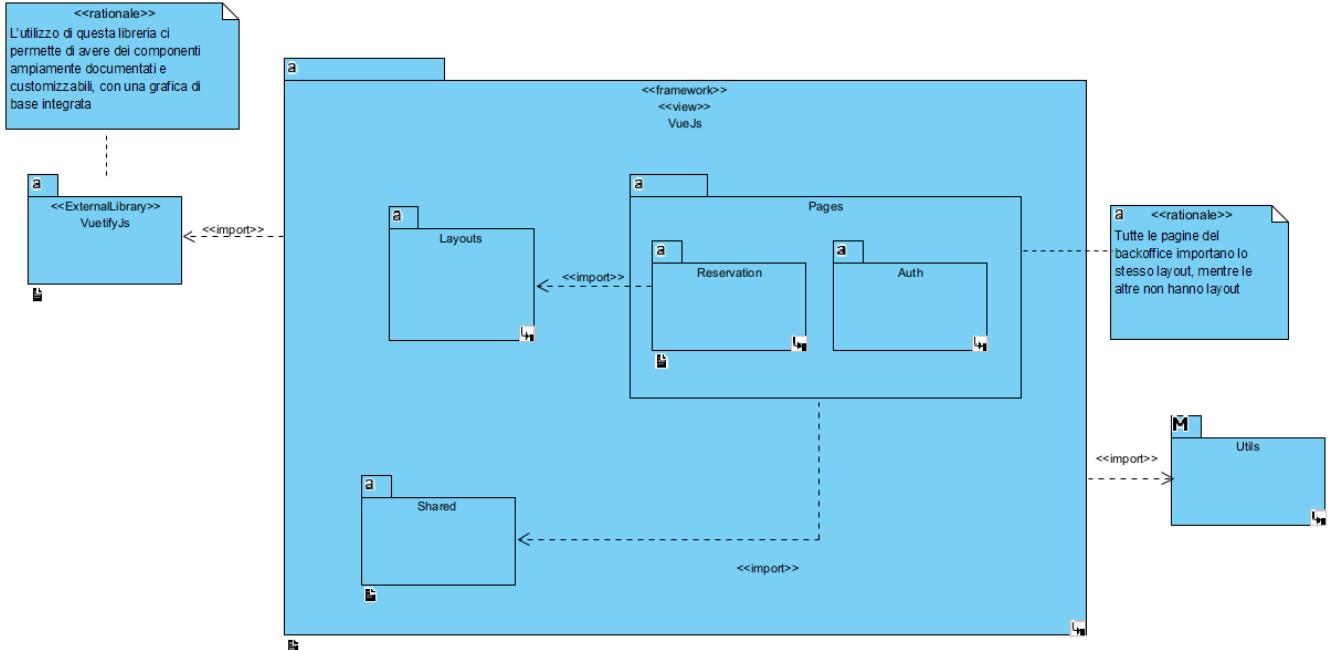


Figura 3.11: VueJs Package

Shared: Contiene i componenti vue implementati.

Pages/Reservation: Contiene le pagine vue (che sono comunque dei componenti) relative alla gestione delle prenotazioni da parte dell'operatore sanitario:

Pages/Auth: Qui risiedono le pagine vue relative all'autenticazione per accedere al backoffice.

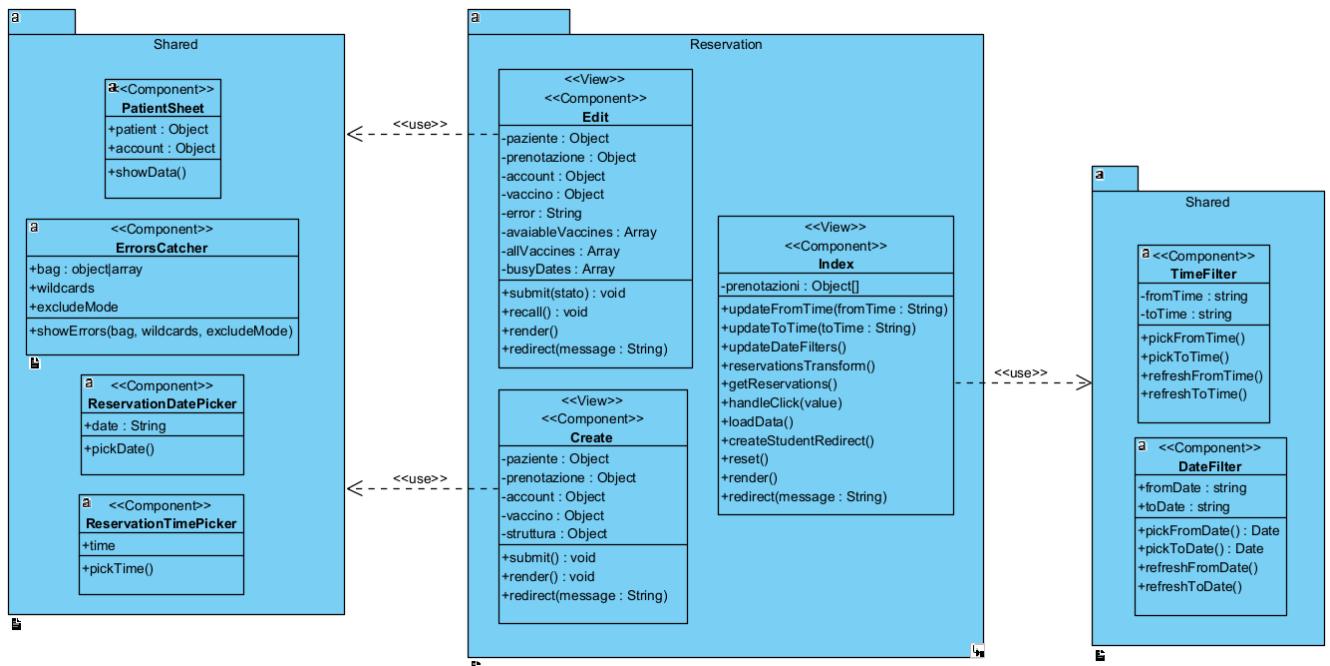


Figura 3.12: Pages Class Diagram

3.3.1.2 Middleware

Definiscono operazioni che vengono effettuate dal framework tra l'elaborazione delle varie richieste. Ad esempio il middleware Auth (definito dal framework), prima che la pagina venga caricata, controlla per le route per le quali è stato aggiunto tale middleware, che l'utente sia loggato e in caso contrario fa redirect alla home.

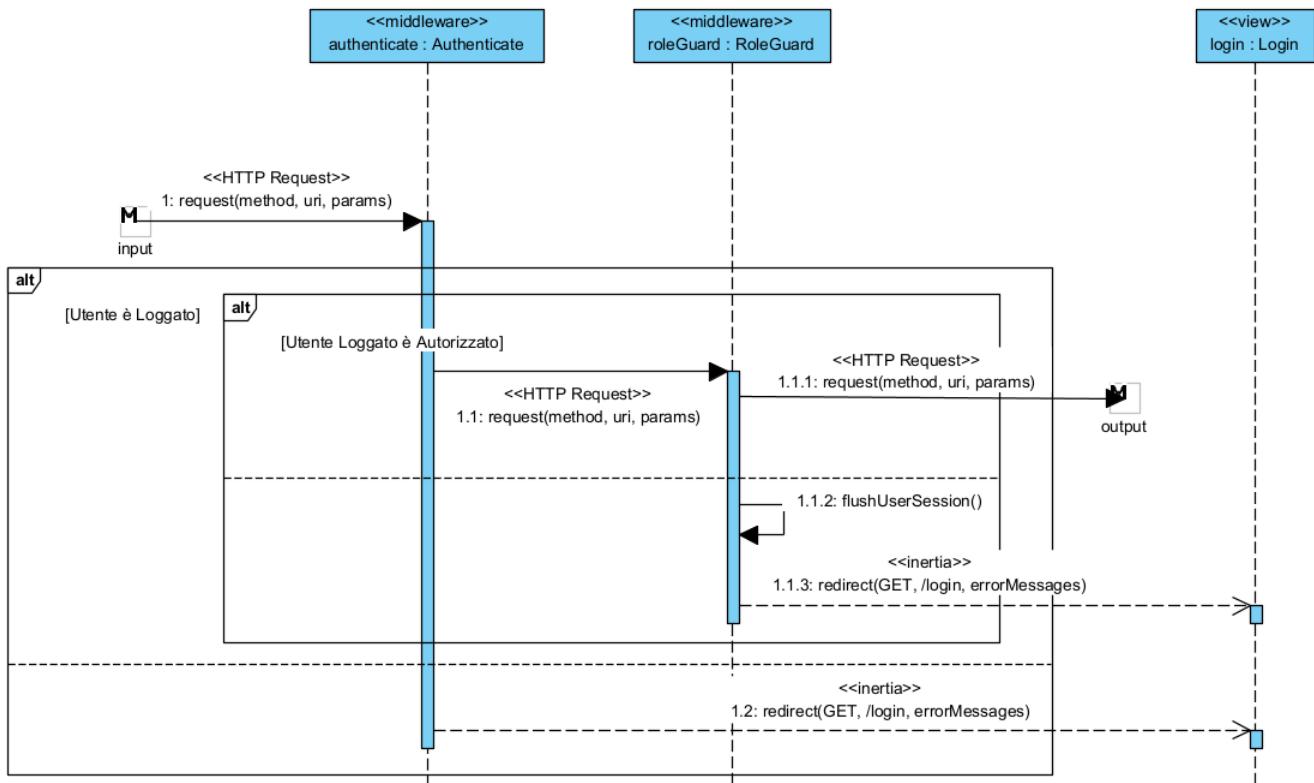


Figura 3.13: Middleware Backoffice Guard Sequence Diagram

3.3.1.3 Policies

Definiscono le politiche di accesso relativamente ad una certa risorsa in maniera più specifica rispetto ai middleware.

3.3.1.4 Repositories e Validators

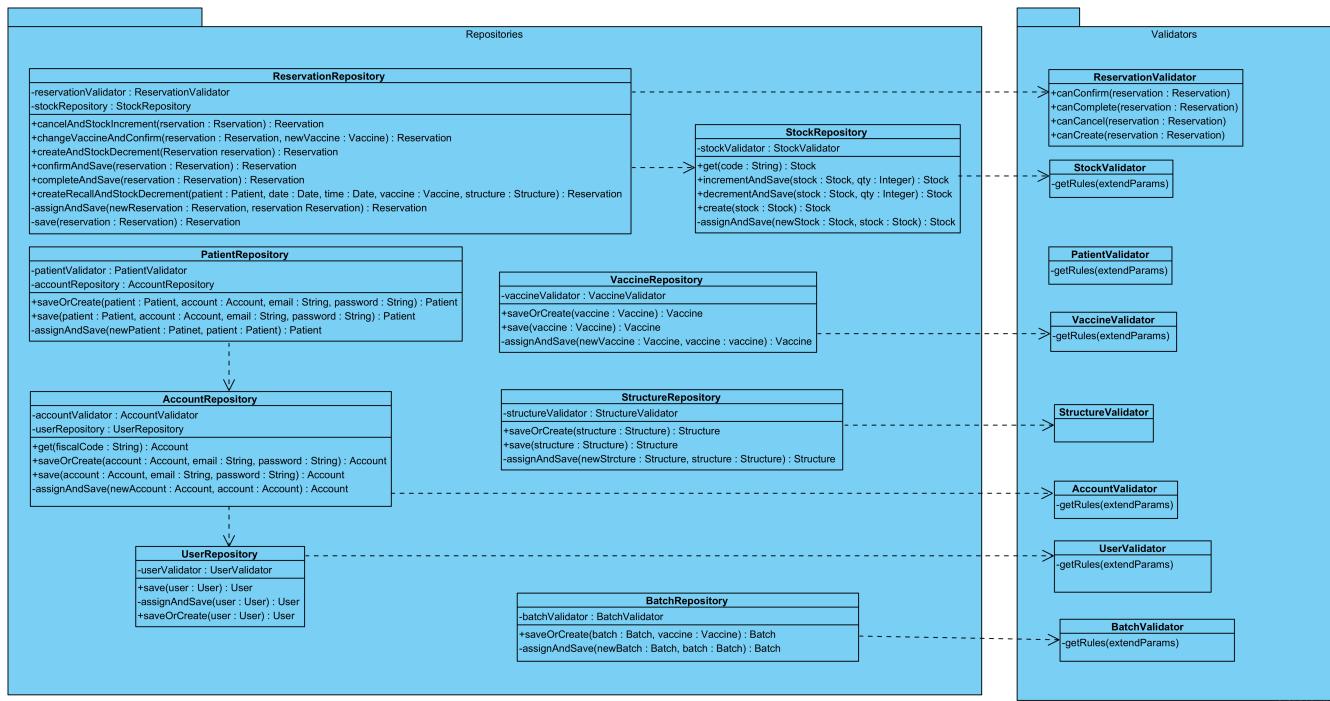


Figura 3.14: Repository e Validations class diagram

Grazie alle classi dei package middleware, validator e policies il sistema rispetta tutti i vincoli di sicurezza relativi ai requisiti non funzionali elencati nel capitolo sulla Fase di Avvio.

3.3.1.5 Rules

Usate per definire delle regole di validazione più complesse rispetto a quelle di base fornite da Laravel.

3.3.1.6 Config

Contiene i file PHP all'interno dei quali sono registrati e associati i metodi dei controller con le routes, nonché i metodi dei controller con gli endpoint delle API.

3.3.1.7 Seeders

Qui è definita la logica di riempimento del database a inizio fase di produzione.

3.3.1.8 Migrations

Le migration definiscono lo schema delle tabelle del Database.

