

UNIVERSITÀ FEDERICO II DI NAPOLI

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

CORSO DI SOFTWARE ARCHITECTURE DESIGN LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

ANNO ACCADEMICO 2021-2022



DOCUMENTAZIONE SOFTWARE ARCHITECTURE DESIGN

SelfTestCOVID19

WEB APPLICATION ORIENTATA ALLA PREVENZIONE DEL SARS-CoV2

Prof.ssa

Ing. Anna Rita Fasolino

Autori:

Antonio Romano - M63001315

Giuseppe Riccio - M63001314

Indice

1 Introduzione	1
1.1 Contesto generale dell'applicazione	1
1.1.1 Che cos'è il COVID-19	1
1.1.2 Necessità dei tamponi	2
1.2 Reengineering del sistema di predizione: Probability Infection Checker dell'autore	2
1.2.1 Il dataset	2
1.2.2 Le fasi di Machine Learning sviluppate	2
1.2.3 Package Diagram del progetto Probability Infection Checker SARS-CoV2 COVID19	4
1.2.4 UI del progetto Probability Infection Checker SARS-CoV2 COVID19	5
2 Avvio della progettazione	6
2.1 Attori e obiettivi generali	6
2.2 Funzionalità generali del sistema software	7
2.3 Assunzioni e dipendenze	8
2.4 Storie utente (Requisiti funzionali)	8
2.5 Requisiti di interfaccia esterna	8
2.5.1 Interfaccia Utente	8
2.5.2 Interfaccia Hardware	14
2.5.3 Interfaccia Software	14
2.5.4 Interfaccia di Comunicazione	14
2.6 Requisiti di Consegna	14
2.7 Requisiti di Riservatezza	14
2.8 Requisiti non funzionali	15
2.9 Vincoli generali	17
2.10 Vincoli di implementazione	17
2.11 Componenti acquistati e free trial	17
2.12 Componenti open source	17
2.13 Glossario	18
2.14 Stima dei costi	20
2.14.1 Unadjusted Use Case Weight (UUCW)	20
2.14.2 Unadjusted Actor Weight (UAW)	21
2.14.3 Technical Complexity Factor (TCF)	22
2.14.4 Environmental Complexity Factor (ECF)	22
2.14.5 Total Use Case Points (UCP)	23
2.15 System Context Diagram	24
3 Processo di sviluppo	25
3.1 Framework di sviluppo	25
3.1.1 SCRUM	25
3.1.2 Suddivisione e sviluppo del progetto	25
3.1.3 Pratiche agili	30
3.2 Software per la condivisione del lavoro e per il supporto di SCRUM	36
3.2.1 Jira	36
3.2.2 Teams	39
3.2.3 Overleaf	39
3.2.4 GitHub	40
3.3 Software utilizzati per la progettazione e lo sviluppo	41

3.3.1 Visual Paradigm	41
3.3.2 Visual Studio Code	42
3.3.3 PythonAnyWhere	42
3.3.4 SQLite	43
4 Analisi dei Requisiti	44
4.1 Textual Analysis	44
4.2 Mappa Web App	45
4.3 Mockup e Design Concepts	46
4.4 Modello dei Casi d'Uso	48
4.4.1 Casi d'uso in dettaglio	49
4.5 System Sequence Diagram	57
4.5.1 System Sequence Diagram - Inserimento sintomi	57
4.5.2 System Sequence Diagram - Verifica Disponibilità Farmacie	58
4.5.3 System Sequence Diagram - Prenotazione tampone	59
4.5.4 System Sequence Diagram - Richiesta esito tampone	60
4.5.5 System Sequence Diagram - Richiesta lista prenotazioni	60
4.5.6 System Sequence Diagram - Creazione Disponibilità tamponi	61
4.5.7 System Sequence Diagram - Creazione farmacie	61
4.5.8 System Sequence Diagram - Ricerca farmacie	62
4.6 System Domain Model	63
4.7 State Machine Diagram degli stati Paziente	63
5 Architettura e progettazione del software	64
5.1 Scelte progettuali	64
5.2 Pattern Architettonico - Model View Controller (MVC) and Routes	66
5.2.1 Routes	66
5.2.2 Model	67
5.2.3 Vista	67
5.2.4 Controller	67
5.3 Stile Architettonico - Microservizi	67
5.4 REST	68
5.4.1 Client - Server in REST	69
5.4.2 Stateless	69
5.4.3 Cacheable	70
5.4.4 Interfaccia uniforme	70
5.4.5 Sistema stratificato	70
5.5 Vista componenti e connettori	71
5.6 Sequence Diagram di progettazione	73
5.6.1 Sequence Diagram di progettazione - Inserimento sintomi	73
5.6.2 Sequence Diagram di progettazione - Verifica disponibilità farmacie	74
5.6.3 Sequence Diagram di progettazione - Creazione disponibilità tampone	75
5.6.4 Sequence Diagram di progettazione - Prenotazione tampone	76
5.7 Activity Diagram	77
5.7.1 Activity Diagram - Inserimento sintomi	77
5.7.2 Activity Diagram - Verifica disponibilità farmacie	78
5.7.3 Activity Diagram - Creazione disponibilità tampone	78
5.7.4 Activity Diagram - Prenotazione tampone	79
5.8 Communication Diagram	79
5.8.1 Communication Diagram - Inserimento sintomi	79

5.8.2 Communication Diagram - Verifica disponibilità farmacie	80
5.8.3 Communication Diagram - Creazione disponibilità tampone	81
5.8.4 Communication Diagram - Prenotazione tampone	81
5.9 Server MVC Class Diagram	83
6 Implementazione del software	84
6.1 Scelte di sviluppo	84
6.2 Framework di sviluppo	85
6.2.1 Back-End: FLASK	85
6.2.2 Front-End: Bootstrap	87
6.3 Package Diagram	88
6.3.1 La struttura del progetto	88
6.4 Dinamica del Sistema MVC	90
6.4.1 Sequence Diagram di dettaglio - Inserimento Sintomi	90
6.4.2 Sequence Diagram di dettaglio - Verifica disponibilità farmacia	91
6.4.3 Sequence Diagram di dettaglio - Prenotazione tampone	92
6.4.4 Sequence Diagram di dettaglio - Creazione disponibilità tampone	93
6.5 Design Pattern	93
6.5.1 Pattern Grasp	94
6.5.2 Pattern Proxy	94
6.5.3 Pattern Observer	95
6.6 Deployment Diagram - Locale	95
6.7 Entity Relationship Diagram	96
7 Testing	98
7.1 Test funzionale	98
7.1.1 Test blackbox	98
7.1.2 Test di unità	102
7.1.3 Test di accettazione - Alfa Test	107
7.2 Test non funzionale	110
7.2.1 Test di carico	110
8 Rilascio del software	114
8.1 Rilascio - PythonAnywhere	114
8.1.1 Gestione delle collisioni e degli errori di esecuzione	114
8.1.2 Potenza di calcolo	116
8.1.3 Il funzionamento generale	116
8.2 Performance Test - post Rilascio	117
8.3 Test di accettazione - Beta Test - post Rilascio	119
8.4 Deployment Diagram - Remoto	120
9 Uso del prodotto software	121
9.1 Manuale d'utilizzo	121
9.1.1 Dashboard Admin	121
9.1.2 Home Page e inserimento sintomi, malattie e informazioni COVID19	121
9.1.3 Dashboard Farmacia	123
9.1.4 Il mio profilo	124
9.1.5 La prenotazione	125
9.2 Video dimostrativo e link alla Web Application SelfTestCOVID19	126
9.3 Scopi futuri	127

1 Introduzione

Contestualizzazione del progetto e illustrazione dell'idea di partenza

Contenuti

1.1	Contesto generale dell'applicazione	1
1.1.1	Che cos'è il COVID-19	1
1.1.2	Necessità dei tamponi	2
1.2	Reengineering del sistema di predizione: Probability Infection Checker dell'autore	2
1.2.1	Il dataset	2
1.2.2	Le fasi di Machine Learning sviluppate	2
1.2.3	Package Diagram del progetto Probability Infection Checker SARS-CoV2 COVID19	4
1.2.4	UI del progetto Probability Infection Checker SARS-CoV2 COVID19	5

1.1 Contesto generale dell'applicazione

Si desidera realizzare il sistema **SelfTest COVID-19** che sulla base dell'esito del test attiva la relativa prenotazione di un tampono in una delle farmacie più vicine al paziente, dunque un semplice servizio gestione prenotazioni tamponi nelle farmacie. Il SelfTest è un modello in grado di predire una certa probabilità di infezione al SARS-CoV-2(COVID-19) in base ai sintomi e/o alle malattie che il paziente manifesta e guidarlo a come comportarsi in caso di alta probabilità di infezione. Ad esempio, se il sistema predice che il paziente X abbia una **probabilità di infezione tra il 75% e il 100%** allora verrà consigliato di recarsi in un centro di tamponi (farmacia, ecc.) e di fare un **tampono molecolare**. Se il sistema predice che il paziente Y abbia una **probabilità di infezione tra il 50% e il 75%** allora verrà consigliato di fare un **tampono rapido** alla farmacia più vicina ad esso. Il sistema cerca di diminuire il numero di persone che si recano nelle farmacie per effettuare i tamponi al fine di evitare disservizi nelle stesse e per risparmiare sul costo dei tamponi da parte dei pazienti. In che modo? Prevedendo la probabilità potenziale al COVID-19 e in base al valore ottenuto può effettuare o meno una prenotazione del tampono alla farmacia più vicina.

1.1.1 Che cos'è il COVID-19

Covid-19 è il nome della malattia respiratoria causata dal nuovo coronavirus. *Co* sta infatti per corona, *vi* per virus, *d* per disease (ossia malattia in inglese) e *19* fa invece riferimento all'anno della sua comparsa in Cina, ossia il 2019.

Il Covid-19 è dunque causato da una tipologia virale appartenente alla famiglia dei coronavirus, responsabili di varie patologie che colpiscono l'apparato respiratorio, e che vanno dal raffreddore, alle più gravi SARS (dall'acronimo Middle East Respiratory Syndrome ossia il virus della sindrome respiratoria del Medioriente) e MERS (ossia della Sindrome Acuta Respiratoria Severa, una malattia contagiosa delle vie respiratorie).

Il Covid-19 può provocare moderati disturbi all'apparato respiratorio, fino anche alla polmonite, e non risponde al trattamento con antibiotici (gli antibiotici, infatti, non hanno effetti contro i virus, e dunque non vengono prescritti in caso di infezione virale).

1.1.2 Necessità dei tamponi

Il tampone molecolare è il test di riferimento per la diagnosi di infezione da SARS-CoV-2.

Ad oggi, il test molecolare su tampone nasofaringeo od orofaringeo è riconosciuto e validato a livello nazionale ed internazionale per sensibilità e specificità, in linea con le indicazioni riportate dalla World Health Organization (WHO) per il controllo della pandemia di COVID-19. Per l'effettuazione del tampone, i pazienti con presunti sintomi si recano principalmente in Farmacia.

1.2 Reengineering del sistema di predizione: Probability Infection Checker dell'autore

Si adotta la reingegnerizzazione della seguente applicazione:

(Probability Infection Checker SARS-CoV2 COVID19)

<https://covid-probability-checker.herokuapp.com/>

a partire dalla sua progettazione, al fine di migliorarne o aggiungervi: funzionalità, interfacciamento con altri processi o sistemi, piattaforme di supporto, qualità (incluse facilità d'uso, manutenibilità, leggibilità), eventualmente implementandolo con nuove tecnologie al posto di quelle precedentemente utilizzate. L'obiettivo del software che si vuole reingegnerizzare è dunque predire una certa probabilità di infezione al COVID-19 in base ai sintomi o alle malattie che il paziente manifesta e guidarlo a come comportarsi in caso di alta probabilità di infezione. Il compito degli autori sarà quello di aggiungere funzionalità, attori estendendo l'applicazione del modello di Machine Learning (AI) che esso **ingloba**.

1.2.1 Il dataset

Si è trovato un dataset che contiene una serie di possibili sintomi legati al COVID-19 e il risultato del tampone o test sierologico per capire se una X persona, con quei sintomi, ha il COVID oppure no (target). I sintomi (le colonne) presenti nel dataset sono indicate nella **Tabella 1.1**:

Dall' analisi dei sintomi presenti nel dataset si nota come il mal di gola, la tosse secca, la febbre e il problema di respirazione siano i sintomi più probabili in presenza del COVID-19. Altre colonne presenti nel dataset sono: abroad travel viaggi all'estero), contact with COVID Patient (contatti con un paziente COVID), attended large gathering (partecipazioni a raduni affollati), visited public exposed places (luoghi pubblici esposti visitati), family working in public exposed places (famiglia che lavora in luoghi pubblici esposti), wearing masks (maschera indossata), sanitization from market (sanificazione dal mercato) e la colonna che determina se una persona è affetta o meno dal virus: COVID-19.

N.B *L'origine dei dati è puramente studiata per essere utilizzata nella modellazione del modello di Machine Learning e dunque non potranno essere utilizzati per "contesti reali" ma solo per scopi didattici*

1.2.2 Le fasi di Machine Learning sviluppate

Per il supporto del modello di Machine Learning e del suo corretto funzionamento sono state seguite le seguenti fasi di Machine Learning:

- **Raccolta e preparazione dei dati:** Viene scelto il dataset e preparato. In particolare il dataset viene mischiato per essere sicuri che sia il training set che il test set siano rappresentativi dei dati.

Sintomo	SI (Presenza del sintomo) %	NO (Non presenza del sintomo) %
Breathing problem Problemi di respirazione	67%	33%
Fever Febbre	79%	21%
Dry Cough Tosse secca	79%	21%
Sore throat Mal di gola	73%	27%
Running nose Naso che cola	54%	46%
Asthma Asma	46%	54%
Chronic Lung Disease Malattia polmonare cronica	47%	53%
Headache Mal di testa	50%	50%
Heart Disease Malattia cardiaca	46%	54%
Diabetes Diabete	48%	52%
Hyper Tension Ipertensione	49%	51%
Fatigue Fatica	47%	53%
Gastrointestinal Problemi gastrointestinali	45%	55%

Tabella 1.1: Dataset Sintomi Sars-CoV2 (COVID19)

- **Scelta del modello e fitting dei parametri:** In questa fase viene scelto il modello di Regressione Logistica. Viene fatto poi il "fitting" che consiste nel calcolare i parametri del modello.
- **Valutazione delle prestazioni del modello :** Si passa alla valutazione delle prestazioni del modello sul test set tramite le metriche di valutazione in termini di accuracy e delle altre metriche derivanti dalla matrice di confusione.
- **La Prediction:** In questa fase si prova ad effettuare una predizione con il modello opportunamente addestrato. A tal proposito forniamo ad esso l'input, ossia i sintomi del paziente, per valutare la probabilità di infezione al COVID-19.

Le fasi di Machine Learning del modello in esame sono state approfondite e sviluppate nel progetto del Corso Accademico di Calcolo Numerico dove si è studiata tutta la parte di analisi matematica e probabilistica:

(Link alla documentazione completa)

https://github.com/giuseppericcio/Seminario_MachineLearning_CN

Inoltre si noti che le fasi sono state sviluppate in *MatLab* ma nel progetto in esame saranno sviluppate in Python per la realizzazione della Web Application, come si nota nella Figura 1.1.

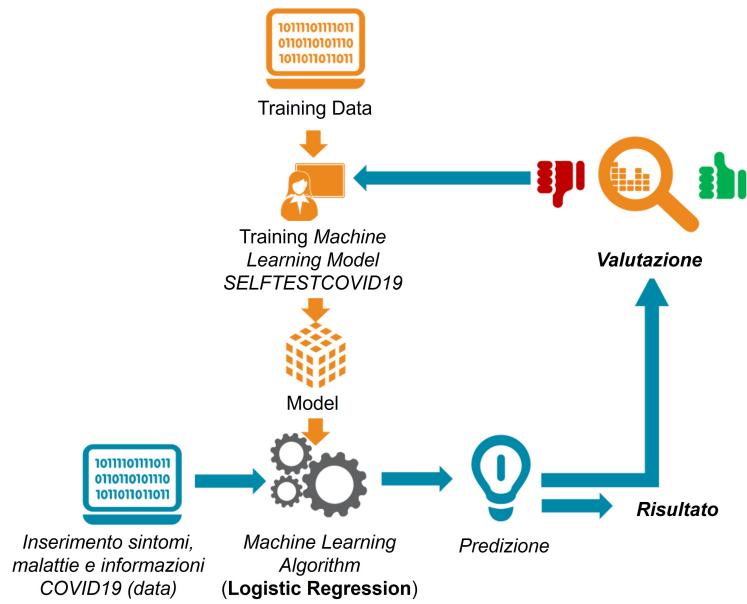


Figura 1.1: Come funziona il test

1.2.3 Package Diagram del progetto Probability Infection Checker SARS-CoV2 COVID19

Si mostra (Figura 1.2) il namespace del progetto di base in quanto a partire da esso si svilupperà e si provvederà a nuove soluzioni sia architettonicali e sia implementativi.

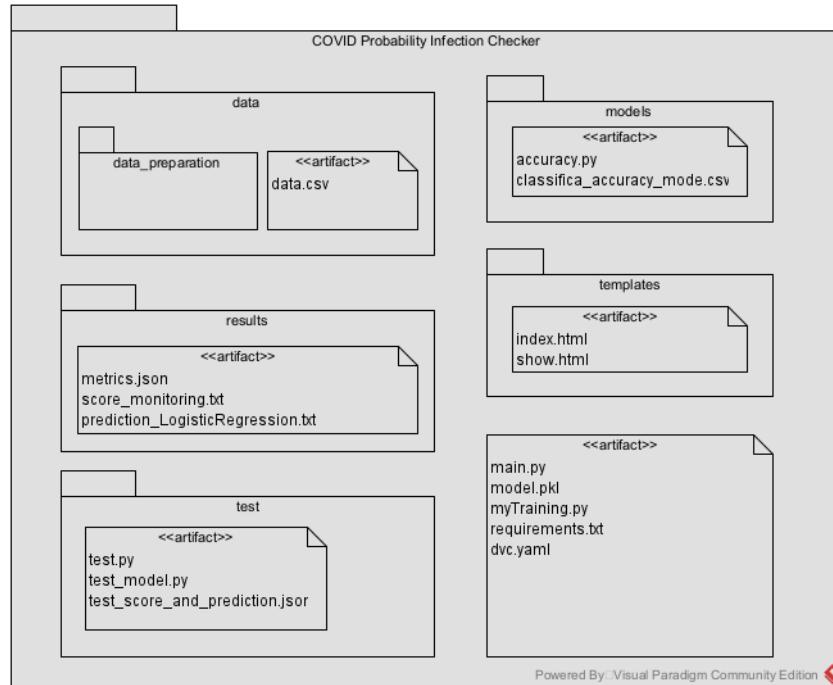
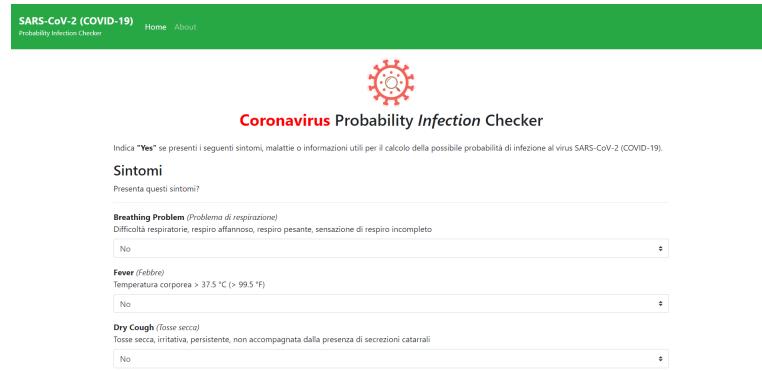


Figura 1.2: Package Diagram del progetto Probability Infection Checker SARS-CoV2 COVID19

1.2.4 UI del progetto Probability Infection Checker SARS-CoV2 COVID19

Si mostra anche la user interface (Figura 1.3) in quanto anche essa sarà da fondamenta per lo sviluppo del front-end del prodotto software in esame.



The screenshot shows a web-based application titled "SARS-CoV-2 (COVID-19) Probability Infection Checker". The header includes links for "Home" and "About". Below the header is a red circular icon representing the coronavirus. The main title is "Coronavirus Probability Infection Checker". A subtitle indicates it "Indica "Yes"" se presenti i seguenti sintomi, malattie o informazioni utili per il calcolo della possibile probabilità di infezione al virus SARS-CoV-2 (COVID-19)". The form consists of three sections: "Sintomi" (Symptoms), "Breathing Problem (Problema di respirazione)", and "Fever (Fevere)". Each section contains a brief description and a dropdown menu with "No" selected. The "Sintomi" section also includes a link to "Presenta questi sintomi?".

Figura 1.3: UI del progetto Probability Infection Checker SARS-CoV2 COVID19

Pertanto, da i prossimi paragrafi, il progetto prenderà il nome di **SelfTestCOVID19**.

2 Avvio della progettazione

Si delineano tutti gli obiettivi del progetto a partire dalla definizione degli attori alle funzioni che ciascuno deve svolgere

Contenuti

2.1 Attori e obiettivi generali	6
2.2 Funzionalità generali del sistema software	7
2.3 Assunzioni e dipendenze	8
2.4 Storie utente (Requisiti funzionali)	8
2.5 Requisiti di interfaccia esterna	8
2.5.1 Interfaccia Utente	8
2.5.2 Interfaccia Hardware	14
2.5.3 Interfaccia Software	14
2.5.4 Interfaccia di Comunicazione	14
2.6 Requisiti di Consegnna	14
2.7 Requisiti di Riservatezza	14
2.8 Requisiti non funzionali	15
2.9 Vincoli generali	17
2.10 Vincoli di implementazione	17
2.11 Componenti acquistati e free trial	17
2.12 Componenti open source	17
2.13 Glossario	18
2.14 Stima dei costi	20
2.14.1 Unadjusted Use Case Weight (UUCW)	20
2.14.2 Unadjusted Actor Weight (UAW)	21
2.14.3 Technical Complexity Factor (TCF)	22
2.14.4 Environmental Complexity Factor (ECF)	22
2.14.5 Total Use Case Points (UCP)	23
2.15 System Context Diagram	24

2.1 Attori e obiettivi generali

Si descrivono, ad alto livello, gli attori del sistema e i relativi obiettivi generali:

Paziente

- Il Paziente svolgerà il SelfTest per la valutazione dei propri sintomi. Il SelfTest è basato su un modello di **Machine Learning (ML)**.
- Il Paziente, al fine di verificare la disponibilità di farmacie, inserirà il CAP o il Nome del comune di residenza per cercare le farmacie che hanno disponibilità del tampone rapido o molecolare. Sceglierà l'ora e il giorno nel quale prenotare.

Paziente Registrato

- In caso di disponibilità, il Paziente per effettuare la prenotazione, dovrà rilasciare dati anagrafici (nome, cognome, email, password, codice fiscale, telefono, accettazione delle normative sulla privacy). Ad ogni nuovo Paziente Registrato è assegnato un codice alfanumerico d'identificazione. Riceverà via Mail la prenotazione effettuata con successo e se vuole, in secondo momento modificare e/o cancellare la prenotazione. Prenotazione che dovrà poi essere presentata tramite QR code in farmacia per fare il tampone.
- Il Paziente Registrato, alla ricezione dell'esito del tampone, potrà aggiungere i suoi possibili sintomi.

Farmacia

- La Farmacia aggiornerà la disponibilità dei tamponi indicando l'orario di apertura e di chiusura e i relativi orari di disponibilità per svolgere il tampone (la durata dell'inalazione del tampone è circa di 15-20 minuti) al fine di permettere ai pazienti di effettuare autonomamente le loro prenotazioni online, o di visionare semplicemente gli orari ancora disponibili in agenda.
- La Farmacia potrà modificare e/o rimuovere le prenotazioni effettuate dai pazienti presso di essa.
- La Farmacia dopo aver processato il tampone, aggiungerà l'esito di quest'ultimo, ed il paziente riceverà una Mail che lo avvisa della disponibilità dell'esito del tampone.

Admin di Sistema

- L'admin di sistema aggiornerà il database delle farmacie, inserendo, modificando ed eliminando nuove farmacie che aderiranno al sistema.
- L'admin di sistema aggiornerà il modello di Machine Learning del Self Test. Per semplicità, verrà scelto un dataset per l'estrazione dei dati, verranno preparati i dati e poi addestrati. Successivamente, il modello di ML verrà automatizzato e distribuito secondo alcuni principi e tools ed infine il monitoraggio con l'obiettivo di ottimizzare il modello e renderlo sempre disponibile e aggiornato ad effettuare predizioni sempre più precise.

2.2 Funzionalità generali del sistema software

Il sistema software "SelfTestCOVID19" deve:

- **Predire infezione da COVID19 al paziente (SELFTEST)** tramite modello di ML.
- **Gestire la prenotazione dei tamponi alla farmacia più vicina al paziente.**
 - Prenotazione tampone rapido o molecolare in base al risultato ottenuto al test.
 - Variazione (modifica) data prenotazione.
 - Annullamento prenotazione.
 - Riepilogo prenotazioni per paziente.
 - Riepilogo prenotazioni per data.
- **Gestire prenotazioni lato Farmacia**
 - Aggiornamento dati prenotazione.
 - Riepilogo prenotazioni per data.
 - Inserire l'esito del tampone del paziente X.

- Aggiornamento disponibilità dei tamponi.
- **Gestire lista delle Farmacie**
- **Gestire Modello di ML**

Le informazioni trattate devono poter essere gestite da diverse postazioni (terminali).

2.3 Assunzioni e dipendenze

Il **Sistema** complessivamente deve essere semplice:

- Il sistema software “SelfTestCOVID19” dovrà essere utilizzato su qualsiasi macchina con browser e connessione ad Internet.
- I pazienti potranno svolgere il test e con semplicità prenotare il tampone negli orari a loro più comodi soltanto se il test abbia avuto esito con percentuale maggiore del 50%.

2.4 Storie utente (Requisiti funzionali)

Per la definizione dei requisiti funzionali del sistema in esame, si fa uso degli epic (un macro requisito), ognuno dei quali presenta più storie utente.

La storia utente è composta da:

- **ID (SUXY):** identificativo della storia utente
- **Titolo:** titolo della storia utente
- **Descrizione:** descrizione della storia utente formulati in questo modo: *Come [ruolo], Io Voglio [obiettivo], In modo da [beneficio]*
- **Priorità:** priorità della storia utente in modo da avere un ordine di sviluppo delle stesse storie
- **Story Points:** punteggio della storia utente, tanto più è alto tanto più è complicata la realizzazione della stessa
- **Epic Link:** individuazione dell'epic di appartenenza
- **Etichetta:** individuazione dell'attore di appartenenza

2.5 Requisiti di interfaccia esterna

2.5.1 Interfaccia Utente

Il sistema software “SelfTestCOVID19” deve essere dotato di un’interfaccia amichevole, con menu, pagine e pulsanti. Con soli pochi click il Paziente e la Farmacia potranno gestire la prenotazione negli orari più comodi. Per l’Admin di sistema, semplicità di gestione delle farmacie.

Storia Utente	SU01
Titolo	Inserimento sintomi
Descrizione	Come paziente lo voglio inserire i miei sintomi all'interno del form In modo da sapere se devo o meno effettuare un tampone per il Covid-19
Priorità	Massima
Story Points	3
Epic Link	Gestione sintomi
Etichette	Requisito Paziente

Tabella 2.1: Storia Utente 01: Requisito Paziente - Gestione sintomi

Storia Utente	SU02
Titolo	Verifica disponibilità farmacie
Descrizione	Come paziente lo voglio verificare la disponibilità delle farmacie e delle relative disponibilità di tipologie di tamponi in un determinato giorno e orario In modo da scegliere un appuntamento per effettuare il tampone
Priorità	Alta
Story Points	8
Epic Link	Gestione prenotazione
Etichette	Requisito Paziente

Tabella 2.2: Storia Utente 02: Requisito Paziente - Gestione prenotazione

Storia Utente	SU03
Titolo	Prenotazione tampone
Descrizione	Come paziente registrato lo voglio prenotare un tampone In modo da certificare il mio stato di salute in merito al virus Covid-19
Priorità	Alta
Story Points	13
Epic Link	Gestione prenotazione
Etichette	Requisito Paziente Registrato

Tabella 2.3: Storia Utente 03: Requisito Paziente Registrato - Gestione prenotazione

Storia Utente	SU04
Titolo	Modifica prenotazione
Descrizione	Come paziente registrato lo voglio modificare la prenotazione fissata in caso di eventuali errori o in base al proprio interesse In modo da aggiornare i dettagli dell'appuntamento in farmacia.
Priorità	Media
Story Points	3
Epic Link	Gestione prenotazione
Etichette	Requisito Paziente Registrato

Tabella 2.4: Storia Utente 04: Requisito Paziente Registrato - Gestione prenotazione

Storia Utente	SU05
Titolo	Rimozione prenotazione
Descrizione	Come paziente registrato lo voglio rimuovere la prenotazione fissata In modo da eliminare l'appuntamento dalla lista delle prenotazioni della farmacia
Priorità	Media
Story Points	3
Epic Link	Gestione prenotazione
Etichette	Requisito Paziente Registrato

Tabella 2.5: Storia Utente 05: Requisito Paziente Registrato - Gestione prenotazione

Storia Utente	SU06
Titolo	Riepilogo prenotazione
Descrizione	Come paziente registrato lo voglio ricevere tramite e-mail la conferma dell'appuntamento con i dettagli In modo da poterla mostrare in farmacia tramite QR-code
Priorità	Media
Story Points	5
Epic Link	Gestione prenotazione
Etichette	Requisito Paziente Registrato

Tabella 2.6: Storia Utente 06: Requisito Paziente Registrato - Gestione prenotazione

Storia Utente	SU07
Titolo	Richiesta esito tampone
Descrizione	Come paziente registrato lo voglio richiedere al sistema di visualizzare l'esito del tampone effettuato presso una farmacia attraverso le credenziali inserite durante la prenotazione In modo da conoscere il mio stato di salute e prendere i dovuti comportamenti.
Priorità	Alta
Story Points	5
Epic Link	Resoconto tampone
Etichette	Requisito Paziente Registrato

Tabella 2.7: Storia Utente 07: Requisito Paziente Registrato - Resoconto tampone

Storia Utente	SU08
Titolo	Aggiunta sintomi al modello ML
Descrizione	Come paziente registrato lo voglio poter aggiungere l'esito del tampone, e dei relativi sintomi In modo da migliorare l'accuratezza del modello di predizione.
Priorità	Bassa
Story Points	8
Epic Link	Resoconto tampone
Etichette	Requisito Paziente Registrato

Tabella 2.8: Storia Utente 08: Requisito Paziente Registrato - Resoconto tampone

Storia Utente	SU09
Titolo	Richiesta lista prenotazioni
Descrizione	Come farmacia lo voglio richiedere di visualizzare tutte le prenotazioni fissate dai pazienti presso la propria farmacia In modo da organizzare i turni per effettuare i tamponi della lista
Priorità	Alta
Story Points	5
Epic Link	Resoconto prenotazioni
Etichette	Requisito Farmacia

Tabella 2.9: Storia Utente 09: Requisito Farmacia - Resoconto prenotazioni

Storia Utente	SU10
Titolo	Modifica prenotazioni
Descrizione	Come farmacia lo voglio modificare le note In modo da aggiornare i dettagli delle prenotazioni
Priorità	Media
Story Points	3
Epic Link	Resoconto prenotazioni
Etichette	Requisito Farmacia

Tabella 2.10: Storia Utente 10: Requisito Farmacia - Resoconto prenotazioni

Storia Utente	SU11
Titolo	Rimozione prenotazioni
Descrizione	Come farmacia lo voglio rimuovere le prenotazioni In modo da aggiornare la lista delle prenotazioni
Priorità	Media
Story Points	3
Epic Link	Resoconto prenotazioni
Etichette	Requisito Farmacia

Tabella 2.11: Storia Utente 11: Requisito Farmacia - Resoconto prenotazioni

Storia Utente	SU12
Titolo	Creazione disponibilità tamponi
Descrizione	Come farmacia lo voglio creare la disponibilità dei tamponi In modo da mettere a disposizione del paziente le date e le fasce orarie per effettuare il tampone.
Priorità	Alta
Story Points	13
Epic Link	Gestione disponibilità tamponi
Etichette	Requisito Farmacia

Tabella 2.12: Storia Utente 12: Requisito Farmacia - Gestione disponibilità tamponi

Storia Utente	SU13
Titolo	Modifica disponibilità tamponi
Descrizione	Come farmacia lo voglio modificare i dati relativi alla disponibilità dei tamponi In modo da aggiornare i dettagli di disponibilità per i pazienti.
Priorità	Media
Story Points	5
Epic Link	Gestione disponibilità tamponi
Etichette	Requisito Farmacia

Tabella 2.13: Storia Utente 13: Requisito Farmacia - Gestione disponibilità tamponi

Storia Utente	SU14
Titolo	Rimozione disponibilità tamponi
Descrizione	Come farmacia lo voglio rimuovere i tamponi nel momento in cui essi diventino non disponibili In modo da aggiornare i dettagli di disponibilità per i pazienti.
Priorità	Bassa
Story Points	5
Epic Link	Gestione disponibilità tamponi
Etichette	Requisito Farmacia

Tabella 2.14: Storia Utente 14: Requisito Farmacia - Gestione disponibilità tamponi

Storia Utente	SU15
Titolo	Aggiunta esito tampone
Descrizione	Come farmacia lo voglio aggiungere l'esito del tampone In modo da notificare tramite Mail il paziente del proprio stato di salute.
Priorità	Media
Story Points	5
Epic Link	Gestione esito tampone
Etichette	Requisito Farmacia

Tabella 2.15: Storia Utente 15: Requisito Farmacia - Gestione esito tampone

Storia Utente	SU16
Titolo	Verifica prenotazione
Descrizione	Come farmacia lo voglio verificare la prenotazione del paziente In modo da scansionare il QR code all'arrivo del paziente in farmacia
Priorità	Media
Story Points	5
Epic Link	Resoconto prenotazioni
Etichette	Requisito Farmacia

Tabella 2.16: Storia Utente 16: Requisito Farmacia - Resoconto prenotazioni

Storia Utente	SU17
Titolo	Creazione farmacie
Descrizione	Come admin di sistema lo voglio creare la lista di farmacie In modo da permettere ad ogni farmacia di accedere al sistema in esame
Priorità	Alta
Story Points	5
Epic Link	Gestione farmacie
Etichette	Requisito Admin di sistema

Tabella 2.17: Storia Utente 17: Requisito Admin di sistema - Gestione farmacie

Storia Utente	SU18
Titolo	Ricerca farmacie
Descrizione	Come admin di sistema lo voglio cercare la lista di farmacie In modo da visualizzare la lista delle farmacie presenti sul sistema
Priorità	Alta
Story Points	5
Epic Link	Gestione farmacie
Etichette	Requisito Admin di sistema

Tabella 2.18: Storia Utente 18: Requisito Admin di sistema - Gestione farmacie

Storia Utente	SU19
Titolo	Modifica farmacie
Descrizione	Come admin di sistema lo voglio modificare le informazioni delle farmacie In modo da mantenere aggiornati i dettagli delle farmacie
Priorità	Media
Story Points	3
Epic Link	Gestione farmacie
Etichette	Requisito Admin di sistema

Tabella 2.19: Storia Utente 19: Requisito Admin di sistema - Gestione farmacie

Storia Utente	SU20
Titolo	Rimozione farmacie
Descrizione	Come admin di sistema lo voglio rimuovere le farmacie In modo da aggiornare la lista delle farmacie disponibili
Priorità	Media
Story Points	3
Epic Link	Gestione farmacie
Etichette	Requisito Admin di sistema

Tabella 2.20: Storia Utente 20: Requisito Admin di sistema - Gestione farmacie

2.5.2 Interfaccia Hardware

Il sistema software "SelfTestCOVID19" non deve interfacciarsi con nessun sistema hardware.

2.5.3 Interfaccia Software

Il sistema software "SelfTestCOVID19" deve interfacciarsi con il sistema software per la gestione del database "SQLite" attraverso query e comandi SQL.

2.5.4 Interfaccia di Comunicazione

Il sistema software "SelfTestCOVID19" richiede l'uso di una particolare interfaccia di comunicazione per l'invio delle Mail di conferma della prenotazione e per la notifica della disponibilità dell'esito del tampone effettuato attraverso il servizio di "Register.it" utilizzando il protocollo SMTP.

2.6 Requisiti di Consegna

Deve essere consegnata una demo del progetto entro un mese.

2.7 Requisiti di Riservatezza

Ogni attore del sistema può accedere solo alle aree ad esso riservate. La *Farmacia* non può accedere ai dati dell'*Admin di sistema* e nè dei *Pazienti* registrati, vale anche il viceversa. Inoltre, il *Paziente* registrato può accedere esclusivamente al proprio profilo personale.

2.8 Requisiti non funzionali

I requisiti non funzionali sono tutte quelle caratteristiche del software non richieste dal cliente, non descrivono "il cosa", ma "il come" il sistema fa ad eseguire certi compiti.

RNF01	Requisiti Prestazionali - Tempi di risposta
Descrizione	Il sistema DEVE consentire all'utente di ottenere risposte in tempo reale. Nel caso in esame, il sistema è distribuito su un Server, che deve essere accessibile in modalità remota (tramite Web Application), il requisito fondamentale è di disporre di un Web Server con buone prestazioni in grado di eseguire uno script Python (Flask). Si richiede, dunque, che per una buona operatività da parte dell'utente è opportuno che il sistema analizzi il carico di rete in modo da scegliere la linea più libera, diminuendo anche la probabilità di perdita di pacchetti dati.
Priorità	Obbligatorio

Tabella 2.21: Requisito non funzionale 01: Requisiti Prestazionali - Tempi di risposta

RNF02	Requisiti Prestazionali - Multiutenza
Descrizione	Il sistema DEVE garantire che le informazioni tratte dal sistema software "SelfTestCovid19" devono poter essere gestite da diverse postazioni(terminali). Infatti, sia il Farmacia che il paziente devono disporre delle funzionalità di gestione/consultazione delle prenotazioni.
Priorità	Obbligatorio

Tabella 2.22: Requisito non funzionale 02: Requisiti Prestazionali - Multiutenza

RNF03	Requisiti di sicurezza - Integrità dei dati
Descrizione	Il sistema DEVE garantire che solo gli admin di sistema possano effettuare alcune operazioni critiche, come: aggiungere o rimuovere farmacie.
Priorità	Obbligatorio

Tabella 2.23: Requisito non funzionale 03: Requisiti di sicurezza - Integrità dei dati

RNF04	Requisiti di sicurezza - Disponibilità
Descrizione	Il sistema DEVE garantire che le funzionalità di prenotazione e di test sui sintomi siano sempre disponibili. In maniera tale da garantire che i pazienti possano in qualsiasi momento usare il sistema.
Priorità	Obbligatorio

Tabella 2.24: Requisito non funzionale 04: Requisiti di sicurezza - Disponibilità

RNF05	Vincoli di sistema - Persistenza dei dati
Descrizione	I dati inseriti dagli utenti e dai farmacisti del sistema DOVREBBERO essere memorizzati in maniera persistente all'interno di un RDBMS.
Priorità	Obbligatorio

Tabella 2.25: Requisito non funzionale 05: Vincoli di sistema - Persistenza dei dati

RNF06	Usabilità di sistema - Comprensibilità
Descrizione	Il sistema DEVE permettere agli utilizzatori di essere comprensibile per aumentare le caratteristiche di affidabilità e di modificabilità
Priorità	Obbligatorio

Tabella 2.26: Requisito non funzionale 06: Usabilità di sistema - Comprensibilità

RNF07	Usabilità di sistema - Apprendibilità
Descrizione	Il sistema DEVE essere capace di permettere all'utente di imparare ad usare l'applicazione.
Priorità	Obbligatorio

Tabella 2.27: Requisito non funzionale 07: Usabilità di sistema - Apprendibilità

RNF08	Transazioni - Proprietà ACID
Descrizione	<p>Il sistema DEVE garantire il corretto indirizzamento delle richieste del Client al Server, che nel caso in esame, devono essere completate interamente o per niente. Queste richieste prendono il nome di <i>transazioni</i> ed esse devono rispettare le cosiddette proprietà ACID:</p> <ul style="list-style-type: none"> ● Atomicity (A): una transazione è un'unità indivisibile. O viene completata (committed) o è annullata (rolled back); ● Consistency (C): dopo la transazione il sistema deve essere in uno stato consistente, ovvero tutti i vincoli di integrità definiti devono essere rispettati; ● Isolation (I): una transazione non dipende da altre transazioni; ● Durability (D): le modifiche devono essere permanenti dopo l'esecuzione della transazione. Devono sopravvivere in caso di fail.
Priorità	Obbligatorio

Tabella 2.28: Requisito non funzionale 08: Transazioni - Proprietà ACID

2.9 Vincoli generali

Il sistema software deve essere sviluppato rispettando le regole imposte dal GDPR (General Data Protection Regulation) per il rispetto della privacy dell'utente e i termini contrattuali relativi ai dati. I Terms of Service e la Privacy Policy verranno letti e accettati dall'utente in fase di registrazione.

Durante la fase di progettazione si devono garantire i tre elementi fondamentali quali:

- **Scopo**, che consiste nel delineare cosa deve essere fatto e come. Indicato nei precedenti paragrafi.
- **Tempo**, individuando il "timing" di realizzazione. Indicato nei paragrafi successivi.
- **Costo**, inteso come l'insieme delle risorse necessarie. Indicato nei paragrafi successivi.

Graficamente possiamo rappresentare questi tre vincoli di progettazione tramite un triangolo 2.1, in cui ciascun lato rappresenta un elemento, e dunque, sta ai progettisti trovare il miglior equilibrio tra di essi affinchè il triangolo sia equilatero.

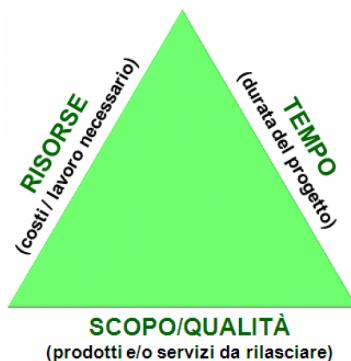


Figura 2.1: Triangolo dei vincoli di Progetto (tempo, costi e scopo)

2.10 Vincoli di implementazione

Si necessita di utilizzare un linguaggio di programmazione che supporta il Modello di Machine Learning e le sue fasi attraverso una vasta gamma di librerie e algoritmi: **Python**.

2.11 Componenti acquistati e free trial

Si è acquistato gratuitamente il dominio host da *Register.it* su cui verrà caricato la Web Application e il servizio *Email* per il trasferimento di essi che sarà necessario in seguito.

Si è acquistato lo spazio PythonAnyWhere per lo sblocco di alcune feature per il supporto della Web Application.

Il **costo totale** per la progettazione è di 5€.

2.12 Componenti open source

Si è usato un componente open source per l'applicazione del QR Code.

2.13 Glossario

Si definiscono alcuni termini specifici utilizzanti durante la stesura del documento.

Termino	Alias	Descrizione
SelfTestCOVID19		Il nome del prodotto software.
Tampone		È generalmente costituito da un bastoncino con del cotone sterile ad una estremità (tipo un lungo cotton fioc) che viene poggiate, o leggermente strofinato, sulla zona da indagare. Il materiale prelevato viene quindi analizzato dal laboratorio di microbiologia.
Dashboard		Una dashboard fornisce viste a colpo d'occhio agli utilizzatori per dare la possibilità di gestire le impostazioni, prenotazioni, disponibilità in un'unica posizione.
Hypertext Markup Language	HTML	È un sistema standardizzato per l'aggiunta di tag ai file di testo per ottenere effetti di carattere, colore, grafica e collegamento ipertestuale sulle pagine World Wide Web.
Machine Learning	ML	Un sottosistema dell'intelligenza artificiale (AI) che si occupa di creare sistemi che apprendono o migliorano le performance in base ai dati che utilizzano.
Web API		Sono delle API alle quali è possibile accedere utilizzando il protocollo HTTP. Come le API classiche, anche loro consentono agli sviluppatori di accedere a funzionalità o dati specifici di un'applicazione, sistema operativo o altri servizi.
Web Application		Una Web Application, cioè un'applicazione web chiamata più comunemente Web-App, è un programma applicativo distribuito che permette di usufruire, tramite accesso da browser, di determinati servizi residenti su un Server remoto.
Service Oriented Architecture	SOA	SOA è un approccio architettonico in cui le applicazioni utilizzano i servizi disponibili nella rete.

Tabella 2.29: Glossario dei termini

Termino	Alias	Descrizione
Separation of Concerns	SOC	È un principio di progettazione software nel quale individua la separazione in livelli e componenti tale che ciascuno abbia funzionalità distinte con la minor sovrapposizione possibile
Platform as a service	PAAS	è un modello di cloud computing in cui un provider di terze parti fornisce strumenti hardware e software agli utenti su Internet.
Framework		È un'architettura logica di supporto sul quale un software può essere progettato e realizzato facilitandone lo sviluppo da parte del programmatore.
Hyper Text Transfer Protocol	HTTP	Hyper Text Transfer Protocol è la sigla apposta davanti a un dominio per visualizzare una pagina Web. Questo protocollo di trasmissione di documenti ipertestuali ordina al Server di far visualizzare su un computer una determinata pagina web
Object Oriented Programming	OOP	La programmazione orientata agli oggetti è un paradigma di programmazione che si basa sul concetto di classi e oggetti. Viene utilizzato per strutturare un programma software in parti di codice semplici e riutilizzabili (solitamente chiamate classi), che vengono utilizzate per creare singole istanze di oggetti.
Diagramma UML		presentazione visiva unificata del sistema UML (Unified Modeling Language) che intende consentire agli sviluppatori o ai proprietari di aziende di comprendere, analizzare e intraprendere la struttura e i comportamenti del loro sistema. Si dividono in diagrammi di strutturali (class, component, deployment, package ...) e in diagramma comportamentali (use case, state, activity, communication, sequence ...)

Tabella 2.30: Glossario dei termini pt.2

2.14 Stima dei costi

Per la valutazione della stima dei costi si avvale degli **Use Case Points(UCP)**. È una tecnica di stima del software utilizzata per **prevedere** le dimensioni del software per i progetti di sviluppo software. UCP viene utilizzato quando le metodologie UML (Unified Modeling Language) vengono utilizzate per la progettazione e lo sviluppo del software. Il concetto di UCP si basa sui requisiti per il sistema scritto utilizzando casi d'uso, che fa parte del set UML di tecniche di modellazione. La dimensione del software (UCP) viene calcolata in base agli elementi dei casi d'uso del sistema con il factoring per tenere conto di considerazioni tecniche e ambientali. L'UCP per un progetto può quindi essere utilizzato per calcolare lo sforzo stimato per completare un progetto.

2.14.1 Unadjusted Use Case Weight (UUCW)

I valori di riferimento assegnati a ciascun caso d'uso, detti anche **USE CASE CLASSIFICATION**, sono:

- *Simple* (1 - 3 transiction) Weight 5.
- *Average* (4 - 7 transiction) Weight 10.
- *Complex* (8 or more transiction) Weight 15.

USE CASE	COMPLESSITA'
PAZIENTE - Inserire sintomi	Simple
PAZIENTE REGISTRATO - Effettua prenotazione	Average
PAZIENTE REGISTRATO - Aggiungere sintomi	Average
FARMACIA - Gestione tamponi	Complex
FARMACIA - Gestione prenotazione	Complex
ADMIN DI SISTEMA - Gestione farmacie	Complex
ADMIN DI SISTEMA - Gestione modello ML	Complex

Tabella 2.31: Elenco casi d'uso Paziente, Paziente Registrato, Farmacia, Admin di Sistema

Valutazione dell'UUCW:

$$\begin{aligned}
 \text{UCCW} = & (\text{TotalNo.ofSimpleUC} * 5) + \\
 & + (\text{TotalNo.AverageUC} * 10) + \\
 & + (\text{TotalNo.ComplexUC} * 15)
 \end{aligned} \tag{2.1}$$

COMPLESSITA'	Peso	No. UC	No.UC * Peso
Simple	5	1	5
Average	10	2	20
Complex	15	4	60
Totale			85

Tabella 2.32: Calcolo UCCW

2.14.2 Unadjusted Actor Weight (UAW)

Valutazione dell'associazione attori del sistema e complessità di realizzazione.

I valori di riferimento:

- *Simple*: Sistema esterno che deve interagire con il sistema utilizzando un'API ben definita - Weight 1.
- *Average*: Sistema esterno che deve interagire con il sistema utilizzando protocolli di comunicazione standard (es. TCP/IP, FTP, HTTP, database) - Weight 2.
- *Difficult*: Attore umano che utilizza un'interfaccia applicativa GUI - Weight 3.

Valutazione dell' UAW:

$$\begin{aligned}
 \textbf{UAW} = & (\text{Tot.No.ofSimpleActors} * 1) + \\
 & + (\text{Tot.No.AverageActors} * 2) + \\
 & + (\text{Tot.No.ComplexActor} * 3)
 \end{aligned} \tag{2.2}$$

ACTORS	COMPLESSITA'
PAZIENTE	Simple
PAZIENTE REGISTRATO	Complex
FARMACIA	Complex
ADMIN DI SISTEMA	Complex

Tabella 2.33: Associazione ACTORS - COMPLESSITA'

COMPLESSITA'	Peso	No. Actors	No.Actors * Peso
Simple	1	1	1
Average	2	0	0
Complex	3	3	9
Totale			10

Tabella 2.34: Calcolo UAW

2.14.3 Technical Complexity Factor (TCF)

Il TCF è uno dei fattori applicati alla dimensione stimata del software al fine di tenere conto delle considerazioni tecniche del sistema. È determinato assegnando un punteggio compreso tra 0 (il fattore è irrilevante) e 5 (il fattore è essenziale) a ciascuno dei 13 fattori tecnici elencati nella tabella seguente. Questo punteggio viene quindi moltiplicato per il valore ponderato definito per ciascun fattore.

I valori di riferimento:

- T_1 - Sistema distribuito, 2.0.
- T_2 - Tempo di risposta/obiettivi prestazionali, 1.0.
- T_3 - Efficienza dell'utente finale, 1.0.
- T_4 - Complessità di elaborazione interna, 1.0.
- T_5 - Riusabilità del codice, 1.0.
- T_6 - Facile da installare, 0.5.
- T_7 - Facile da usare, 0.5.
- T_8 - Portabilità su altre piattaforme, 2.0.
- T_9 - Manutenzione del sistema 1.0.
- T_{10} - Elaborazione simultanea/parallela, 1.0.
- T_{11} - Funzioni di sicurezza, 1.0.
- T_{12} - Accesso per terze parti, 1.0.
- T_{13} - Formazione dell'utente finale, 1.0.

Per il sistema software in esame si usano i seguenti valori di fattori:

T_1 (3), T_2 (4), T_3 (5), T_4 (2), T_5 (1), T_6 (4), T_7 (5), T_8 (4), T_9 (4), T_{10} (3), T_{11} (3), T_{12} (3), T_{13} (3).

Il totale di tutti i valori calcolati è il fattore tecnico (TF), calcolato come segue:

$$TF = 3 * 2 + 4 * 1 + 5 * 1 + 2 * 1 + 1 * 1 + 4 * 0.5 + 5 * 0.5 + 4 * 2 + 4 * 1 + 3 * 1 + 3 * 1 + 3 * 1 = 42.5 \quad (2.3)$$

Il TF viene quindi utilizzato per calcolare il TCF con la seguente formula:

$$TCF = 0.6 + \left(\frac{TF}{100} \right) = 0.6 + \left(\frac{42.5}{100} \right) = 0.6 + 0.425 = 1.025 \quad (2.4)$$

2.14.4 Environmental Complexity Factor (ECF)

L'ECF è un altro fattore applicato alla dimensione stimata del software al fine di tenere conto delle considerazioni ambientali del sistema. È determinato assegnando un punteggio compreso tra 0 (nessuna esperienza) e 5 (esperienza) a ciascuno degli 8 fattori ambientali elencati nella tabella seguente. Questo punteggio viene quindi moltiplicato per il valore ponderato definito per ciascun fattore.

Valori di riferimento:

- E_1 - Familiarità con il processo di sviluppo utilizzato, 1.5.

- $E2$ - Esperienza applicativa, 0.5.
- $E3$ - Esperienza di team orientata agli oggetti, 1.0.
- $E4$ - Capacità di lead analyst, 0.5.
- $E5$ - Motivazione del team, 1.0.
- $E6$ - Stabilità dei requisiti, 2.0.
- $E7$ - Personale part-time, -1.0.
- $E8$ - Linguaggio di programmazione difficile, -1.0.

Per il sistema software in esame si usano i fattori: $E1$ (3) + $E2$ (4) + $E3$ (2) + $E4$ (3) + $E5$ (5) + $E6$ (4) + $E7$ (1) + $E8$ (3).

Il totale di tutti i valori calcolati è il fattore ambiente (EF), calcolato come segue:

$$\mathbf{EF} = 3 * 1.5 + 4 * 0.5 + 2 * 1 + 3 * 0.5 + 5 * 1 + 4 * 2 + 1 * (-1) + 3 * (-1) = 19 \quad (2.5)$$

L'EF viene quindi utilizzato per calcolare l'ECF con la seguente formula:

$$\mathbf{ECF} = 1.4 + (-0.03 \times 19) = 0.83 \quad (2.6)$$

2.14.5 Total Use Case Points (UCP)

Infine, una volta determinate le dimensioni del progetto non aggiustate (UUCW e UAW), e calcolati il fattore tecnico (TCF) e il fattore ambientale (ECF) si può calcolare l'UCP. L'UCP viene calcolato in base alla seguente formula:

$$\mathbf{UCP} = (UUCW + UAW) * TCF * ECF = (85 + 10) * 1.025 * 0.83 = 80.82 \quad (2.7)$$

Da quest'ultimo parametro è possibile valutare il numero di ore di lavoro totali considerando che un singolo UC venga sviluppato in 8 ore mediamente:

$$\mathbf{Th} = UCP * 8 = 80.82 * 8 = 646.56h \quad (2.8)$$

In definitiva, se si suppone che un membro del progetto lavori per circa 60 ore a settimana (Wh), ed il team sia composto da 2 membri, essi lavoreranno per un totale di 120 ore a settimana (TWh). Th è il numero di ore di lavoro totali necessarie a terminare il progetto e TWh il numero di ore di lavoro a settimana del gruppo di lavoro (team), allora il numero di settimane necessarie per terminare il lavoro è dato da:

$$\frac{\mathbf{Th}}{\mathbf{TWh}} = \frac{646.56}{120} = 5.39 \quad (2.9)$$

Dunque si stima che occorreranno circa 5 settimane, ovvero circa 3 iterazioni, per sviluppare l'intero progetto.

2.15 System Context Diagram

Il diagramma seguente mostra ad alto livello come gli attori esterni (il paziente, il paziente registrato, la farmacia e l'admin di sistema) interagiscono con il sistema. In particolare, all'accesso del sistema, al **paziente** viene mostrato il form del **Modello di ML** per la predizione al COVID19, esso inserirà i sintomi e malattie che esso manifesta e il sistema restituirà l'esito.

Se l'**esito del test** è compreso tra il 50% e il 75% verrà indirizzato in maniera automatica al sistema di prenotazione di un tampone rapido alla farmacia più vicina. Se l'esito è maggiore del 75% allora verrà indirizzato in maniera automatica al sistema di prenotazione di un tampone molecolare alla farmacia più vicina. Effettuando la prenotazione, il paziente si registra al sistema, se già non lo è, diventando così un **paziente registrato**. La **farmacia** aggiornerà la disponibilità dei tamponi rapidi/molecolari e aggiungerà l'esito del tampone effettuato al paziente. Inoltre può visualizzare, modificare ed eliminare le prenotazioni. L'**admin di sistema** aggiornerà la lista delle farmacie aderenti al sistema. Inoltre aggiornerà il modello di ML per migliorare l'accuratezza dei risultati.

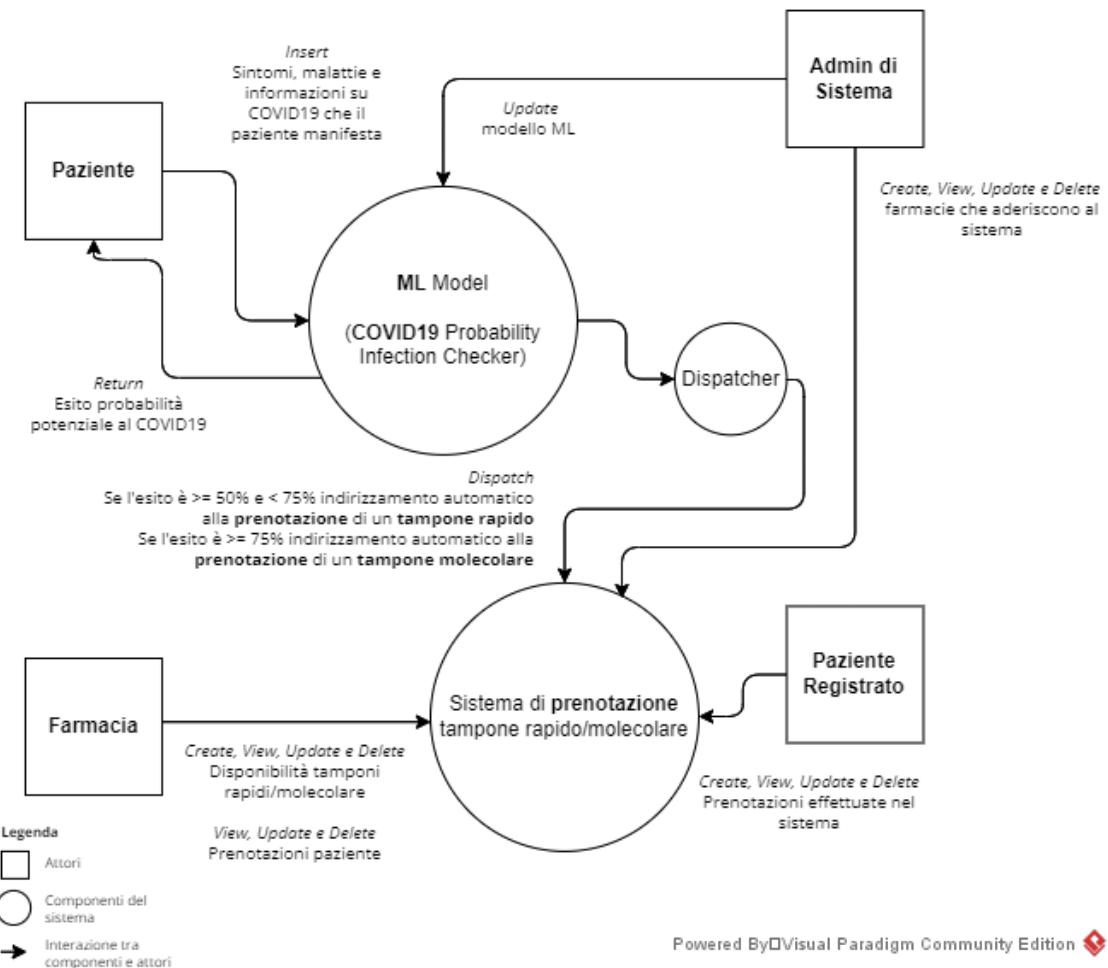


Figura 2.2: System Context Diagram

3 Processo di sviluppo

Si definiscono tutte le attività necessarie per lo sviluppo di un sistema software.

Contenuti

3.1	Framework di sviluppo	25
3.1.1	SCRUM	25
3.1.2	Suddivisione e sviluppo del progetto	25
3.1.3	Pratiche agili	30
3.2	Software per la condivisione del lavoro e per il supporto di SCRUM	36
3.2.1	Jira	36
3.2.2	Teams	39
3.2.3	Overleaf	39
3.2.4	GitHub	40
3.3	Software utilizzati per la progettazione e lo sviluppo	41
3.3.1	Visual Paradigm	41
3.3.2	Visual Studio Code	42
3.3.3	PythonAnyWhere	42
3.3.4	SQLite	43

3.1 Framework di sviluppo

Il framework che si seguirà per la realizzazione del prodotto software sarà quella che gestirà meglio il *cambiamento*. Si sa che il cambiamento è inevitabile in quanto ogni momento di sviluppo e progettazione può e possono esserci nuove idee, possibili errori, nuove tecnologie e dunque nuovi requisiti che possono obbligare a rielaborare e dunque rifare componenti del prodotto. Pertanto questo implica a costi detti di *rework* che possono inficiare nei tempi di realizzazione del prodotto e non solo. Dunque serve ridurre tali costi, come? O anticipando i possibili cambiamenti (**approccio predittivo**) oppure utilizzando il concetto di consegna *incrementale* (**approccio adattivo**), in cui gli incrementi di sistema vengono consegnati al cliente per ottenere commenti e per validarli. Ciò consente di evitare le modifiche inutili. Pertanto si seguirà il framework di sviluppo SCRUM dove al centro c'è il concetto di **agilità, incremento e sentimento**.

3.1.1 SCRUM

Si è scelto dunque di utilizzare un processo di sviluppo agile, seguendo le indicazioni fornite dal framework Scrum.

3.1.2 Suddivisione e sviluppo del progetto

Come previsto da SCRUM, durante la fasi di realizzazione del prodotto software in esame, a scopo puramente didattico, si divide il team nei seguenti ruoli (Tabella 3.1):

Ruolo	Descrizione	Assegnatari
Product Owner	Figura responsabile a massimizzare il valore del prodotto da realizzare. Esso decide cosa deve essere fatto, quali funzionalità deve avere il prodotto, le priorità con cui farle e quando rilasciarle. Esso deve avere un'ampia conoscenza dei bisogni dell'utilizzatore finale	Antonio Romano
Scrum Master	Figura responsabile di promuovere e sostenere Scrum, aiutando il team a comprendere appieno i principi su cui si fonda. Ha il compito di rimuovere gli impedimenti riscontrati dal team di sviluppo, gestendo le interazioni tra coloro al di fuori del team e il team stesso.	Giuseppe Riccio
Development Team	È un insieme di professionisti il cui compito è quello di consegnare incrementi realizzati del prodotto potenzialmente rilasciabili	Antonio Romano, Giuseppe Riccio

Tabella 3.1: Suddivisione ruoli SCRUM Team

Durante il processo di sviluppo si è cercato di rispettare quanto più fedelmente possibile gli eventi previsti da Scrum (Tab. 3.2):

In modo speciale, il Development Team, con il supporto di Jira Software (si mostrerà successivamente cos'è e come l'abbiamo utilizzato) ha seguito i report che generava in maniera automatica il software. I report seguiti, in particolar modo, sono:

- **Grafico burn-down:** si monitora e si gestisce il lavoro totale rimanente all'interno di uno sprint.
- **Grafico burn-up:** Visualizza il lavoro completato su uno sprint e confrontarlo con il suo ambito complessivo.
- **Diagramma cumulativo:** Mostra lo stato dei ticket di progetto nel tempo.
- **Diagramma di Gantt:** Consente a chi gestisce un progetto di avere sempre sotto controllo il lavoro svolto, ovvero le attività già eseguite da ogni componente del team coinvolto nel progetto stesso.

Evento SCRUM	Descrizione	TimeBox
Sprint	Evento container maggiore, contenente gli eventi che verranno descritti successivamente. Nello sprint viene creato un incremento di prodotto utilizzabile e potenzialmente rilasciabile.	1 settimana
Sprint Planning	In questo evento viene definito cosa deve essere realizzato e come deve essere svolto il lavoro. Inoltre viene definito lo Sprint Goal , ovvero l'obiettivo.	3 ore
Daily Scrum	Evento quotidiano in cui il team di sviluppo pianifica il lavoro da svolgere nelle prossime 24 ore e di assicurarsi quanto fatto fino a quel momento risolvendo, se necessario, possibili problemi.	15 minuti
Sprint Review	Evento rilasciato al termine di un sprint. Utile per revisionare l'incremento rilasciato. Viene illustrato agli stakeholder il lavoro al fine di ricevere feedback sull'incremento in modo da conoscere cosa realizzare nel prossimo incremento	2 ore
Sprint Retrospective	Evento in cui lo SCRUM team analizza quanto fatto nello sprint completo al fine di migliorare il proprio rendimento nello sprint successivo (Attraverso il RADAR).	2 ore

Tabella 3.2: Eventi SCRUM seguiti durante il processo di sviluppo

Come definito dall'analisi dei costi, sono stati realizzati 3 sprint della durata di 1 settimana, ciascuna delle quali sono stati sviluppati e prodotti gli artefatti previsti da SCRUM (Tabella 3.3):

Artefatto	Descrizione
Product Backlog	Elenco di user stories ordinato che definiscono tutto ciò che è necessario al prodotto software. Quando vengono definiti, questi ultimi non saranno mai terminati, in quanto durante lo sviluppo possono evolvere.
Sprint Backlog	Si dividono i product backlog in Sprint Backlog così da dividerli in incrementi
Incremento	Realizzazione dello sprint in termini di sviluppo dei componenti del prodotto software. È bene osservare che al termine di ogni sprint, l'incremento deve essere potenzialmente rilasciabile.

Tabella 3.3: Artefatti SCRUM realizzati durante il processo di sviluppo

Oltre agli artefatti richiesti da Scrum, per scopi didattici sono stati prodotti diversi diagrammi UML, cercando di mantenere comunque un approccio agile. Per questo motivo è stato prodotto solo lo stretto necessario per una piena comprensione del sistema, infatti sono stati sviluppati soltanto i casi d'uso con priorità elevata. Si mostrano gli eventi SCRUM seguiti dal team di sviluppo di questo progetto (Figura 3.1).

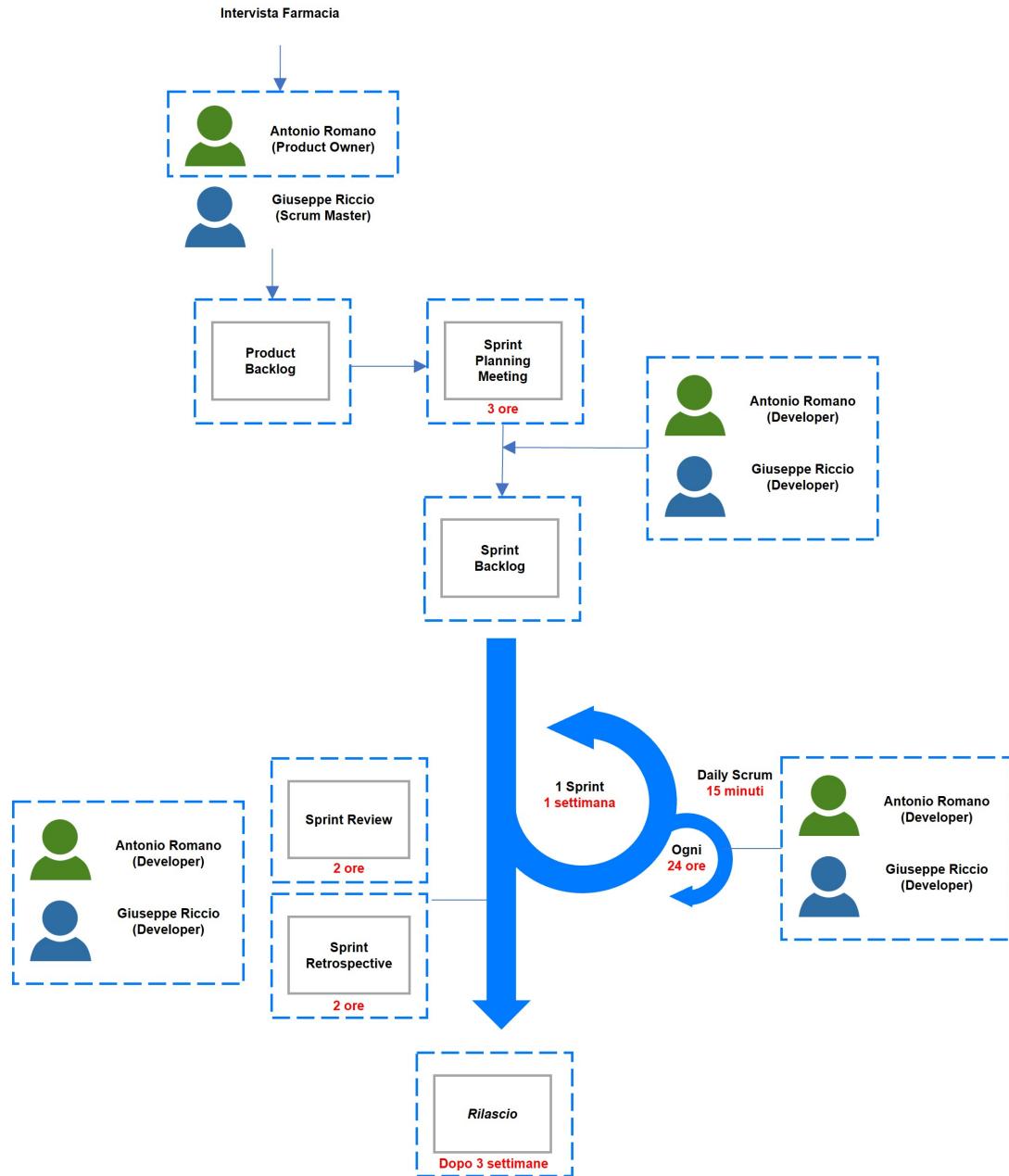


Figura 3.1: SCRUM - Step seguiti dal team del progetto

3.1.3 Pratiche agili

Le pratiche agili seguite sono:

- refactoring del codice;
- comunicazione stretta tra i membri del team;
- sviluppo iterativo;
- time-boxing, suddivisione del progetto in intervalli temporali ben definiti;
- semplicità di progettazione, modellazione, documentazione e codice;
- pair programming;

Si mostrano i grafici e le implementazioni delle storie utente negli sprint:

- SPRINT 1:

- Nello Sprint 1 si sono implementate le storie utente relative agli Epic **Gestione Sintomi** e **Gestione Farmacie**
- In particolare, abbiamo progettato e sviluppato le seguenti Storie Utente:
 - Inserimento Sintomi (2.1)
 - Creazione farmacie (2.17)
 - Ricerca farmacie (2.18)
 - Modifica farmacie (2.19)
 - Rimozione farmacie (2.20)
- I grafici Burn-up e Burn-down dello Sprint 1 relativi agli Epic realizzati sono i seguenti:

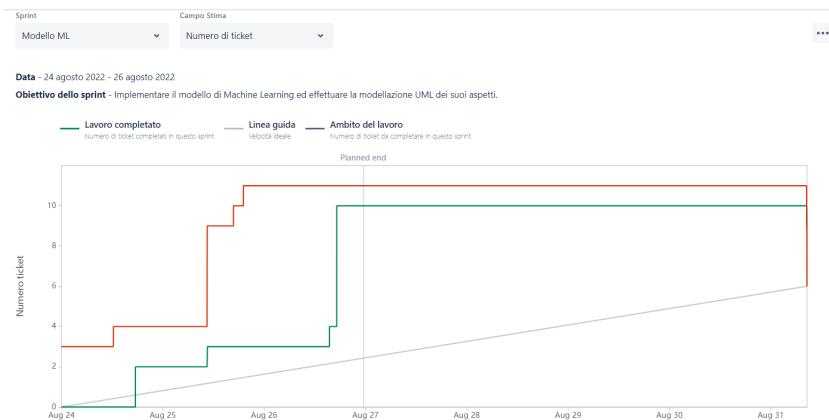


Figura 3.2: Grafico burn-up - Modello ML - Jira

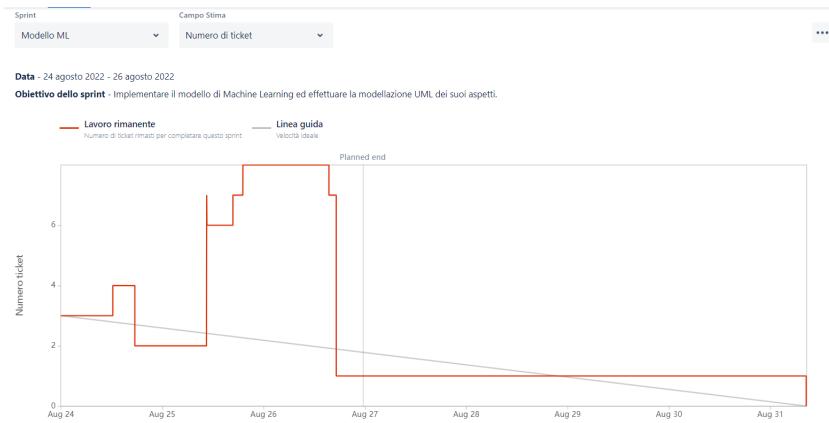


Figura 3.3: Grafico burn-down - Modello ML - Jira



Figura 3.4: Grafico burn-up - Gestione Farmacie Admin - Jira



Figura 3.5: Grafico burn-down - Gestione Farmacie Admin - Jira

● SPRINT 2:

- Nello Sprint 2 si sono implementate le storie utente relative agli Epic **Gestione Esiti Tamponi Farmacie e Gestione Ordini Farmacia**
- In particolare, abbiamo progettato e sviluppato le seguenti Storie Utente:
 - Richiesta lista prenotazioni (2.9)
 - Modifica prenotazioni (2.10)
 - Rimozione prenotazioni (2.11)
 - Creazione disponibilità tamponi (2.12)
 - Modifica disponibilità tamponi (2.13)
 - Rimozione disponibilità tamponi (2.14)
 - Aggiunta esito tampone (2.15)
- I grafici Burn-up e Burn-down dello Sprint 2 relativi agli Epic realizzati sono i seguenti:

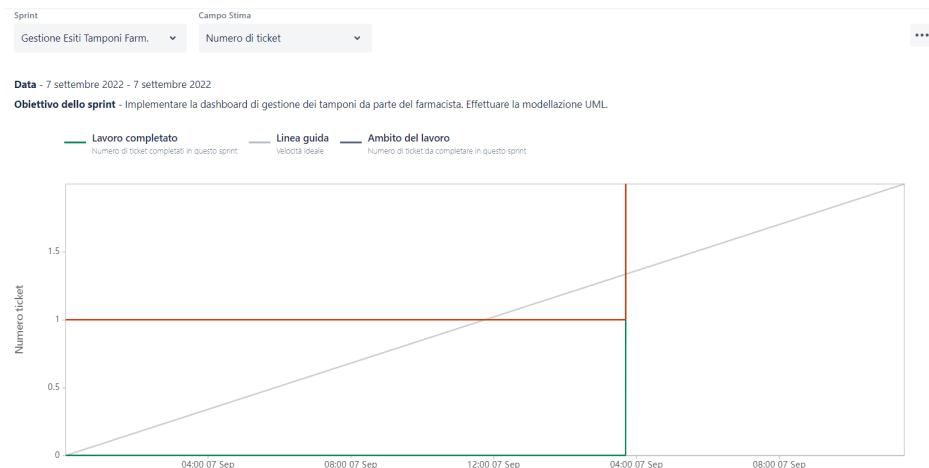


Figura 3.6: Grafico burn-up - Gestione Esiti Tamponi Farmacie - Jira

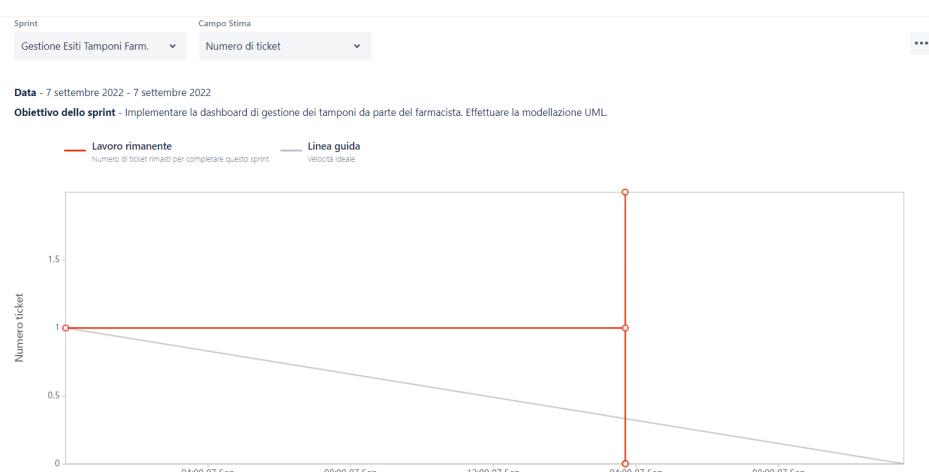


Figura 3.7: Grafico burn-down - Gestione Esiti Tamponi Farmacie - Jira

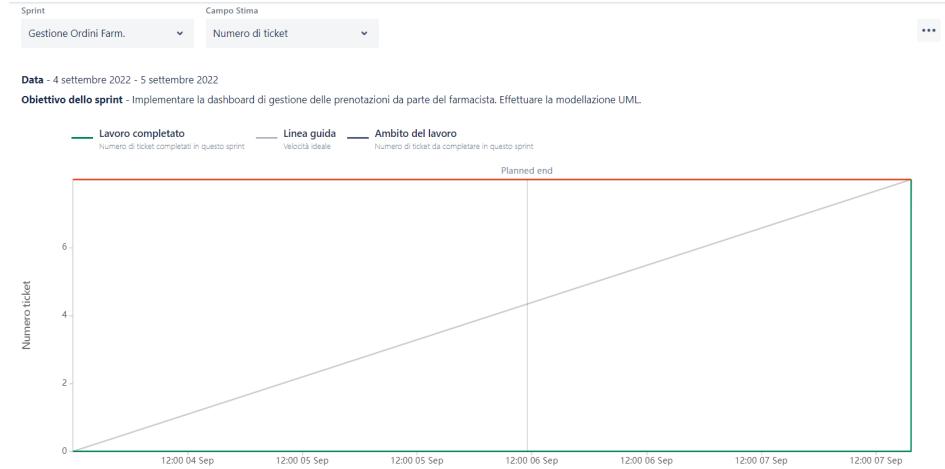


Figura 3.8: Grafico burn-up - Gestione Ordini Farmacia - Jira

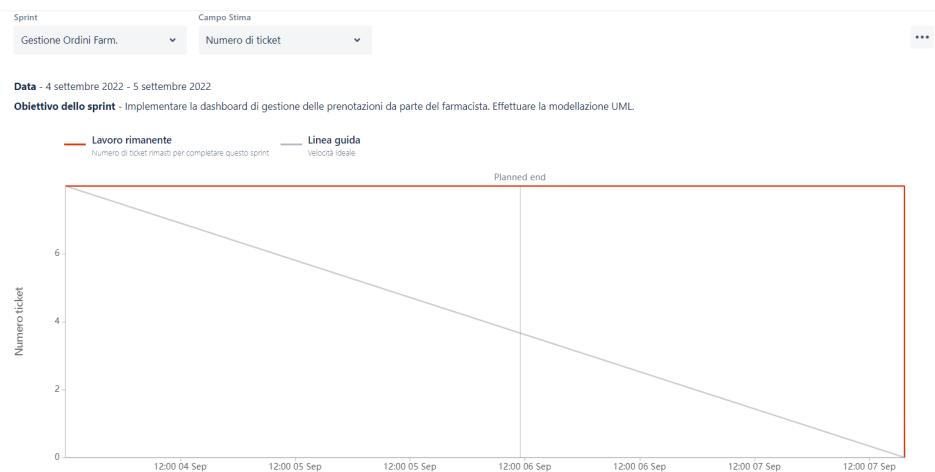


Figura 3.9: Grafico burn-down - Gestione Ordini Farmacia - Jira

● SPRINT 3:

- Nello Sprint 3 si sono implementate le storie utente relative all'Epic **Gestione Prenotazione Paziente** e si è terminato l'Epic **Gestione Ordini Farmacia**
- In particolare, abbiamo progettato e sviluppato le seguenti Storie Utente:
 - Verifica disponibilità farmacie (2.2)
 - Prenotazione tampone (2.3)
 - Modifica prenotazione (2.4)
 - Rimozione prenotazione (2.5)
 - Riepilogo prenotazione (2.6)
 - Richiesta esito tampone (2.7)
 - Verifica prenotazione (2.16)
- I grafici Burn-up e Burn-down dello Sprint 3 relativi agli Epic realizzati sono i seguenti:

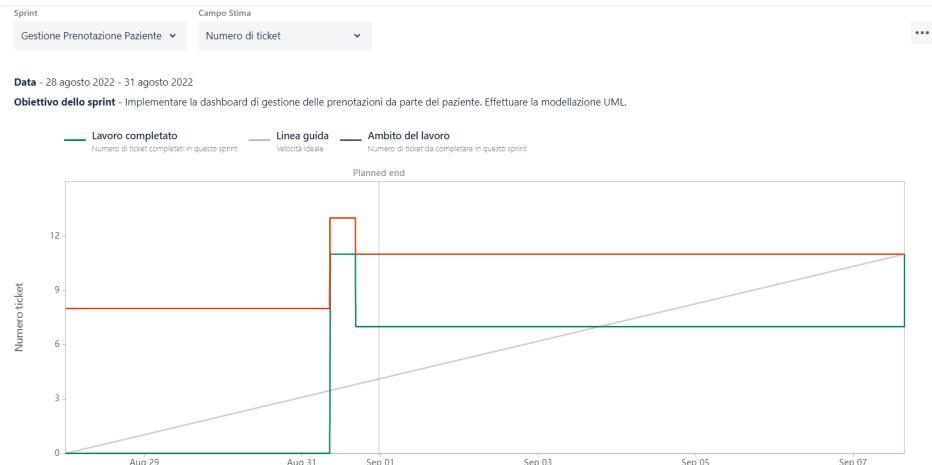


Figura 3.10: Grafico burn-up - Gestione Prenotazione Paziente - Jira

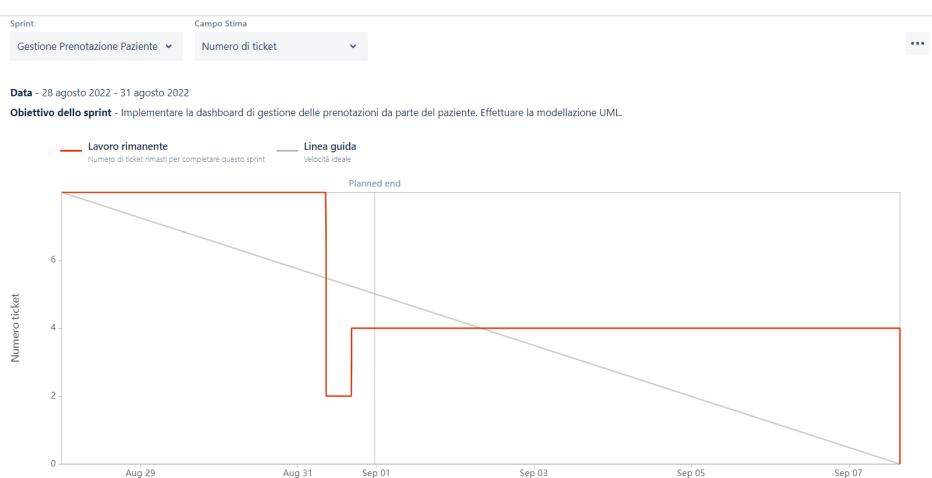


Figura 3.11: Grafico burn-down - Gestione Prenotazione Paziente - Jira

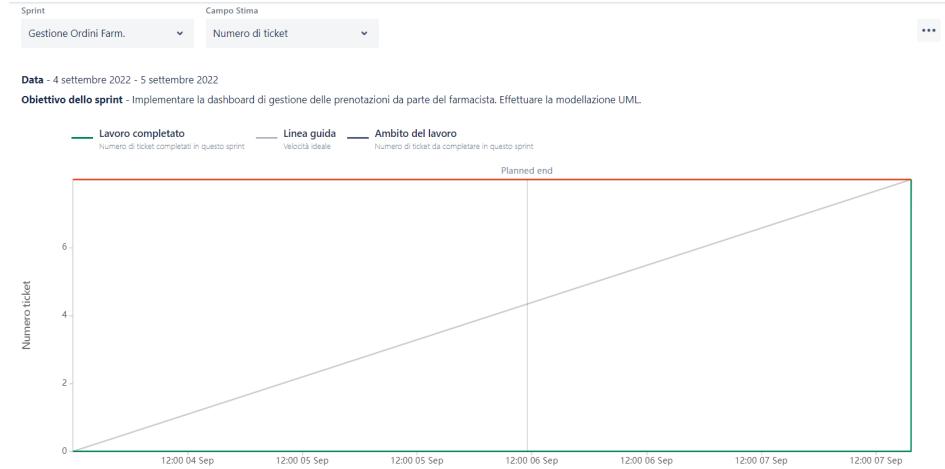


Figura 3.12: Grafico burn-up - Gestione Ordini Farmacia - Jira

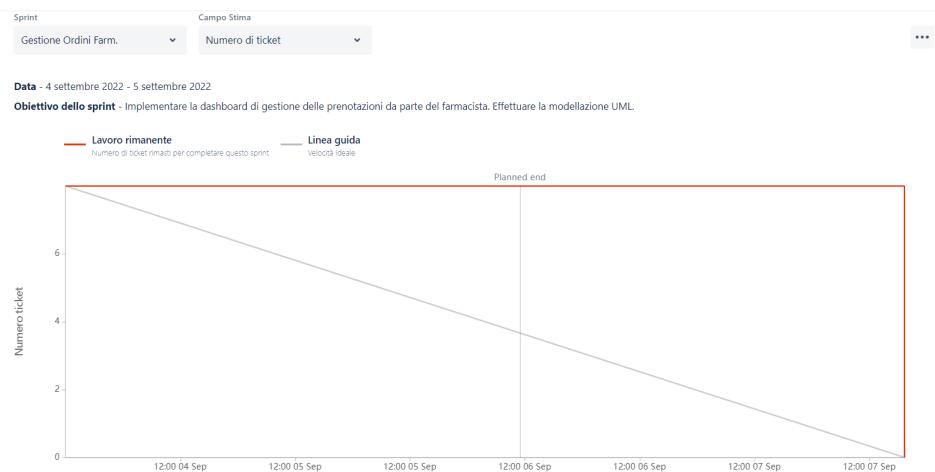


Figura 3.13: Grafico burn-down - Gestione Ordini Farmacia - Jira

3.2 Software per la condivisione del lavoro e per il supporto di SCRUM

Al fine di supportare gli step necessari di SCRUM, si è utilizzato **JIRA**, **Teams**, **Overleaf**, **GitHub**.

3.2.1 Jira

Come supporto all'intero processo di sviluppo è stato adottato la piattaforma Jira Software. In particolare tra le funzionalità offerte sono state utilizzate le seguenti:

- definizione di una storia utente;
- definizione di un epic;
- definizione del product backlog;
- definizione di una storia utente;
- assegnazione di priorità alle storie utente
- assegnazione di storypoints alle storie utente
- assegnazione di una storia utente ad un epic

Grazie all'uso di questo strumento è stato possibile organizzare il lavoro in maniera sistematica e flessibile. Infatti, ad ogni membro del team per conoscere la situazione dello Sprint attuale, gli bastava visionare la **Board del progetto** riportata in Figura 3.14.

Figura 3.14: Board - Jira

All'interno di tale board venivano inseriti per ogni Sprint, i ticket (che corrispondono alle Storie Utente del progetto) da sviluppare in quello Sprint (**Sprint backlog**), i ticket del progetto possono essere visualizzati anche singolarmente, Figura 3.15.

ticket						
		Cerca i ticket		Salva filtro		
Tipo	Chiave	Riepilogo	Assegnatario	Reporter	P	Stato
✓	STC-58	Verifica Prenotazione (QR code) - Progettazione web	(GR) Giuseppe Riccio	(GR) Giuseppe Riccio	=	COMPLETATA
✓	STC-57	Login Admin - Progettazione WebApp	(GR) Giuseppe Riccio	(GR) Giuseppe Riccio	=	COMPLETATA
✓	STC-56	State Machine Diagram Paziente	(GR) Giuseppe Riccio	(GR) Giuseppe Riccio	=	COMPLETATA
✓	STC-55	SDM System Domain Model	(A) antonio.romano45	(A) antonio.romano45	=	COMPLETATA
✓	STC-53	SSD e Caso d'uso di dettaglio CREAZIONE DISPONIBILITA' TAMPONI	(GR) Giuseppe Riccio	(A) antonio.romano45	=	COMPLETATA
✓	STC-52	SSD e Caso d'uso di dettaglio RICHIESTA LISTA PRENOTAZIONI	(A) antonio.romano45	(A) antonio.romano45	=	COMPLETATA
✓	STC-51	SSD e Caso d'uso di dettaglio RICHIESTA ESITO TAMPONE	(A) antonio.romano45	(A) antonio.romano45	=	COMPLETATA
✓	STC-50	SSD e Caso d'uso di dettaglio VERIFICA DISPONIBILITA' FARMACIE	(A) antonio.romano45	(A) antonio.romano45	=	COMPLETATA

Figura 3.15: Ticket - Jira

Inoltre, per visionare l'andamento del progetto generale e dei suoi Epic, è stato usato anche un **diagramma di Gantt**, che riportava sulle ordinate proprio gli Epic con il loro relativo stato, Figura 3.16.



Figura 3.16: Diagramma di Gantt - Jira

Infine, si è fatto uso anche di un **diagramma di flusso cumulativo** che ci permette di conoscere lo stato generale dei ticket del progetto con le relative fasi in cui possono trovarsi secondo quanto visto per la board, Figura 3.17.

Diagramma di flusso cumulativo

[Come leggere questo report](#)

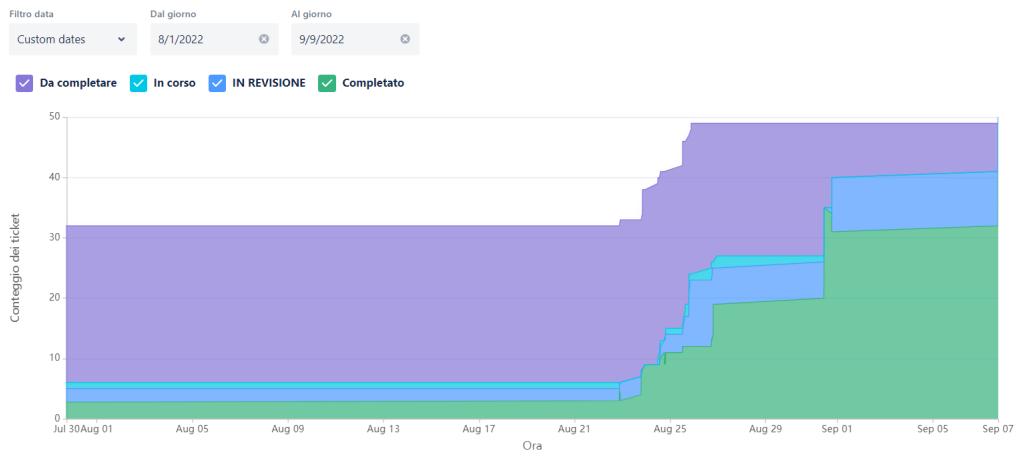


Figura 3.17: Diagramma di flusso cumulativo - Jira

3.2.2 Teams

Si è utilizzato Teams per lavorare online, ogni membro del team ha comunicato attraverso una chat o call vocali per la maggior parte degli appuntamenti del ciclo di sviluppo del progetto. Si è sfruttata la funzionalità di condivisione schermo, utile nei Daily Scrum, grazie alla quale ogni membro del gruppo mostra quanto fatto, Figura 3.18.

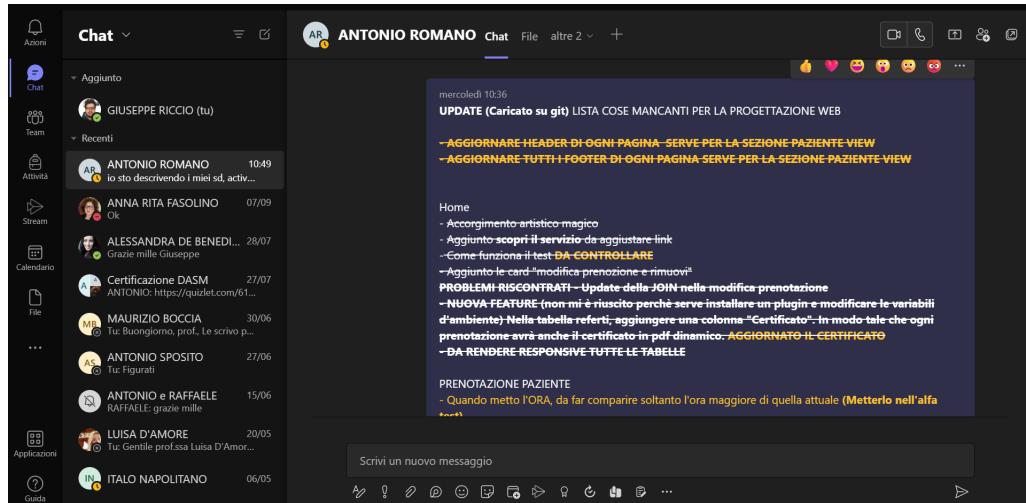


Figura 3.18: Teams

3.2.3 Overleaf

Si è utilizzato Overleaf per documentare in \LaTeX le fasi di realizzazione del sistema "SelfTestCOVID19" al fine di poter lavorare contemporaneamente su di un unico documento condiviso come dalla Figura 3.19.

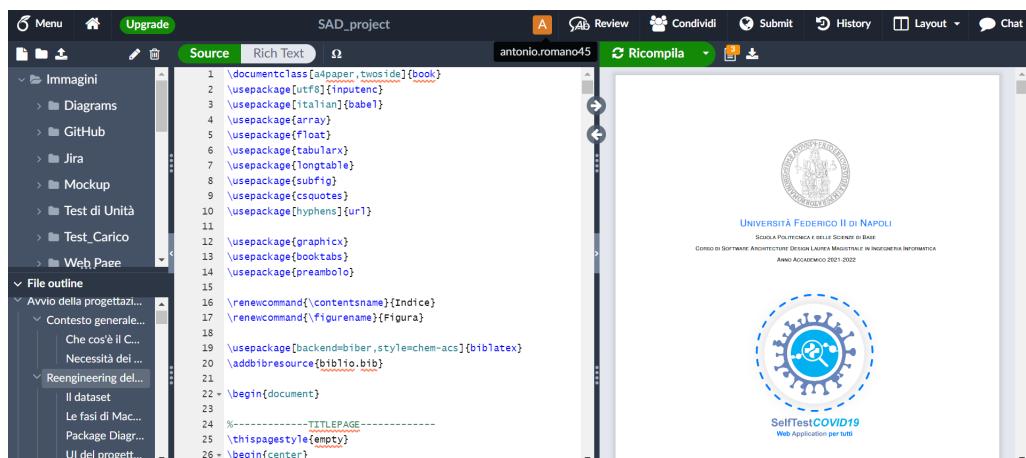


Figura 3.19: Overleaf

3.2.4 GitHub

Per supportare la collaborazione tra i diversi membri del team è stato utilizzato GitHub. In questo modo è stato possibile lavorare in parallelo sul codice senza avere particolari conflitti. Inoltre, alla fine di ciascuno Sprint si è generato un rilascio funzionante delle funzionalità implementate fino a quel momento.

Nelle seguenti Figure è possibile visualizzare quanto appena detto.

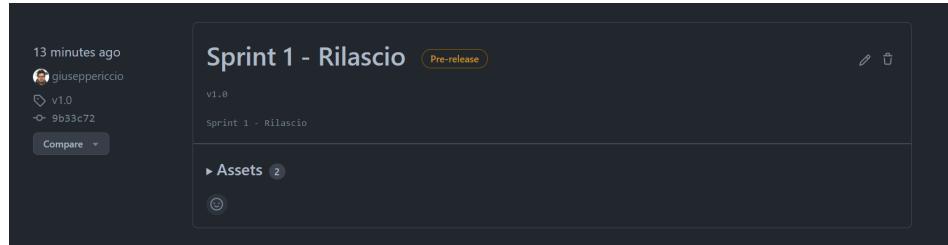


Figura 3.20: Sprint 1 - Rilascio

This screenshot displays two GitHub release pages side-by-side. The top one is for 'Sprint 3 - Rilascio' (v1.2) and the bottom one is for 'Sprint 2 - Rilascio' (v1.1). Both pages show a commit history with one entry each by 'giuseppericcio' (11 minutes ago), a 'Compare' button, and an 'Assets' section that is empty.

Figura 3.21: Sprint 2 e Sprint 3 - Rilascio

The screenshot shows the GitHub release page for 'Rilascio finale' (v1.3). This is the final release, indicated by the 'Latest' badge. The page includes a commit history with one entry by 'giuseppericcio' (10 minutes ago), a 'Compare' button, and an 'Assets' section containing two items: 'Source code (zip)' and 'Source code (tar.gz)', both uploaded 24 minutes ago.

Figura 3.22: Rilascio finale

Al seguente link è possibile accedere al:

Repository del progetto SelfTestCOVID19
<https://github.com/giuseppericcio/SelfTestCovid19>

3.3 Software utilizzati per la progettazione e lo sviluppo

I software utilizzati per la progettazione e lo sviluppo sono:

3.3.1 Visual Paradigm

Si sono progettati i diagrammi aderenti allo standard UML con Visual Paradigm. Grazie al concetto di "repository" messo a disposizione da Visual Paradigm, si è lavorato contemporaneamente utilizzando il comando "update" e "commit". Attraverso esso, si sono creati Sequence Diagram, Activity Diagram e Use Case Diagram per la documentazione relativa al sistema (Figura 2.7).

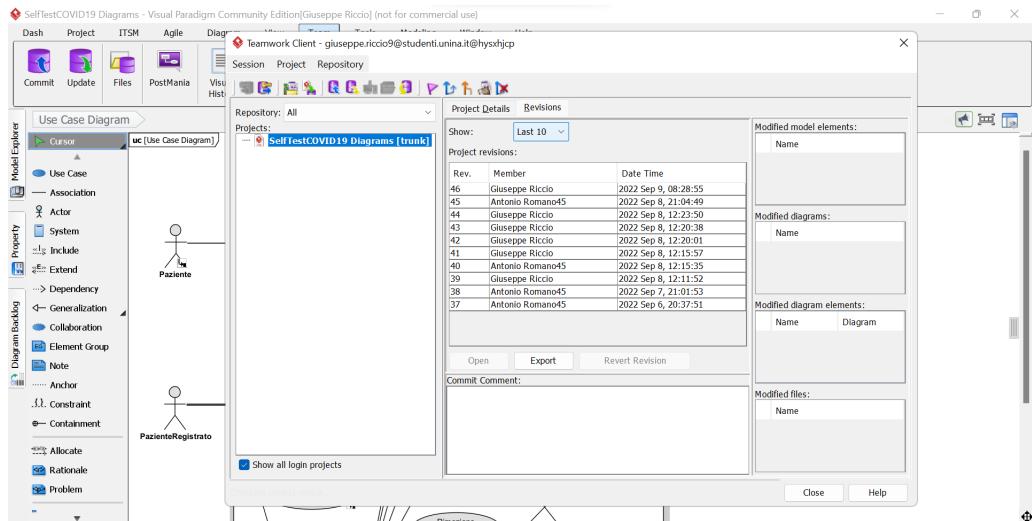


Figura 3.23: Visual Paradigm

3.3.2 Visual Studio Code

Per lo sviluppo dell'applicazione web, si è fatto uso della semplicità di Visual Studio Code.

The screenshot shows the Visual Studio Code interface. The left sidebar displays a project structure for 'SELFTESTCOVID19' containing files like main.py, model.pkl, myTraining.py, README.md, and requirements.txt. The main editor area shows the code for 'main.py'. Below the editor is a terminal window displaying the output of a development server:

```

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
Debugger PIN: 319-211-774
  
```

At the bottom, status bar information includes: Riga 174, colonna 45 - Spazi: 4 - UTF-8 - CRLF - Python 3.10.6 64-bit.

Figura 3.24: Visual Studio Code

3.3.3 PythonAnyWhere

PythonAnyWhere è un ambiente di sviluppo integrato online e web hosting. Si descrive dettagliatamente nel capitolo del **Rilascio**

The screenshot shows the PythonAnywhere dashboard. At the top, there are links for 'Send feedback', 'Forums', 'Help', 'Blog', 'Account', and 'Log out'. The main area is titled 'Dashboard' and shows a 'Welcome' message for 'selftestcovid19'. It features four main sections: 'Recent Consoles', 'Recent Files', 'Recent Notebooks', and 'All Web apps'. The 'Recent Consoles' section shows a 'Bash console 25412940'. The 'Recent Files' section lists files such as checkQRcode.html, prenota.html, GestioneprenotazioniController.py, requirements.txt, and MLModelController.py. The 'Recent Notebooks' section indicates 'You have no recent notebooks.' The 'All Web apps' section shows a single entry: 'www.selftestcovid19.it'. There are also buttons for 'Upgrade Account', 'Open Web tab', 'Add new (Python 3.9)', and 'Browse all files'.

Figura 3.25: PythonAnywhere

3.3.4 SQLite

Il DBMS utilizzato è SQLite, un DBMS leggero e adeguato per lo storage locale nel caso di applicazioni web. L'interfacciamento a tale database avviene attraverso delle apposite librerie fornite da Python.

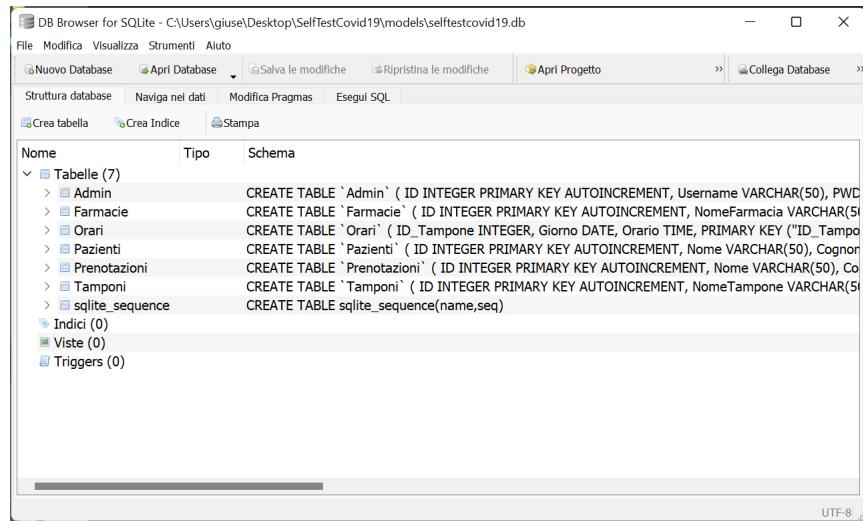


Figura 3.26: SQLite

ID	NomeFarmacia	Citta	CAP	Indirizzo	Email	PWD
1	1 Don Bosco	Napoli	80100	Via Roma 45	donbosco@farmacia.it	prova1
2	2 Mastrelia	Torre del Greco	80059	Via Nazionale 245	mastrelia@farmacia.it	prova2
3	3 Del Panda	Torre del Greco	80059	Via Molise 23	delpanda@farmacia.it	prova3
4	4 Leone	San Giovanni a Teduccio	80146	Via Corso Protopisani, 53	leonefarmacia@farmacia.it	farmacialeone

Figura 3.27: SQLite

4 Analisi dei Requisiti

In questa fase si ha l'obiettivo di chiarire e documentare le funzioni e i servizi che vengono offerti dal prodotto software in esame

Contenuti

4.1	Textual Analysis	44
4.2	Mappa Web App	45
4.3	Mockup e Design Concepts	46
4.4	Modello dei Casi d'Uso	48
4.4.1	Casi d'uso in dettaglio	49
4.5	System Sequence Diagram	57
4.5.1	System Sequence Diagram - Inserimento sintomi	57
4.5.2	System Sequence Diagram - Verifica Disponibilità Farmacie	58
4.5.3	System Sequence Diagram - Prenotazione tampone	59
4.5.4	System Sequence Diagram - Richiesta esito tampone	60
4.5.5	System Sequence Diagram - Richiesta lista prenotazioni	60
4.5.6	System Sequence Diagram - Creazione Disponibilità tamponi	61
4.5.7	System Sequence Diagram - Creazione farmacie	61
4.5.8	System Sequence Diagram - Ricerca farmacie	62
4.6	System Domain Model	63
4.7	State Machine Diagram degli stati Paziente	63

4.1 Textual Analysis

A partire dalle specifiche di alto livello 2.1, si può procedere ad una prima analisi testuale al fine di individuare gli attori principali, secondari ed i casi d'uso del sistema in esame. In particolare, si userà il colore **giallo** per individuare gli **Attori del sistema**, il **verde** per gli **Attori secondari**, l' **azzurro** per i casi d'uso ed infine, il **magenta** per un caso d'uso non implementato per il momento.

Si descrivono, ad alto livello, gli attori del sistema e i relativi obiettivi generali:

Paziente

- Il **Paziente** svolgerà il **SelfTest** per la valutazione dei propri sintomi. Il SelfTest è basato su un modello di Machine Learning (ML).
- Il **Paziente**, al fine di verificare la disponibilità di farmacie, inserisce il CAP e il Nome del comune di residenza per cercare le farmacie che hanno disponibilità del tampone rapido o molecolare. Sceglierà l'ora e il giorno nel quale prenotare.

Paziente Registrato

- In caso di disponibilità, il **Paziente** per effettuare la prenotazione, dovrà rilasciare dati anagrafici (nome, cognome, email, password, codice fiscale, telefono, accettazione delle normative sulla privacy). Ad ogni nuovo **Paziente Registrato** è assegnato un codice alfaniumerico d'identificazione. Riceverà via Mail la prenotazione effettuata con successo e se vuole, in secondo momento, modificare e/o cancellare la prenotazione. La **prenotazione che dovrà poi essere presentata tramite QR code in farmacia** per fare il tampone.
- Il **Paziente Registrato**, alla ricezione dell'esito del tampone, potrà aggiungere i suoi possibili sintomi.

Farmacia

- La **Farmacia** aggiornera la disponibilità dei tamponi indicando l'orario di apertura e di chiusura e i relativi orari di disponibilità per svolgere il tampone (la durata dell'analisi del tampone è circa di 15-20 minuti) al fine di permettere ai pazienti di effettuare autonomamente le loro prenotazioni online, o di visionare semplicemente gli orari ancora disponibili in agenda.
- La **Farmacia** potrà modificare e/o rimuovere le prenotazioni effettuate dai pazienti presso di essa.
- La **Farmacia** dopo aver processato il tampone, aggiungerà l'esito di quest'ultimo, ed il **paziente** riceverà una Mail che lo avvisa della disponibilità dell'esito del tampone.