3.3.1.9 Models

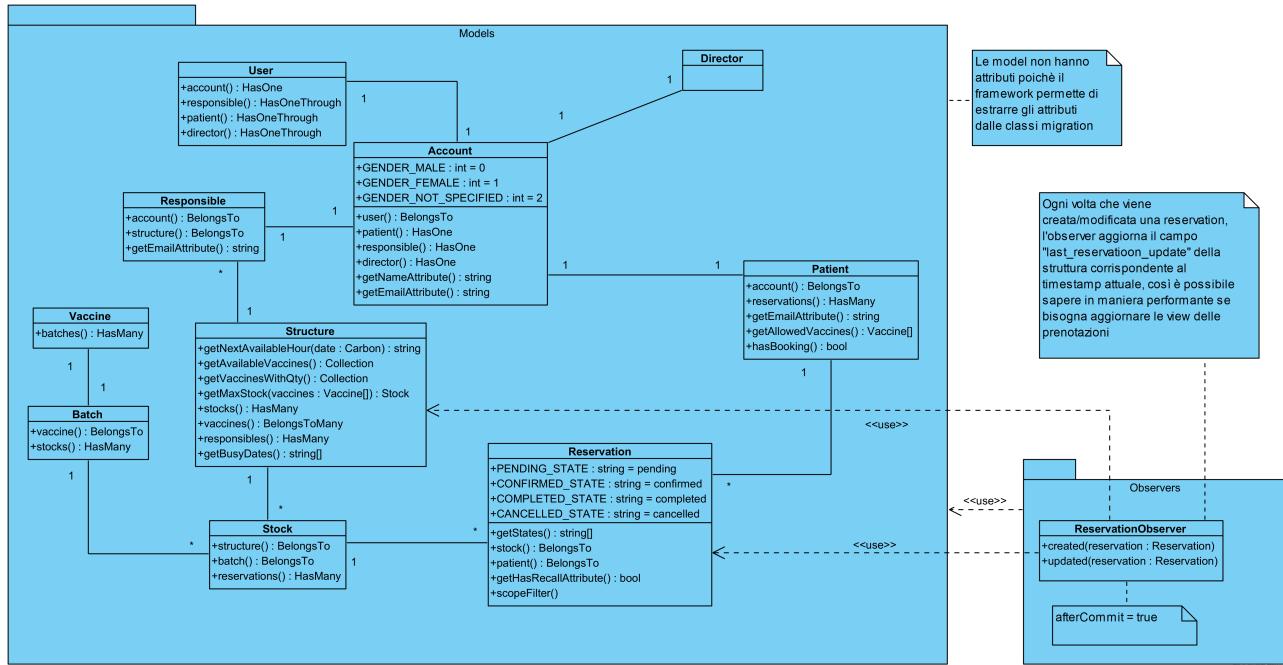


Figura 3.15: Models class diagram

3.3.1.10 Controllers

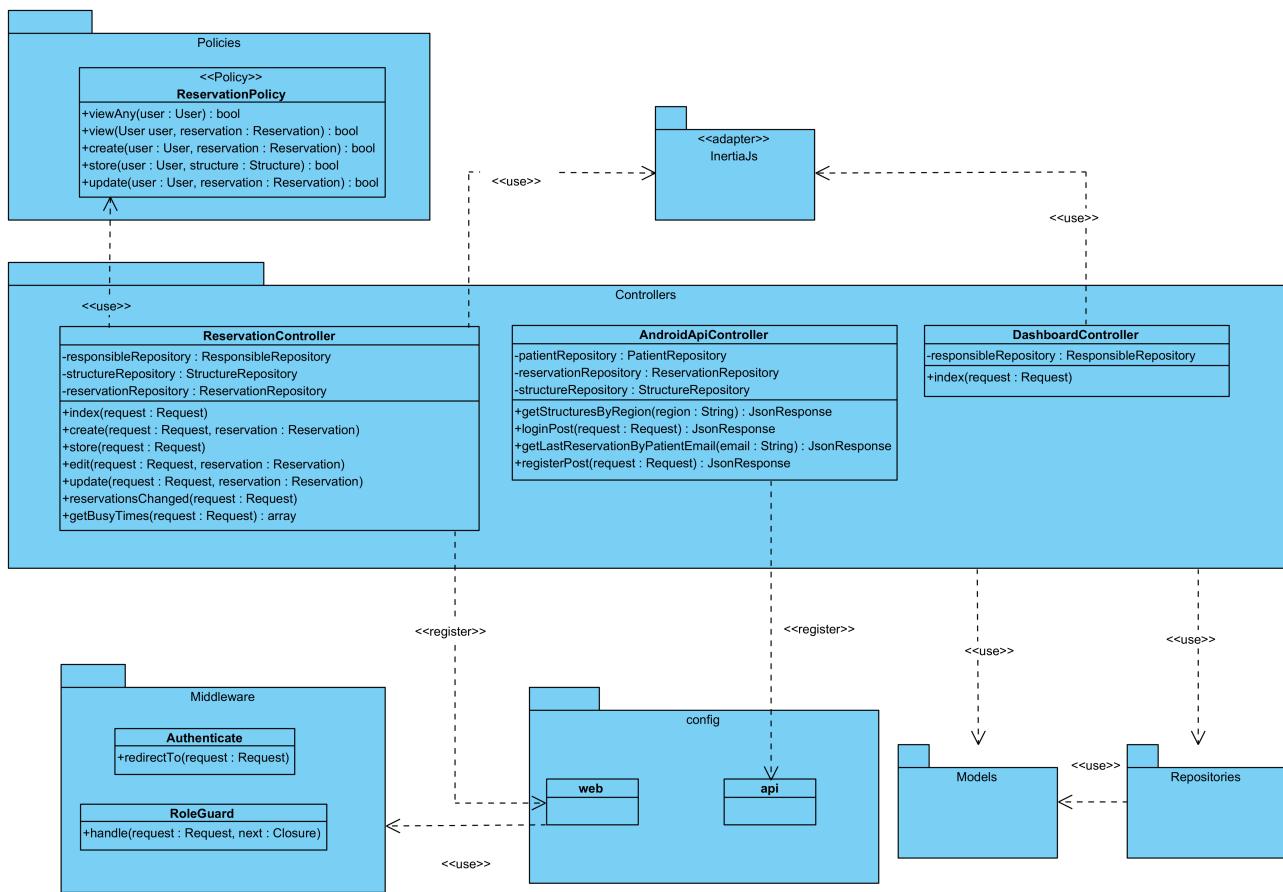


Figura 3.16: Controller class diagram

3.3.2 App Android

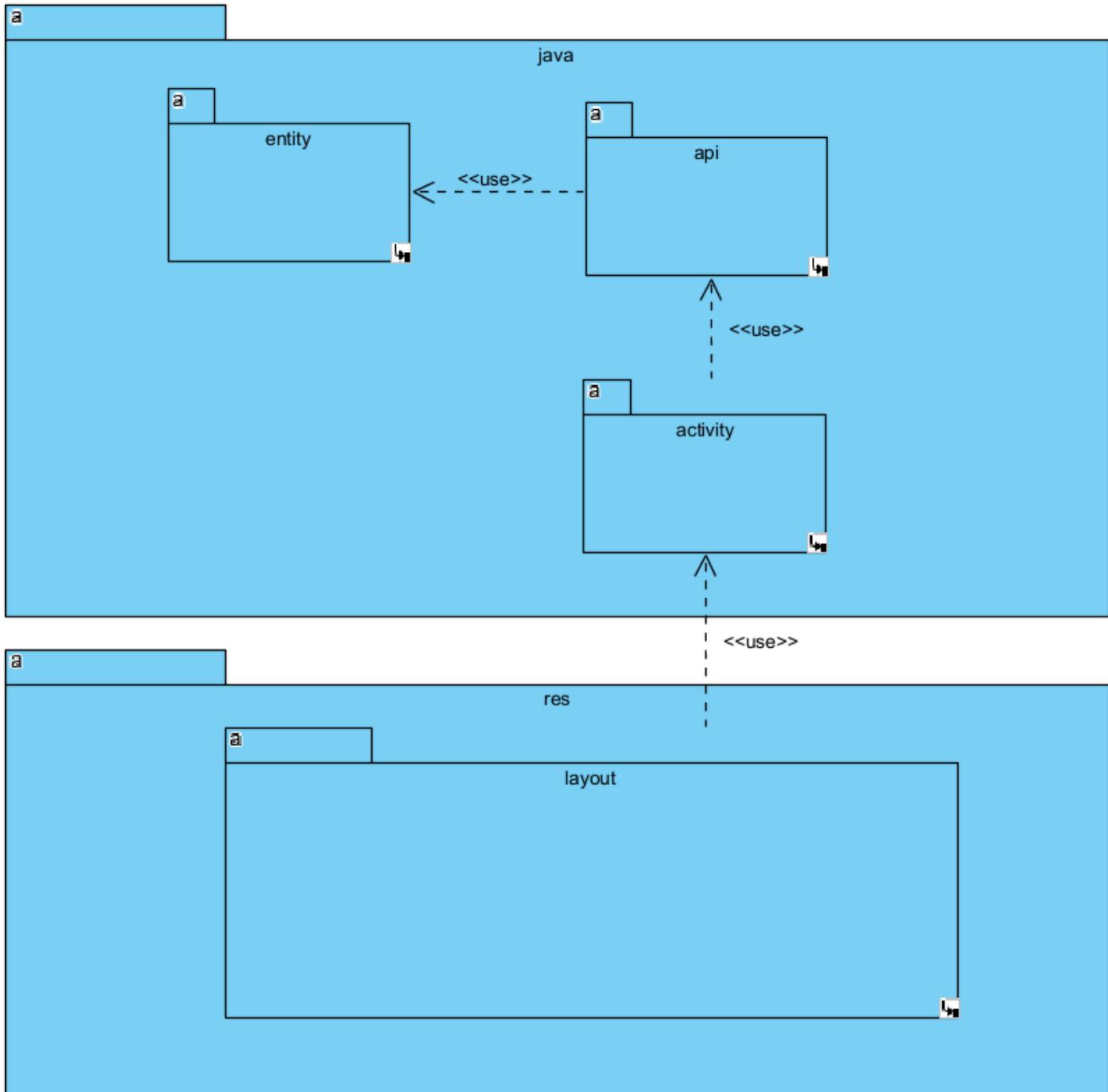


Figura 3.17: Package App Android

I componenti dell'applicazione sono elementi costitutivi di base di un'applicazione Android. Questi componenti sono accoppiati in modo lasco con `AndroidManifest.xml` che descrive ogni componente dell'applicazione e come interagiscono.

Di seguito sono riportati i componenti principali utilizzati per l'applicazione.

3.3.2.1 Activity

Sono gli elementi usati per realizzare l'interfaccia utente delle applicazioni. Ogni activity è qualcosa che l'utente può fare (es. scorrimento, click, ecc.). Un'app può avere una sola activity o più activities. Un' activity può ospitare

uno o più fragment alla volta.

Per creare un'Activity è necessario fare due cose:

- estendere la classe Activity o una da essa derivata, appartenente al framework Android;
- registrare l'Activity nell'AndroidManifest.xml mediante l'uso dell'apposito tag XML <activity>.

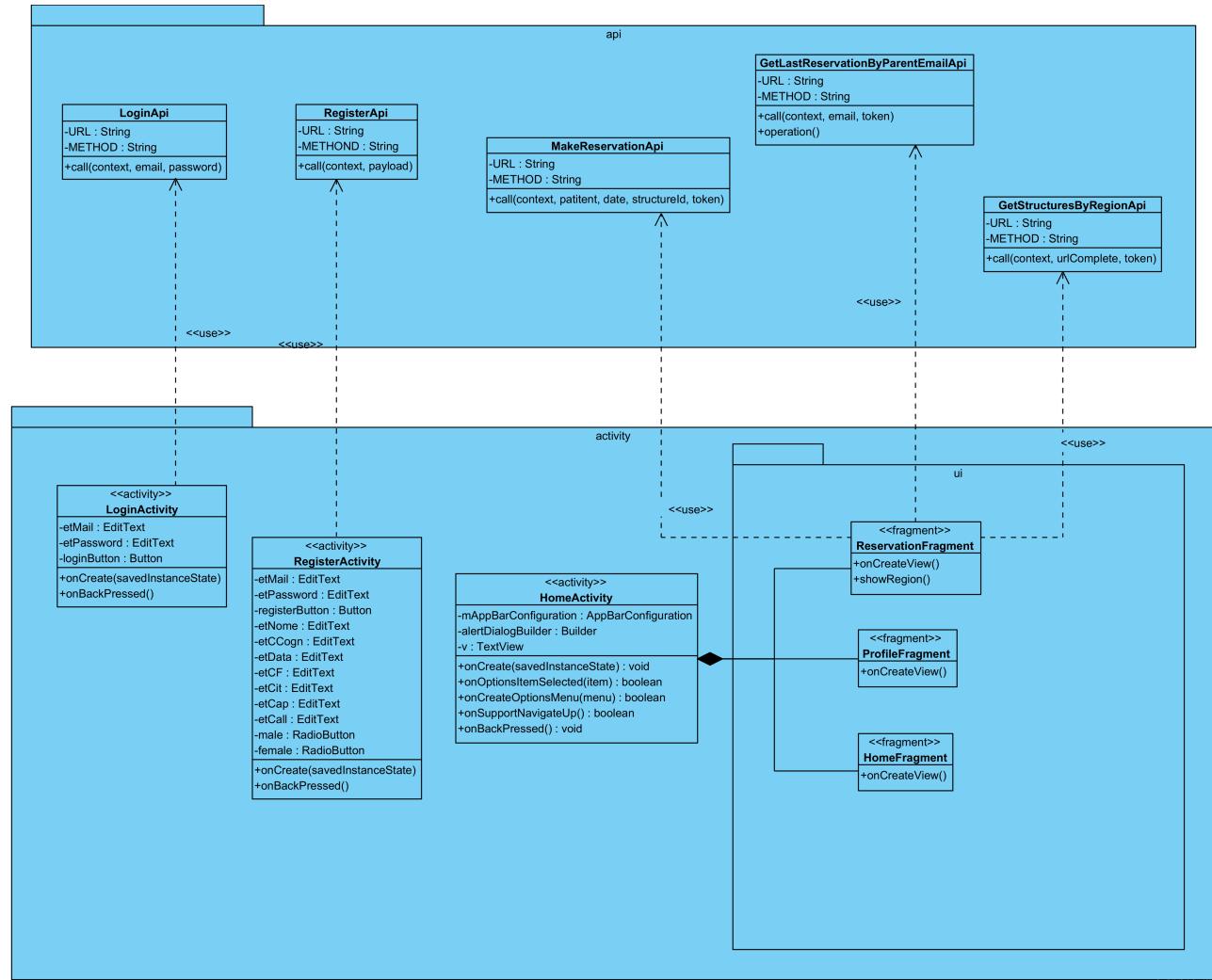


Figura 3.18: Diagramma delle classi Activity Android

3.3.2.2 Componenti di Vincolo

Componenti di vincolo utilizzati:

- **Manifest** - Il Manifest (chiamato `AndroidManifest.xml`) raccoglie informazioni basilari sull'app, informazioni necessarie al sistema per far girare qualsiasi porzione di codice della stessa. Si occupa delle seguenti cose: da un nome al package Java dell'applicazione, descrive le componenti dell'applicazione, determina quali processi ospiteranno componenti dell'applicazione, dichiara i permessi dell'app e il livello minimo di API Android, elenca librerie necessarie
- **Res** - Padre della cartella Layout, contiene e fornisce testo, immagini, colori, layout alla nostra app.
- **layout** - Contiene l'architettura grafica dei componenti dell'interfaccia utente. Sono definiti in XML in una maniera simile a come viene usato HTML per strutturare le pagine web;

- Fragments - Un Fragment è una porzione di Activity. Non si tratta solo di un gruppo di controlli o di una sezione del layout. Può essere definito più come una specie di sub-activity con un suo ruolo funzionale molto importante ed un suo ciclo di vita.
Inoltre, un Fragment non può vivere senza un'Activity.
- Views - Mostrano sullo schermo elementi dell'interfaccia utente, inclusi pulsanti, elenchi, ecc.
- Intents - Metodo con cui una componente può richiedere l'esecuzione di un'azione da parte di un'altra componente.