Admin di Sistema

- L'admin di sistema aggiornera il database delle farmacie, inserendo, modificando ed eliminando nuove farmacie che aderiranno al sistema.
- L'admin di sistema aggiornera il modello di Machine Learning del Self Test. Per semplicità, verrà scelto un dataset per l'estrazione dei dati, verranno preparati i dati e poi addestrati. Successivamente, il modello di ML verrà automatizzato e distribuito secondo alcuni principi e tools ed infine il monitoraggio con l'obiettivo di ottimizzare il modello e renderlo sempre disponibile e aggiornato ad effettuare predizioni sempre più precise.

Figura 4.1: Textual Analysis

Si mostra il risultato ottenuto con Visual Paradigm (Figura 4.2)

No.	Candidate Class	Extracted Text	Type	Occurrence	Highlight
1	Paziente	Paziente	Actor	5	
2	Paziente Registrato	Paziente Registrato	Actor	3	
3	Admin di Sistema	Admin di Sistema	Actor	3	
4	Farmacia	Farmacia	Actor	5	
5	aggiorerà il modello di Machine Learning c aggiorerà il modello di Machine Learning del S	aggiorerà il modello di Machine Learning c aggiorerà il modello di Machine Learning del S	Use Case	1	
6	Riceverà via Mail	Riceverà via Mail	Actor	1	
7	prenotazione che dovrà poi essere presenta	prenotazione che dovrà poi essere presentata tr	Actor	1	
8	riceverà una Mail che lo avvisa della disponibit	riceverà una Mail che lo avvisa della disponibilità	Actor	1	
9	svolgerà il SelfTest	svolgerà il SelfTest	Use Case	1	
10	verificare la disponibilità di farmacie	verificare la disponibilità di farmacie	Use Case	1	
11	effettuare la prenotazione	effettuare la prenotazione	Use Case	1	
12	modificare e/o cancellare la prenotazione	modificare e/o cancellare la prenotazione	Use Case	1	
13	potrà aggiungere i suoi possibili sintomi	potrà aggiungere i suoi possibili sintomi	Use Case	1	
14	aggiorerà la disponibilità dei tamponi	aggiorerà la disponibilità dei tamponi	Use Case	1	
15	aggiungerà l'esito di quest'ultimo	aggiungerà l'esito di quest'ultimo	Use Case	1	
16	inserendo, modificando ed eliminando nuovi	inserendo, modificando ed eliminando nuove fa	Use Case	1	
17	modificare e/o rimuovere le prenotazioni	modificare e/o rimuovere le prenotazioni	Use Case	1	
18	ricezione dell'esito del tampone	ricezione dell'esito del tampone	Use Case	1	

Figura 4.2: Textual Analysis

4.2 Mappa Web App

Si illustra la mappa della web application per organizzare gerarchicamente tutte le sezioni di SelfTestCOVID19.

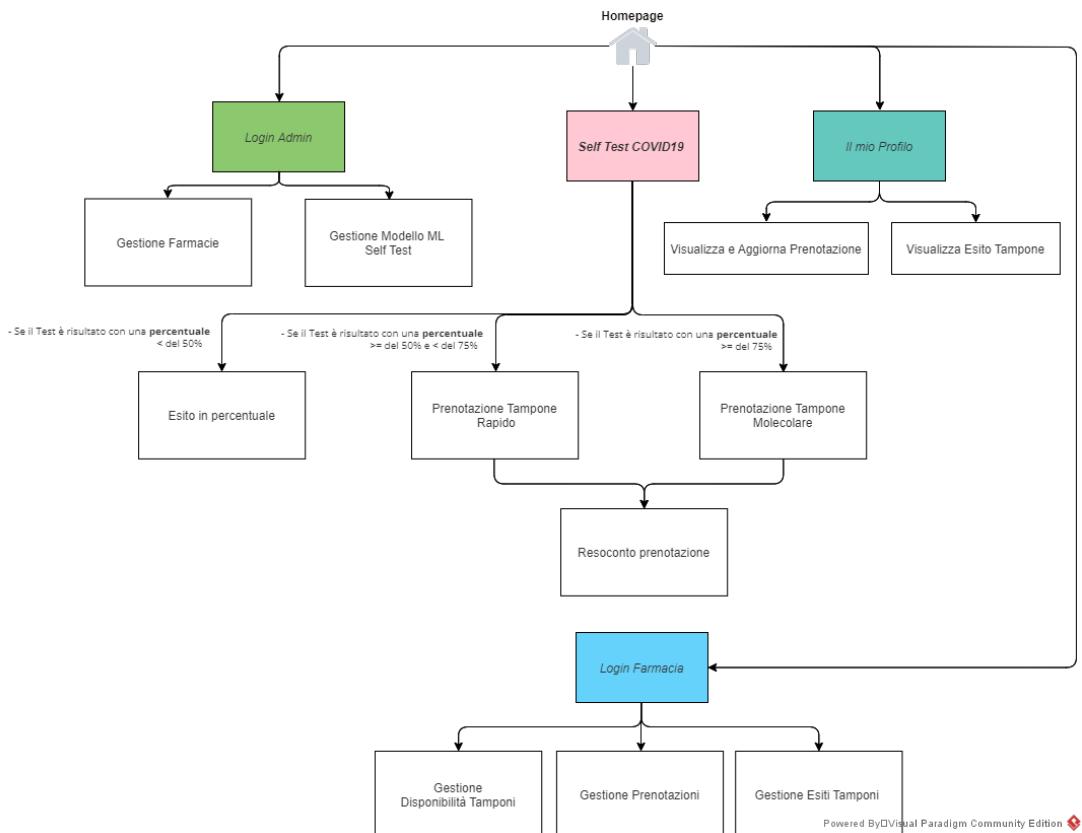


Figura 4.3: Mappa della Web App

4.3 Mockup e Design Concepts

Nel seguente paragrafo si illustrano le possibili User Interface (UI) degli attori. Nella Figura 4.4 si mostra il Mockup del **Paziente**, nella Figura 4.5 il Mockup della **Farmacia** e nella Figura 4.6 il Mockup dell'**Admin di Sistema**. Si noti che vengono delineate alcune mockup del prodotto software ma si sottolinea che a partire da queste idee si delineeranno le altre UI del progetto.

PazienteUI

Logo

Presentazione del sistema
e redirezione dell'utente al FORM SelfTes

Button for FORM

Informazioni di utilizzo del sistema

Footer

PazienteUI

Logo

Sintomo 1
Yes/No

Sintomo 2
Yes/No

⋮

Yes/No

Submit

Footer

PazienteUI

Logo

RISULTATO DEL TEST: XY,ZW %
Informazioni sui risultati
(il pulsante compare solo se il test è maggiore di)

Prenota tampone

Footer

PazienteUI

Logo

Inserisci la farmacia più vicina

Nome Farmacia

Città

CAP

Cerca Farmacie

Footer

PazienteUI

Logo

Farmacie disponibili

Farmacia 1
Farmacia 2
Farmacia 3
Farmacia 4
Farmacia 5
Farmacia 6

Prenota un esame

Footer

PazienteUI

Logo

Referti: Nome Cognome

Lista tamponi con esito

Carica sintomi

Footer

Figura 4.4: Mockup - PazienteUI e PazienteRegistratoUI

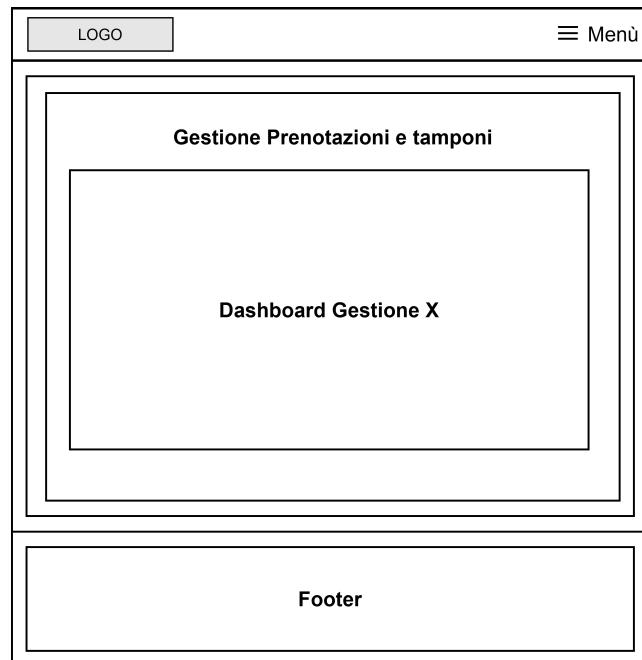


Figura 4.5: Mockup - FarmaciaUI

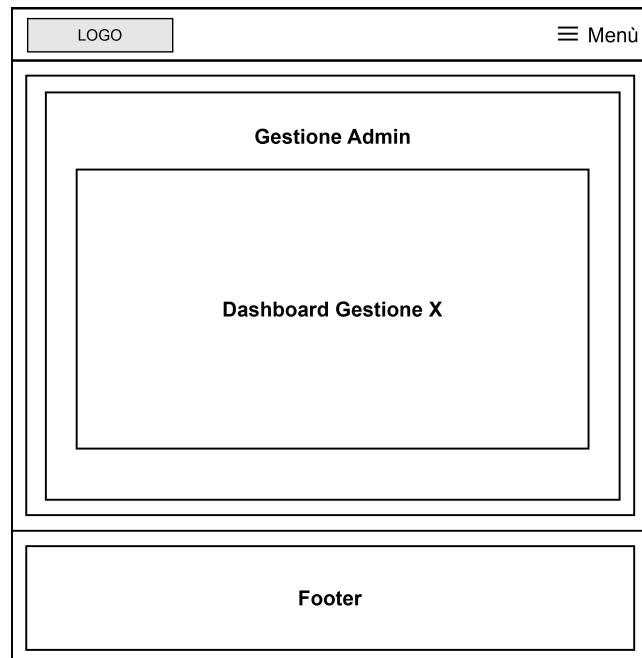


Figura 4.6: Mockup - AdminSistemaUI

4.4 Modello dei Casi d'Uso

Si determinano graficamente le funzionalità offerte dal sistema (caso d'uso), l'interazione con gli attori e come essi interagiscono con il sistema. Ogni caso d'uso si presenta come uno scenario con cui l'attore interagisce per il raggiungimento di un determinato obiettivo. Tra i casi d'uso rappresentati in figura 4.7 si è poi proceduto al raffinamento e all'implementazione di quelli con priorità (vedere il capitolo Storie Utente) massima o alta. Si noti la diversificazione del paziente negli attori *Paziente* e *PazienteRegistrato* in quanto si prevede una fase di registrazione, mentre sia per la *Farmacia* e sia per l'*Admin di sistema* è previsto a priori la fase di registrazione (login) nel sistema e pertanto non determinato nell'UCD.

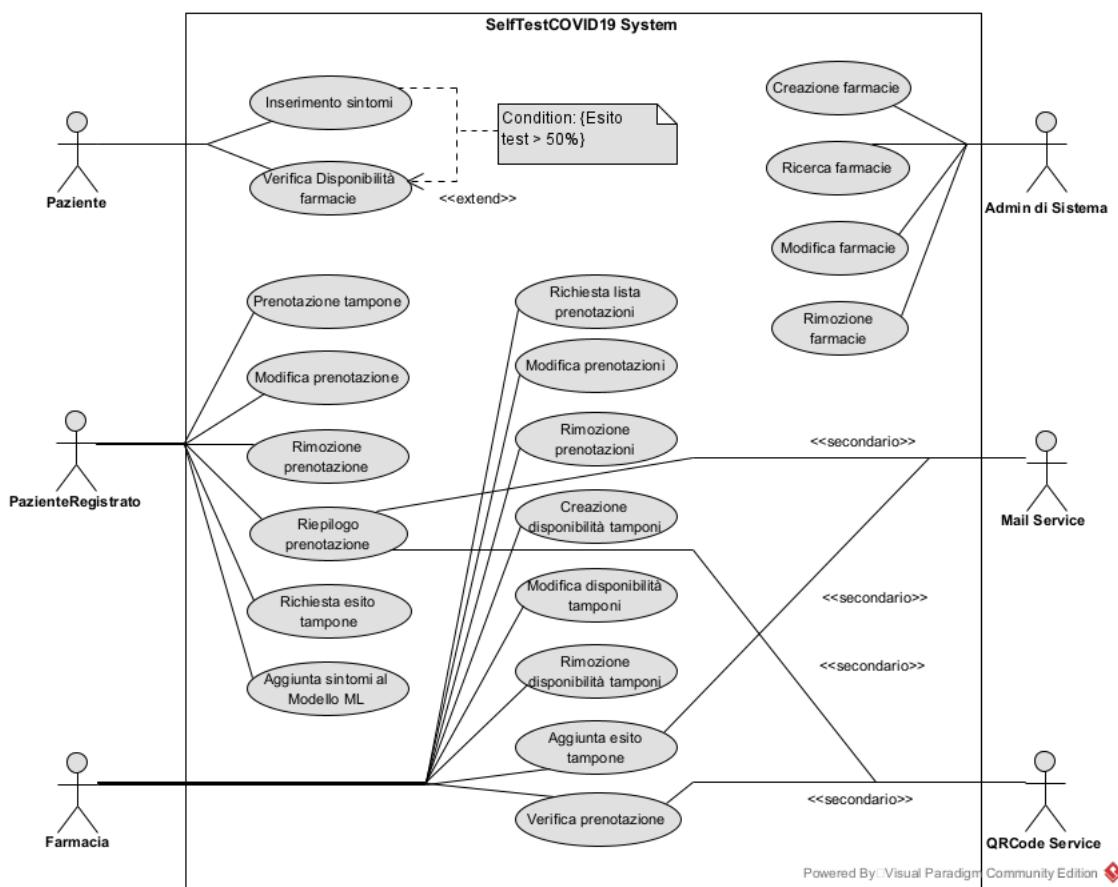


Figura 4.7: Use Case Diagram di alto livello

4.4.1 Casi d'uso in dettaglio

In seguito alla definizione dell'Use Case Diagram, si procede alla stesura dettagliata dei casi d'uso, si è scelto di effettuare quest'operazione solo per i requisiti considerati a priorità più alta, come indicato nelle Storie Utente. In particolare, si descriveranno i seguenti use case:

- Inserimento Sintomi
- Verifica disponibilità farmacie
- Prenotazione tampone
- Richiesta esito tampone
- Richiesta lista prenotazioni
- Creazione disponibilità tamponi
- Creazione farmacie
- Ricerca farmacie

Partiamo con la descrizione dei casi d'uso in dettaglio:

Inserimento sintomi

Livello	Obiettivo <i>Paziente</i>
Attore primario	Paziente
Parti interessate e interessi	Paziente vuole inserire i sintomi per il test
Garanzia di successo	Viene mostrato il risultato del test in percentuale
Scenario principale	<p>Fasi scenario:</p> <ul style="list-style-type: none"> ● 1. Il paziente accede alla mainPage della web app ● 2. Attraverso il form, il paziente inserisce i sintomi, malattie e informazioni sul COVID19 indicati ● 3. Il paziente clicca sul bottone "Calcola la probabilità di infezione al Sars-CoV2 COVID19"
Estensioni	Nessun estensione

Tabella 4.1: Use Case in dettaglio - Inserimento sintomi

Verifica disponibilità farmacie

Livello	Obiettivo <i>Paziente</i>
Attore primario	Paziente
Parti interessate e interessi	Paziente vuole verificare la disponibilità dei tamponi nella farmacia più vicina
Garanzia di successo	Viene mostrata la lista delle farmacie più vicine con annessa disponibilità del rispettivo tampone
Scenario principale	<p>Fasi scenario:</p> <ul style="list-style-type: none"> ● 1. Il paziente, dopo aver svolto il test, clicca sul pulsante "Prenota un tampone rapido/molecolare" ● 2. Attraverso un box, il paziente cerca la farmacia con Nome Farmacia, Città (e/o CAP) e clicca sul pulsante "Cerca farmacia" ● 3. Il sistema mostrerà i risultati (Nome Farmacia, Città, Disponibilità tampone rapido/molecolare(data, ora, prezzo)) ● 4. Il paziente, clicca sul pulsante "Prenota tampone" della farmacia selezionata ● 5. Il sistema mostra la pagina di prenotazione del tampone
Estensioni	<ul style="list-style-type: none"> ● 2a. La farmacia cercata non esiste nel sistema: <ul style="list-style-type: none"> – 1. Il sistema segnala l'errore – 2. Il sistema richiedere di rifare la ricerca al paziente

Tabella 4.2: Use Case in dettaglio - Verifica disponibilità farmacie

Prenotazione tampone

Livello	Obiettivo <i>Paziente Registrato</i>
Attore primario	<i>PazienteRegistrato</i>
Parti interessate e interessi	<i>PazienteRegistrato vuole prenotare il tampone nella farmacia selezionata</i>
Garanzia di successo	Prenotazione del tampone avvenuto con successo
Scenario principale	<p>Fasi scenario:</p> <ul style="list-style-type: none"> ● 1. Il paziente, dopo aver verificato la disponibilità delle farmacie, clicca sul pulsante "Prenota tampone" ● 2. Il sistema mostrerà il form di prenotazione del tampone con le seguenti informazioni: <i>SE il paziente NON è registrato</i> <ul style="list-style-type: none"> – Giorno – Ora – Nome – Cognome – E-mail – Password – Codice Fiscale – Telefono – Accettazione delle normative sulla privacy <i>SE il paziente è registrato</i> <ul style="list-style-type: none"> – Giorno – Ora – E-mail – Password ● 3. Il sistema mostra a video il riepilogo della prenotazione effettuata
Estensioni	<ul style="list-style-type: none"> ● 3a. SE le informazioni inserite NON sono corrette: <ul style="list-style-type: none"> – 1. Il sistema segnala l'errore – 2. Il sistema richiedere di reinserire le informazioni

Tabella 4.3: Use Case in dettaglio - Prenotazione tampone

Richiesta esito tampone

Livello	Obiettivo <i>Paziente Registrato</i>
Attore primario	PazienteRegistrato
Parti interessate e interessi	PazienteRegistrato vuole visualizzare l'esito del tampone effettuato in farmacia
Garanzia di successo	Visualizzazione esito tampone
Scenario principale	<p>Fasi scenario:</p> <ul style="list-style-type: none"> ● 1. Il paziente registrato riceve via Mail la disponibilità dell'esito del tampone ● 2. Il paziente accede nella sezione "Il mio profilo" e clicca sul bottone "Vai al referto" ● 3. Il sistema mostra l'esito del tampone/dei tamponi ● 4. Il paziente registrato clicca sul bottone "Certificato disponibile" ● 5. Il sistema restituisce il certificato in formato .PDF
Estensioni	<ul style="list-style-type: none"> ● 3a. SE l'esito del tampone NON è disponibile: <ul style="list-style-type: none"> – 1. Il sistema mostra l'esito "da effettuare" – 2. Il sistema mostra "Certificato NON disponibile"

Tabella 4.4: Use Case in dettaglio - Richiesta esito tampone

Richiesta lista prenotazioni

Livello	Obiettivo <i>Farmacia</i>
Attore primario	Farmacia
Parti interessate e interessi	Farmacia vuole visualizzare tutte le prenotazioni fissate dai pazienti presso la propria farmacia
Garanzia di successo	Visualizzazione della lista delle prenotazioni
Scenario principale	<p>Fasi scenario:</p> <ul style="list-style-type: none"> ● 1. La Farmacia, attraverso la dashboard, clicca sul pulsante "Vai alla lista prenotati della giornata" ● 2. Il sistema mostrerà la lista dei prenotati in modo da riconoscere il paziente in presenza.
Estensioni	Nessuna estensione

Tabella 4.5: Use Case in dettaglio - Richiesta lista prenotazioni

Creazione disponibilità tamponi

Livello	Obiettivo <i>Farmacia</i>
Attore primario	Farmacia
Parti interessate e interessi	Farmacia vuole creare la disponibilità di tamponi nel sistema
Garanzia di successo	La lista dei tamponi disponibili viene incrementata
Scenario principale	<p>Fasi scenario:</p> <ul style="list-style-type: none"> ● 1. La farmacia accede alla dashboard di gestione dei tamponi ● 2. Attraverso la dashboard, la farmacia clicca sul bottone "Crea disponibilità tamponi" ● 3. Il sistema mostra alla farmacia il form per l'inserimento delle informazioni relative ai tamponi ● 4. La farmacia attraverso il form, inserisce le info sulla disponibilità di tamponi (nometamponne, tipo, npezzi, data, orainizio, orafine, prezzo) ● 5. Il sistema aggiorna la lista dei tamponi disponibili e la mostra a video
Estensioni	<ul style="list-style-type: none"> ● 5a. SE le informazioni inserite NON sono corrette: <ul style="list-style-type: none"> – 1. Il sistema segnala l'errore – 2. Il sistema richiedere di reinserire le informazioni

Tabella 4.6: Use Case in dettaglio - Creazione disponibilità tamponi

Creazione farmacie

Livello	Obiettivo <i>Admin</i>
Attore primario	Admin di Sistema
Parti interessate e interessi	Admin di Sistema vuole inserire le farmacie nel sistema
Garanzia di successo	La lista delle farmacie disponibili viene aggiornata
Scenario principale	<p>Fasi scenario:</p> <ul style="list-style-type: none"> ● 1. L'admin accede alla dashboard di gestione delle farmacie ● 2. Attraverso la dashboard, l'admin clicca sul bottone "Crea farmacia" ● 3. L'admin inserisce le informazioni (NomeFarmacia, Città, CAP, Indirizzo, Email, PWD) nel form di creazione della farmacia ● 4. Il sistema aggiornerà la lista delle farmacie disponibili ● 5. Il sistema conferma l'avvenuta creazione con un messaggio a video
Estensioni	<p>Scenari alternativi:</p> <ul style="list-style-type: none"> ● 3a. La farmacia esiste già nel sistema: <ul style="list-style-type: none"> – 1. Il sistema segnala l'errore – 2. Il sistema richiede di reinserire le informazioni

Tabella 4.7: Use Case in dettaglio - Creazione farmacie

Ricerca farmacie

Livello	Obiettivo <i>Admin</i>
Attore primario	Admin di Sistema
Parti interessate e interessi	Admin di Sistema vuole ricercare le farmacie disponibili nel sistema
Garanzia di successo	La lista delle farmacie disponibili viene mostrata
Scenario principale	<p>Fasi scenario:</p> <ul style="list-style-type: none"> ● 1. L'admin accede alla dashboard di gestione delle farmacie ● 2. Attraverso la dashboard, l'admin clicca sul bottone "Ricerca farmacie" ● 3. L'admin ricerca le farmacie per Nome, Città e/o CAP inserendo le relative informazioni nel form ● 4. Il sistema ricerca le farmacie disponibili corrispondenti alle informazioni inserite ● 5. Il sistema mostra a video la lista delle farmacie trovate
Estensioni	<p>Scenari alternativi:</p> <ul style="list-style-type: none"> ● 5a. Il sistema non trova farmacie corrispondenti alle informazioni inserite <ul style="list-style-type: none"> – 1. Il sistema segnala l'errore – 2. Il sistema richiede di rifare la ricerca della farmacia

Tabella 4.8: Use Case in dettaglio - Ricerca farmacie

4.5 System Sequence Diagram

Il System Sequence Diagram descrive le relazioni che intercorrono, in termini di messaggi, tra gli Attori e il Sistema che si sta rappresentando. Un messaggio è un'informazione che viene scambiata tra due entità. Solitamente chi invia il messaggio, la parte attiva, è l'Attore. Il messaggio è sincrono, se l'emittente rimane in attesa di una risposta, o asincrono, nel caso l'emittente non aspetti la risposta e questa può arrivare in un secondo momento. Il messaggio che viene generato in risposta ad un precedente messaggio, al quale si riferisce anche come contenuto informativo, è detto messaggio di risposta. Un messaggio, in cui il ricevente è nello stesso tempo l'emittente, è detto ricorsivo. Dalla versione 2 dell'UML è stata introdotta la possibilità di indicare nello stesso diagramma anche delle sequenze alternative.

4.5.1 System Sequence Diagram - Inserimento sintomi

Si riporta l'interazione tra il Paziente e il Sistema. L'obiettivo è effettuare il test. Segue dunque la sequenza mostrata in Figura 4.8. Non sono presenti sequenze alternative perché non c'è ne la necessità.

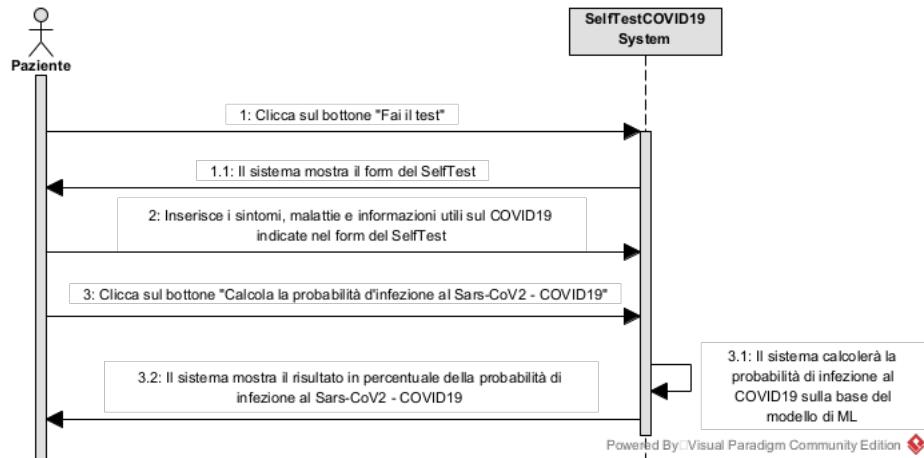


Figura 4.8: System Sequence Diagram - Inserimento sintomi

4.5.2 System Sequence Diagram - Verifica Disponibilità Farmacie

Il Paziente interagisce con il Sistema per verificare quali sono le farmacie nella sua città che hanno dei tamponi (rapido o molecolare) disponibili per prenotarne uno, o semplicemente per verificare gli orari della farmacia.

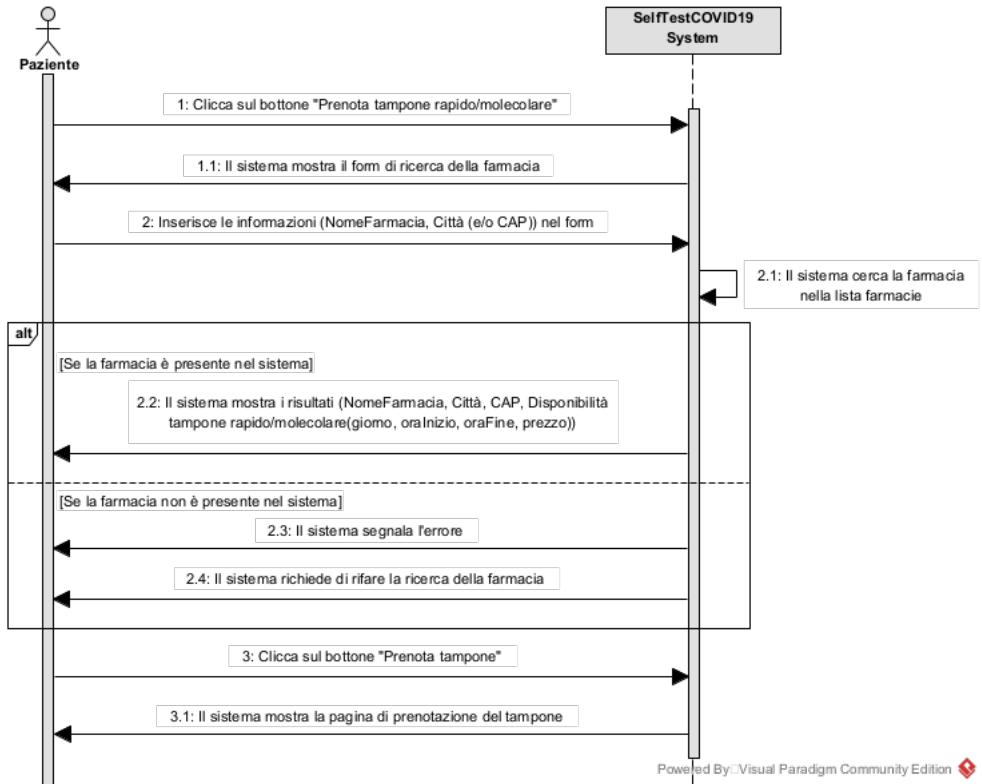


Figura 4.9: System Sequence Diagram - Verifica Disponibilità Farmacie

4.5.3 System Sequence Diagram - Prenotazione tampone

Il Paziente per prenotare il tampone selezionato deve inserire le proprie generalità al fine di permettere la sua identificazione all'arrivo in farmacia, nonché per permettergli di recuperare le informazioni sulle prenotazioni effettuate sul Sistema.

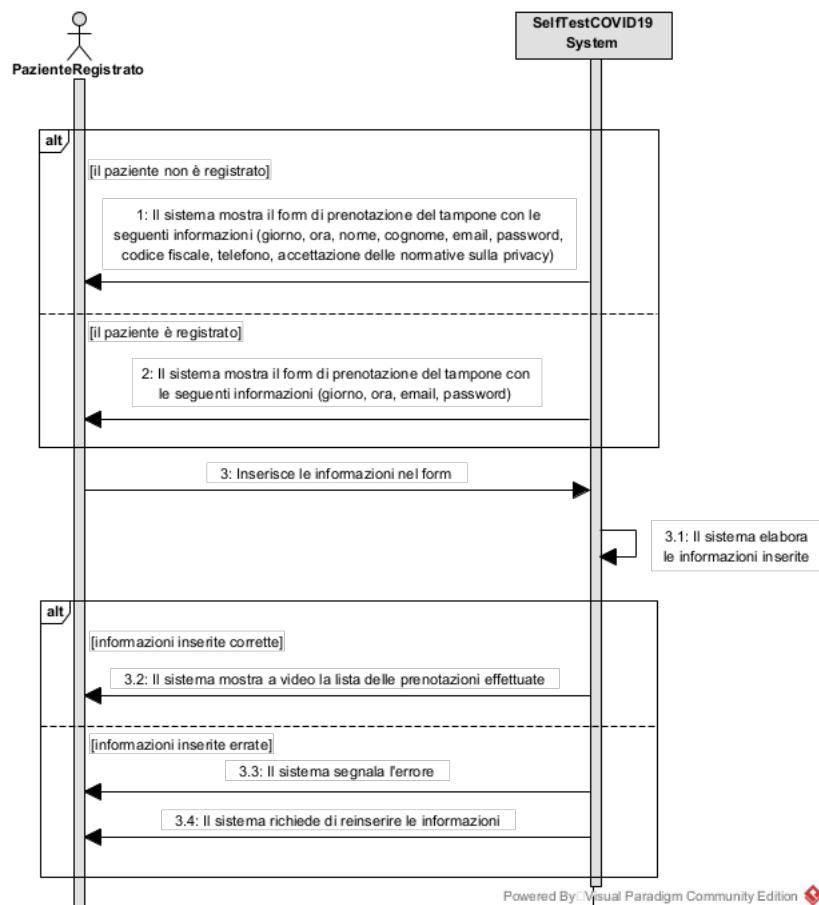


Figura 4.10: System Sequence Diagram - Prenotazione tampone

4.5.4 System Sequence Diagram - Richiesta esito tampone

Il Paziente Registrato quando il tampone è stato processato e la farmacia ha aggiunto l'esito nel Sistema, riceve una Mail attraverso la quale accede a "Il mio profilo" e può scaricare/visualizzare l'esito del tampone.

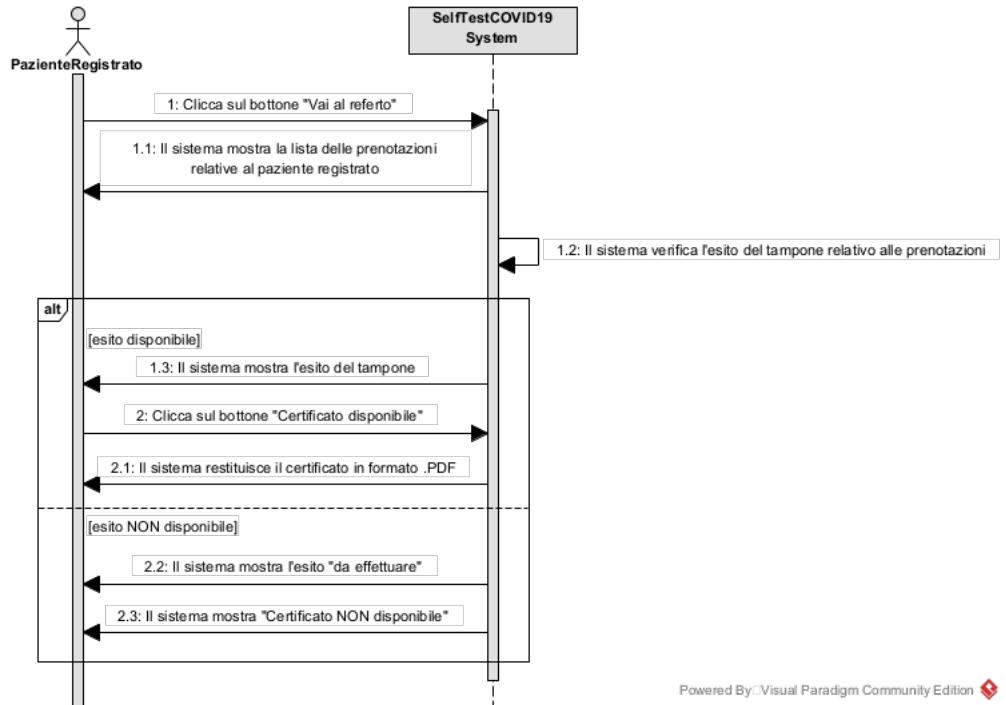


Figura 4.11: System Sequence Diagram - Richiesta esito tampone

4.5.5 System Sequence Diagram - Richiesta lista prenotazioni

La Farmacia può accedere alle proprie prenotazioni previste per la giornata attraverso la dashboard messa a disposizione dal Sistema.

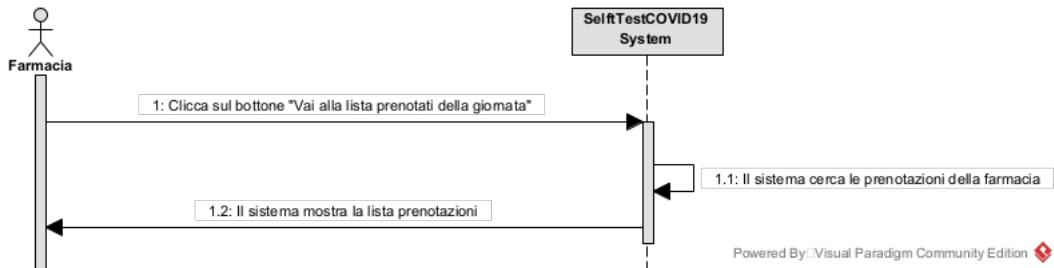


Figura 4.12: System Sequence Diagram - Richiesta lista prenotazioni

4.5.6 System Sequence Diagram - Creazione Disponibilità tamponi

La Farmacia in maniera periodica crea la disponibilità di nuovi tamponi nel Sistema, in maniera tale che i pazienti quando effettueranno la verifica della disponibilità possano visualizzarli.

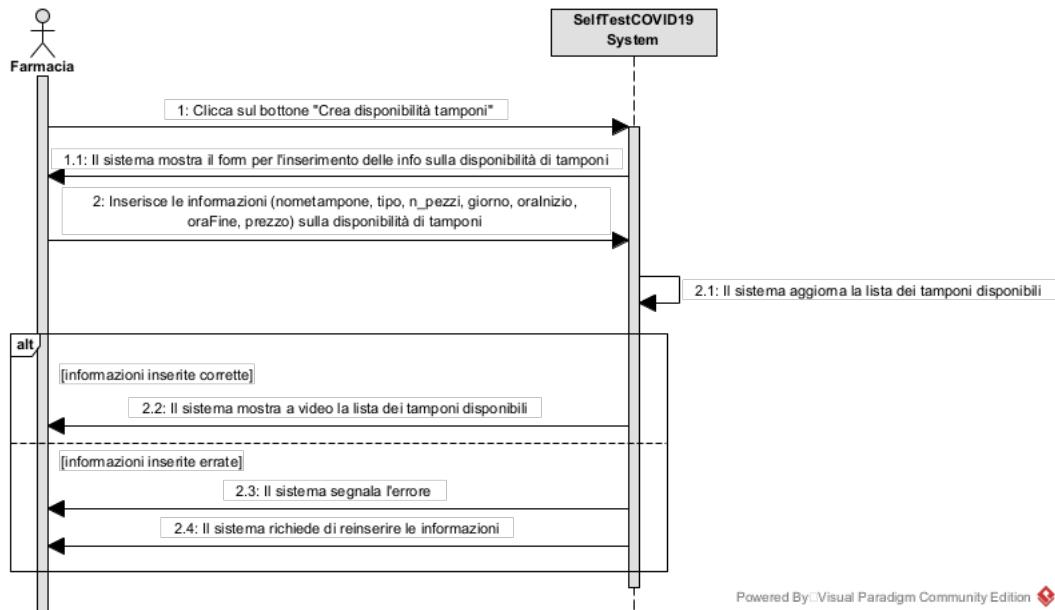


Figura 4.13: System Sequence Diagram - Creazione Disponibilità tamponi

4.5.7 System Sequence Diagram - Creazione farmacie

L'Admin quando riceve delle richieste da nuove farmacie che desiderano aderire alla piattaforma, deve provvedere alla loro aggiunta al Sistema tramite la propria dashboard.

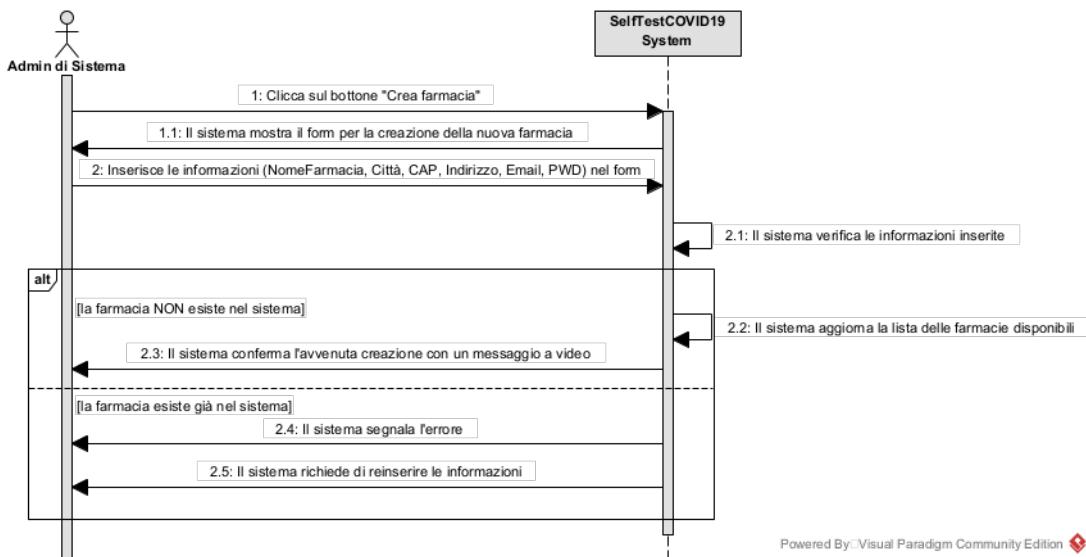


Figura 4.14: System Sequence Diagram - Creazione farmacie

4.5.8 System Sequence Diagram - Ricerca farmacie

L'Admin deve, inoltre, avere la possibilità di ricercare le farmacie presenti sul Sistema per effettuare eventuali verifiche quando lo ritiene opportuno.



Figura 4.15: System Sequence Diagram - Ricerca farmacie

4.6 System Domain Model

Il System Domain Model (SDM) è un modello concettuale che astrae e rappresenta gli elementi e/o entità del sistema software che si vuole realizzare (dominio), descrivendo le relazioni che intercorrono tra di essi, i possibili ruoli e attributi. Tale modello dà una descrizione della struttura (quindi una descrizione statica) del sistema calato nel dominio operativo.

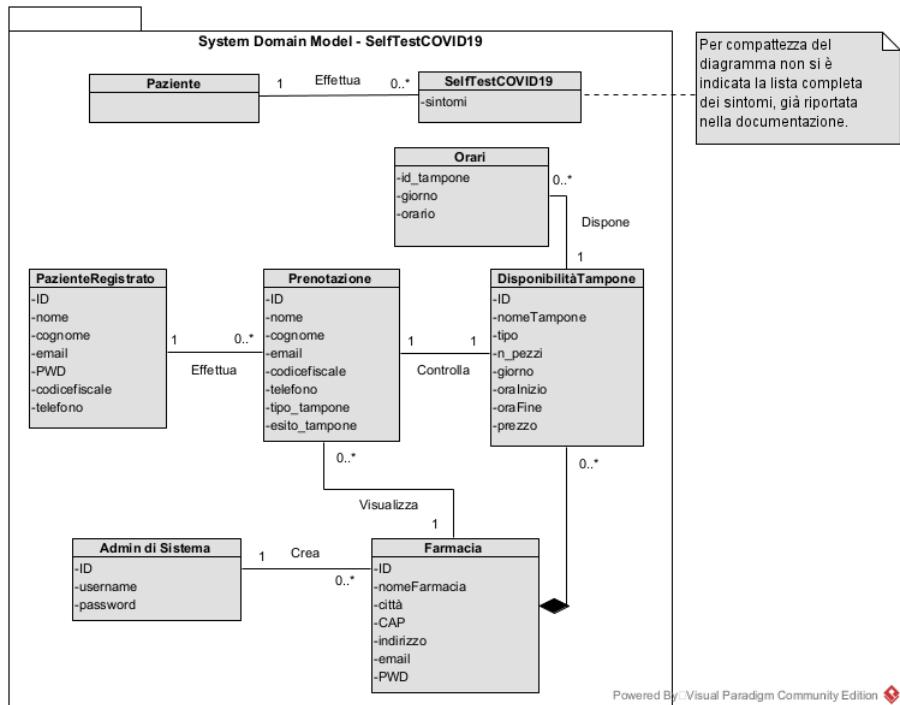


Figura 4.16: System Domain Model

4.7 State Machine Diagram degli stati Paziente

Si mostra un State Machine Diagram in quanto l'attore *Paziente*, dopo aver svolto la prima volta la prenotazione, diventa *PazienteRegistrato* (si è registrato nel sistema).

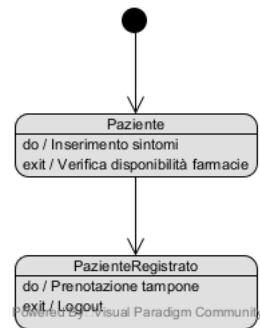


Figura 4.17: State Machine Diagram - Paziente

5 Architettura e progettazione del software

In questa fase si determina l'architettura del prodotto software evidenziando i componenti che costituiscono il sistema e le loro interazioni.

Contenuti

5.1 Scelte progettuali	64
5.2 Pattern Architetturale - Model View Controller (MVC) and Routes	66
5.2.1 Routes	66
5.2.2 Model	67
5.2.3 Vista	67
5.2.4 Controller	67
5.3 Stile Architetturale - Microservizi	67
5.4 REST	68
5.4.1 Client - Server in REST	69
5.4.2 Stateless	69
5.4.3 Cacheable	70
5.4.4 Interfaccia uniforme	70
5.4.5 Sistema stratificato	70
5.5 Vista componenti e connettori	71
5.6 Sequence Diagram di progettazione	73
5.6.1 Sequence Diagram di progettazione - Inserimento sintomi	73
5.6.2 Sequence Diagram di progettazione - Verifica disponibilità farmacie	74
5.6.3 Sequence Diagram di progettazione - Creazione disponibilità tampone	75
5.6.4 Sequence Diagram di progettazione - Prenotazione tampone	76
5.7 Activity Diagram	77
5.7.1 Activity Diagram - Inserimento sintomi	77
5.7.2 Activity Diagram - Verifica disponibilità farmacie	78
5.7.3 Activity Diagram - Creazione disponibilità tampone	78
5.7.4 Activity Diagram - Prenotazione tampone	79
5.8 Communication Diagram	79
5.8.1 Communication Diagram - Inserimento sintomi	79
5.8.2 Communication Diagram - Verifica disponibilità farmacie	80
5.8.3 Communication Diagram - Creazione disponibilità tampone	81
5.8.4 Communication Diagram - Prenotazione tampone	81
5.9 Server MVC Class Diagram	83

5.1 Scelte progettuali

Per definire le scelte progettuali del prodotto software in questione, si delineano gli stili e i pattern che definiranno il progetto. A partire dal seguente grafico (**Figura 5.1**) le scelte saranno:

- **Pattern Architetturale:** *Model View and Controller*, in quanto c'è la necessità di visualizzare operazioni e dati tramite interfaccia grafica (GUI) controllandoli opportunamente. L'obiettivo è avere una separazione netta dei componenti che gestiscono i dati e le view.

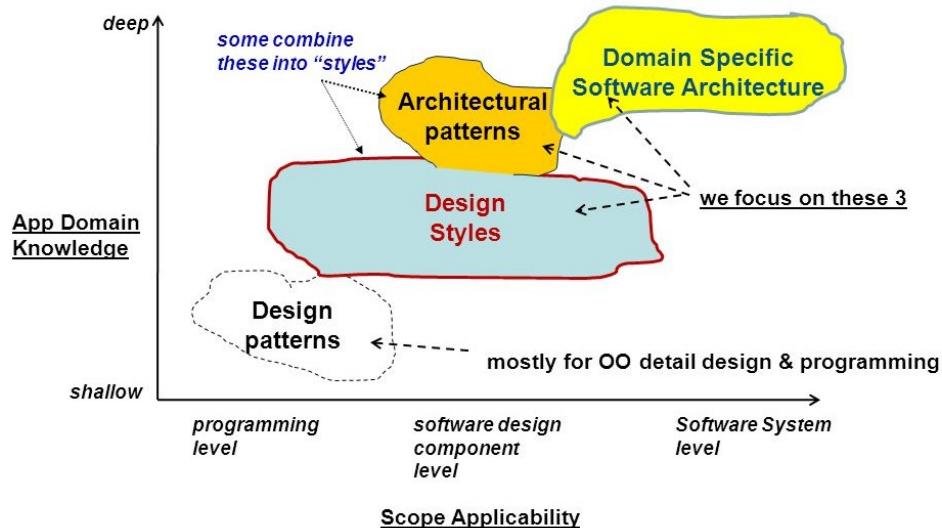


Figura 5.1: Application domain knowledge and Scope

- **Stile Architetturale:** *Microservizi*, in quanto c'è la necessità di dover garantire un certo livello di modularità per permettere un'agilità durante la fase di progettazione e sviluppo offrendo la possibilità di scalare l'architettura ad un livello di granularità più fine.
- **Pattern di Design:** *GRASP, Proxy e Observer* per descrivere le responsabilità dei componenti che compongono il sistema software a livello implementativo.

Si vuole, come già scritto nei precedenti paragrafi, realizzare una Web Application. La scelta di realizzare una Web Application è dovuta dal fatto che c'è la completa indipendenza dal sistema nel quale viene eseguita. Questo vantaggio quindi semplifica di molto il lavoro degli sviluppatori che non devono sviluppare un Client per uno specifico computer o sistema operativo; chiunque abbia accesso ad Internet e disponga di un browser compatibile, può utilizzare la Web-App senza bisogno di alcuna installazione. Il loro obiettivo è quello di adattarsi al meglio con la piattaforma sulla quale vengono lanciate. Ulteriori vantaggi di questa soluzione sono:

- aggiornamento automatico e indipendente dalle scelte degli utenti, che quindi eseguiranno tutta la medesima, e anche ultima, versione;
- bassi costi di realizzazione e manutenzione;
- alta scalabilità;
- modularità
- fase di testing poco complessa;
- facilità di manutenzione;
- rari problemi di compatibilità se non a livello di posizionamento di bottoni, form e icone tra i vari browser esistenti.

5.2 Pattern Architetturale - Model View Controller (MVC) and Routes

Come definito in precedenza, la scelta del Pattern Architetturale è ricaduta sul Model-View-Controller (MVC). Consente la realizzazione di un'architettura che permette la **separazione netta** tra come si presentano i dati e le relative informazioni (**View**) e come si gestiscono i dati stessi (**Model**). Nel caso in esame, principalmente per le caratteristiche dei framework di sviluppo che verranno mostrati successivamente, per progettare l'architettura del prodotto, si aggiunge il componente "Routes".

Lo schema di **Alto livello** del Pattern Architetturale MVC è il seguente:

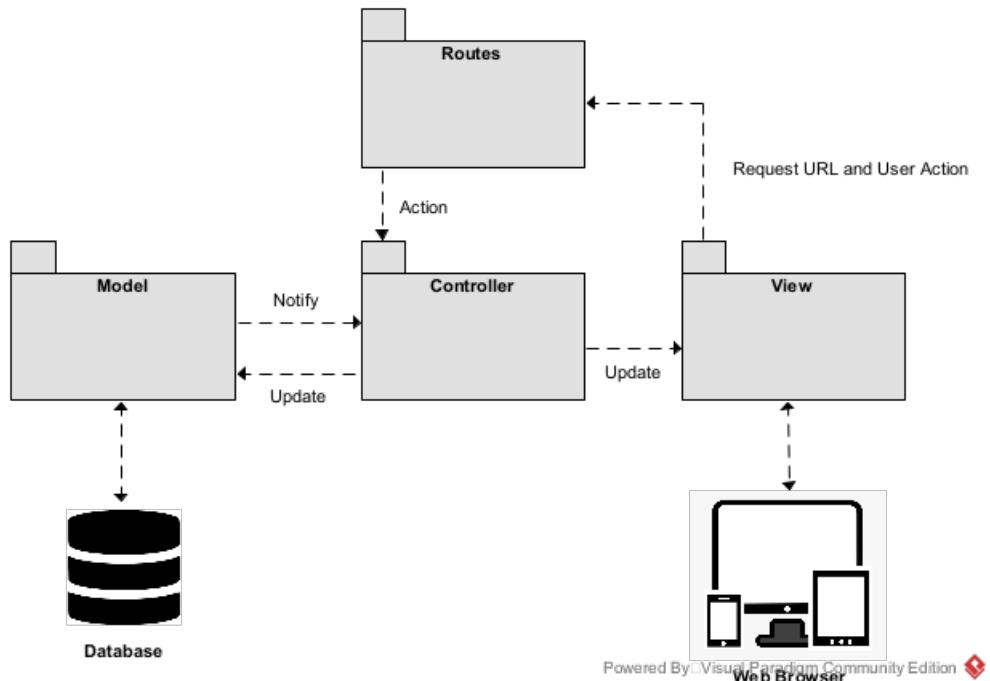


Figura 5.2: Model View Controller and Routes - Alto livello

In breve:

- 1. Un attore richiede di visualizzare la pagina della web application immettendo dal browser un URL.
- 2. L'applicazione abbina l'URL a una **route** predefinita.
- 3. Viene chiamata l'azione del **controller** associata alla route
- 4. L'azione del controller utilizza i modelli per recuperare tutti i dati necessari da un database, inserisce i dati in una struttura dati e li carica in una **view**.
- 5. La **view** dunque accede alla struttura dei dati e la utilizza per eseguire il rendering della pagina richiesta, che viene quindi presentata all'utente nel proprio browser.

5.2.1 Routes

Ad ogni percorso della "Routes" è associato un "Controller". Pertanto, quando si immette un URL, l'applicazione tenta di trovare una route corrispondente e, se ha esito positivo, chiama l'azione del controller associata a tale route.

5.2.2 Model

Il "Model" viene utilizzato per recuperare tutti i dati necessari da un database, e tali dati vengono passati a una "View", che esegue il rendering della pagina richiesta. Un modello può quindi essere descritto utilizzando un diagramma entità-relazione, che mostra tutti i tipi di oggetti, i loro attributi e il modo in cui le entità sono correlate tra loro.

5.2.3 Vista

Dunque, una "View" è l'interfaccia utente in grado di presentare dati provenienti da un modello. Le views contengono elementi di input come pulsanti, campi e cursori. Quando questi elementi di input sono attivati, il "Controller" deve decidere come rispondere. Le visualizzazioni sono spesso scritte come modelli con segnaposto per i dati. Per la programmazione Web, un modello "View" viene spesso realizzato utilizzando HTML, CSS e JS come nel caso in esame.

5.2.4 Controller

Un "Controller" risponde all'input modificando una vista o un modello. Un tipo comune di "Controller" è guidato da un'interfaccia utente grafica, che utilizza elementi come menu, campi e pulsanti in modo che un utilizzatore possa fare clic su questi elementi per eseguire delle azioni. Un ulteriore tipo di "Controller" è un'API, che viene in genere utilizzata da altri software (piuttosto che da un essere umano) per far sì che l'applicazione faccia qualcosa.

5.3 Stile Architetturale - Microservizi

I Microservizi sono architetture che consentono di supportare la creazione del software con una logica orientata alla gestione separata dei singoli servizi che compongono l'applicazione completa. Sfruttando le potenzialità degli ambienti di sviluppo in Cloud ed in particolare delle PaaS (Platform as a Service), esse consentono di svolgere in maniera agile le fasi di sviluppo, test e implementazione delle applicazioni.

Le architetture a microservizi nascono sulla scia di modelli come il SOA (Service Oriented Architecture) con un focus specifico sullo sviluppo del software, ribaltando totalmente il concetto monolitico in favore di uno modello agile, basato sulla decomposizione del "monolite" tradizionale in più pacchetti, cui corrispondono i singoli elementi funzionali che compongono il software.

Tali pacchetti vengono disaccoppiati, in modo da risultare il più possibile indipendenti tra loro durante le fasi di sviluppo e risultare combinabili nell'applicazione di riferimento. Questo comporta quindi un'agilità intrinseca data dal possibile riutilizzo dei vari pacchetti per strutturare applicazioni differenti oltre al fatto di poter aggiornare i vari moduli del software senza dover ricompilare l'intera struttura. Si riporta il diagramma architettonico ad alto livello dell'architettura a Microservizi del sistema "SelfTestCOVID19":

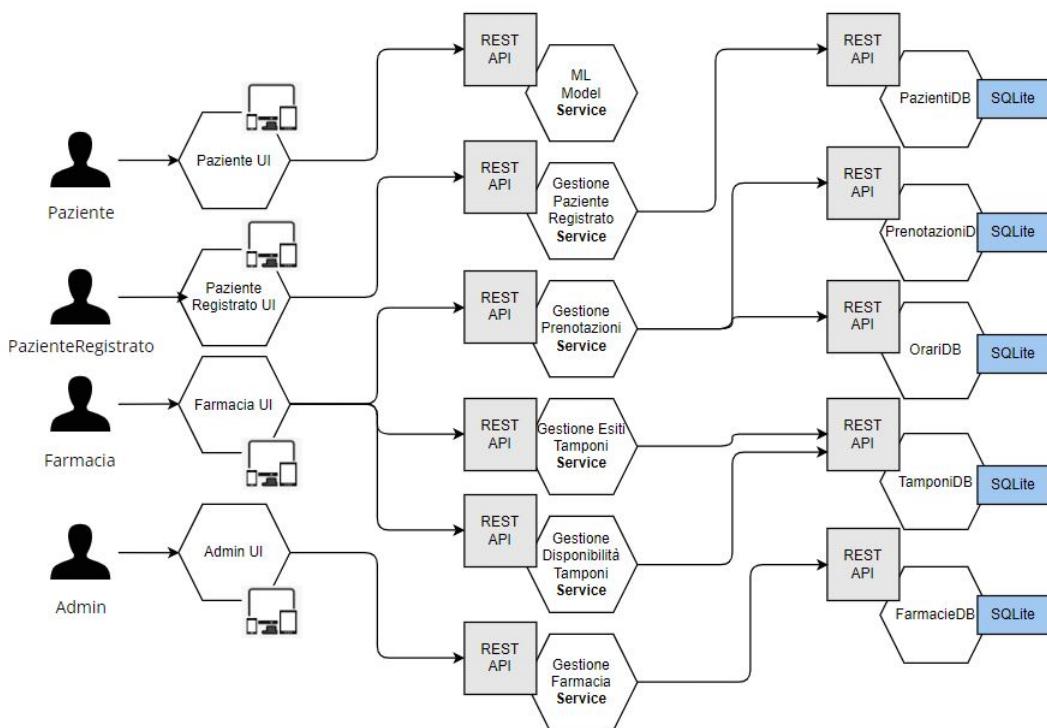


Figura 5.3: Microservizi - Alto livello - SelfTestCOVID19

Dalla Figura 5.3 si notano come gli attori invocano i Microservizi attraverso le UI dedicate e con il protocollo REST, ogni servizio fornisce i suoi clienti l'accesso alle sue funzionalità.

La suddivisione in Microservizi ha fatto sì di sviluppare e distribuire ogni servizio in modo indipendente in quanto ogni componente è **lasciamente accoppiato**.

5.4 REST

Vale la pena dunque definire **REST (REpresentational State Transfer)**. Esso pone dei requisiti architetturali alla quale il sistema deve soddisfare, utilizzando degli standard per la rappresentazione dei dati e l'HTTP per il trasferimento di questi dati o altri. Quando il **Client** chiama un'API RESTful, il Server trasferirà al Client una *rappresentazione dello stato* della risorsa richiesta.

Quando il Client richiede al Server una risorsa, dovrà fornirgli un'**identificatore** per la risorsa che gli interessa (URL) e l'**operazione** che si desidera che il Server esegua su tale risorsa sottoforma di metodi HTTP:

- POST: crea una nuova risorsa;
- GET: restituisce una o più risorse esistenti;
- PUT: aggiorna una risorsa o, se non esiste, la crea;
- DELETE: elimina una risorsa.

Vantaggi della soluzione REST

Separazione tra Client e Server che consente di sviluppare i diversi componenti in maniera indipendente tra loro,

alta scalabilità dovuta al fatto che Server e Client possono risiedere su macchine diverse, indipendenza delle API REST dal linguaggio o dalle tecnologie utilizzate per implementare i componenti.

Riguardo quest'ultimo punto bisogna, per ogni linguaggio, mantenere solamente la coerenza sul formato delle richieste e delle risposte che ci si aspetta dal Client o dal Server in base a come esse siano state definite. Nel paragrafo successivo si elencano i principi base da seguire per costruire un'architettura REST affinché le API siano definite RESTful.

5.4.1 Client - Server in REST

Basato sul paradigma SoC(separation of concerns), questo principio stabilisce la separazione dei compiti tra Client e Server. Infatti in un'architettura distribuita bisogna andare a definire le figure di Client e Server che avranno il compito di svolgere una determinata funzionalità. Si ha dunque che il Client invocherà, tramite un messaggio di richiesta, un servizio esposto dal Server, che a sua volta si occuperà di elaborare la richiesta e di fornire una risposta al Client.

Come è formata una richiesta:

- **Endpoint:** URL in cui il Server REST è in ascolto.
- **Metodo:** utile per permettere di richiedere dati o modificarli in modo tale che il Server saprà che operazione ha richiesto il Client. Come già detto in precedenza i metodi sono GET, POST, PUT, DELETE e altri.
- **Intestazione:** i dettagli aggiuntivi forniti per la comunicazione tra Client e Server (si ricorda che REST è senza stato). Alcune delle intestazioni comuni sono:
 - **Request:**
 - * host: IP del Client
 - * accept-language: lingua comprensibile dal cliente
 - * user-agent: informazioni relativi sul Client (sistema operativo, etc...)
 - **Response:**
 - * status: lo stato della richiesta o codice HTTP
 - * content-type: tipo di risorsa inviata dal Server.
 - * set-cookie: imposta i cookie per Server
- **Dati:** contiene le informazioni che si desiderano inviare al Server (chiamato anche corpo o messaggio).

Pertanto, nel caso in esame, possiamo individuare questa separazione dalla Figura 5.4

5.4.2 Stateless

La comunicazione tra il Client e il Server deve essere "senza stato", cioè la richiesta del Client verso il Server deve contenere tutte le informazioni utili in modo tale che essa possa essere soddisfatta sul servizio chiamato. Viene quindi eliminata la correlazione tra le varie richieste, facendo così apparire ogni richiesta come se fosse la prima verso quel Server. Inoltre questo tipo di comunicazione dà la possibilità di poter scalare il sistema molto più facilmente, poiché non vi è più il problema di sincronizzare le informazioni sulla sessione degli utenti tra i vari nodi del Server; questa informazione viene conservata lato Client oppure demandata a un database.

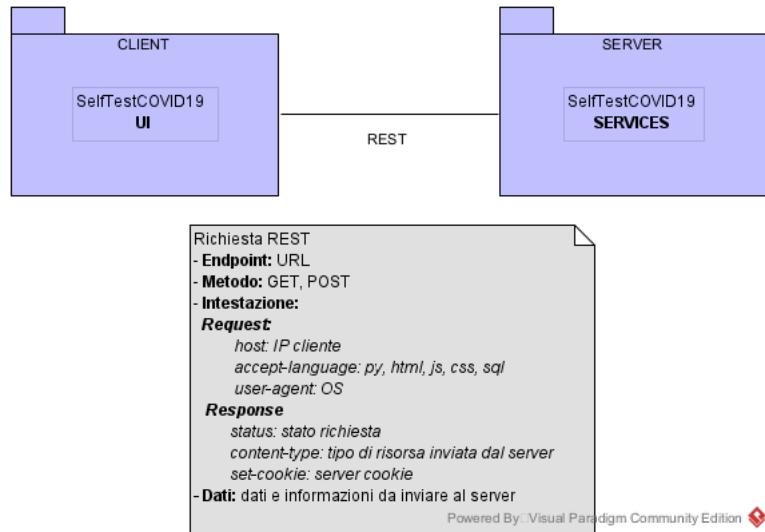


Figura 5.4: Client - Server REST

5.4.3 Cacheable

Le risposte fornite dal Server possono essere etichettate come cachabili o non cachabili. Nel caso di risposte cachabili, le informazioni contenute al suo interno possono essere riutilizzate nelle richieste successive evitando l'interazione con il Server poiché esse sono informazioni ancora valide ed attuali. Questo porta ad un conseguente miglioramento della scalabilità e delle performance del Server. Contrariamente, le risposte possono essere definite non cachabili se contengono informazioni su stati che in futuro potrebbero non essere più validi.

5.4.4 Interfaccia uniforme

La comunicazione tra Client e Server deve avvenire tramite un'interfaccia uniforme che preveda la definizione delle risorse e del formato nella quale esse devono essere rappresentate. Una risorsa è l'elemento fondamentale del servizio REST: essa rappresenta un oggetto identificante un qualcosa di inherente al dominio applicativo al quale si accede tramite un identificatore globale (URI) e di cui Client e Server si scambiano rappresentazioni tramite un'interfaccia standard come HTTP.

La rappresentazione più usata per le risorse è il formato JSON, ma vi sono altri formati utilizzabili come HTML come utilizzato per questo progetto.

5.4.5 Sistema stratificato

Il sistema stratificato permette ad un'architettura di essere composta da più livelli intermedi posti tra Client e Server, indipendenti tra loro ed ognuno dei quali svolge un determinato compito. Ad esempio, strati intermedi possono essere composti da sistemi che svolgono azioni di caching, di sicurezza o di load-balancing. Il vantaggio di avere un sistema stratificato con livelli indipendenti sta nella possibilità di poter aggiungere, cancellare o spostare i livelli intermedi in base al comportamento che si vuole ottenere nell'architettura, senza influenzare gli altri livelli già presenti.

5.5 Vista componenti e connettori

Si rappresenta inizialmente la struttura interna del sistema software in termini dei suoi componenti principali e delle relazioni fra di essi. I **componenti** forniscono funzionalità specifiche per il prodotto software che si sta realizzando mentre i **connettori** forniscono meccanismi di interazione indipendenti dall'applicazione. Con questo diagramma (Figura 5.5) si cerca di separare l'elaborazione dall'interazione esplicitandone i collegamenti che vi sono tra i componenti. I connettori consentono lo scambio di informazioni fra i componenti (es. dati, messaggi, risultati) di tipo *procedure call* tra il componente *Main Handler* e i componenti servizi mentre tra i componenti servizi e i database i connettori sono di tipo *data access*.

Nel caso in esame, i componenti rappresentati sono il **Client** e il **Server**, in particolare nel Server si organizzano i vari componenti come da definizione dei Microservizi, infatti si dividono i componenti in servizi per differenti operazioni che gli attori dovranno eseguire. Questi ultimi sono gestiti dal componente *Main Handler* che smisterà opportunamente le richieste (lollipop collegato con un semicerchio) verso i servizi forniti (lollipop collegato ad una linea retta):

- **MLModelService** (connector *MLModelServiceInterface*), il microservizio che gestisce il modello Machine Learning per la predizione al COVID19;
- **GestionePrenotazioniService** (connector *GestionePrenotazioniInterface*), il microservizio che gestisce le prenotazioni con le operazioni CRUD (lato farmacia);
- **GestionePazienteRegistratoService** (connector *GestionePazienteRegistratoService*), il microservizio che gestisce il profilo del paziente: visualizzazione dell'esito del tampone e delle operazioni CRUD (lato paziente)
- **GestioneEsitiTamponiService** (connector *GestioneEsitiTamponiService*), il microservizio che gestisce l'esito del tampone (lato farmacia)
- **GestioneDisponibilitàTamponiService** (connector *GestioneDisponibilitaService*), il microservizio che gestisce la disponibilità dei tamponi in farmacia (lato farmacia)
- **GestioneFarmaciaService** (connector *GestioneFarmaciaService*), il microservizio che gestisce le farmacie che aderiscono nel sistema con le operazioni CRUD (lato admin)

Si esplicitano inoltre le interfacce tra i componenti e il DB di ogni servizio sono:

- **PazientiDB** (connector *PazientiDatabaseAccess*)
- **PrenotazioniDB** (connector *PrenotazioniDatabaseAccess*)
- **OrariDB** (connector *OrariDatabaseAccess*)
- **TamponiDB** (connector *TamponiDatabaseAccess*)
- **FarmacieDB** (connector *FarmacieDatabaseAccess*)

Un discorso a parte merita il componente **MainHandler**, in quanto svolge il ruolo di dispatch delle richieste provenienti dal Client, ed inoltre, ha anche alcune operazioni interne che non hanno uno scopo funzionale, ma principalmente illustrativo delle caratteristiche generali del Sistema.

In seguito si descriveranno in termini di sequence, activity, e communication diagram le funzionalità con priorità più alta del software in questione.

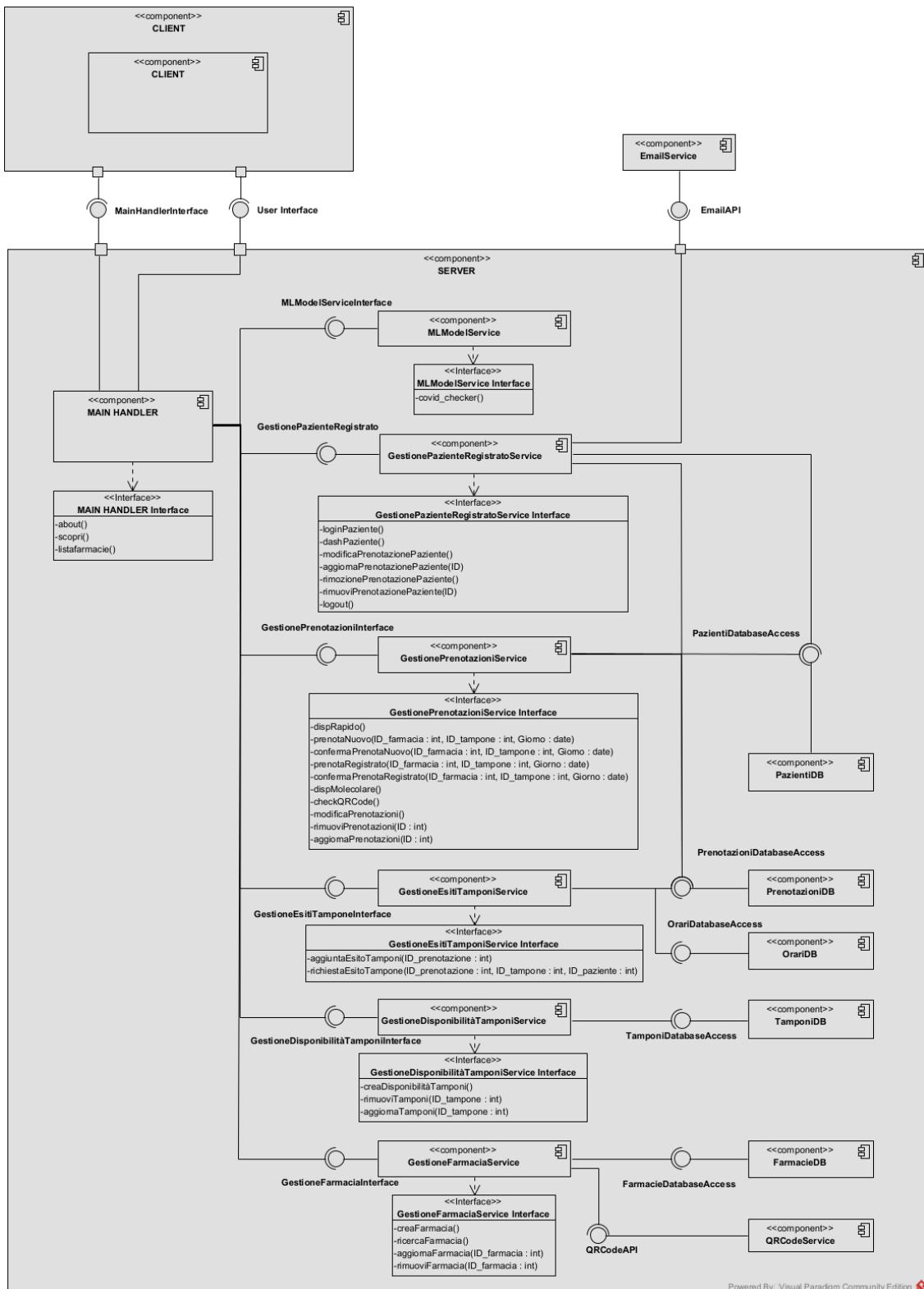


Figura 5.5: View, component and connector diagram

5.6 Sequence Diagram di progettazione

Si è scelto di realizzare diagrammi di sequenza con un livello medio di astrazione, detti di progettazione sfruttando i componenti già individuati e descritti nel diagramma 5.5.

5.6.1 Sequence Diagram di progettazione - Inserimento sintomi

Il paziente, inserirà i sintomi, malattie e informazioni relative al COVID19 attraverso il form, il *Client* le catturerà e le invierà al *Main Handler* che richiederà al *MLModelService* di effettuare il calcolo sui dati inseriti dal Client. Quest'ultimo restituirà il risultato in percentuale.