3.3.2.3 Entity

Classi contenenti i dati delle entità che interagiscono con l'applicazione.

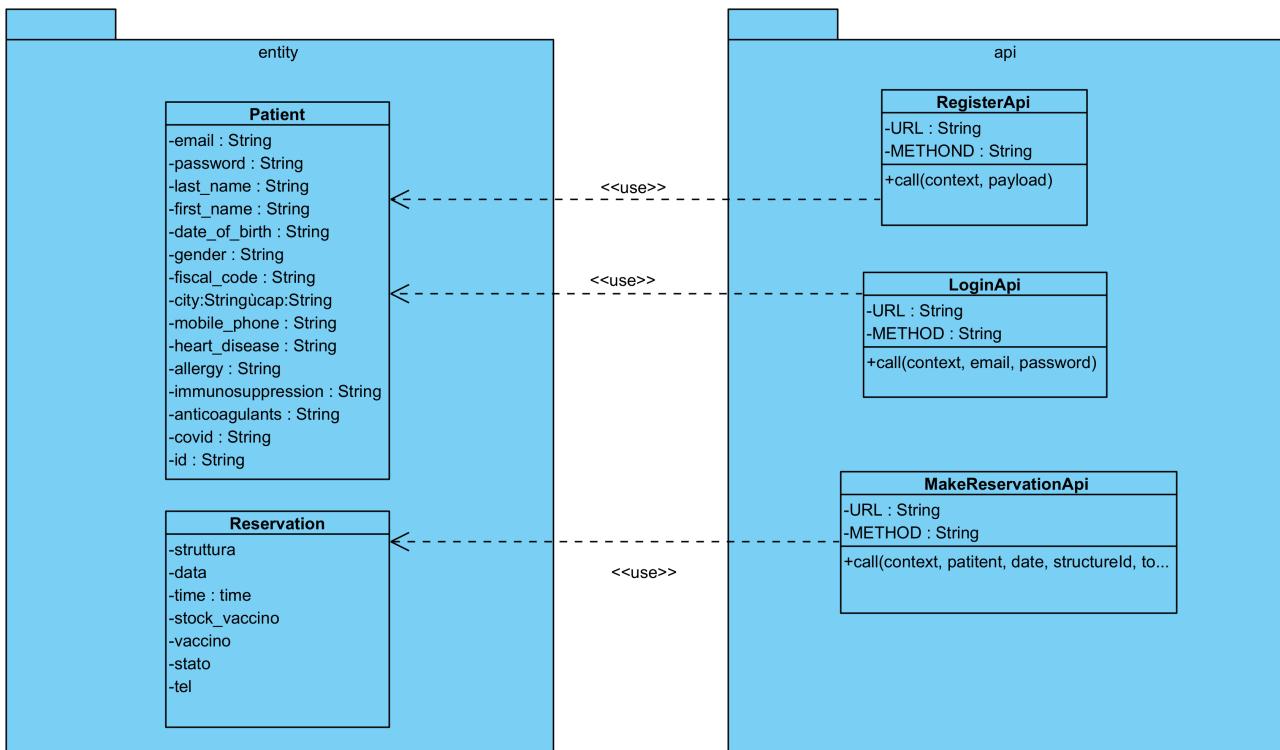


Figura 3.19: Diagramma delle classi Entity Android

3.3.2.4 Api

A causa dell'eccessivo spazio occupato dal codice relativo alle chiamate API su android, si è deciso di separare la logica delle chiamate da quella delle activity. Sono l'equivalente dei Repository dell'architettura.

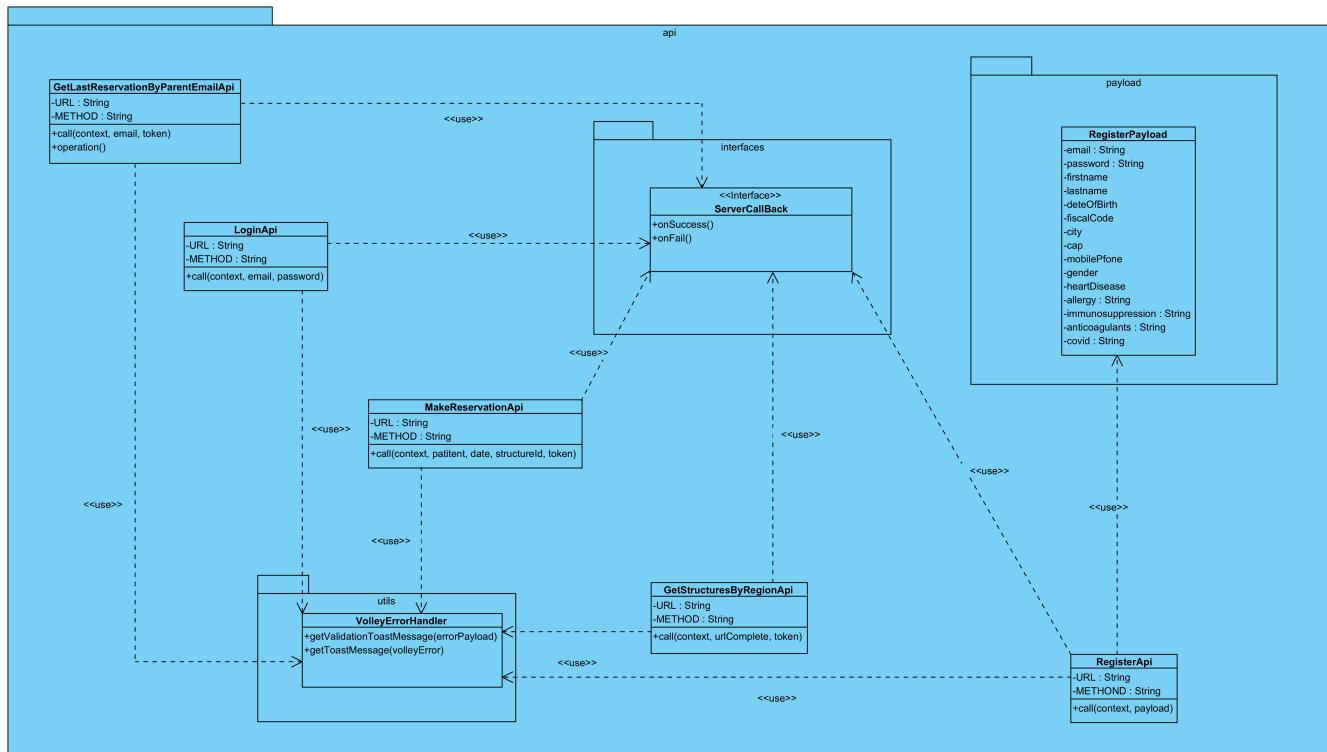


Figura 3.20: Diagramma delle classi Api Android

Il seguente component diagram mostra come si è ottenuto questo risultato:

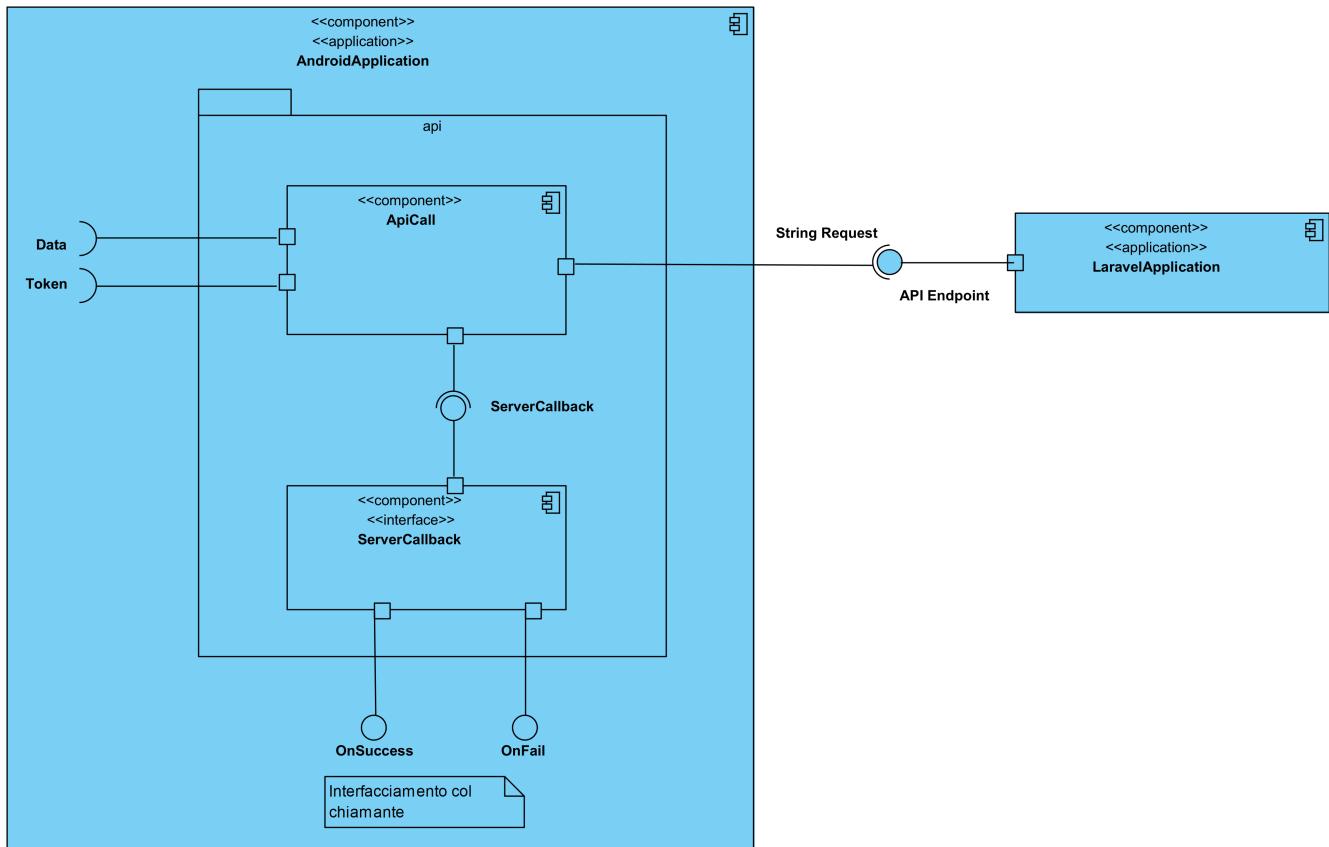


Figura 3.21: Diagramma dei componenti android API

Alla classe responsabile di effettuare la chiamata API viene passata una callback definita all'interno dell'Activity chiamante rispettando l'interfaccia di ServerCallback.

3.4 Database

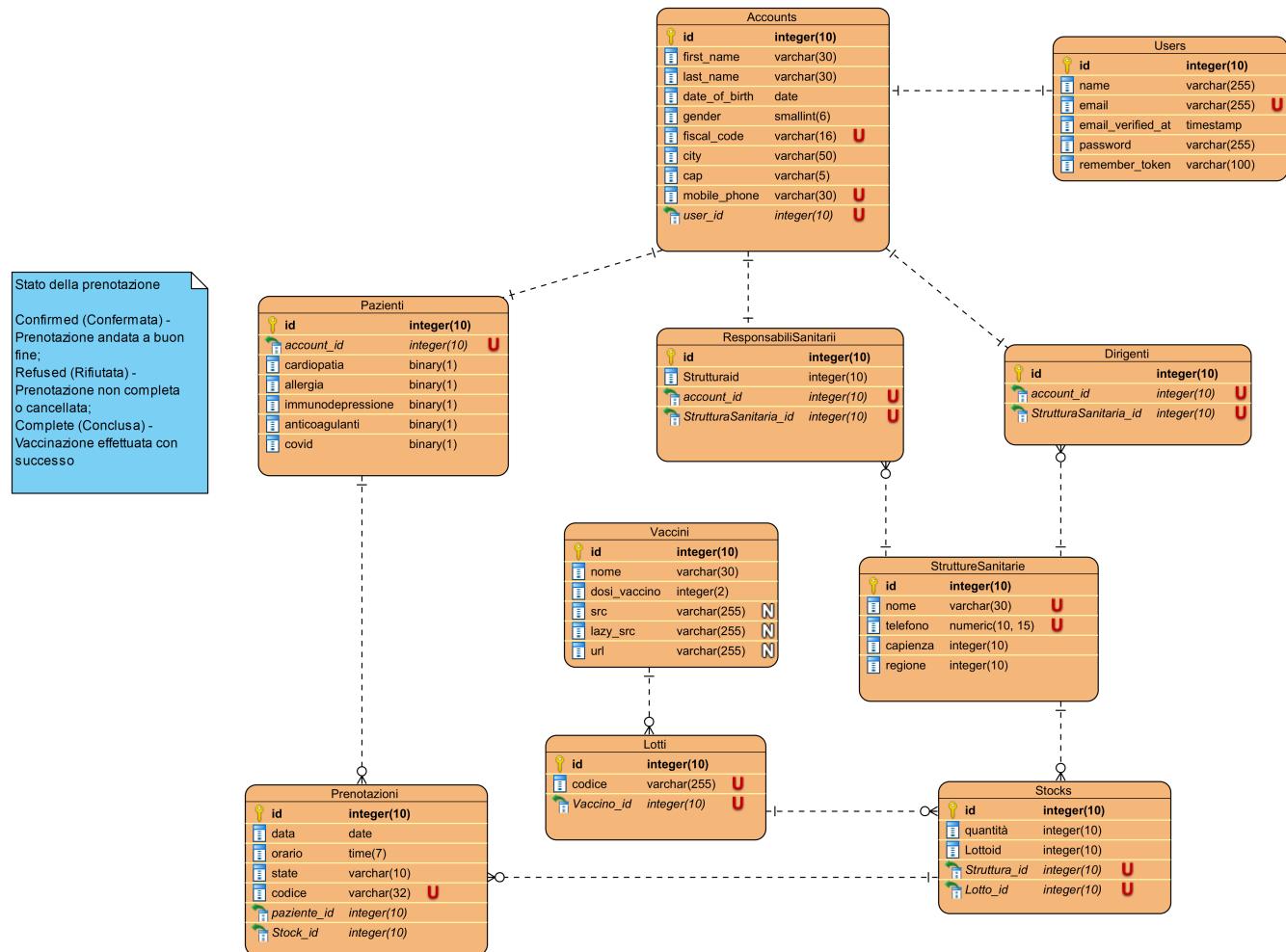


Figura 3.22: Diagramma ER database

3.5 Diagrammi Funzionalità Implementate

3.5.1 Gestione delle Eccezioni

La gestione delle eccezioni ha costituito un punto fondamentale nella realizzazione dell'applicazione per cercare di rendere il sistema più tollerante agli errori, **migliorando la reliability del software**. Non si deve mai verificare, infatti, il caso in cui un'eccezione:

- Generi comportamenti non previsti nell'applicazione
- Generi inconsistenze nei dati
- Faccia crashare l'applicazione
- Costituisca un punto di partenza per trovare eventuali exploit sulla sicurezza

3.5.1.1 Lato Server

Laravel possiede una classe chiamata Handler che intercetta le eccezioni che vengono sollevate e le gestisce secondo la logica definita dagli sviluppatori. Quella di base prevede che le eccezioni siano visualizzate sul frontend con tanto di stacktrace e dati, ciò chiaramente non può avvenire quando l'applicazione si trova in produzione. Il seguente sequence diagram mostra come sono state gestite:

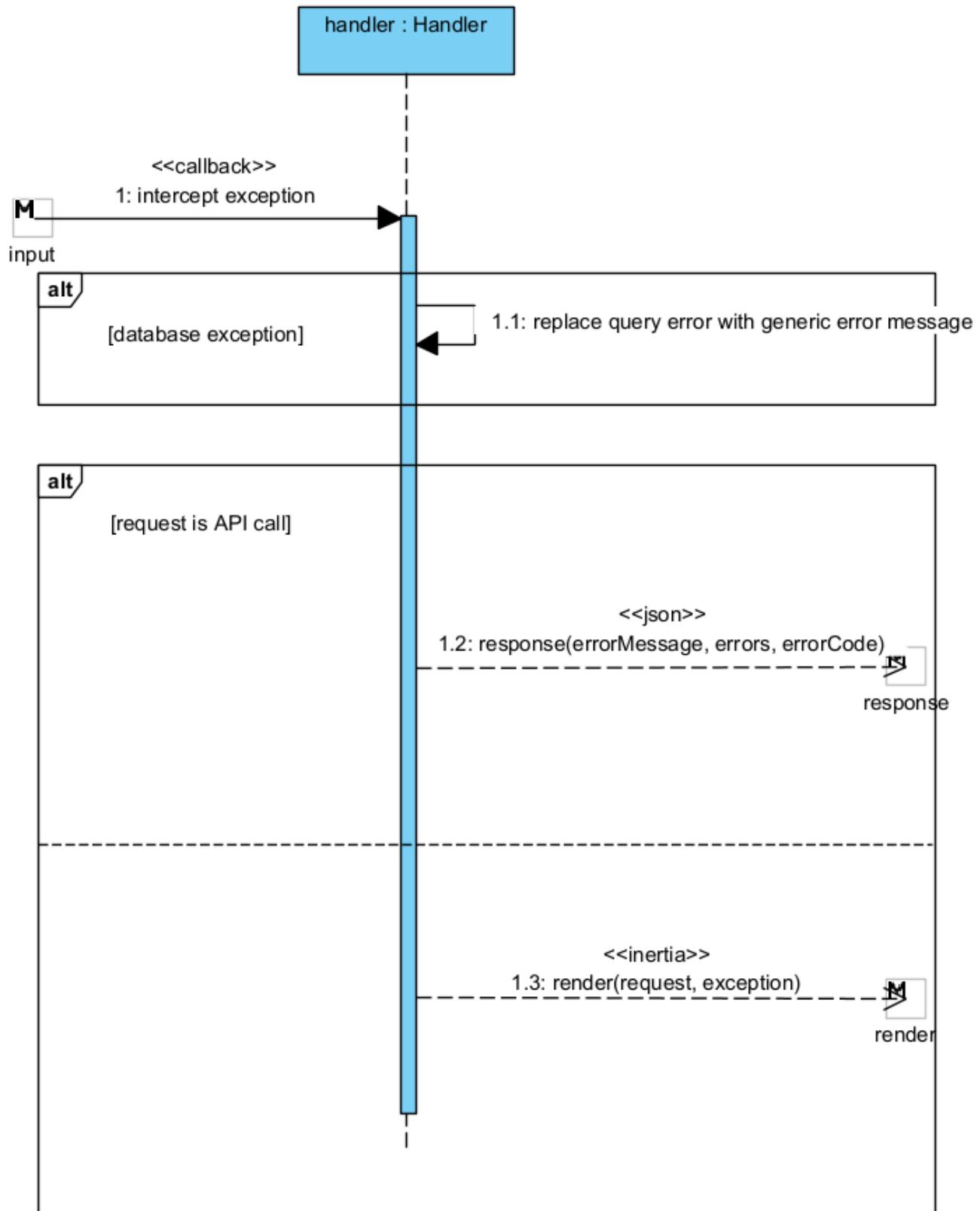


Figura 3.23: Sequenza gestione eccezioni lato server

3.5.1.2 Lato Frontend (Vue)

Il server, tramite InertiaJs, restituisce al frontend una «bag» contenente tutti gli errori che si sono verificati. Tale bag verrà inserita come elemento all'interno del campo «errors» contenuto nel form di Inertia nella pagina vue. Ad esempio, se abbiamo sbagliato a digitare l'email e gli errori sono

- Non è un'email
- E' troppo lunga

Il campo «errors» del form sarà così riempito:

```
"errors" => [
  "validation" => [
    "email" => [
      "not_email" => "Non è un'email",
      "length" => "Troppo lunga"
    ]
  ]
]
```

Figura 3.24: Errori form

Se vogliamo visualizzare sotto il campo di testo «email» i due messaggi d'errore, dobbiamo scrivere una funzione che estragga tale informazione dal form. A tal proposito si è scritta una funzione, inserita nel component principale di vue, che usa il pattern wildcard per estrarre gli errori. Quindi se vogliamo estrarre tutti i messaggi d'errore relativi all'email, basta passare come wildcard «email*», come mostrato nel seguente sequence:

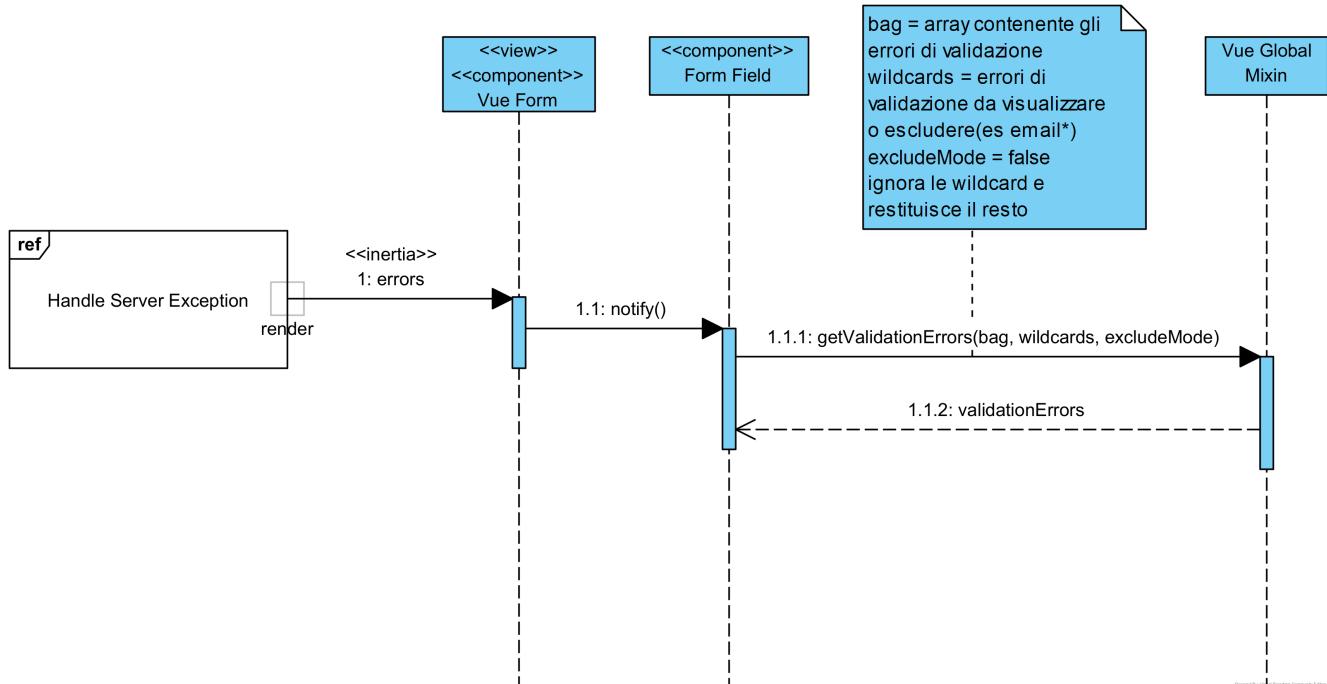


Figura 3.25: Sequence gestione eccezioni lato frontend

3.5.1.3 App Android

Per Android si è scritta una classe ad hoc che esamina il payload d'errore e trasforma il messaggio d'errore o i messaggi di validazione in una stringa visualizzabile tramite toast sul display.

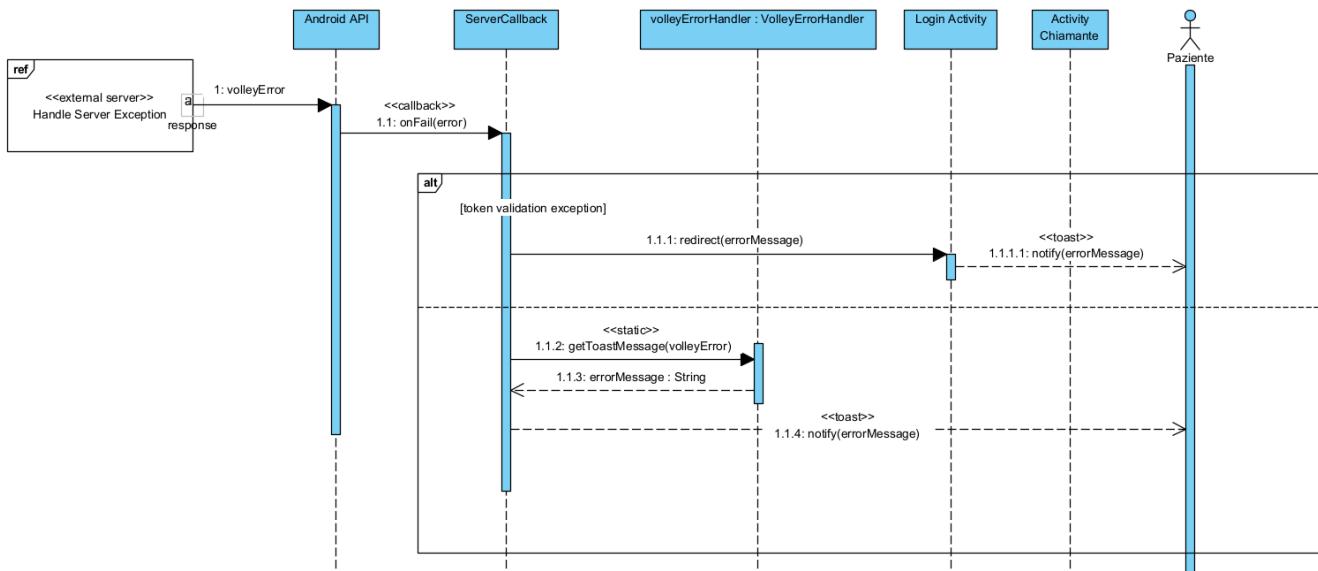


Figura 3.26: Sequence gestione eccezioni lato frontend

3.5.2 Visualizza Prenotazioni

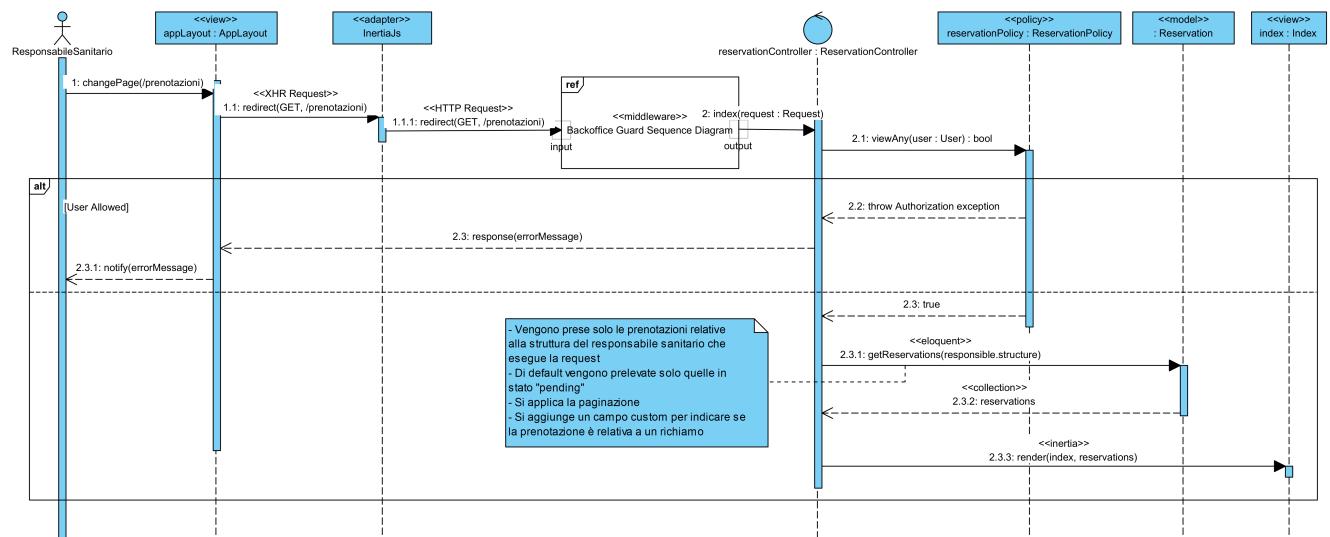


Figura 3.27: Sequence visualizza prenotazioni

C'è da considerare che il numero di pazienti, e di conseguenza il numero di prenotazioni, potrebbe essere particolarmente elevato, quindi sarebbe impensabile caricare nella griglia tutte le prenotazioni presenti nel database, ragion per cui verranno caricate in un determinato momento solo un tot di prenotazioni. Quando l'operatore cambierà pagina, verrà effettuata una chiamata asincrona al server per caricare le prenotazioni della pagina successiva, come nel seguente sequence diagram:

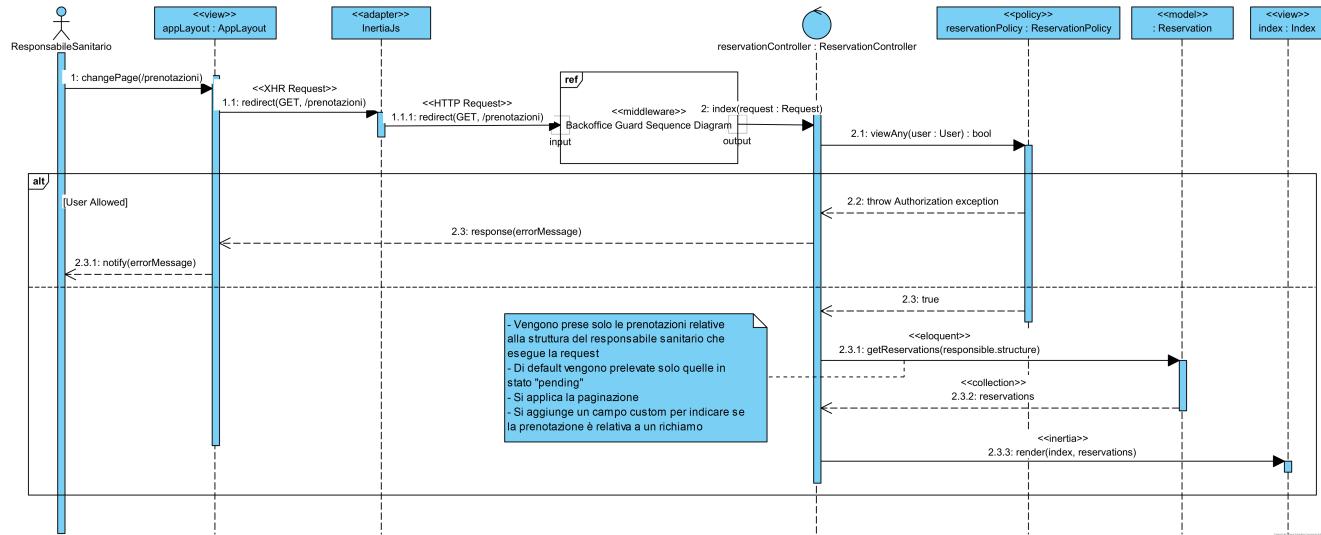


Figura 3.28: Sequence Filtri Prenotazioni

3.5.3 Gestisci Prenotazione

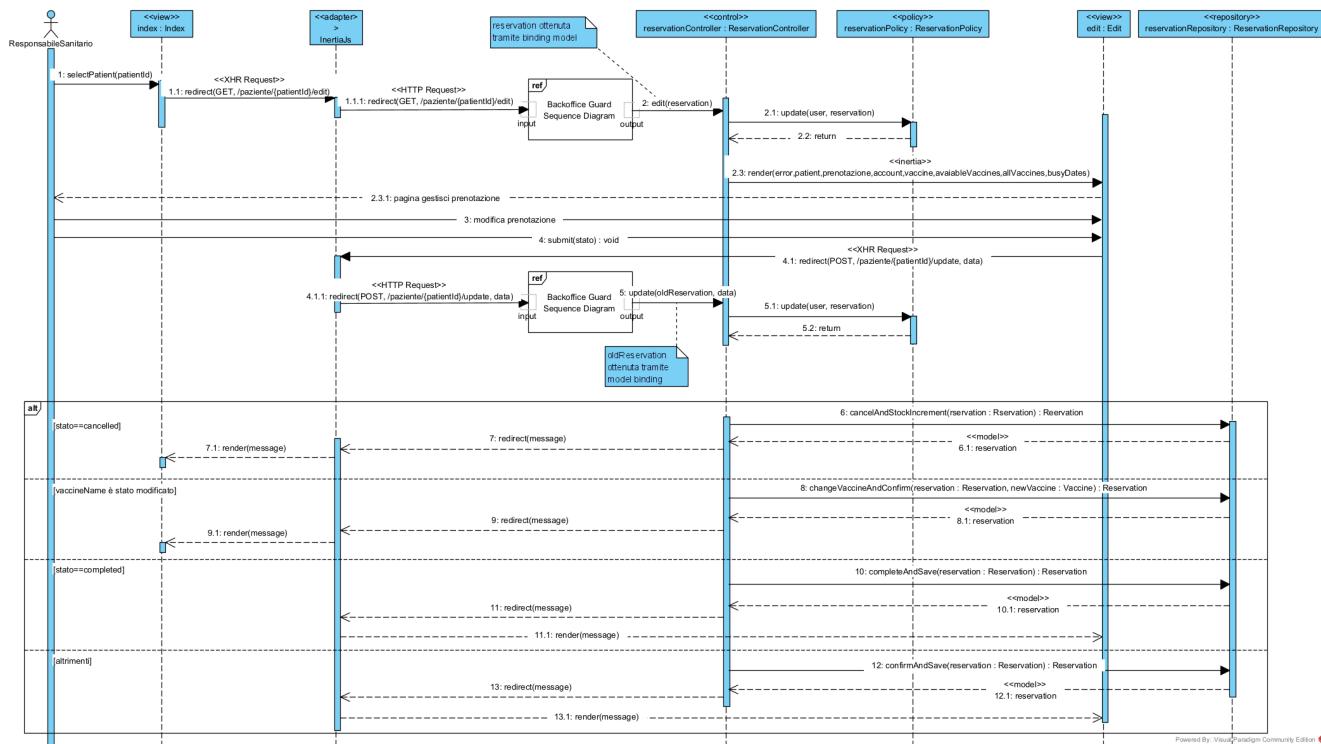


Figura 3.29: Sequence gestisci prenotazione

3.5.4 Prenota Richiamo

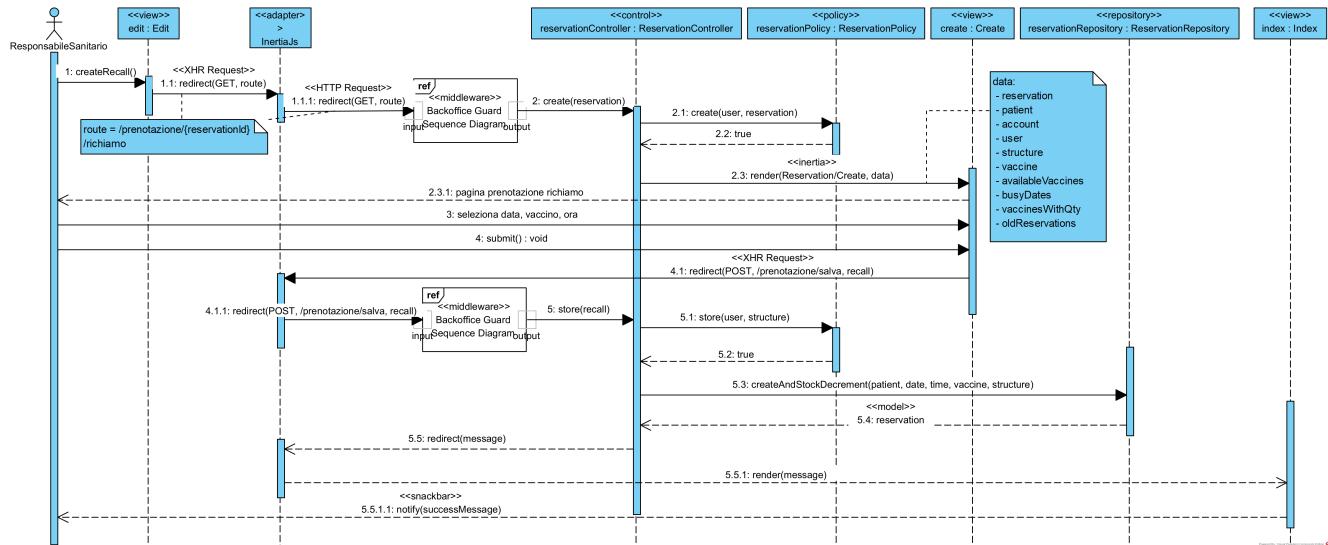


Figura 3.30: Sequence prenota richiamo

3.5.5 Effettua Prenotazione

La prenotazione può essere effettuata tramite applicazione Android che si connette al server.