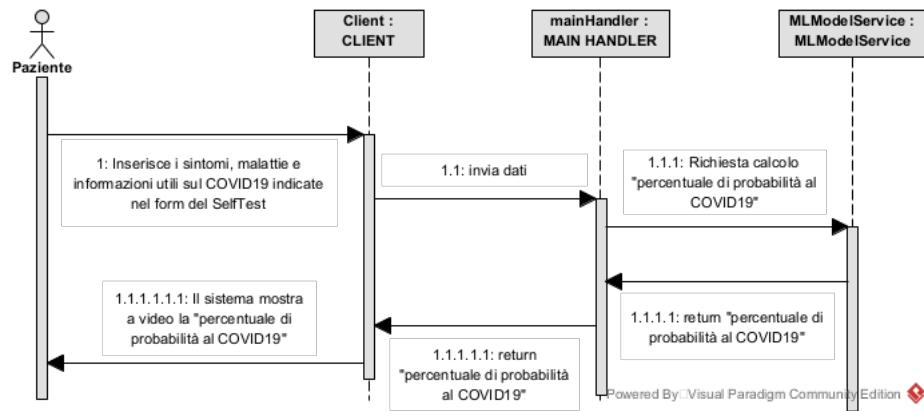


Figura 5.6: Sequence Diagram di progettazione - Inserimento sintomi

5.6.2 Sequence Diagram di progettazione - Verifica disponibilità farmacie

Il paziente, dopo aver svolto il test ed è risultato potenzialmente positivo (esito in percentuale maggiore del 50%), richiede di prenotare un tampone. Verrà dunque sollecitato il servizio *GestionePrenotazioniService* quando il Paziente, dopo aver inserito le informazioni richieste dal form, effettua una richiesta della farmacia al *Main Handler* che smista la richiesta verso il servizio che a sua volta svolge una query verso la *FarmaciaDB*. Quest'ultima restituirà le informazioni sulla farmacia, con la relativa disponibilità, se la query ha avuto esito positivo, altrimenti restituirà che la farmacia è inesistente.

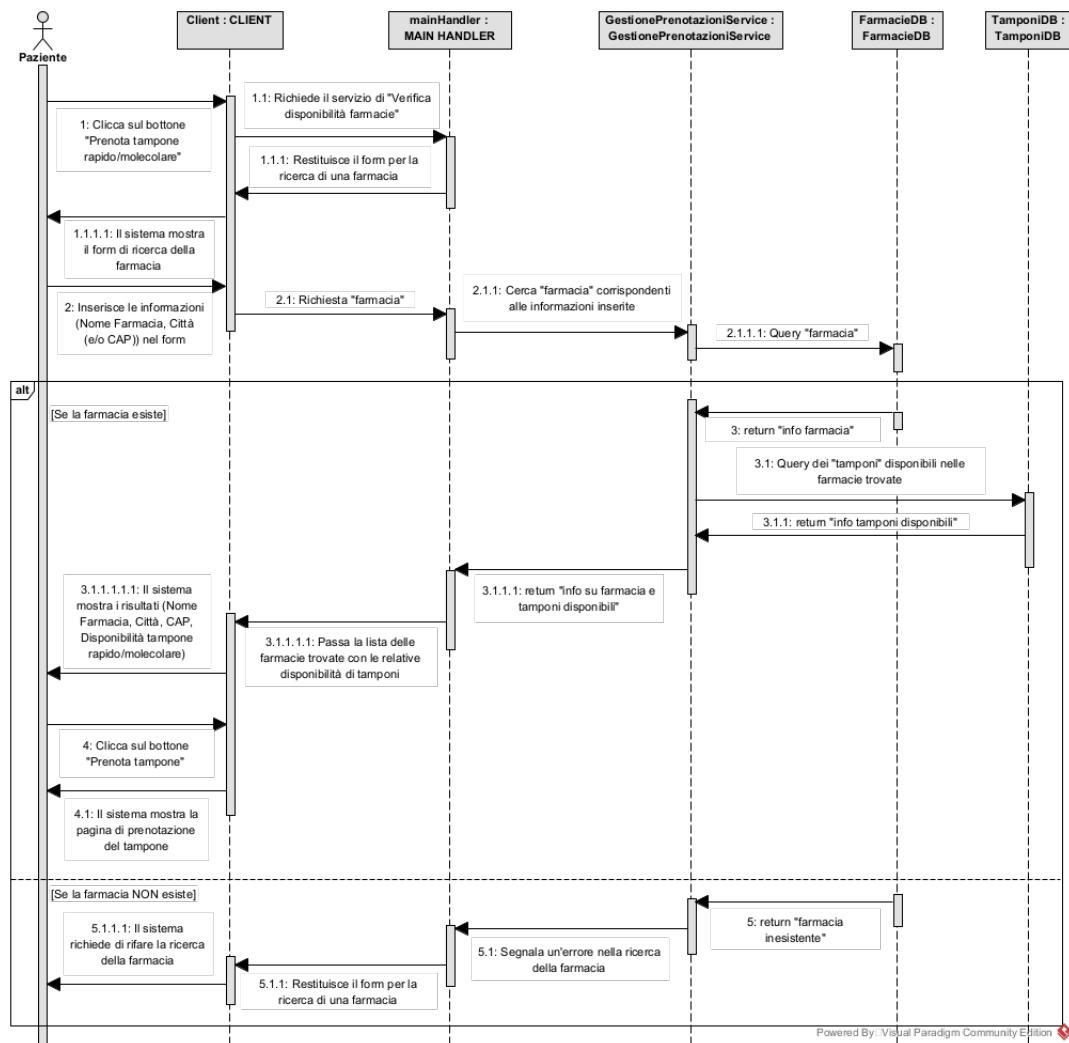


Figura 5.7: Sequence Diagram di progettazione - Verifica disponibilità farmacie

5.6.3 Sequence Diagram di progettazione - Creazione disponibilità tampone

La farmacia, dopo aver fatto il login alla propria dashboard, richiede di creare la disponibilità di nuovi tamponi. Verrà dunque sollecitato il servizio *GestioneDisponibilitàTamponiService* quando la farmacia, dopo aver inserito le informazioni richieste dal form, effettua una richiesta di creazione disponibilità al *Main Handler* che smista la richiesta verso il servizio che a sua volta svolge una INSERT verso il *TamponiDB*. Quest'ultimo restituirà le informazioni sulla disponibilità di tamponi, dopodiché vengono aggiunti anche gli orari di disponibilità relativi ai tamponi appena inseriti nel sistema, viene quindi effettuata un'altra INSERT in *OrariDB*.

Se le informazioni inserite relative al tampone risultano errate, tuttavia, le precedenti operazioni di INSERT non vengono effettuate ed il sistema restituisce un errore e chiede alla farmacia di reinserire nuovamente le informazioni.

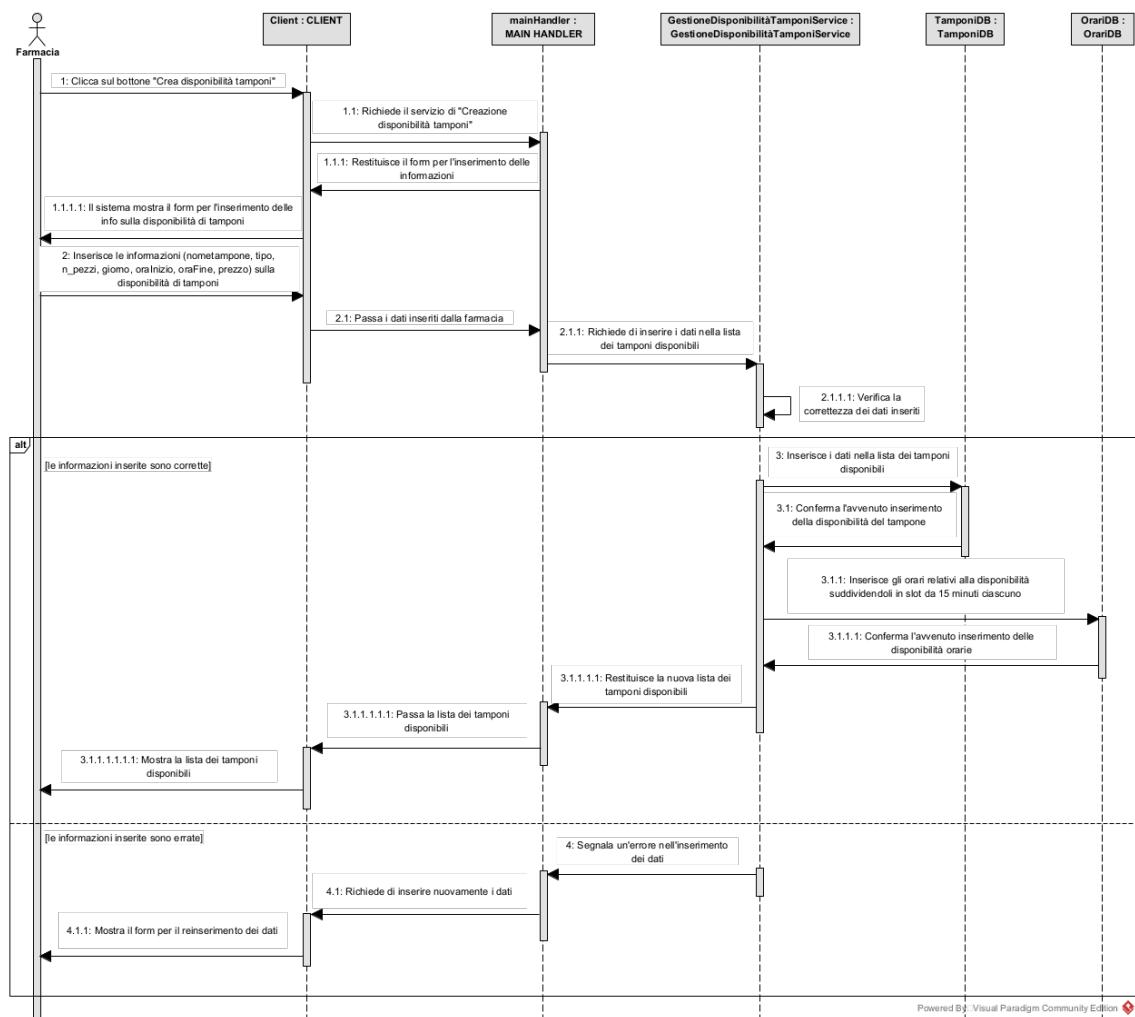


Figura 5.8: Sequence Diagram di progettazione - Creazione disponibilità tampone

5.6.4 Sequence Diagram di progettazione - Prenotazione tampone

Il paziente, dopo aver verificato la disponibilità delle farmacie, richiede di prenotare un tampone. Verrà dunque sollecitato il servizio *GestionePrenotazioniService* quando il paziente, dopo aver indicato di essere registrato o meno, effettua una richiesta di prenotazione al *Main Handler*, quest'ultimo smista la richiesta verso il servizio che a seconda dello stato del paziente (registrato o meno) mostrerà il form relativo affinché il paziente diventi un paziente registrato. Il paziente registrato inserisce le informazioni relative alla prenotazione e le sue generalità, dopodiché il *GestionePrenotazioniService* effettua una INSERT in *PrenotazioniDB*, a seguito dell'inserimento della prenotazione avviene l'UPDATE in *TamponiDB* della disponibilità di tamponi e la DELETE da *OrariDB* dell'Orario di disponibilità relativo. Inoltre, se il paziente non era registrato viene anche effettuata una INSERT in *PazientiDB*. Se le informazioni inserite relative alla prenotazione risultano errate, tuttavia, le precedenti operazioni di INSERT, UPDATE e DELETE non vengono effettuate ed il sistema restituisce un errore e chiede al paziente di reinserire nuovamente le informazioni.

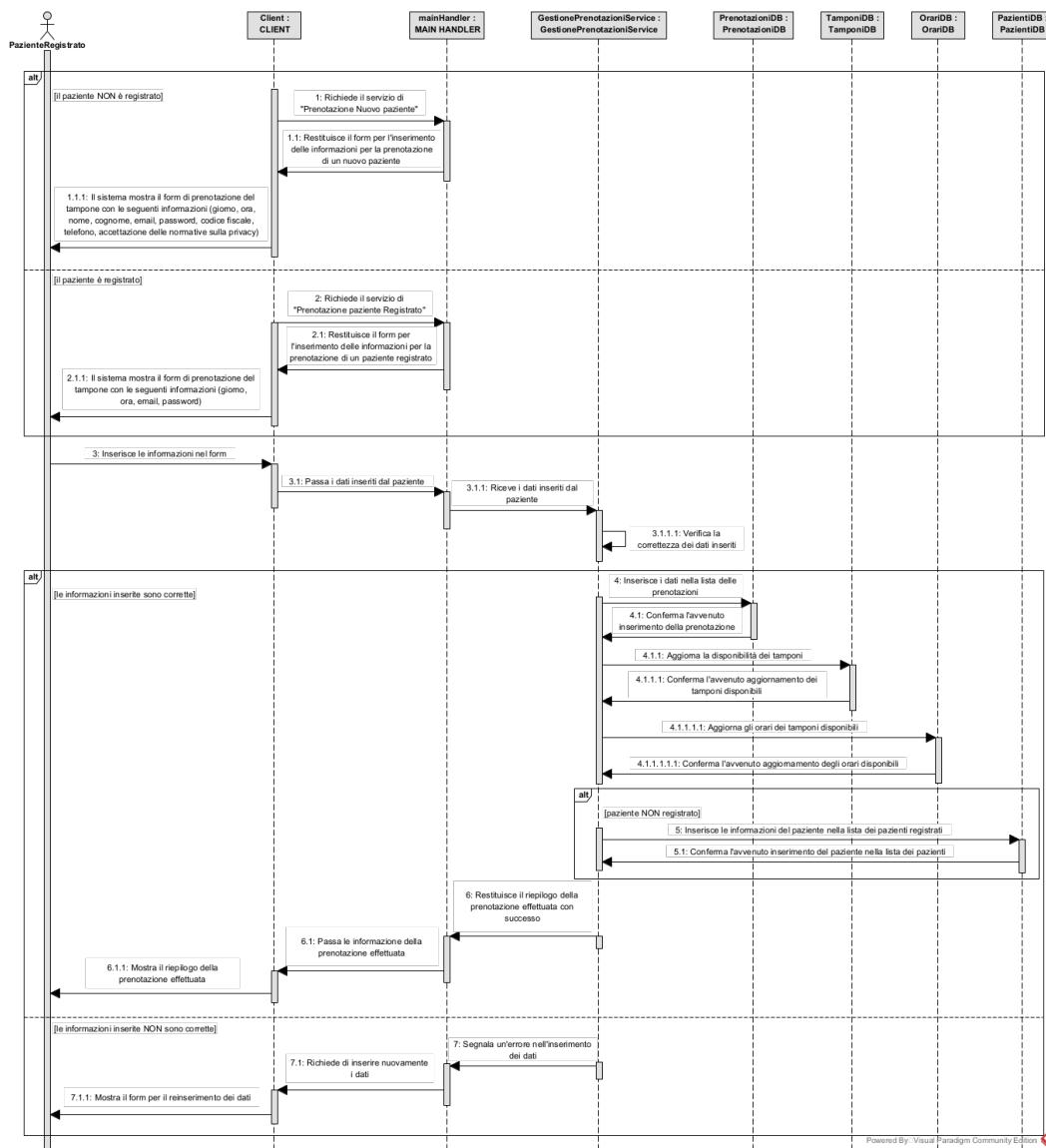


Figura 5.9: Sequence Diagram di progettazione - Prenotazione tampone

5.7 Activity Diagram

Per descrivere gli aspetti dinamici del software (flusso di esecuzione di una funzione dell'applicazione) si utilizza l'*Activity Diagram*. Esso ha lo scopo di modellare i flussi di dati e di operazioni che si intersecano con le attività correlate. Si è scelto la variante *Swimlane* (corsie di una piscina) per separare al meglio le fasi di esecuzione del flusso tra le componenti presenti nel software. Nel caso in esame, le swimlane sono il *Client*, il *Main Handler* e il *servizio corrispondente*.

5.7.1 Activity Diagram - Inserimento sintomi

Nel caso d'uso *Inserimento sintomi*, le swimlane sono il *Client*, il *Main Handler* e il *MLModelService*. Attraverso il Client (Browser), il paziente inserirà i sintomi (nel form). Il Main Handler invierà i dati nel modello e il MLModelService calcolerà la percentuale di probabilità al COVID19. Il Main handler riceverà il risultato, valuterà il risultato e mostrerà al Client il relativo esito in base alla percentuale.

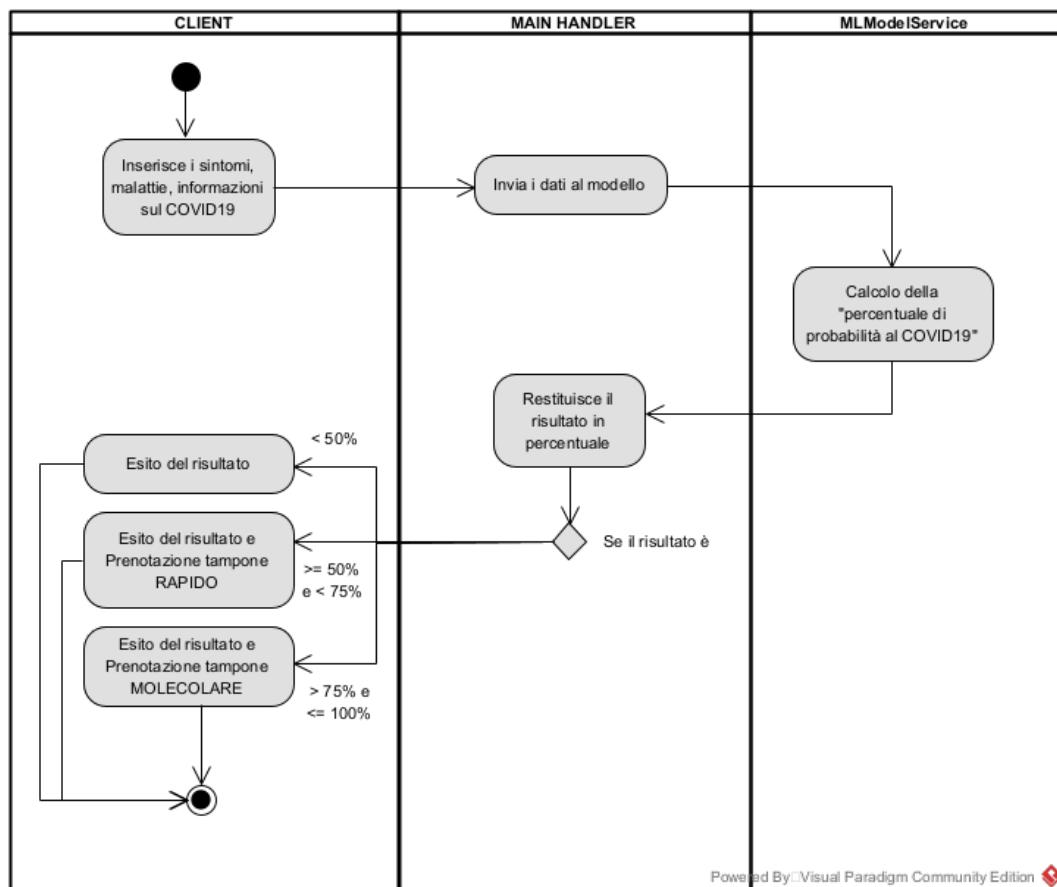


Figura 5.10: Activity Diagram - Inserimento sintomi

5.7.2 Activity Diagram - Verifica disponibilità farmacie

Nel caso d'uso *Verifica disponibilità farmacie*, le swimlane sono il *Client*, il *Main Handler*, la *GestionePrenotazioneService*, la *FarmaciaDB* e il *TamponiDB*. Nel Client inserisco le informazioni relative alla farmacia ed effettua una richiesta della stessa al GestionePrenotazioniService che a sua volta effettua una query nella FarmaciaDB. Se la farmacia non esiste, ritornerà con un messaggio di inesistenza altrimenti il servizio GestionePrenotazioniService cercherà i tamponi disponibili nelle farmacie trovate e dunque svolgerà una query nei TamponiDB restituendo la disponibilità al servizio GestionePrenotazioniService.

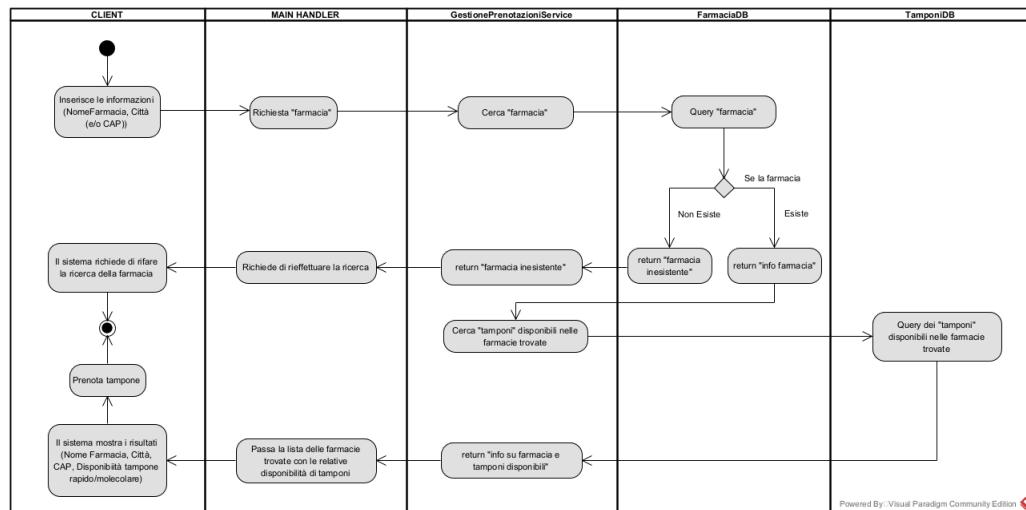


Figura 5.11: Activity Diagram - Verifica disponibilità farmacie

5.7.3 Activity Diagram - Creazione disponibilità tampone

Nel caso d'uso *Creazione disponibilità tampone*, le swimlane sono il *Client*, il *Main Handler*, la *GestioneDisponibilitàTamponiService*, il *TamponiDB* e l'*OrariDB*. Nel Client inserisco le informazioni relative alla nuova disponibilità di tamponi ed effettua una richiesta della stessa al GestioneDisponibilitàTamponiService che a sua volta, se le informazioni sono corrette, effettua una INSERT nel TamponiDB ed una INSERT nell'OrariDB. Se le informazioni inserite sono errate, ritornerà con un messaggio di errore e chiederà alla farmacia di reinserire le informazioni.

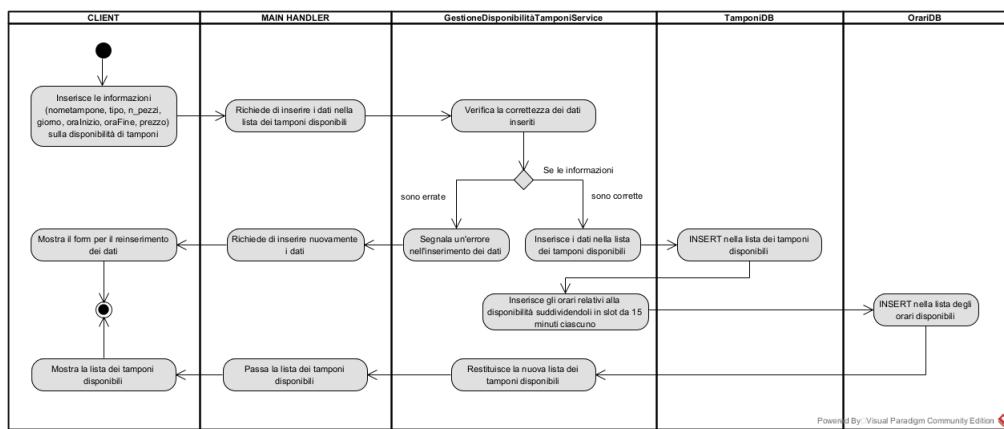


Figura 5.12: Activity Diagram - Creazione disponibilità tampone

5.7.4 Activity Diagram - Prenotazione tampone

Nel caso d'uso *Prenotazione tampone*, le swimlane sono il *Client*, il *Main Handler*, la *GestionePrenotazioneService*, il *PrenotazioniDB*, il *TamponiDB*, l'*OrariDB* ed il *PazientiDB*. Nel Client inserisco le informazioni relative alla prenotazione, in base al fatto se il paziente è registrato o meno, ed effettua una richiesta della stessa al GestionePrenotazioneService che a sua volta, se le informazioni sono corrette, effettua una INSERT nel PrenotazioniDB, fa un UPDATE della disponibilità in TamponiDB ed una DELETE dell'orario in OrariDB. Inoltre, se il paziente non era registrato viene fatta anche una INSERT in PazientiDB. Se le informazioni inserite sono errate, ritornerà con un messaggio di errore e chiederà al paziente di reinserire le informazioni.

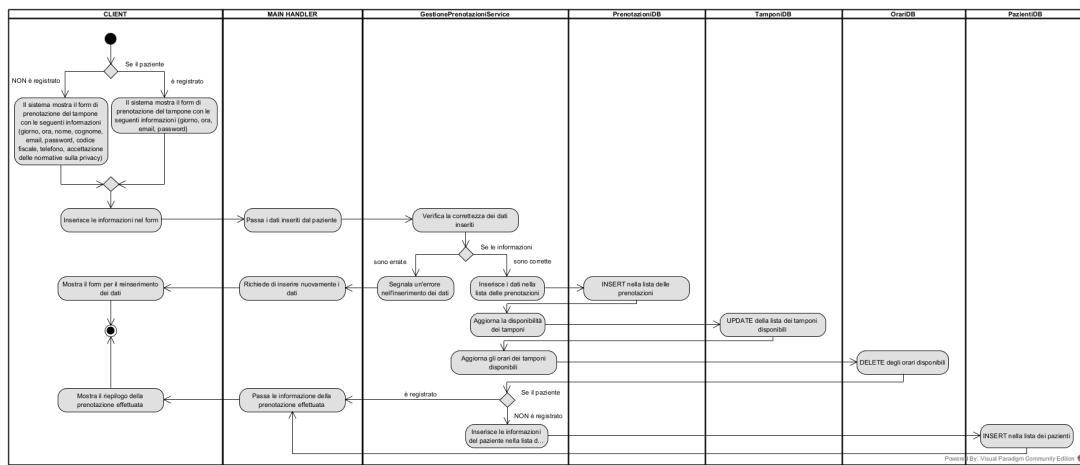


Figura 5.13: Activity Diagram - Prenotazione tampone

5.8 Communication Diagram

Per descrivere l'interazione tra attore e funzionalità si fa uso del *Communication Diagram*. Esso definisce i *partecipanti* e la sequenza di *messaggi* che intercorrono tra i partecipanti stessi. I partecipanti possono essere elementi che compongono un caso d'uso. Anche in questo caso si descrivono i Communication Diagram per i casi d'uso con priorità più elevata.

5.8.1 Communication Diagram - Inserimento sintomi

Nel seguente diagramma si mostrano dunque i *partecipanti* della funzionalità *Inserimento sintomi* e sono il *FormInserimentoSintomi* nel quale il Paziente, come già ampiamente descritto, inserirà sintomi, malattie e informazioni COVID19 e invierà un messaggio al successivo partecipante: *Risultato* che mostrerà l'esito del risultato in percentuale. Da notare la suddivisione in quanto, sulla base della percentuale, verranno inviati opportuni messaggi ai seguenti partecipanti: *PrenotazioneTamponeRapido* se il risultato è maggiore del 50% e minore del 75%, *PrenotazioneTamponeMolecolare* se il risultato è maggiore del 75% e minore uguale del 100%, *Esito* se il risultato è minore del 50%. (Figura 5.14)

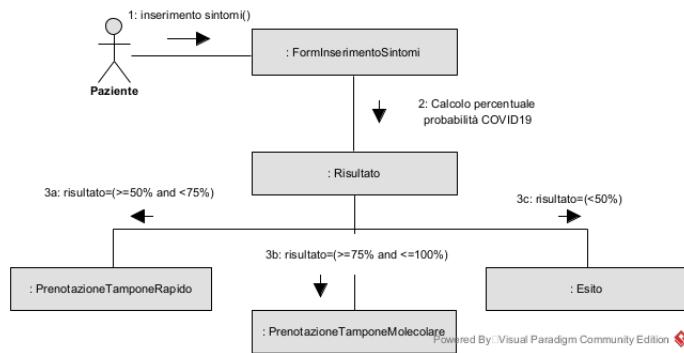


Figura 5.14: Communication Diagram - Inserimento sintomi

5.8.2 Communication Diagram - Verifica disponibilità farmacie

I partecipanti del caso d'uso *Verifica disponibilità farmacie* sono il *FormRicercaFarmacia* nel quale il paziente inserisce le informazioni di ricerca, quest'ultimo invierà un messaggio di *cerca farmacia* al partecipante *ListaFarmacieDisponibili* che mostrerà la lista delle farmacie. Verificherà la disponibilità dei tamponi sollecitando *VerificaTamponeDisponibile*, se i tamponi sono disponibili allora invia il messaggio al partecipante *PrenotazioneTampone* altrimenti torna al partecipante *ListaFarmacieDisponibili*. (Figura 5.15)

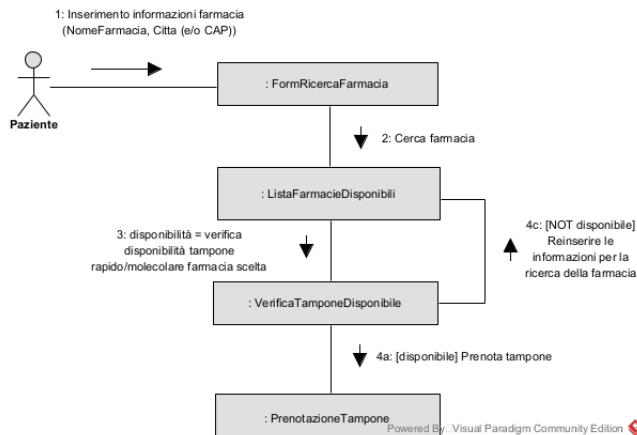


Figura 5.15: Communication Diagram - Verifica disponibilità farmacie

5.8.3 Communication Diagram - Creazione disponibilità tampone

I partecipanti del caso d'uso *Creazione disponibilità tampone* sono il *FormCreazioneDisponibilitàTampone* nel quale la farmacia inserisce le informazioni di creazione, quest'ultimo invierà un messaggio di *verifica informazioni* al partecipante *VerificaInformazionilnserite* che se le informazioni sono corrette invierà un messaggio di *crea disponibilità tamponi* al partecipante *AggiuntaDisponibilitàTampone* altrimenti torna al partecipante *FormCreazioneDisponibilitàTampone*. Il partecipante *AggiuntaDisponibilitàTampone* invia un messaggio di *crea orari disponibili* al partecipante *AggiuntaOrariDisponibili*, il quale invia un messaggio di *visualizza tamponi disponibili* al partecipante *MostraListaTamponiDisponibili*. (Figura 5.16)

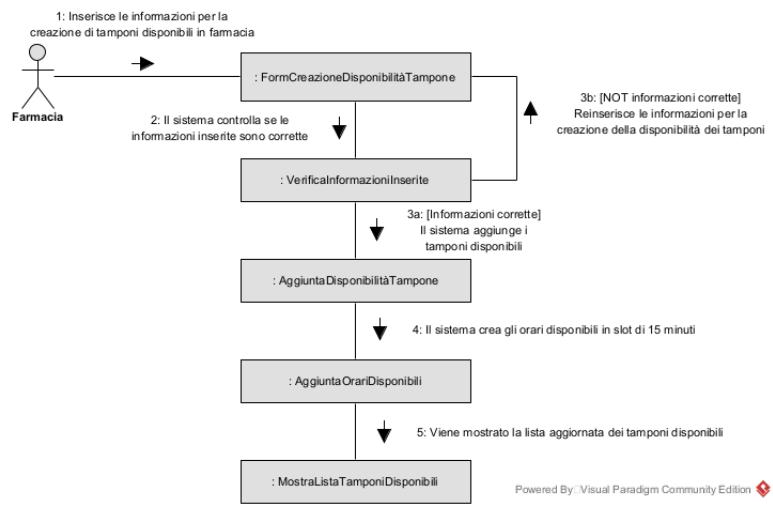


Figura 5.16: Communication Diagram - Creazione disponibilità tampone

5.8.4 Communication Diagram - Prenotazione tampone

I partecipanti del caso d'uso *Prenotazione tampone* sono il *VerificaStatoPaziente* nel quale il paziente indica il proprio stato, se quest'ultimo è registrato invierà un messaggio di *inserisci informazioni prenotazione registrato*, altrimenti invierà un messaggio di *inserisci informazioni prenotazione nuovo*. Dopo di che il partecipante *ValidazioneDatiInseriti* controlla se le informazioni sono corrette invierà un messaggio di *inserisci prenotazione* al partecipante *InserimentoPrenotazione* altrimenti torna al partecipante *VerificaStatoPaziente*. Il partecipante *InserimentoPrenotazione* invia un messaggio di *aggiorna tamponi disponibili* al partecipante *AggiornaTamponiDisponibili*, il quale invia un messaggio di *aggiorna orari disponibili* al partecipante *AggiornaOrariDisponibili*. Il partecipante *AggiornaOrariDisponibili* se il paziente non è registrato invia un messaggio di *registra paziente* al partecipante *InserimentoPaziente*, altrimenti manda un messaggio di *mostra riepilogo prenotazione* al partecipante *RiepilogoPrenotazioneEffettuata*. (Figura 5.17)

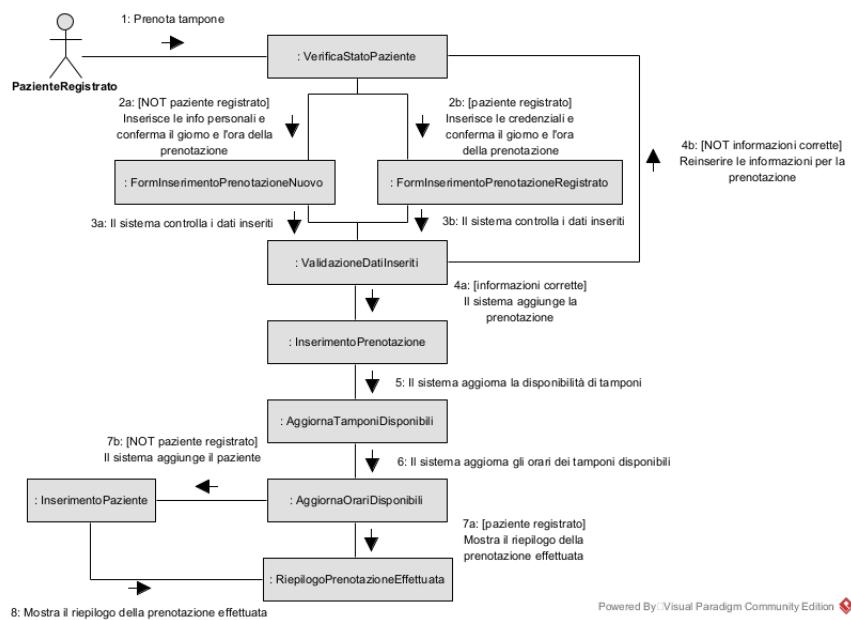


Figura 5.17: Communication Diagram - Prenotazione tampone

5.9 Server MVC Class Diagram

Il Server, come anticipato nelle scelte progettuali del pattern architetturale, è stato progettato attraverso il MVC. Il diagramma ci mostra 4 package tra loro interconnessi, in particolare, abbiamo le *View*, le *Routes*, il *Controller* ed il *Model*. Le *View* effettuano delle "REQUEST URL and ACTION" ai *Routes* quando l'utente interagisce con l'interfaccia, quest'ultimo richiama "ACTION" sul *Controller*.