3.5.5.1 Lato Android

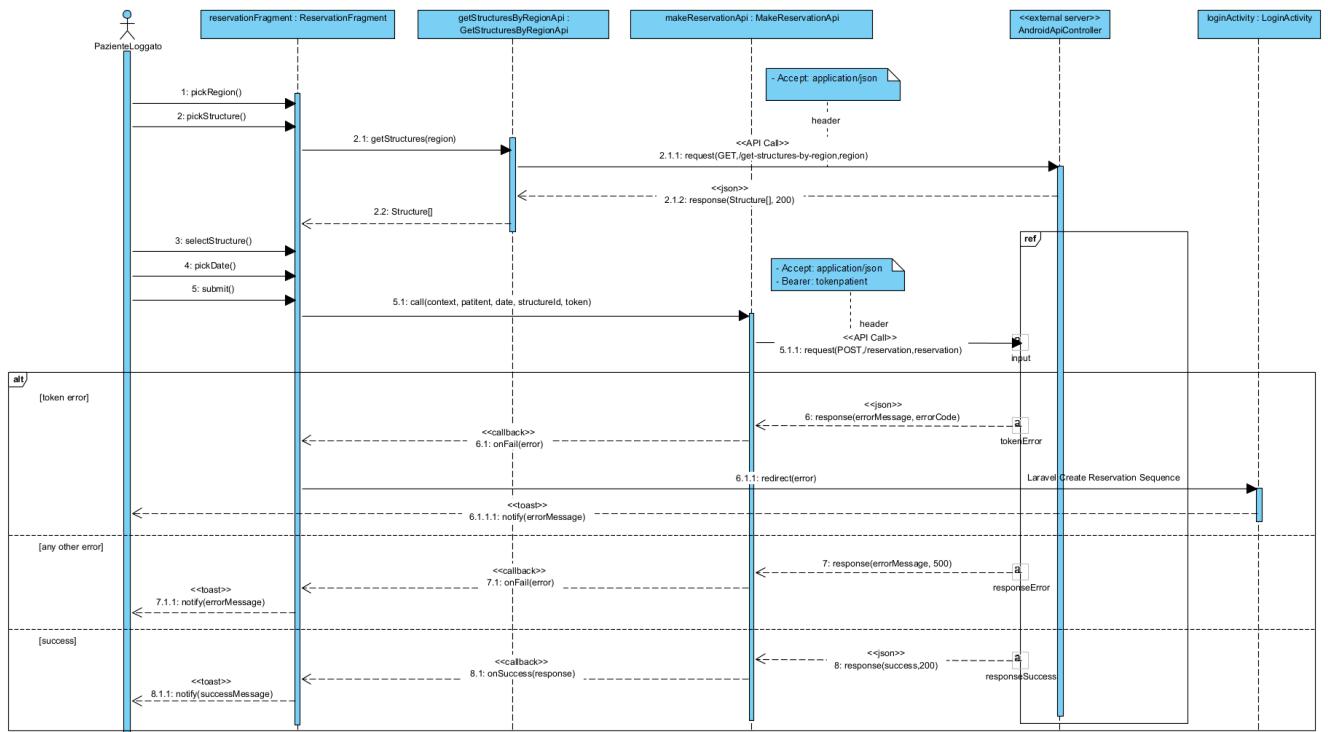


Figura 3.31: Sequence effettua prenotazione lato Android

3.5.5.2 Lato Server

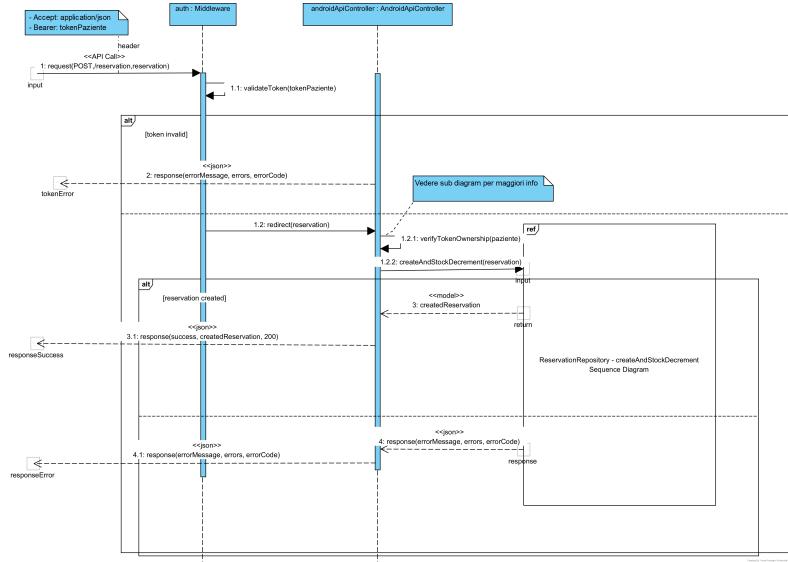


Figura 3.32: Sequenza effettua prenotazione lato Server

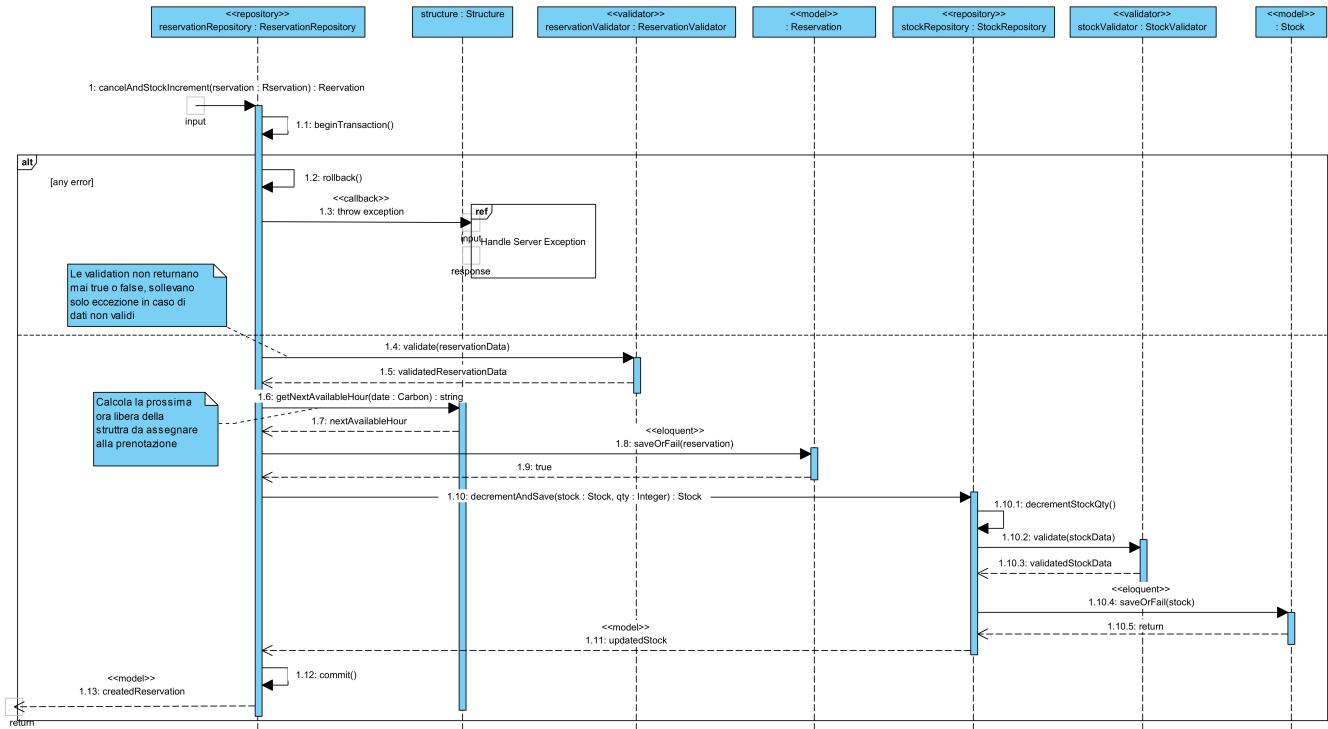


Figura 3.33: Sequenza effettua prenotazione repository

3.6 Diagrammi Globali

3.6.1 Web Application Component Diagram

Il componente centrale dell'architettura è il componente LaravelApplication, basato sul framework MVC Laravel, che comprende tutta la logica di business dell'intero sistema di gestione delle prenotazioni. Questo componente

si interfaccia ad un DBMS MySQL per consentire alle classi model di estrarre i dati dal database. La Laravel application fornisce inoltre un’interfaccia per permettere la connessione alla web application sia da browser web (per responsabili sanitari e pazienti) sia da un prototipo di App Android (solo per i pazienti). L’app Android si interfaccia con il controller dell’applicazione Laravel tramite metodi REST API.

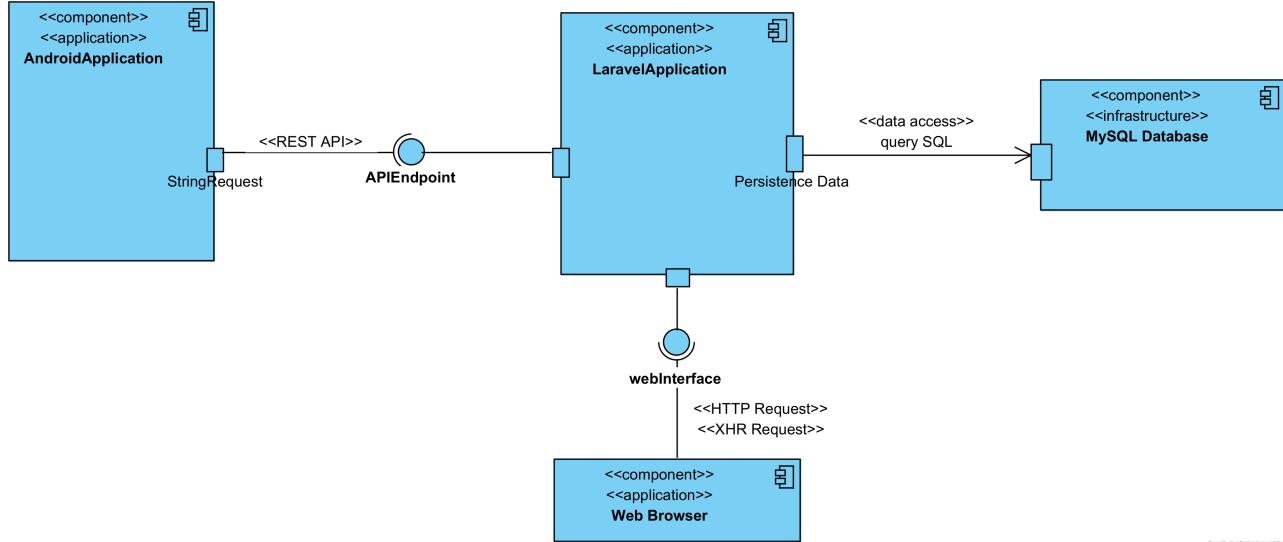


Figura 3.34: Diagramma dei componenti

3.6.2 Deployment diagram

Definisce come gli artefatti prodotti sono distribuiti sui nodi fisici dell'applicazione.

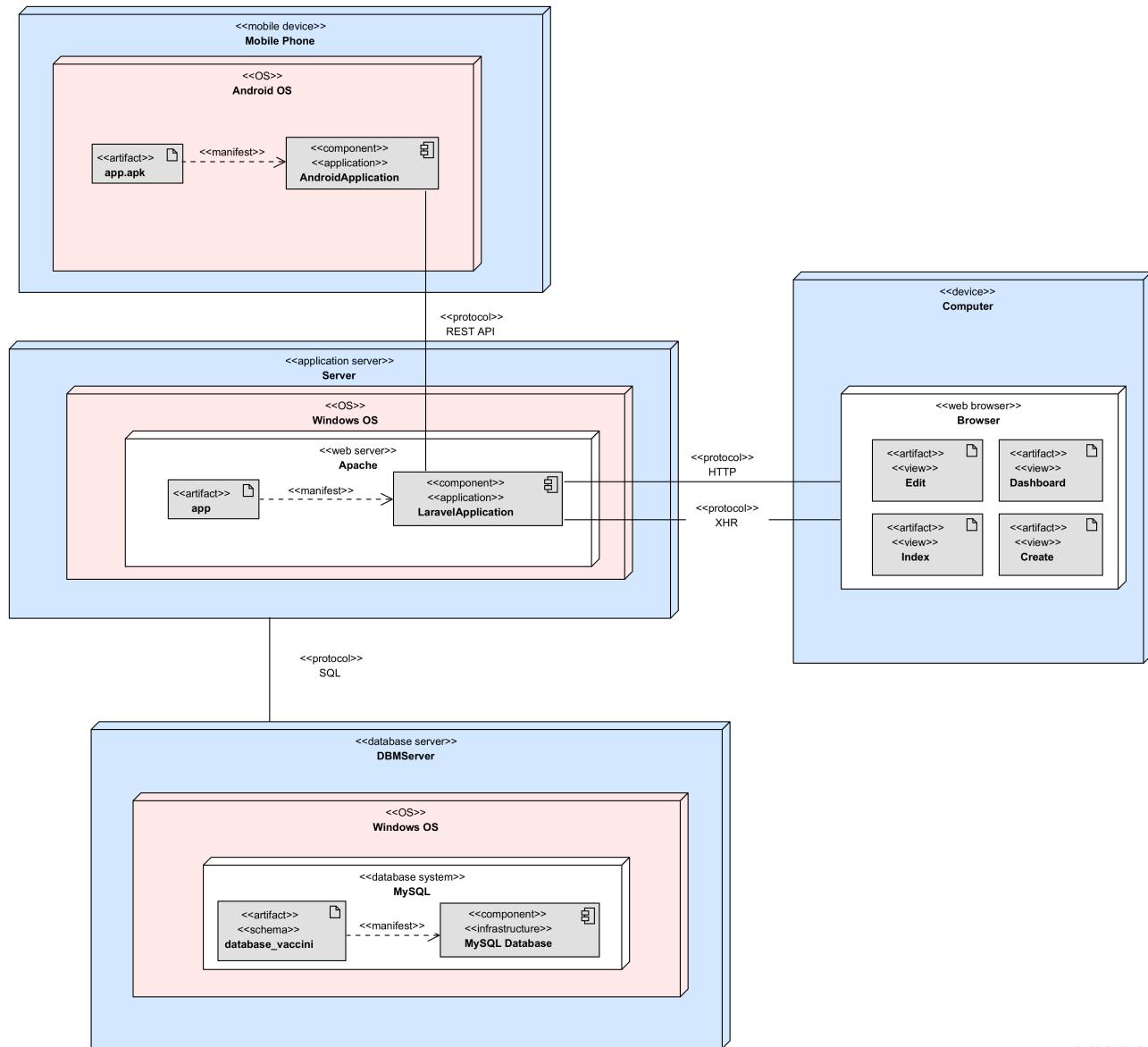


Figura 3.35: Diagramma di distribuzione

3.6.3 Rest API

3.6.3.1 Diagramma UML

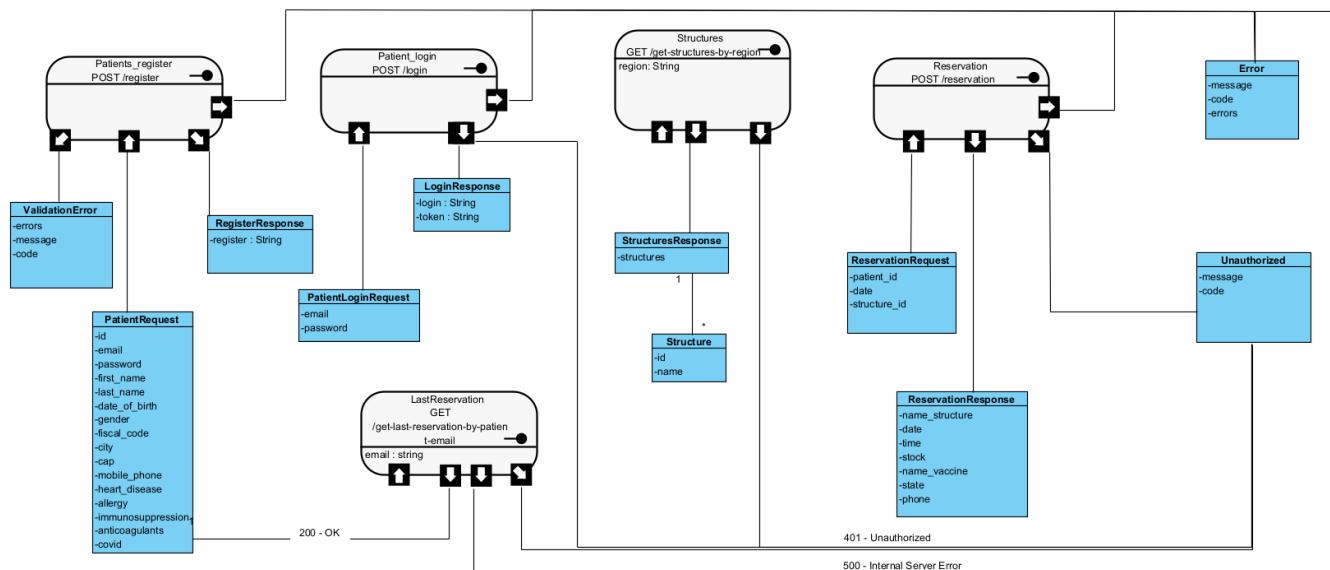


Figura 3.36: Diagramma Rest API

3.6.3.2 SWAGGER

Può essere consultato sull'applicazione all'uri /api/documentation

The SWAGGER interface displays the following API endpoints:

- API Endpoints (1.0.0 OAS3):**
 - `http://127.0.0.1:8000/docs/api-docs.json`
 - API per interfacciarsi con l'applicazione Android
- reservation:**
 - `GET /get-structures-by-region/{region}` Prendi Strutture
 - `GET /get-last-reservation-by-patient-email/{email}` Prende ultima prenotazione
 - `POST /reservation` Crea prenotazione
- auth:**
 - `POST /login` Effettua Login
 - `POST /registerPost` Effettua Registrazione

Figura 3.37: SWAGGER

Nel dettaglio sono state inserite le specifiche dettagliate delle singole API:

The screenshot shows the SWAGGER interface for the `/registerPost` endpoint. The top bar indicates a `POST` method and the endpoint `/registerPost`. The description is "Effettua la registrazione del paziente".

Parameters

No parameters

Request body required

application/json

Info richieste per la registrazione

[Example Value](#) | [Schema](#)

```
{
  "first_name": "marco",
  "last_name": "predoni",
  "email": "marco.predoni@email.it",
  "password": "secret1234",
  "date_of_birth": "2021-05-16",
  "fiscal_code": "MRCPRD95F839G",
  "mobile_phone": "3951302553",
  "city": "milano",
  "address": "via napoleone 57",
  "cap": "80931"
}
```

Responses

Code	Description	Links
200	Registrazione effettuata con successo	No links
500	Errore interno del server	No links

200 Response:

Media type: application/json

Controls Accept header.

[Example Value](#) | [Schema](#)

```
{
  "message": "success",
  "code": 200
}
```

500 Response:

Media type: application/json

[Example Value](#) | [Schema](#)

```
{
  "message": "validation",
  "errors": {"email": ["length': 'email troppo lunga']}",
  "code": 500
}
```

Figura 3.38: SWAGGER Details

Capitolo 4

DOCUMENTAZIONE DI IMPLEMENTAZIONE

4.1 FRAMEWORK E LIBRERIE

Per l'implementazione del progetto si è scelto di utilizzare dei framework per i seguenti motivi:

- Implementano tutta una serie di funzionalità di base, fornendo un ottimo punto di partenza per realizzare la propria applicazione
- Svincolano gli sviluppatori dal risolvere tutta una serie di problemi di basso livello, non costringendo a reinventare la ruota ogni volta
- Per realizzare applicazioni moderne che siano effettivamente utilizzabili nel mondo reale e che non rimangano puramente accademiche, la scelta di un framework è d'obbligo

Si è deciso, piuttosto che aggiungere features particolarmente complesse, di sfruttare le potenzialità dei framework utilizzati per rendere l'applicazione polished e farle rispettare i criteri necessari ad essere utilizzata in un contesto reale. Anche una feature semplice, in quest'ottica, assume una certa complessità.

4.1.0.1 Laravel 8



Figura 4.1: Logo Laravel

Laravel è un framework open source di tipo MVC scritto in PHP per lo sviluppo di applicazioni web creato come derivazione di Symfony.

Distribuito con licenza MIT, mantiene tutto il codice disponibile su GitHub e viene indicato, in base al punteggio GitHub e StackOverflow, come il framework PHP più popolare, seguito da Symfony, CodeIgniter e altri; ad agosto 2014 risulta essere il progetto PHP più seguito su GitHub.

Un framewok fornisce una struttura e un punto di partenza per creare la propria applicazione, permettendo agli sviluppatori di scrollarsi la responsabilità di gestire problematiche di basso livello e concentrandosi sulla logica applicativa.

Perché Laravel? Laravel è stato scelto in base alle seguenti motivazioni (oltre a quelle già esposte relativamente all'architettura):

1. Espressività: è un framework molto espressivo con una sintassi chiara ed elegante
2. Semplicità di utilizzo: la formula utilizzata è «easy to use, hard to master», permette di scrivere applicazioni di una certa complessità anche a chi ha poca esperienza
3. Documentazione ufficiale: è chiara, ricca di esempi e spiegazioni riguardo ogni singolo aspetto del framework
4. Documentazione non ufficiale: essendo uno dei framework più utilizzati al mondo, vanta di decine di migliaia di issues risolte su forum non ufficiali come Stack Overflow
5. Librerie Open Source: sempre grazie alla sua diffusione, esistono migliaia di librerie pubblicate e gratuitamente scaricabili da github per implementare le funzionalità più disparate

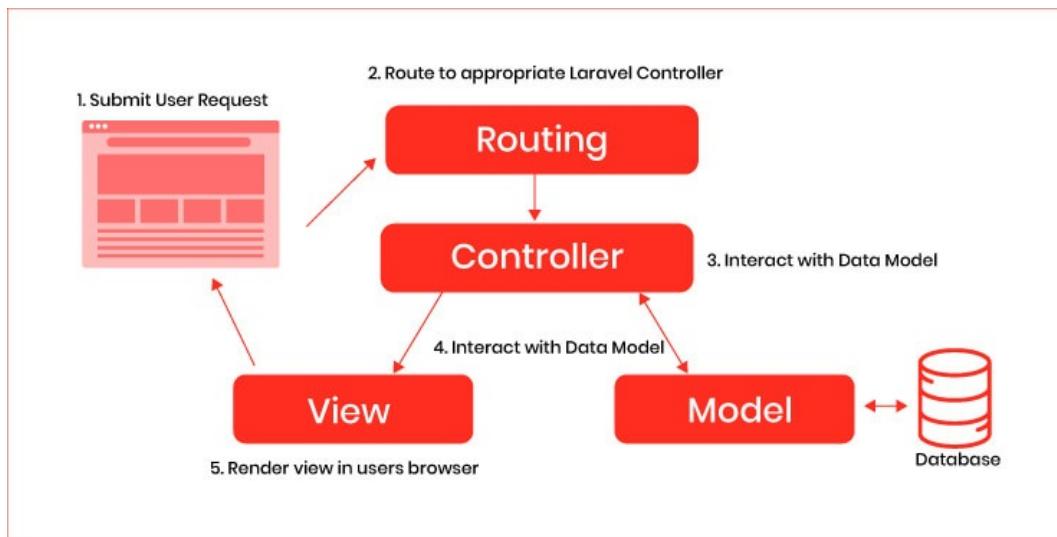


Figura 4.2: Architettura MVC di Laravel

Architettura del Framework Laravel presenta un'architettura di tipo MVC, dove però la model non esegue mai l'update della view ma tale compito è sempre delegato ai controller sotto esplicita richiesta. Si compone dei seguenti componenti:

- Routing: è un registro (un semplice file php) all'interno del quale vengono registrati i binding tra i metodi dei controller e le uri del sito
- Controller: espone i metodi responsabili di renderizzare le pagine del frontend o di eseguire operazioni come il salvataggio dei dati tramite model. In particolare si dovrebbe seguire il seguente [standard](#)

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Figura 4.3: Standard per i Controller

Alcune operazioni, come richieste di tipo Ajax, vanno inserite in dei metodi ad hoc che non rientrano in queste categorie.

- **Model:** laravel usa un ORM (Object Relational Mapper) per l'interazione con il database. A ciascuna tabella del DB corrisponde una model che viene usata per interagire con tale tabella, permettendo di inserire, estrarre, modificare e cancellare i dati in essa contenuti. La model non supporta un binding diretto con la view, tuttavia esiste la possibilità di implementare un binding statico che permette al controller di caricare direttamente la model scelta in base all'id specificato nell'url.
In questa maniera, ad esempio, se vado su /prenotazioni/5/edit, il controller già sa che deve caricare la model di tipo Reservation con il record nel DB di ID 5.
- **View:** viste in html/phtml renderizzate sul frontend. In questo progetto, tuttavia, è stato utilizzato un altro framework per il frontend (Vue), del quale si parlerà più avanti

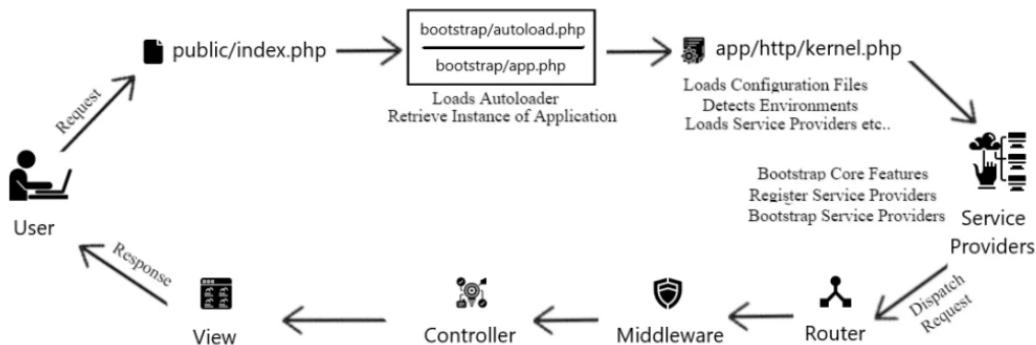


Figura 4.4: Ciclo di Vita di una Richiesta

Ciclo di Vita di una Richiesta L'utente effettua la richiesta (ad esempio fa il submit di un form sul frontend), la richiesta viene dispatchata al router dal Service Provider, individuando così il metodo del controller da richiamare,

prima che questa raggiunga il controller deve passare per dei [Middleware](#) assegnati (sempre all'interno del routing) che possono avere diversi compiti, come quello di verificare che l'utente sia autenticato, o che abbia particolari permessi.

Se la richiesta passa il controllo del middleware arriva al controller, il quale la elabora (nel caso del submit di un form, ad esempio, potrebbe eseguire un salvataggio) ed invia la risposta al client, che potrebbe essere un redirect o un payload contenente dati.

Dependency Injection Grazie ad un tool per gestire le classi chiamato [“Service Container”](#) è possibile effettuare la [Dependency Injection](#) delle dipendenze delle classi, “iniettandole” tramite il costruttore.

Database Per l'inizializzazione del database, laravel fornisce le seguenti classi:

- [Migration](#): permettono di definire uno schema per le tabelle del database, in maniera tale da definire in un solo package quale sarà la struttura del DB senza necessità di modificarla manualmente. Anche in fase di produzione, quando non è possibile ricreare il DB da 0, è possibile definire delle migration che invece di creare tabelle, aggiornano quelle già esistenti
- [Seeders](#): all'interno del database seeder è possibile gestire una logica di riempimento iniziale del DB, ad esempio se ci sono dati immutabili che devono essere caricati, oppure se si vuole testare l'applicazione con dati fittizzi
- [Factories](#): permettono di definire una logica per riempire con dati casuali le model

Testing In laravel è possibile scrivere Unit Test e Feature Test per testare praticamente qualsiasi funzionalità implementata sia sul backend che sul frontend, o sviluppare by testing.

Librerie Utilizzate



Figura 4.5: L5 Swagger

[5.1.7.1 L5-Swagger](#) Laravel non possiede uno swagger built-in, quindi si è utilizzata la libreria L5-Swagger per generare lo swagger

4.1.0.2 [Vue 2](#)



Figura 4.6: Vue

Vue.js è un framework JavaScript open-source in configurazione [Model-view-viewmodel](#) per la creazione di interfacce utente e single-page applications. Vue.js presenta un'architettura adattabile in modo incrementale che si concentra sul rendering dichiarativo e sulla composizione dei componenti. Le funzionalità avanzate richieste per applicazioni complesse come routing, state management e strumenti di compilazione sono offerte tramite librerie e pacchetti di supporto ufficialmente mantenuti, che comprendono Nuxt.js tra le soluzioni più popolari.

Perché Vue? Le view statiche di Laravel sono state scartate a priori, in quanto non ci consentirebbero di realizzare un'applicazione moderna, reactive e single page (o per lo meno non in maniera semplice). Assodato ciò, la scelta è ricaduta tra 3 principali competitors:

- Angular: sviluppato da Google, basato su TypeScript (un framework JavaScript), è il più difficile da apprendere
- React: sviluppato da Facebook, una volta compresi i concetti di base la curva di apprendimento rimane stabile
- Vue: sviluppato da un ex dipendente google e basato su TypeScript, non ha il supporto di grandi compagnie. L'espressività è simile a quella di react, ma permette una maggiore customizzazione dei componenti

La scelta è ricaduta su Vue per i seguenti motivi:

1. La sua elevata personalizzazione facilita la curva di apprendimento
2. È il più flessibile tra i tre
3. È molto supportato dalla community in termini di soluzioni e librerie

Perchè la Versione 2? La 3 è ancora acerba e meno supportata dalla community.

Architettura Vue usa il pattern architettonale View-Model-ViewModel

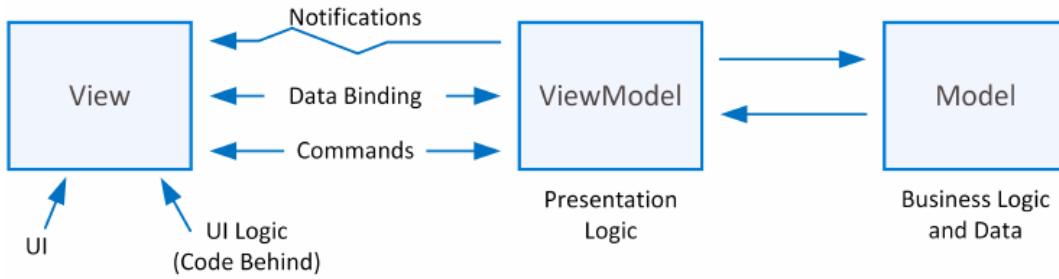
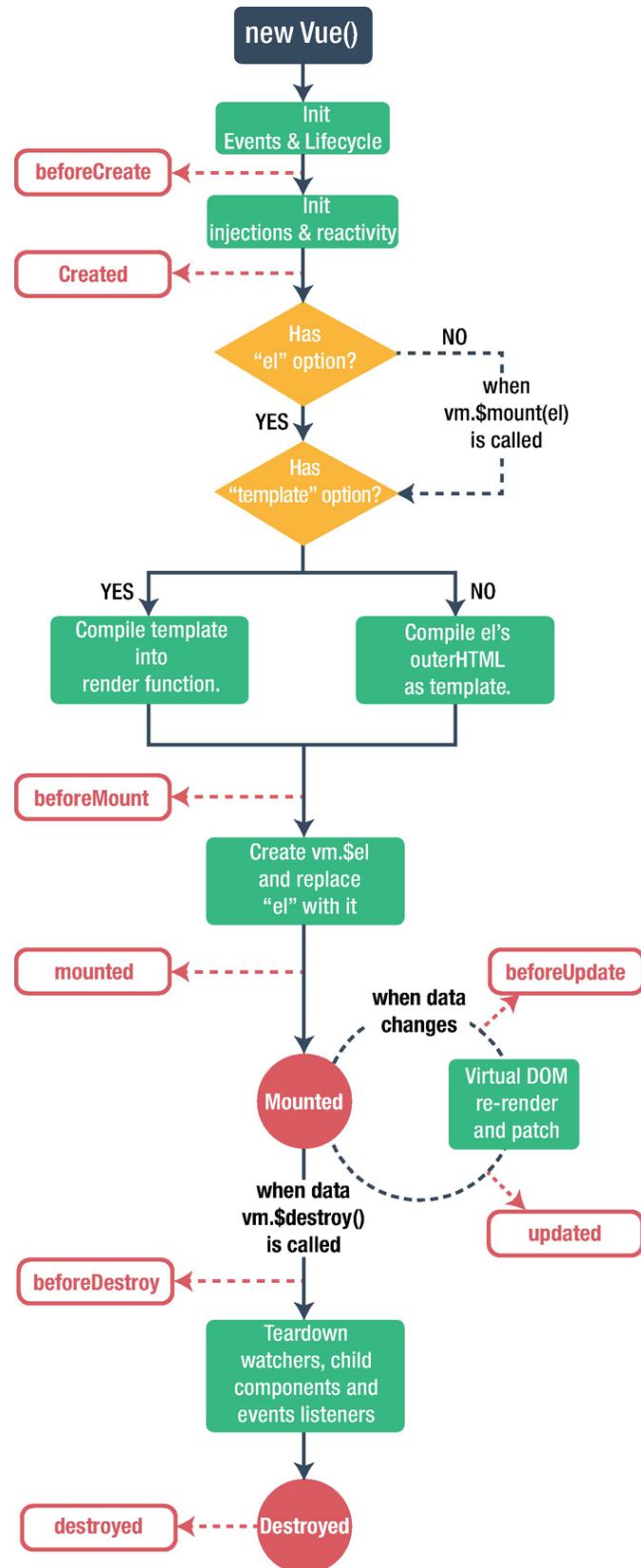


Figura 4.7: Architettura Vue

- **Model Layer**: contiene i dati e la business logic, ad esempio se carichiamo le prenotazioni, queste saranno contenute in un array o in un object definito qui
- **View Layer**: qui è definito il layout della pagina
- **ViewModel Layer**: fa da intermediario agli altri 2 layer definendo come i dati contenuti nelle model devono essere visualizzati sulle view, ad esempio nel caso delle prenotazioni lo stato che viene passato dal server è in inglese, nel layer intermedio possiamo tradurlo in italiano e mandare la stringa tradotta alla view

Componenti L'elemento principale del framework è costituito dai componenti, che rappresentano gli oggetti visualizzati sul frontend. I componenti possono essere innestati e possono comunicare tra loro tramite binding diretto, tanto che le pagine stesse sono dei componenti.