Il *Controller* si occupa di eseguire le funzionalità del software e di fare "UPDATE" sia del *Model* che della *View*, che ha invocato la richiesta.

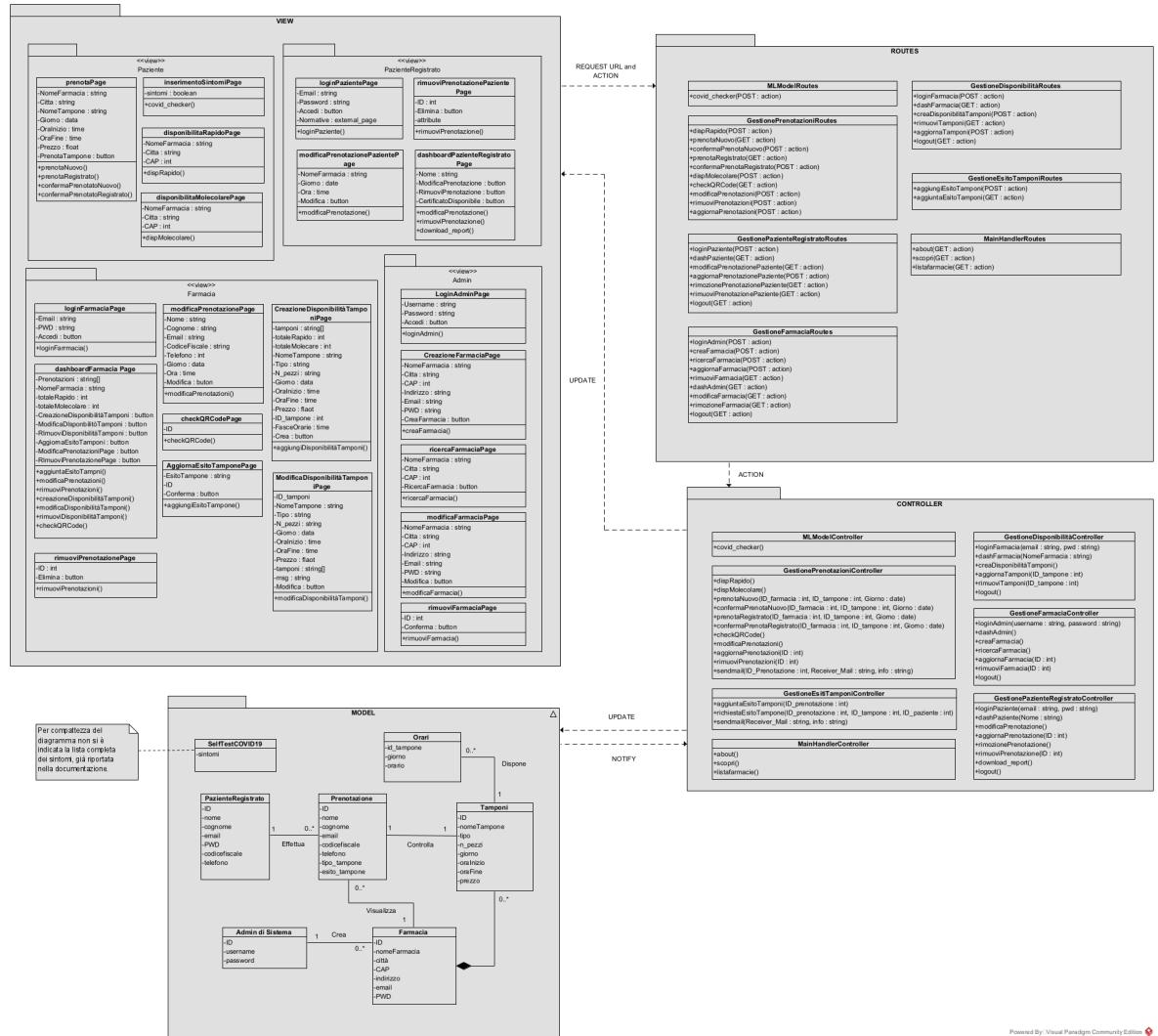


Figura 5.18: Server MVC Class Diagram

Dalla Figura 5.18, si può notare come all'interno del *Model* siano presenti le tabelle del DB. Questo diagramma è stato utile come base di partenza per l'implementazione del sistema "SelfTestCOVID19", come descritto più avanti nel documento.

6 Implementazione del software

Si determinano i framework di sviluppo e le scelte che hanno determinato la realizzazione del prodotto software

Contenuti

6.1	Scelte di sviluppo	84
6.2	Framework di sviluppo	85
6.2.1	Back-End: FLASK	85
6.2.2	Front-End: Bootstrap	87
6.3	Package Diagram	88
6.3.1	La struttura del progetto	88
6.4	Dinamica del Sistema MVC	90
6.4.1	Sequence Diagram di dettaglio - Inserimento Sintomi	90
6.4.2	Sequence Diagram di dettaglio - Verifica disponibilità farmacia	91
6.4.3	Sequence Diagram di dettaglio - Prenotazione tampone	92
6.4.4	Sequence Diagram di dettaglio - Creazione disponibilità tampone	93
6.5	Design Pattern	93
6.5.1	Pattern Grasp	94
6.5.2	Pattern Proxy	94
6.5.3	Pattern Observer	95
6.6	Deployment Diagram - Locale	95
6.7	Entity Relationship Diagram	96

6.1 Scelte di sviluppo

Le scelte di sviluppo del prodotto software sono state influenzate dal garantire la semplicità di supporto per il modello di Machine Learning che sarà inglobato nella web application.

Essa comprende sia la parte relativa all'interfaccia visibile all'utente con la quale esso interagisce, sia la logica lato Client nella quale si vanno a definire le funzionalità degli elementi visibili e le loro interazioni con il back-end; quest'ultimo contiene tutte le funzioni che svolge il Server, come ad esempio il salvataggio e l'elaborazione delle informazioni.

La si divide in livelli logico funzionali (in tre livelli: Figura 6.1):

- Livello di Presentazione: associabile al front-end, rappresenta l'interfaccia dell'applicazione con cui l'utente interagisce per immettere i dati e leggere i risultati attraverso il Web Browser (**User Interface visualizzata tramite pagine HTML,CSS,JS**);
- Livello Intermedio: associabile al back-end, rappresenta la logica di elaborazione poiché elabora i dati e in base alle richieste dell'utente genera dei risultati (**WebServer con i relativi moduli della web application**);
- Livello Dati: rappresenta un DBMS, indipendente dalle applicazioni Server, che viene usato per la persistenza dei dati (**SQLite**).

Ognuno di questi livelli riesce a comunicare solo con i livelli adiacenti assumendo nella comunicazione il ruolo di Client o di Server. Inoltre essi sono moduli indipendenti, per cui una modifica a uno dei livelli non si ripercuote direttamente su un altro livello. Parlando più specificamente del front-end, esso è costruito usando linguaggi come

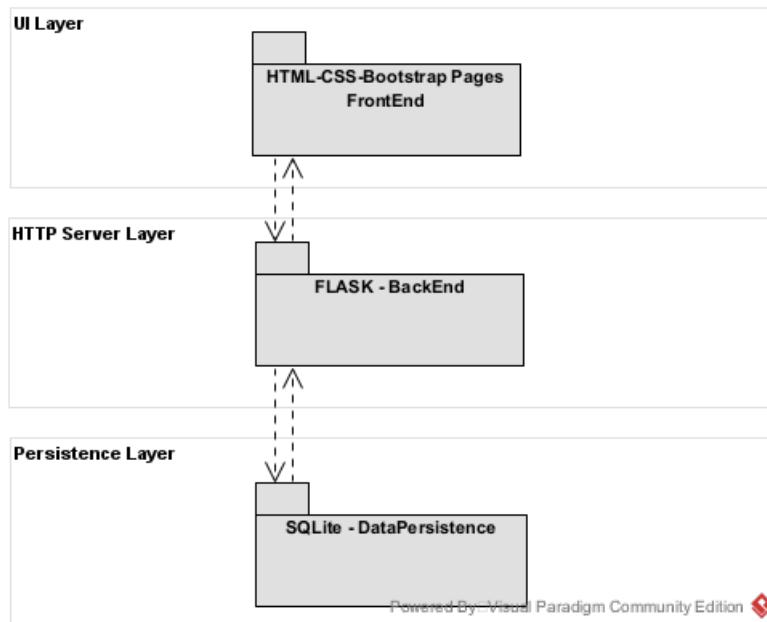


Figura 6.1: Organizzazione Three Tier

HTML, CSS e JavaScript, i quali sono supportati da gran parte dei browser.

Pertanto si è deciso di **NON adottare uno stile di programmazione di tipo OOP**, sebbene tale metodologia di sviluppo sia eccezionale e consenta un'enorme flessibilità al fine di avere più sistemi che interagiscono con gli stessi dati/logica; tuttavia, nello sviluppo di una web application c'è sicuramente una situazione in cui non ci si vorrebbe preoccupare di caricare molti oggetti. Vale a dire, essendo la maggior parte delle funzioni basate sull'interrogazione di un database e sulla selezione di un insieme di tuple da esso, tali operazioni spesso non richiedono effettivamente l'intervento di OOP. In effetti si potrebbe voler eludere completamente l'OOP poiché i dati presenti nelle tabelle del DB di solito sono strettamente collegate con altre informazioni (unione di più record) e normalmente non si vuole estrarre più dati dal database di quelli che si sta effettivamente utilizzando.

Ora, inserire informazioni in un DB di solito implica assicurarsi che venga seguita una determinata logica aziendale, come ad esempio, che la password segua determinate regole, in queste situazioni sfruttare uno stile OOP mantiene le cose troppo incapsulate e difficilmente trasferibili tra le applicazioni.

6.2 Framework di sviluppo

Per l'implementazione della Web Application si è utilizzato il framework web¹ **FLASK**, mentre per la realizzazione del front-end si è utilizzato il framework **Bootstrap**.

6.2.1 Back-End: FLASK

La scelta è influenzata dall'accorpamento del modello di **Machine Learning** nel SelfTest che è totalmente sviluppato e supportato con Python. Dunque serve un framework web che si basi su Python e possa interagire con il modello di ML in questione. La soluzione è FLASK. L'idea è mostrata nella Figura 6.2

¹Un Web Framework rappresenta una raccolta di librerie e moduli che consentono agli sviluppatori di applicazioni Web di scrivere applicazioni senza preoccuparsi di dettagli di basso livello come protocollo, gestione dei thread e così via.

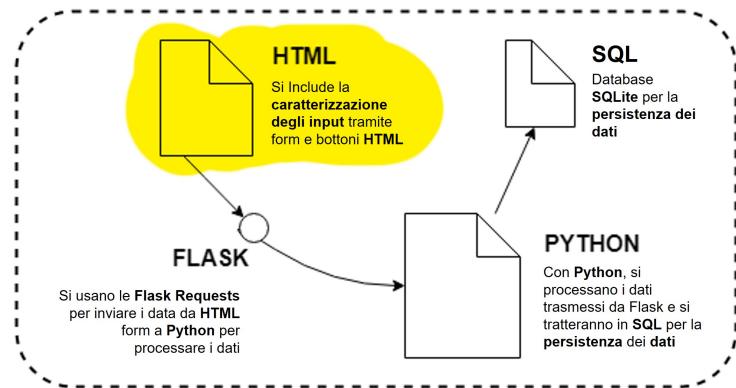


Figura 6.2: Passaggio HTML- FLASK - SQL

Flask offre agli sviluppatori varietà di scelta durante lo sviluppo di applicazioni WEB, fornisce strumenti, librerie e meccanismi che consentono di creare applicazioni senza dipendenze esterne. Flask è basato sul toolkit *Werkzeug WSGI* e sul motore di template *Jinja2*. In particolare:

- **WSGI:** Web Server Gateway Interface è un'interfaccia utilizzata come standard per lo sviluppo di applicazioni Web Python. È un protocollo di trasmissione che stabilisce e descrive comunicazioni ed interazioni tra Server ed applicazioni web scritte nel linguaggio Python.
- **Werkzeug:** Werkzeug è un toolkit WSGI che implementa richieste, oggetti di risposta e funzioni di utilità. Ciò consente di costruire un frame Web su di esso.
- **Jinja2:** è un motore di modelli per Python. Un sistema di modelli Web che combina un modello con un'origine dati specifica per eseguire il rendering di una pagina Web dinamica.

Per entrare più nel dettaglio, dalla Figura si nota come ci siano due lati coinvolti nel rispondere alla richiesta HTTP di un Client: il **Server Web** e l'**applicazione Web**.

- Il Server gestisce la complessità delle connessioni di rete, ovvero, riceve la richiesta dal Client e invia la risposta.
- L'applicazione, prende i *dati* della richiesta, agisce su di essi e crea la risposta per il Server da inviare. Cioè l'applicazione, contiene quella funzione che prende i dati input, esegue la procedura e rimanda in output la risposta da inviare.

L'interazione tra il **Server Web** e l'**applicazione Web** è **WSGI** (Web Server Gateway Interface), ma per garantire che si stia realizzando un'applicazione Web Python, quest'ultima deve assicurarsi che le funzioni create accettino determinati parametri indicati nell'intestazione HTTP e i dati del modulo di input. Se invece si vuole un Server Web che serva applicazioni Python, allora quest'ultimo deve essere in grado di richiamare quella funzione nell'applicazione web ogni qual volta che arriva una richiesta HTTP. WSGI specifica esattamente quali devono essere i parametri in input per quella funzione e restituisce al Server l'output inviato da quella funzione.

Werkzeug fornisce una serie di utility per lo sviluppo di applicazioni conformi a WSGI. Queste utility sono l'analisi dell'intestazioni, l'invio e la ricezione di cookie, l'accesso ai dati del modulo, la generazione di reindirizzamenti, la generazione di pagine di errore quando c'è un'eccezione, persino la fornitura di un Server per un debugger interattivo che viene eseguito nel browser.

Per completare dunque l'applicazione, serve comunque un template che possa essere presentabile e interattivo

all'utente. **Jinja2** può essere integrato nei formati di markup (HTML nel caso in esame) in modo da restituire all'utente tramite una richiesta HTTP una pagina web dinamica. E' detto motore di modello in quanto esso permette di poter aggiungere tra i tag HTML variabili che vengono *sostituite* dai valori passati dopo aver **eseguito il rendering** della pagina HTML (come nell'esempio mostrato di seguito).

Codice 6.1: Esempio di passaggio dei parametri in HTML grazie al template Jinja2

```
<!-- /templates/index.html -->
<html>
  <head>
    <title> {{ title }} </title>
  </head>
  <body>
    <h1> Hello {{ username }} </h1>
  </body>
</html>
```

Codice 6.2: Esempio Python FLASK

```
# app.py
from flask import Flask, render_template # importing the render_template function

app = Flask(__name__)
# home route
@app.route("/")
def hello():
    return render_template('index.html', username = 'John', title = "Hello Ninja")

app.run(debug = True)
```

Il particolare vantaggio di Jinja2 in HTML è quello di poter riuscire ad utilizzare statements ciclici come il for, oppure condizionali come l'if e tante altre skill che essa mette a disposizione nella sua documentazione. Questo permette dunque di avere dinamicità nelle pagine HTML (Figura 6.3).

In definitiva i **vantaggi** del framework rispettano le esigenze agili del prodotto software in esame. Esso infatti è **scalabile** quindi si ha la possibilità di incrementare rapidamente in qualsiasi direzione (verticale o orizzontale) consentendo di funzionare senza problemi anche mentre si espande e aumenta (approccio adattivo), è **leggero** quindi non si basa su un gran numero di estensioni per funzionare, c'è più controllo sotto questo aspetto, è **flessibile** e molto **documentato** dunque un ottimo supporto durante lo sviluppo dell'applicativo.

6.2.2 Front-End: Bootstrap

Bootstrap è un framework che uniforma i vari componenti che ne realizzano l'interfaccia web. Bootstrap infatti è una raccolta di strumenti per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su **HTML** e **CSS**, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione, così come alcune estensioni opzionali di **JavaScript**.

Anche in questo caso, la scelta di questo framework è pesata dall'esigenza agile del prodotto in esame in quanto i suoi **vantaggi** lo permettono. Il framework Bootstrap è **open source**, **facile da usare**, **compatibile** con la maggior parte dei browser, **responsiveness**, **coerente** ovvero tutti i componenti del framework condividono gli

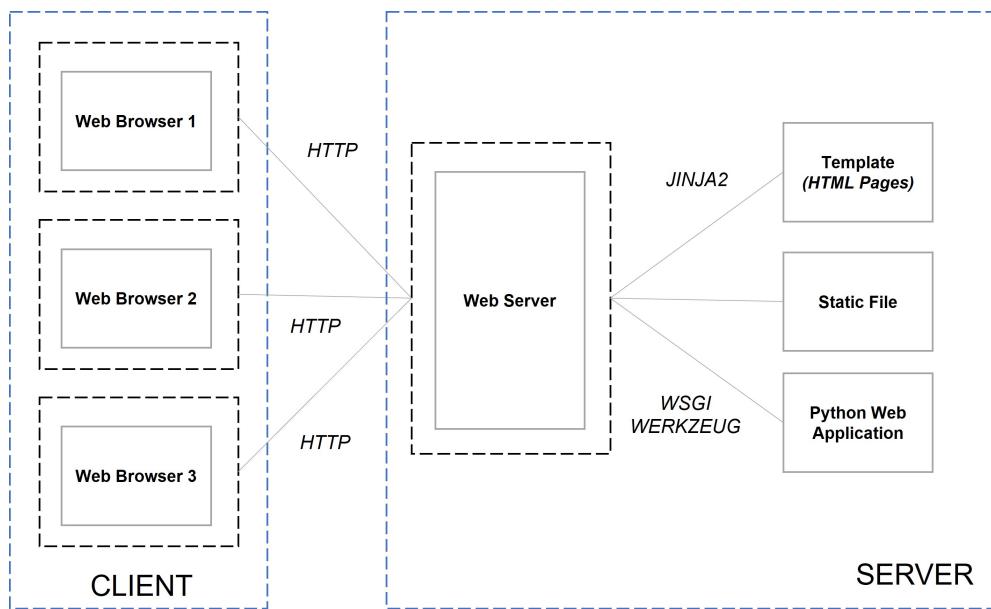


Figura 6.3: Interazione Client - Server (Web Server - Web Application Python)

stessi modelli e stili di progettazione attraverso una libreria centrale e dunque il design e il layout delle pagine saranno coerenti, **personalizzabile**.

6.3 Package Diagram

Per raggruppare gli elementi e fornire un namespace per gli elementi raggruppati, si utilizza un package diagram, utile per fornire un'organizzazione gerarchica dei package consentendo così una maneggevole organizzazione delle miriadi di elementi che un modello UML comporta. In modo speciale, si segue il pattern MVC descritto nel precedente paragrafo. (Figura 6.4)

6.3.1 La struttura del progetto

La struttura del progetto sarà dunque:

SelfTestCOVID19

- *templates*
 - AdminView
 - FarmaciaView
 - PazienteRegistratoView
 - PazienteView

- *controllers*
 - GestioneDisponibilitàController
 - GestioneEsitiTamponiController
 - GestioneFarmaciaController

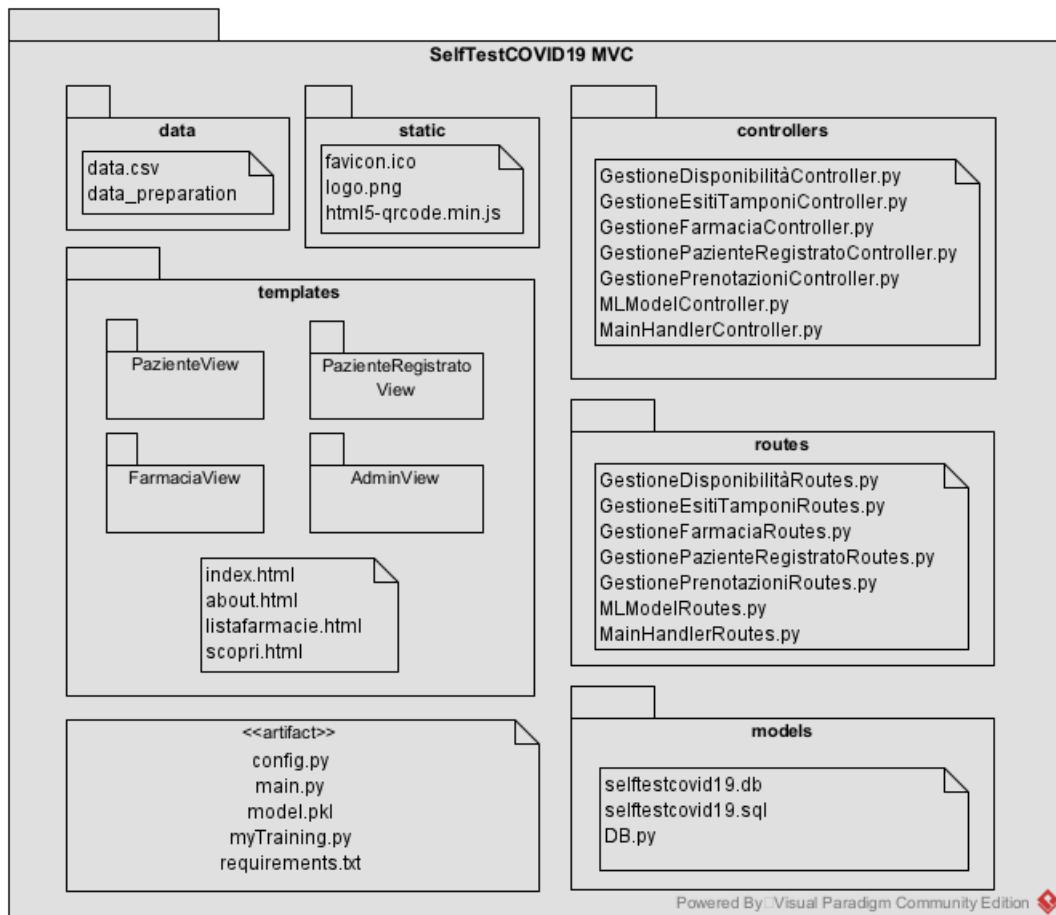


Figura 6.4: Package Diagram

- GestionePazienteRegistratoController
 - GestionePrenotazioniController
 - MLModelController
 - MainHandlerController
- *routes*
 - GestioneDisponibilitàRoutes
 - GestioneEsitiTamponiRoutes
 - GestioneFarmaciaRoutes
 - GestionePazienteRegistratoRoutes
 - GestionePrenotazioniRoutes
 - MLModelRoutes
 - MainHandlerRoutes
 - *model*
 - DB.py
 - selftestconvid19
 - main.py
 - myTraining.py
 - config.py

Le altre cartelle sono: *data*, *static* e il file per la console in Cloud per l'installazione delle librerie *requirements.txt*.

6.4 Dinamica del Sistema MVC

Si riportano i diagrammi dinamici di dettaglio che descrivono come sono implementate le funzionalità offerte dal sistema seguendo il design pattern MVC. In particolare, si entra nel dettaglio tecnico dei *Sequence Diagram* precedentemente descritti.

6.4.1 Sequence Diagram di dettaglio - Inserimento Sintomi

La dinamica del seguente diagramma mostra il paziente che interagirà con la view *InserimentoSintomiPage*, dal quale farà una request al servizio *MLModelService* attraverso *MLModelRoutes*. Quest'ultimo aziona il controller *MLModelController* che aggiornerà il modello *SelfTestCOVID19*. Esso calcolerà la probabilità, notificherà il controller dell'avvenuto calcolo passandogli il risultato che verrà mostrata alla view *InserimentoSintomiPage*.

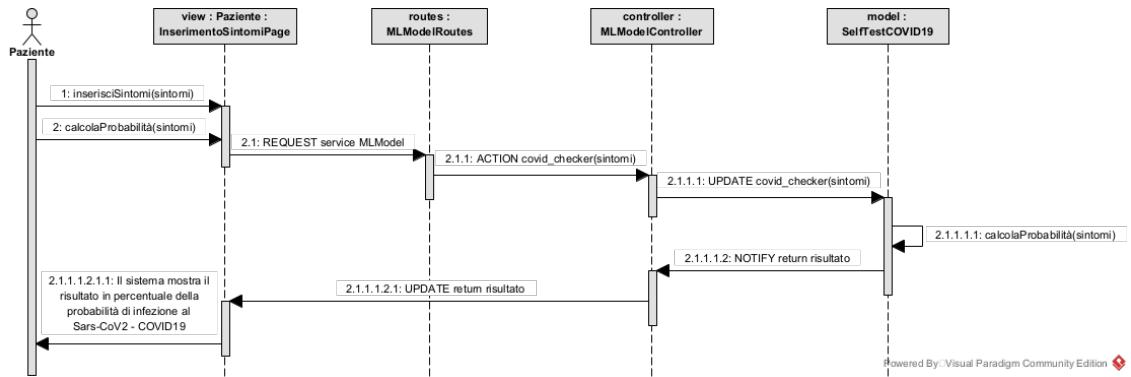


Figura 6.5: Sequence Diagram di dettaglio - Inserimento Sintomi

6.4.2 Sequence Diagram di dettaglio - Verifica disponibilità farmacia

Nel seguente diagramma il paziente interagirà con la view *disponibilitàRapidoPage* (caso *tampone rapido*) dove inserirà le informazioni della Farmacia (Nome, Città, CAP). La view richiederà il servizio *GestioneFarmacia* attraverso la routes *GestioneFarmaciaRoutes* che a sua volta azionerà il controller *GestioneFarmaciaController*. Esso accederà al model *Farmacia* verificando se la farmacia cercata è presente nel database. Dunque il model *Farmacia* notificherà il controller e mostrerà la lista delle farmacie trovate. A questo punto, la view richiede il servizio *GestionePrenotazioni* attraverso la routes *GestionePrenotazioniRoutes* che a sua volta azionerà il controller *GestionePrenotazioneController* che verificherà la disponibilità nel model *Tamponi*. Esso notificherà la disponibilità al controller *GestionePrenotazioniController* e aggiornerà la view *disponibilitàRapidoPage*.

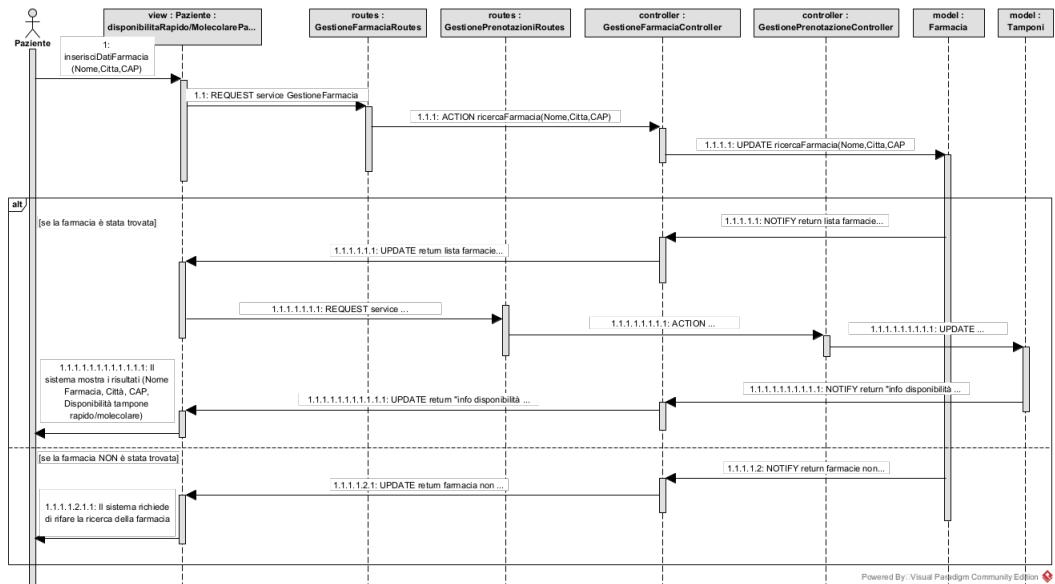


Figura 6.6: Sequence Diagram di dettaglio - Verifica disponibilità farmacia

6.4.3 Sequence Diagram di dettaglio - Prenotazione tampone

Nel seguente diagramma il paziente interagirà con la view *PrenotaPage* dove inserirà le informazioni relative alla prenotazione, nel caso in cui esso è registrato inserirà (giorno, ora, email, password), mentre nel caso in cui il paziente non è registrato inserirà (giorno, ora, nome, cognome, email, codice fiscale, password, telefono, accettazione delle normative sulla privacy). La view richiederà il servizio *GestionePrenotazioni* attraverso la routes *GestionePrenotazioniRoutes* che a sua volta azionerà il controller *GestionePrenotazioniController*. Esso accederà al model *Prenotazioni* inserendo la prenotazione nel database. Dunque, il model *Prenotazioni* notificherà il controller dell'avvenuto inserimento. A questo punto, il controller *GestioneDisponibilitàController* aggiorna il model *Tamponi* diminuendo il numero di pezzi del tampone prenotato, il model notifica che l'operazione è stata eseguita. Il controller, infine, elimina dal model *Orari* la disponibilità dell'orario appena prenotato, inoltre, se il paziente non era registrato il controller *GestionePrenotazioniController* lo inserisce nel model *Pazienti*. A questo punto il controller restituisce alla view il riepilogo della prenotazione effettuata che viene mostrata al paziente registrato. Se il controller *GestionePrenotazioniController* verifica che le informazioni inserite dal paziente per la prenotazione non sono corrette, lo segnala ritornando un errore alla view che mostra il messaggio di errore che chiede di reinserire le informazioni.

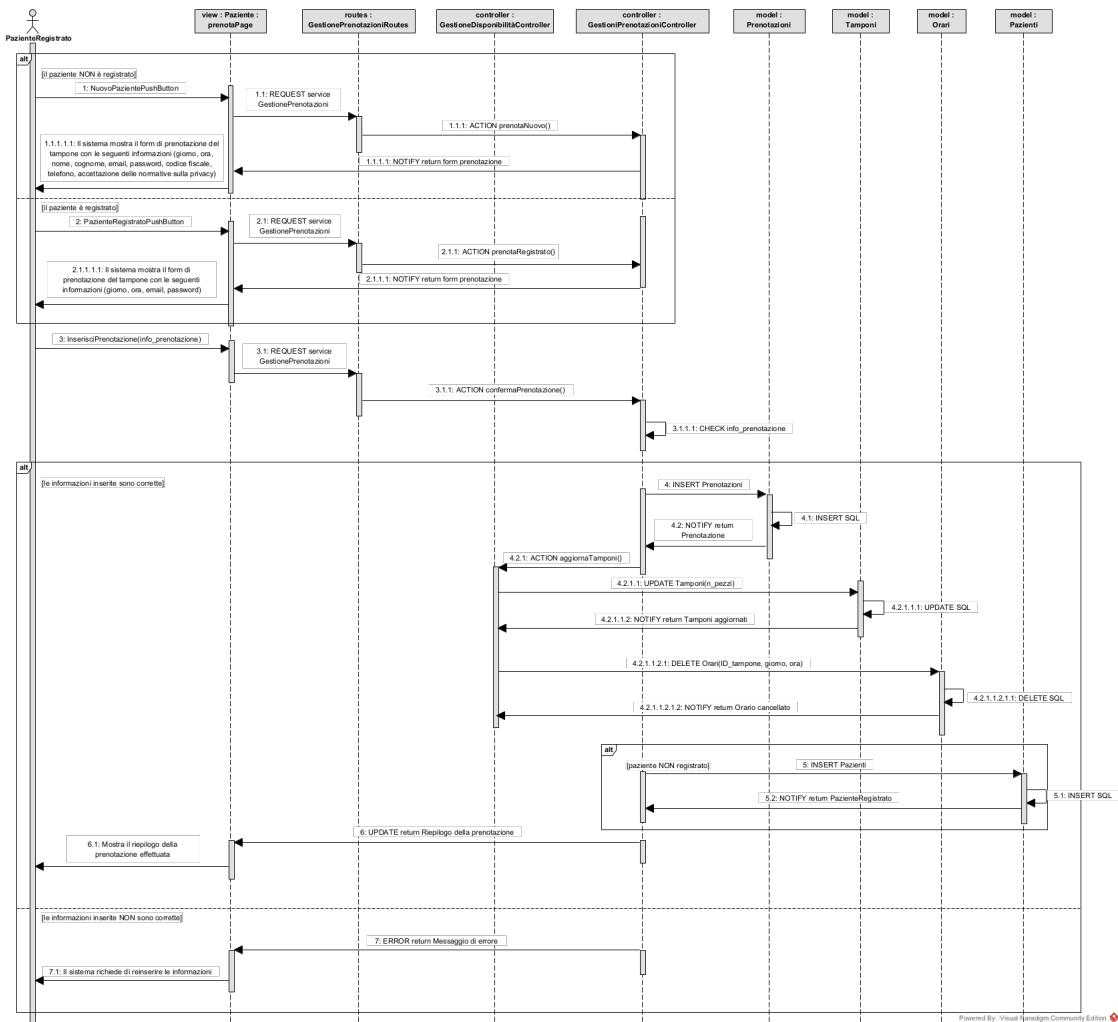


Figura 6.7: Sequence Diagram di dettaglio - Prenotazione tampone

6.4.4 Sequence Diagram di dettaglio - Creazione disponibilità tampone

Nel seguente diagramma la farmacia interagirà con la view *CreazioneDisponibilitàTamponiPage* dove inserirà le informazioni relative al tampone (nometampone, tipo, npezzi, giorno, orainizio, orafine, prezzo). La view richiederà il servizio *GestioneDisponibilità* attraverso la routes *GestioneDisponibilitàRoutes* che a sua volta azionerà il controller *GestioneDisponibilitàController*. Esso accederà al model *Tamponi* inserendo il tampone nel database. Dunque il model *Tamponi* notificherà il controller e mostrerà l'avvenuta operazione. A questo punto, il controller *GestioneDisponibilitàController* inserirà gli orari relativi alla disponibilità del tampone appena inserito nel model *Orari*. Dopo la notifica di conferma della creazione degli orari dal model *Orari*, la view riceve la lista dei tamponi disponibili e la mostra alla farmacia. Se il controller *GestioneDisponibilitàController* verifica che le informazioni inserite dalla farmacia per il tampone non sono corrette, lo segnala ritornando un errore alla view che mostra il messaggio di errore che chiede di reinserire le informazioni.