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

Figura 4.8: Ciclo di Vita di un Componente Vue

Ciclo di Vita di un Componente Vue Quando un componente viene renderizzato passa per diversi stati, come mostrato in figura, ed è possibile intervenire prima o dopo una qualsiasi di queste fasi. La renderizzazione del componente è completa quando arriva in “mounted”, mentre si trova in questa fase alle variabili dichiarate è associato un observer, in maniera tale che se il loro valore cambia, vengono aggiornati i componenti che le utilizzano. Infine quando il componente viene eliminato (ad esempio quando viene renderizzata un’altra pagina), vengono deallocated tutti gli elementi associati.

Comunicazione View-Server Vue è compatibile con qualsiasi framework di backend, in quanto i dati vengono prelevati tramite chiamate col protocollo XHR. Normalmente la comunicazione tra View di Vue e Controller di Laravel avverrebbe tramite chiamate API, avendo lo svantaggio di non poter utilizzare tutte le funzionalità built-in di laravel riguardo la gestione delle richieste HTTP.

Per ovviare a questo problema, come verrà descritto nella prossima sezione, si è utilizzato un Adapter che funge da intermediario tra Laravel e le view di Vue, consentendoci di trattarle come se fossero pagine statiche di Laravel in maniera trasparente.



Figura 4.9: Logo VuetifyJs

Libreria [VuetifyJs](#) Vuetify è una libreria open source per Vue che mette a disposizione una vasta gamma di componenti personalizzabili. Grazie ad essa è stato ridotto al minimo indispensabile il lavoro richiesto per graficare i componenti dell’applicazione.

4.1.0.3 [Inertia.Js](#)

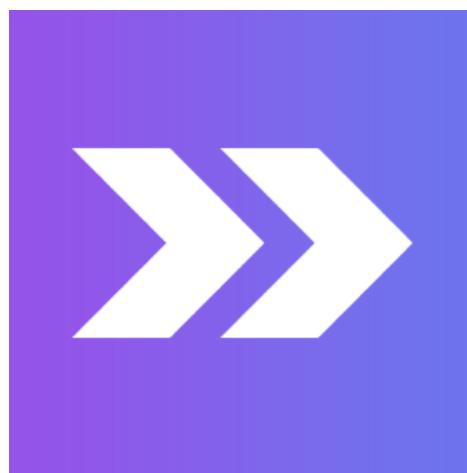


Figura 4.10: Vue

InertiaJs non è un vero e proprio framework, piuttosto può essere definito come una serie di adapter frontend e backend. Costituisce un nuovo approccio rispetto allo sviluppo delle classiche server-driven web app permettendo di creare applicazioni single page completamente renderizzabili dal client senza la complessità delle moderne SPA facendo leva sul framework lato server.

Senza Vue, la navigazione classica consiste in:

1. Client visita il sito tramite browser con richiesta HTTP
2. Il server renderizza la view richiesta mandando al client l'intera pagina e il browser la visualizza
3. Client cambia pagina o effettua operazione (ad esempio submit di un form)
4. Il server ripete il passo 2 anche se bisogna ricaricare la pagina attuale

Nel caso di Vue, le pagine sono renderizzate dal browser, quindi il server viene svincolato dalla renderizzazione dinamica delle pagine:

1. Client visita il sito tramite browser con richiesta HTTP
2. Il server manda codice precompilato html/javascript della prima pagina visitata e di tutte le pagine .vue collegate a questa tramite vue router (a meno che non venga esplicitamente specificato che il codice debba essere mandato man mano che si scorrono le route), il browser costruirà dinamicamente la pagina interpretando tale codice
3. Client cambia pagina
4. Se il codice della pagina target è già presente, la nuova pagina viene immediatamente renderizzata andando a modificare il DOM di quella esistente e se devono essere prelevati dati dal server, vengono effettuate delle chiamate tramite protocollo XHR sugli endpoint designati

Con Inertia è sufficiente effettuare le chiamate con una sintassi ad hoc. In particolare quando viene effettuata una richiesta (ad esempio tramite inertia-link), viene effettuata una chiamata XHR particolare al server, il quale grazie ad Inertia riconosce che si tratta di una chiamata speciale e la tratta come una classica chiamata HTTP con la differenza che invece di restituire una nuova pagina pre-renderizzata, restituisce dati JSON contenenti i dati richiesti dal client.

Ricapitolando, il control flow è il seguente:

1. Client visita il sito tramite browser con richiesta HTTP
2. Il server manda codice precompilato html/javascript solo della pagina visitata, comprendendo i primi dati richiesti dalla stessa
3. Client cambia pagina
4. Inertia effettua (in maniera trasparente) una chiamata speciale XHR inserendo un header contenente un flag che indica che si tratta di una chiamata di tipo inertia, in questa maniera il server riconosce la chiamata e la tratta come una normale richiesta HTTP facendola gestire dai service providers e dai middleware
5. Il controller, anche usando la normale sintassi di Laravel, restituisce invece che una pagina HTML renderizzata al momento, solamente i dati in formato JSON delle nuove informazioni richieste e del codice precompilato dell'eventuale pagina target

4.2 SOFTWARE UTILIZZATI

4.2.1 PhpStorm

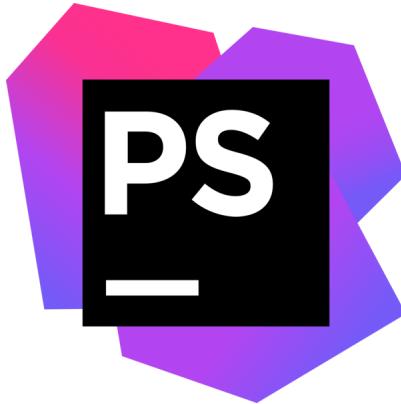


Figura 4.11: Logo PhpStorm

PhpStorm è un ambiente di sviluppo integrato (IDE) commerciale multi-piattaforma pensato per PHP, sviluppato dalla società ceca JetBrains. PhpStorm fornisce un editor per PHP, HTML e JavaScript con analisi del codice on-the-fly, prevenzione degli errori e refactoring automatici per codice PHP e JavaScript. L'autocompletamento del codice in PhpStorm supporta PHP dalla versione 5.3 in poi (progetti moderni e legacy), inclusi generatori, coroutines, la parola chiave `finally`, elenchi in `foreach`, `namespace`, `chiusure`, `traits` e array a sintassi breve. Include un editor SQL completo coi risultati di query modificabili.

PhpStorm è scritto in Java. Gli utenti possono estendere l'IDE installando plugin creati per PhpStorm o scrivendone di propri. Il software comunica anche con origini esterne come XDebug.

Tutte le funzionalità disponibili in WebStorm sono a loro volta incluse in PhpStorm, in aggiunta al supporto a PHP e database. WebStorm viene fornito con i plug-in JavaScript preinstallati (come per Node.js).

Si è dunque usato questo IDE per lo sviluppo del codice relativo al backend e al frontend dell'applicazione.

4.2.2 Android Studio



Figura 4.12: Logo Android Studio

Android Studio è un ambiente di sviluppo integrato (IDE) per lo sviluppo per la piattaforma Android. È stato annunciato il 16 maggio 2013 in occasione della conferenza Google I/O tenuta dal Product Manager Google, Katherine Chou. Android Studio è disponibile gratuitamente sotto licenza Apache 2.0.

Basato sul software di JetBrains IntelliJ IDEA, Android Studio è stato progettato specificamente per lo sviluppo di applicazioni Android. È disponibile il download su Windows, Mac OS X e Linux, e sostituisce gli Android

Development Tools (ADT) di Eclipse, diventando l'IDE primario di Google per lo sviluppo nativo di applicazioni Android.

4.2.3 Composer



Figura 4.13: Logo Composer

Composer è un gestore di pacchetti a livello applicativo per il linguaggio di programmazione PHP, che fornisce un formato standard per la gestione delle dipendenze dei progetti PHP e delle librerie richieste. Composer viene eseguito da riga di comando e si occupa di installare le dipendenze (ad esempio le librerie) dell'applicazione. Consente inoltre agli utenti di installare applicazioni PHP disponibili su "Packagist", il suo repository principale di pacchetti. Fornisce inoltre funzionalità di autoload per le librerie che ne prevedono il supporto, per facilitare l'utilizzo di codice di terze parti.

4.2.4 XAMPP



Figura 4.14: Logo XAMPP

XAMPP è una piattaforma software multipiattaforma e libera costituita da Apache HTTP Server, il database MariaDB e tutti gli strumenti necessari per utilizzare i linguaggi di programmazione PHP e Perl. Il nome è un acronimo dei software sopra citati (la X sta per x-platform, l'abbreviazione di cross-platform in lingua inglese ovvero multipiattaforma).

Nel nostro caso XAMPP è stato utilizzato per configurare e avviare il database MySQL sul quale si appoggia l'applicazione.

4.2.5 NodeJs



Figura 4.15: Logo NodeJs

Node.js è un runtime system open source multipiattaforma orientato agli eventi per l'esecuzione di codice JavaScript.

In origine JavaScript veniva utilizzato principalmente lato client. In questo scenario gli script JavaScript, generalmente incorporati all'interno dell'HTML di una pagina web, vengono interpretati da un motore di esecuzione incorporato direttamente all'interno di un Browser. Node.js consente invece di utilizzare JavaScript anche per scrivere codice da eseguire lato server, ad esempio per la produzione del contenuto delle pagine web dinamiche prima che la pagina venga inviata al Browser dell'utente. Node.js in questo modo permette di implementare il cosiddetto paradigma "JavaScript everywhere" (JavaScript ovunque), unificando lo sviluppo di applicazioni Web intorno ad un unico linguaggio di programmazione (JavaScript).

Node.js ha un'architettura orientata agli eventi che rende possibile l'I/O asincrono. Questo design punta ad ottimizzare il Throughput e la scalabilità nelle applicazioni web con molte operazioni di input/output, è inoltre ottimo per applicazioni web Sistema real-time (ad esempio programmi di comunicazione in tempo reale o browser game).

4.2.6 Vue Devtool



Figura 4.16: Logo Vue Devtool

Vue Devtool è un plugin per Chrome che consente di ispezionare le pagine create con Vue visualizzando i singoli componenti e le variabili contenute al loro interno. Chiaramente tale ispezione funziona solo quando il progetto è compilato in fase di developement.

4.2.7 Visual Paradigm



Figura 4.17: Logo Visual Paradigm

Visual Paradigm è un tool volto alla modellazione UML. E' stato utilizzato per produrre la documentazione relativa al progetto.

4.3 SETUP APPLICAZIONE

Si riportano gli step seguiti per configurare l'applicazione. Dettagli di alcuni sotto-step verranno omessi in quanto si darà per scontato che il lettore possegga competenze tecniche di base (o quanto meno i procedimenti sono ampiamente documentati sul web).

4.3.1 Setup Server

Il cuore dell'applicazione è costituito dal server. Dal momento che non esiste un unico eseguibile, la configurazione comprenderà i seguenti step.

4.3.1.1 Php e Database SQL

Installare Php e il database SQL. A questo proposito basta scaricare e installare XAMPP che comprende entrambe le cose. L'installazione non richiede particolari configurazioni, si riporta al link del download (scaricare la versione 7.4.2):

[XAMPP 7.4.2](#)

Successivamente va aggiunto il path di php all'interno della variabile di sistema Path.

Note In realtà il DBMS può essere installato su una macchina separata rispetto a quella dove si trova il server, nel qual caso va installato XAMPP, o quanto meno PHP, sia dove risiede il server che dove risiede il DBMS.

4.3.1.2 Composer

Bisogna installare [composer](#) per l'installazione delle librerie incluse nel progetto.

4.3.1.3 Download e Installazione Package

L'applicazione risiede su un [repository privato](#) di github, sarà necessario scaricarla e inserirla all'interno di una cartella. In tale cartella bisognerà aprire un terminale ed eseguire in sequenza i seguenti comandi:

1. composer install -> installa le dipendenze del progetto relative a php (composer.json)
2. npm install -> inizializza NodeJs ed installa le dipendenze relative alle librerie javascript (package.json)
3. npm run prod (o "npm run dev" in fase di developement) -> compila i file del frontend scritti in Vue in javascript
4. php artisan storage:link -> linka la cartella public in storage così da renderla accessibile

4.3.1.4 Configurazione File d'Ambiente

Nella root è presente un file chiamato .env.example, rinominarlo in .env e configurarlo in base alla configurazione scelta. Di particolare importanza è la configurazione del database:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=vaccini
DB_USERNAME=root
DB_PASSWORD=
```

4.3.1.5 Configurazione Database

Avviare il database SQL installato tramite XAMPP (o un altro metodo scelto), accedere al pannello di configurazione tramite browser (127.0.0.1:porta_di_phpmyadmin) e creare un database col nome corrispondente a quello impostato nel .env

Tornare nel terminale ed eseguire i seguenti comandi:

1. php artisan migrate -> crea le tabelle settate nelle migration

2. php artisan db:seed -> riempie le tabelle del database con i dati essenziali (ai fini della presentazione con dati fintizi relativi ai pazienti e alle prenotazioni)
3. php artisan serve --host=0.0.0.0 --port=27017 -> hosta il server all'indirizzo della macchina locale alla porta specificata, la quale deve essere aperta sul router e inserita tra le eccezioni nel firewall

4.3.1.6 Generazione Swagger

Eseguire in successione i seguenti comandi:

1. php artisan vendor:publish --provider "L5Swagger\L5SwaggerServiceProvider" -> genera il file di configurazione dello swagger, si rimanda alla [documentazione ufficiale della libreria](#) per la configurazione
2. php artisan l5-swagger:generate

4.3.2 Setup app Android

Prima di scaricare ed installare manualmente l'apk, è necessario seguire una brevissima procedura. Dalle impostazioni del dispositivo bisogna attivare l'installazione da sorgenti sconosciute. Una volta attivata questa impostazione è possibile installare manualmente il file .apk. Per installarlo si deve per prima cosa individuare il file sul dispositivo. Una volta trovato bisogna selezionare l'APK da installare. Fatto ciò l'app sarà installata e pronta all'uso.