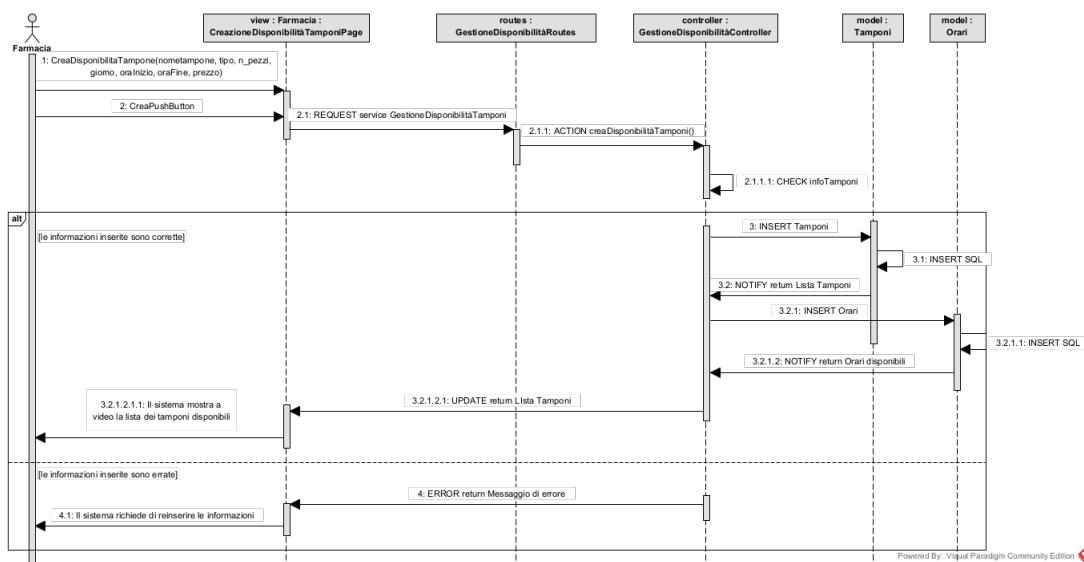


Figura 6.8: Sequence Diagram di dettaglio - Creazione disponibilità tampone

6.5 Design Pattern

I Design Pattern permettono di definire "una soluzione progettuale generale ad un problema ricorrente". In particolare, si tratta di una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi del ciclo di vita del software, ancor prima della definizione dell'algoritmo risolutivo della parte computazionale, e quindi **indipendente dall'approccio di sviluppo OOP o meno**. È un approccio spesso efficace nel **diminuire i difetti che si presentano nel codice**, in quanto ne permettono di identificarli in anticipo.

I Design Pattern possono essere classificati con diversi criteri, i più comuni dei quali sono quelli che evidenziano il tipo di problema che si cerca di risolvere. Il tipo di problema può essere legato ad uno specifico dominio progettuale (come nel caso in esame, relativo ad una Web App) oppure, più comunemente, al problema progettuale in senso più ampio (ad esempio, relativo alla creazione, comportamento, navigazione di strutture dati come un DB).

Nel loro libro la "Gang of Four" identificò 23 tipi di Design pattern, suddivisi in tre categorie:

- Strutturali

- Creazionali
- Comportamentali

6.5.1 Pattern Grasp

Si definiscono le responsabilità delle "classi", intese come componenti del sistema software, in particolare:

- Creator: con lo scopo di definire la responsabilità di creare specifici oggetti.
 - Si considera la *Farmacia* come creator in quanto responsabile alla creazione dei *Tamponi (rapidi/molecolari)*
 - Si considera l'*Admin* come creator in quanto responsabile alla creazione delle *Farmacie*
 - Si considera il *Paziente* come creator in quanto responsabile della creazione della *Prenotazione*
- Information Expert: con lo scopo di definire la responsabilità di conoscere un determinato oggetto a partire dal suo identificatore
 - Si considera l'*Admin* come information expert in quanto conosce tutte le farmacie
- Controller: con lo scopo di definire la responsabilità di ricevere e controllare la gestione di una richiesta dalla User Interface (dalla View)
 - Si considera la *GestioneDisponibilitàTamponi* come controller in quanto riceve e controlla la gestione di una richiesta di disponibilità tamponi dal *PazienteRegistrato*
 - Si considera la *GestioneEsitiTamponi* come controller in quanto riceve e controlla la gestione di una richiesta dell'esito del tampone dal *PazienteRegistrato*
 - Si considera la *GestioneFarmacia* come controller in quanto riceve e controlla la gestione di una richiesta di gestione della farmacia dall'*Admin*
 - Si considera la *GestionePazienteRegistrato* come controller in quanto riceve e controlla la gestione di una richiesta di gestione delle prenotazioni in *farmacia*
 - Si considera la *GestionePrenotazioni* come controller in quanto riceve e controlla la gestione di una richiesta di gestione delle prenotazioni dalla *farmacia*
 - Si considera il *MLModel* come controller in quanto riceve e controlla la gestione di una richiesta di inserimento sintomi, malattie e altre informazioni sul COVID19 dal *paziente*
 - Si considera il *MainHandlerController* come controller in quanto riceve e controlla le richieste di informazioni fatte sulla homepage dagli *attori* del sistema

6.5.2 Pattern Proxy

Il componente *Routes* implementa il pattern Proxy in quanto riceve le richieste dalla View e le ridirige ai Controller (Microservizi). In modo speciale nel caso in esame si è usata la libreria **Flask - Blueprint** utile per associare le richieste provenienti dalla View e reinstrarlarli verso i Controller. Questo Pattern ci permette di separare ed encapsulare le funzioni del sistema, in quanto le View alla ricezione di una particolare richiesta dall'utente non conosce i dettagli implementativi della funzionalità desiderata.

6.5.3 Pattern Observer

Si applica questo pattern per quanto riguarda le modifiche dei dati sugli *Esiti Tamponi* di una particolare *Prenotazione* da parte della *Farmacia* che notificherà alle classi View del *Paziente*, principalmente *dashPaziente*, aggiornando lo stato degli esiti e dunque le informazioni mostrate al paziente. Inoltre, all'aggiunta dell'esito del tampone da parte della Farmacia viene anche notificata la relativa disponibilità al Paziente tramite **Mail**.

6.6 Deployment Diagram - Locale

Per assecondare le scelte fatte nel paragrafo Vista Componenti e Connettori, si descrive il sistema in termini di risorse hardware, dette nodi, e di relazioni fra di esse. Si utilizza questo diagramma per mostrare come le componenti software siano distribuite rispetto alle risorse hardware disponibili sul sistema. Inizialmente si dimostra il deployment diagram a livello locale, successivamente, dopo il rilascio, si dimostrerà il deployment diagram a livello remoto.

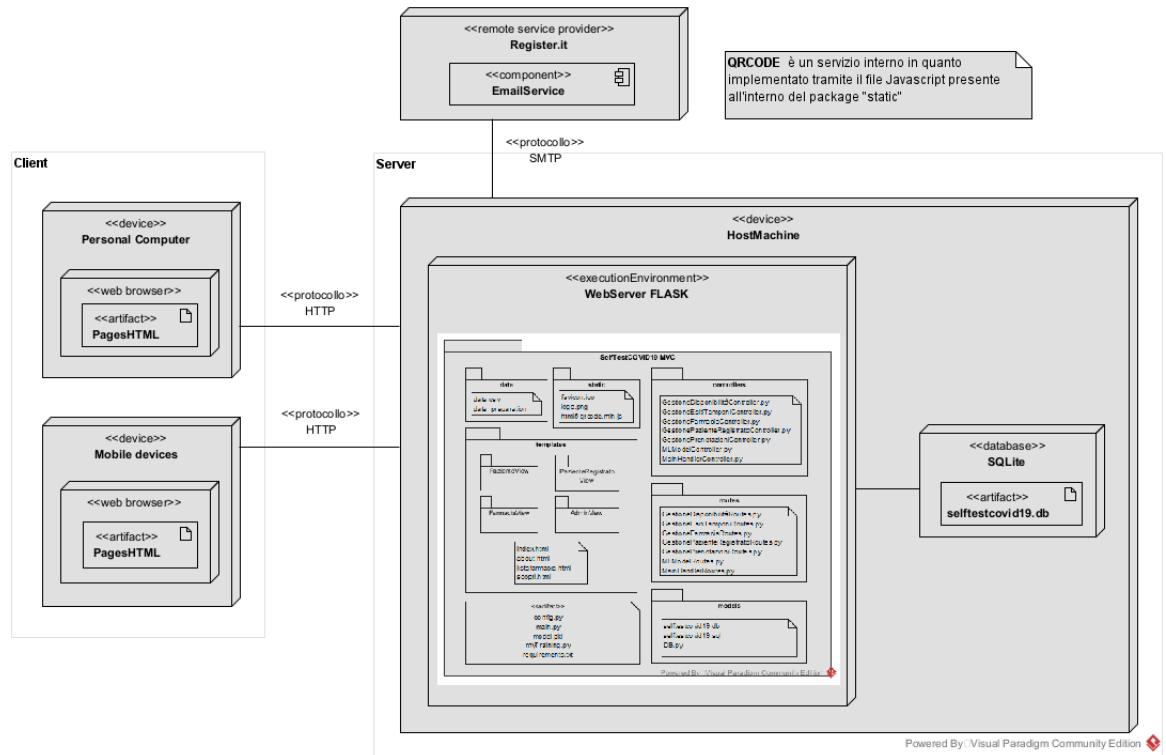


Figura 6.9: Deployment Diagram

Dalla Figura 6.9 si nota il **Client** che a livello hardware può essere sia un *Personal Computer* e sia un *Mobile Devices* (smartphone, tablet). Questi ultimi interagiscono con il **Server** attraverso il protocollo HTTP. Mostrando inizialmente il caso *locale*, il *Server* esegue sulla macchina locale dello sviluppatore tramite l'ambiente di esecuzione "WebServer Flask", al cui interno sono istanziate i package come visto già in Figura 6.4. L'ambiente di esecuzione è collegato al database "SQLite" ed al servizio di invio Mail di Register.it tramite protocollo SMTP.

6.7 Entity Relationship Diagram

Il modello entità-relazione è stato utilizzato nella fase di progettazione del database, nella quale è necessario tradurre le informazioni risultanti dall'analisi del dominio d'esame in uno schema concettuale, detto diagramma entità-relazione (o diagramma E-R Figura 6.10). Le **entità** del dominio sono:

- **Farmacie;**
- **Tamponi;**
- **Orari;**
- **Admin;**
- **Prenotazioni;**
- **Pazienti**

Le **relazioni** sono:

- **Farmacie - Tamponi** con cardinalità $1 - n$;
- **Farmacie - Prenotazioni** con cardinalità $1 - n$;
- **Pazienti - Prenotazioni** con cardinalità $1 - n$;
- **Tamponi - Orari** con cardinalità $1 - n$;
- **Prenotazioni - Tamponi** con cardinalità $1 - 1$

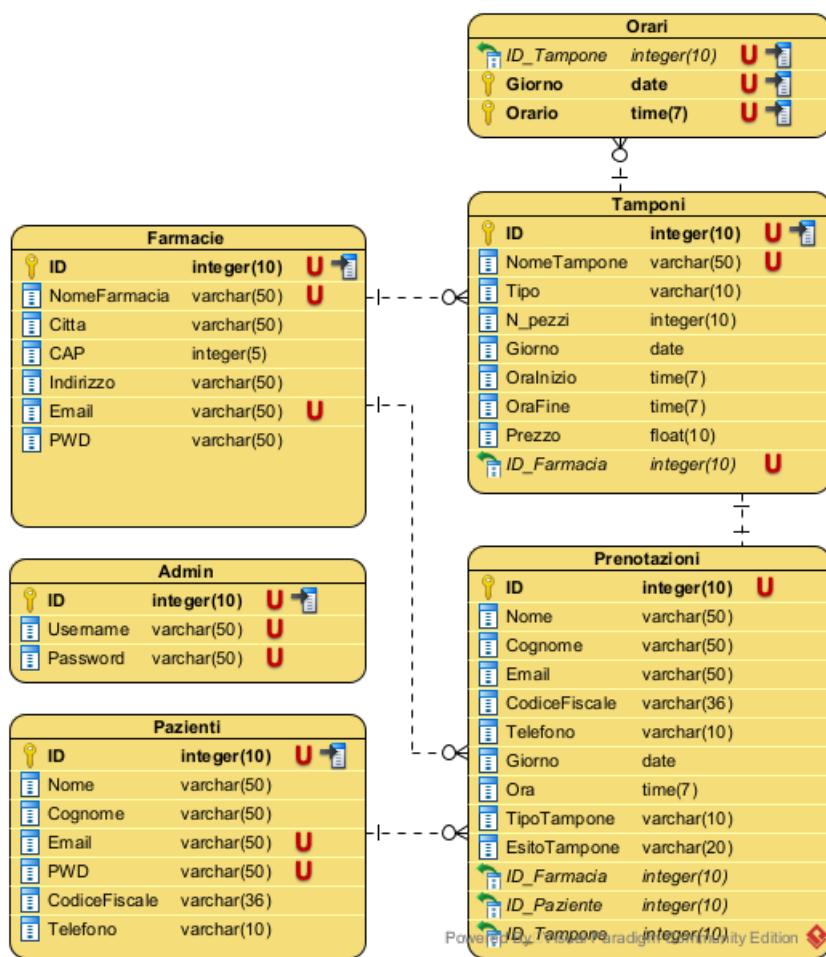


Figura 6.10: Entity Relationship Diagram

7 Testing

Si prova a dimostrare con la fase di testing, se un sistema soddisfa i suoi requisiti funzionali e non-funzionali. Il software viene sottoposto a test rigorosi prima del rilascio al fine di evidenziare le differenze tra il concetto (l'idea) originale e il risultato finale. Si verifica che il software funzioni come pianificato durante la fase di progettazione ed infine convalidare il prodotto finale. E' inoltre importante che il prodotto deve soddisfare i requisiti del cliente valutando le caratteristiche e la qualità del software.

Contenuti

7.1	Test funzionale	98
7.1.1	Test blackbox	98
7.1.2	Test di unità	102
7.1.3	Test di accettazione - Alfa Test	107
7.2	Test non funzionale	110
7.2.1	Test di carico	110

7.1 Test funzionale

Nel Test funzionale si utilizzano le specifiche funzionali (User Stories) fornite nella fase di analisi per verificare se il sistema li rispetta adeguatamente.

7.1.1 Test blackbox

Si controlla solo che il software faccia quello che è stato dichiarato nella definizione degli use case di dettaglio. La conoscenza della codifica non è necessaria poiché si valuta solo a livello di interfaccia utente e il risultato calcolato.

Tentativo di inserimento sintomi, malattie e informazioni relative al COVID19

- Condizioni di validità dell'input
 - Il paziente inserisce i sintomi, malattie e informazioni relative al COVID19 nel form mostrato dal sistema.
- Precondizioni:
 - Non ci sono precondizioni

Tentativo di verifica disponibilità farmacie

- Condizioni di validità dell'input
 - Il paziente inserisce il nome della farmacia o la città o il CAP di dove è ubicata la farmacia.
- Precondizioni:
 - Il risultato del SelfTest sia stato maggiore del 50% e minore del 75% per prenotare un tampone rapido oppure è stato maggiore uguale del 75% e minore uguale del 100% per prenotare un tampone molecolare.

TC	Descrizione	Input	Output	Esito del test
TC1	Inserimento sintomi, malattie e informazioni relative al COVID19 e restituzione del risultato calcolato dal ML Model di predizione	Sintomi, malattie e informazioni relative al COVID19	Risultato del test in percentuale	OK

Tabella 7.1: TC1 - Inserimento sintomi

TC	Descrizione	Input	Output	Esito del test
TC2	Verifica della disponibilità dei tamponi in farmacia	Nome farmacia, città o CAP	Lista delle farmacie con relativa disponibilità	OK

Tabella 7.2: TC2 - Verifica disponibilità farmacie

Tentativo di prenotazione tampone

- Condizioni di validità dell'input
 - Il paziente prenota il relativo tampone (rapido o molecolare) attraverso la registrazione se è un nuovo utente altrimenti utilizza le credenziali utilizzate alla precedente prenotazione.
- Precondizioni:
 - Il paziente ha ricercato la farmacia più vicina ad esso e verificato la disponibilità

TC	Descrizione	Input	Output	Esito del test
TC3	Prenotazione del relativo tampone (rapido o molecolare)	SE NUOVO PAZIENTE Nome, Cognome, Email, PWD, Telefono, Codice Fiscale, Giorno prenotazione, Ora prenotazione e dichiarare i termini della privacy altrimenti SE REGISTRATO IN PRECEDENZA Email, PWD, Giorno prenotazione, Ora prenotazione	Prenotazione avvenuta con successo	OK

Tabella 7.3: TC3 - Prenotazione tampone**Tentativo di richiesta esito tampone**

- Condizioni di validità dell'input
 - Il paziente dopo aver svolto il tampone in farmacia, accede al suo profilo e verifica l'esito nella lista dei referti con la possibilità di scaricare il certificato.
- Precondizioni:
 - Il paziente ha effettuato il tampone presso una farmacia

TC	Descrizione	Input	Output	Esito del test
TC4	Richiesta esito tampone	Email e PWD per accedere al suo profile	Esito tampone nella lista referti	OK

Tabella 7.4: TC4 - Richiesta esito tampone**Tentativo di richiesta lista prenotazioni**

- Condizioni di validità dell'input

- La farmacia accede alla propria dashboard e visualizza la lista delle prenotazioni giornaliere
- Precondizioni:
 - La farmacia ha fatto l'accesso tramite login farmacia

TC	Descrizione	Input	Output	Esito del test
TC5	Richiesta lista prenotazioni	Email e PWD per accedere alla dashboard	Dashboard della farmacia e lista delle prenotazioni	OK

Tabella 7.5: TC5 - Richiesta lista prenotazioni**Tentativo di creazione disponibilità tampone**

- Condizioni di validità dell'input
 - La farmacia attraverso la dashboard, crea la disponibilità dei tamponi rapidi/molecolari.
- Precondizioni:
 - La farmacia ha fatto l'accesso tramite login farmacia ed in loco sono arrivati i tamponi rapidi/molecolari.

TC	Descrizione	Input	Output	Esito del test
TC6	Creazione disponibilità tampone	NomeTampone, Tipo, npezzi, giorno, ora inizio, ora fine e prezzo	Nuovi tamponi disponibili e numero disponibilità relativa al tipo di tampone aggiornato	OK

Tabella 7.6: TC6 - Creazione disponibilità tampone**Tentativo di creazione farmacia**

- Condizioni di validità dell'input
 - L'admin dopo aver avuto l'accordo e la conferma che la farmacia aderisca al sistema, crea una nuova farmacia.
- Precondizioni:
 - L'admin ha fatto l'accesso alla dashboard

TC	Descrizione	Input	Output	Esito del test
TC7	Creazione farmacia	NomeFarmacia, città, CAP, indirizzo, email, password	Nuova farmacia aggiunta	OK

Tabella 7.7: TC7 - Richiesta esito tampone**Tentativo di ricerca farmacia**

- Condizioni di validità dell'input
 - L'admin cerca la farmacia per visualizzare le sue informazioni.
- Precondizioni:
 - L'admin ha fatto l'accesso alla dashboard

TC	Descrizione	Input	Output	Esito del test
TC8	Ricerca farmacia	NomeFarmacia, città, CAP	Farmacia trovata	OK

Tabella 7.8: TC8 - Richiesta esito tampone

7.1.2 Test di unità

I test di unità vengono effettuati sulle singole unità di un programma ed hanno l'obiettivo di esaminare i singoli moduli in maniera isolata rispetto agli altri. In particolare, i test di unità si focalizzano sulla funzionalità e affidabilità del software. Grazie al testing di unità possono essere scovati dei difetti, permettendone la valutazione preventiva sulla loro natura e sull'impatto che questi possono avere sul Sistema generale; evitando così di scoprirli troppo tardi in fase di integrazione o addirittura di rilascio.

Per eseguire i test di unità nel sistema "SelfTestCOVID19" è stato usato il framework **Pytest**, tale framework appositamente pensato per testare moduli software scritti in Python è altamente scalabile e flessibile rispetto alle esigenze dello sviluppatore. Al fine di dare continuità alla trattazione si è deciso di testare in maniera approfondita solo le funzionalità di cui finora si sono mostrati i diagrammi di dettaglio, ovvero:

- Inserimento Sintomi
- Creazione disponibilità tamponi
- Prenotazione tampone
- Verifica disponibilità farmacie

UT1 - Inserimento Sintomi

Per il testing di unità di questo requisito, si è deciso di implementare due casi di test differenti, uno in cui ci si aspetta che il test restituisca esito positivo, mentre nell'altro caso il test deve restituire esito negativo.

Nel primo caso vengono passate al software i sintomi del paziente in formato compatibile con quello richiesto dall'interfaccia, ovvero, attraverso valori booleani (0,1), es. 'Breathing Problem' : '0', 'Fever' : '1', 'Dry Cough' : '1', 'Sore throat' : '0', etc.

Nell'altro caso si sono lasciati dei sintomi con campo vuoto il che solleva un'eccezione dal momento che il sistema si aspetta un valore pari a (0,1) e riceve invece un valore nullo, es. 'Breathing Problem' : "", 'Fever' : '1', 'Dry Cough' : '1', 'Sore throat' : "", etc.

UT2 - Creazione disponibilità tamponi

Allo stesso modo visto prima, si sono implementati due casi di test anche per la funzionalità di creazione disponibilità tamponi, in particolare, per ottenere il risultato positivo si sono inserite delle informazioni relative ad un tampone non esistente già nel Sistema.

Nel caso con esito negativo, si sono inserite delle informazioni relative ad un tampone già presente nel sistema, quindi, in questo scenario il Sistema non inserisce la disponibilità richiesta.

UT3 - Prenotazione tampone

Per la funzionalità di prenotazione del tampone si sono implementati quattro casi di test, due nel caso di un Nuovo Paziente del Sistema, ed altri 2 nel caso di Paziente Registrato. Nel caso di Nuovo Paziente, il caso con esito positivo è quello in cui il paziente effettivamente non è presente nel Sistema, invece, se il paziente in realtà è presente sulla piattaforma viene generato un messaggio di errore.

Per la variante con il Paziente Registrato, i casi di successo e fallimento del Sistema si ottengono invertendo le condizioni esplicitate prima.

UT4 - Verifica disponibilità farmacie

Per la funzionalità di verifica della disponibilità delle farmacie si sono implementati quattro casi di test, infatti, abbiamo che il paziente può verificare la disponibilità sia di tamponi rapidi che molecolari. Sia per il caso dei tamponi rapidi che per quelli molecolari, il caso con esito positivo si ottiene se la ricerca della farmacia con disponibilità restituisce risultati, mentre, il caso con esito negativo è ottenuto quando la ricerca non da nessun risultato.

Automatizzazione dei Test di Unità tramite linguaggio YAML

Descritti i 12 casi di test implementati il cui codice, come quello di tutto il sistema, è lasciato al lettore nel repository di GitHub. Si è deciso di automatizzare la fase del test di unità tramite un workflow scritto con il **linguaggio YAML**, tale linguaggio ha come caratteristica distintiva la leggibilità, infatti, esso deriva da vari linguaggi di programmazione come Perl, C e Python; il linguaggio YAML permette di scrivere degli script che automatizzano alcune delle fasi del ciclo di vita del software.

Per eseguire questa automatizzazione del testing si è fatto uso della piattaforma **GitHub Actions**, la quale mette a disposizione delle risorse Cloud sul quale gli script scritti, come nel caso in esame, in YAML vengono eseguiti in maniera automatica all'accadere di un determinato evento come ad esempio, un PUSH sul repository.

Si è dunque, aggiunto lo script YAML che ad ogni push sul repository di progetto, lancia su una macchina Ubuntu il test di unità scritto con Pytest. Andando ad osservare cosa accade sul repository notiamo la seguente situazione.

La piattaforma ha rilevato un'operazione di push sul repository quindi, ha attivato lo script YAML, il quale viene eseguito su 3 macchine Cloud con differenti versioni di Python (3.8, 3.9, 3.10) per testare anche la compatibilità dei moduli software con versioni differenti, Figura 7.2.

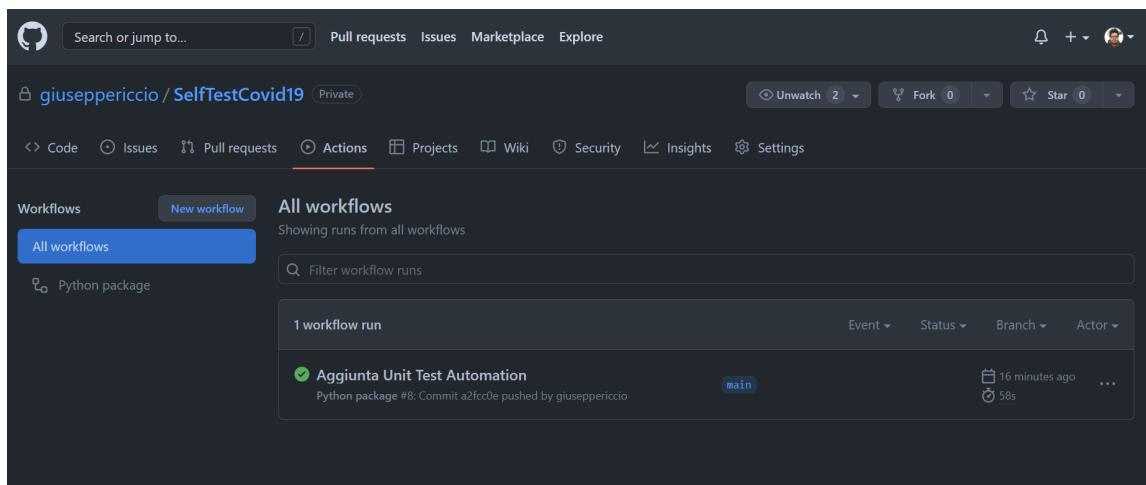


Figura 7.1: GitHub Actions

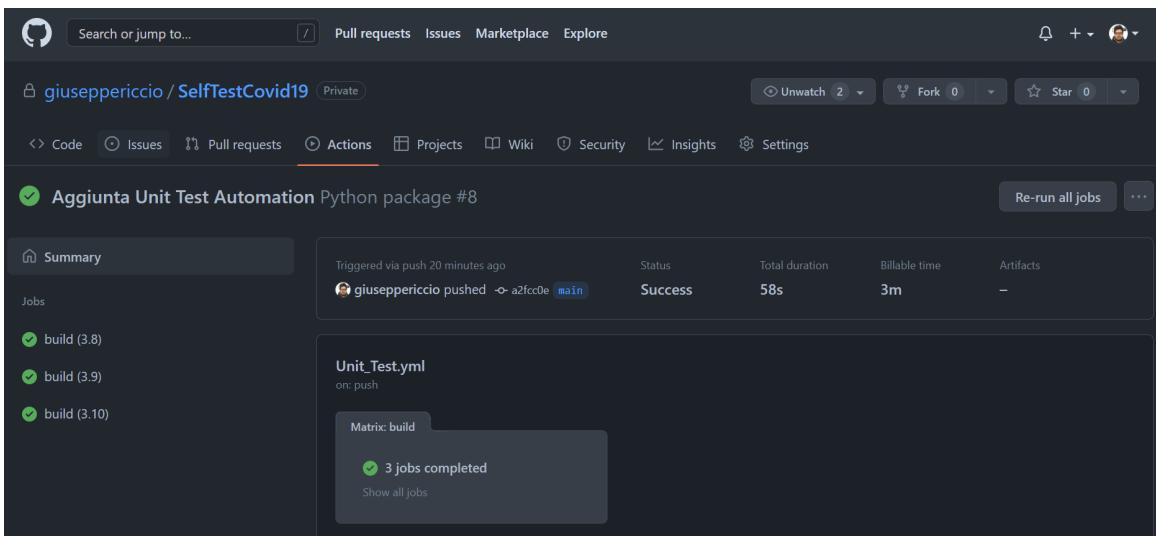


Figura 7.2: GitHub Workflow

Ciascun build si compone di diverse fasi, come si nota dalla Figura 7.3.

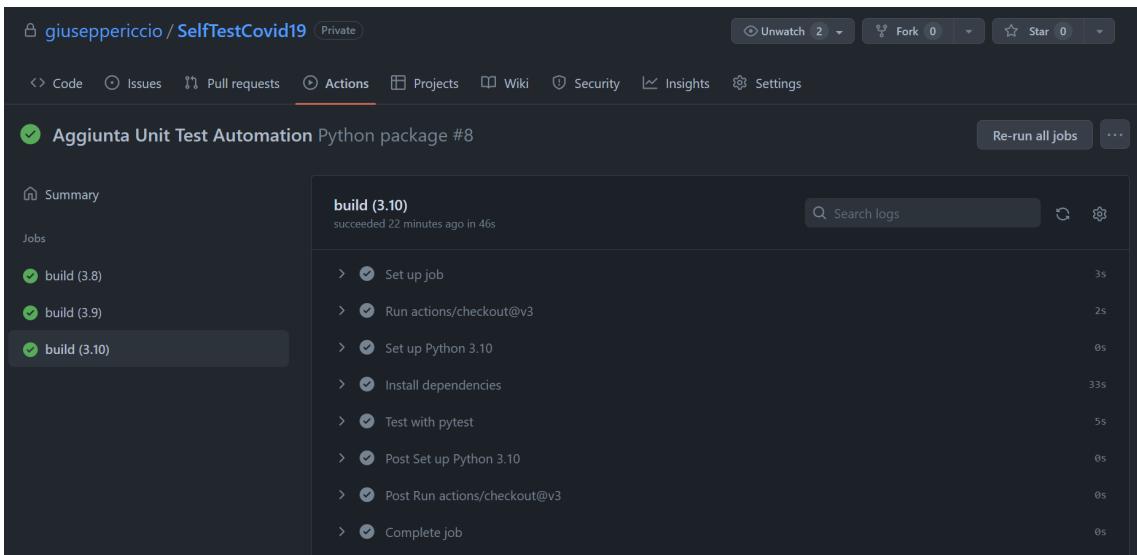


Figura 7.3: GitHub Workflow

Al termine della build, se il test ha avuto successo viene visualizzata la spunta verde e aprendo il build sulla fase "Test with pytest", si può vedere il dettaglio dei test eseguiti e del loro risultato, Figura 7.4.

Tutti i Test hanno avuto esito positivo.

Codice YAML per l'automazione del test:

```
https://github.com/giuseppericcio/SelfTestCovid19/blob/main/.github/
    workflows/Unit_Test.yml
```

Codice Python per il test d'unità

```
https://github.com/giuseppericcio/SelfTestCovid19/blob/main/tests/unit_
    test.py
```

```
===== test session starts =====
platform win32 -- Python 3.10.6, pytest-7.1.3, pluggy-1.0.0 -- C:\Users\giuse\AppData\Local\Programs\Python\Python310\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\giuse\Desktop\SelfTestCovid19
collected 12 items

tests/unit_test.py::test_InserimentoSintomi_corretto PASSED [ 8%]
tests/unit_test.py::test_InserimentoSintomi_sbagliato PASSED [16%]
tests/unit_test.py::test_prenotaNuovo_corretto PASSED [25%]
tests/unit_test.py::test_prenotaNuovo_sbagliato PASSED [33%]
tests/unit_test.py::test_prenotaRegistrato_corretto PASSED [41%]
tests/unit_test.py::test_prenotaRegistrato_sbagliato PASSED [50%]
tests/unit_test.py::test_dispRapido_corretto PASSED [58%]
tests/unit_test.py::test_dispRapido_sbagliato PASSED [66%]
tests/unit_test.py::test_dispMolecolare_corretto PASSED [75%]
tests/unit_test.py::test_dispMolecolare_sbagliato PASSED [83%]
tests/unit_test.py::test_InserimentoSintomi_sbagliato PASSED [16%]
tests/unit_test.py::test_prenotaNuovo_corretto PASSED [25%]
tests/unit_test.py::test_prenotaNuovo_sbagliato PASSED [33%]
tests/unit_test.py::test_prenotaRegistrato_corretto PASSED [41%]
tests/unit_test.py::test_prenotaRegistrato_sbagliato PASSED [50%]
tests/unit_test.py::test_dispRapido_corretto PASSED [58%]
tests/unit_test.py::test_dispRapido_sbagliato PASSED [66%]
tests/unit_test.py::test_dispMolecolare_corretto PASSED [75%]
tests/unit_test.py::test_dispMolecolare_sbagliato PASSED [83%]
tests/unit_test.py::test_CreaDisponibilitaTampone_corretto PASSED [91%]
tests/unit_test.py::test_CreaDisponibilitaTampone_sbagliato PASSED [100%]

===== 12 passed in 2.15s =====
```

Figura 7.4: Test di Unità

7.1.3 Test di accettazione - Alfa Test

Si testa il software su ambiente locale, si creano dei piani di test tali da scoprire bug o altri problemi. Successivamente, gli sviluppatori correggeranno gli eventuali errori e si rifarà il test. Si sviluppano dei test case (TAAT:Test di Accettazione Alfa Test) per costruire il piano di test ALFA TEST nr.1 :

- TAAT1. Controllo inserimento tamponi
- TAAT2. Controllo disponibilità orari farmacia (DASHBOARD FARMACIA)
- TAAT3. Controllo modifica data prenotazioni farmacia (DASHBOARD FARMACIA)
- TAAT4. Controllo inserimento tamponi (DASHBOARD FARMACIA)
- TAAT5. Controllo messaggio di avvenuta conferma della operazione (DASHBOARD FARMACIA)
- TAAT6. Controllo aggiunta tamponi (DASHBOARD PAZIENTE REGISTRATO)

Per dimostrazione, si mostra anche il test case per costruire il piano di test ALFA TEST nr.2 mostrando la risoluzione di alcuni test :

- TAAT2. Controllo disponibilità orari farmacia (DASHBOARD FARMACIA)
- TAAT3. Controllo modifica data prenotazioni farmacia (DASHBOARD FARMACIA)
- TAAT4. Controllo inserimento tamponi (DASHBOARD FARMACIA)
- TAAT5. Controllo messaggio di avvenuta conferma della operazione (DASHBOARD FARMACIA)
- TAAT6. Controllo aggiunta tamponi (DASHBOARD PAZIENTE REGISTRATO)
- TAAT7. *Controllo aggiornamento dataset SelfTestCOVID19 (DASHBOARD ADMIN)*

Si può notare che dalla Tabella 7.10, si è aggiunto un **TAAT7** che ha avuto esito negativo, mentre **TAAT5**, **TAAT4** hanno avuto esito positivo, pertanto al prossimo alfa test verranno rimossi come **TAAT1**.

TAAT	Descrizione	Input	Output atteso	Output risultante	Esito confronto
TAAT1	Controllo inserimento tamponi	Sintomi, malattie e informazioni sul COVID19	Risultato in percentuale del SelfTest	Risultato in percentuale del SelfTest	OK
TAAT2	Controllo disponibilità orari farmacia	Form orario della prenotazione	Lista orari disponibili a partire dall'orario momentaneo	Lista orari disponibili	NO
TAAT3	Controllo modifica data prenotazioni farmacia	Modifica data delle prenotazioni	SE LA DATA è già presente nel database il sistema non permette la modifica, SE LA DATA NON è presente nel database, il sistema permette la modifica	Il sistema permette la modifica in entrambi casi	NO
TAAT4	Controllo inserimento tamponi	Nome Tampone, Tipo, N-prezzi, Giorno e orari	Inserimento non duplicato nel database	Inserimento duplicato nel database	NO
TAAT5	Controllo messaggio di avvenuta conferma della operazione	form delle rispettive pagine di inserimento e modifica	SE LA MODIFICA O L'INSEMENTO ha contenuto già presente nel database, il sistema segnala l'errore e non inserirà o modificherà nel database	Dopo la MODIFICA e L'INSEMENTO il sistema NON segnala errore e inserisce e modifica le entry nel database	NO
TAAT6	Controllo aggiunta tamponi	form di inserimento sintomi, malattie e informazioni COVID19	Avvenuto caricamento sintomi, malattie e informazioni COVID19 nel dataset.csv	Non è presente la funzionalità	NO

Tabella 7.9: TAAT - ALFA TEST 1

TAAT	Descrizione	Input	Output atteso	Output risultante	Esito confronto
TAAT2	Controllo disponibilità orari farmacia	Form orario della prenotazione	Lista orari disponibili a partire dall'orario momentaneo	Lista orari disponibili	NO
TAAT3	Controllo modifica data prenotazioni farmacia	Modifica data delle prenotazioni	SE LA DATA è già presente nel database il sistema non permette la modifica, SE LA DATA NON è presente nel database, il sistema permette la modifica	Il sistema permette la modifica in entrambi casi	NO
TAAT4	Controllo inserimento tamponi	Nome Tampone, Tipo, N-prezzi, Giorno e orari	Inserimento non duplicato nel database	Inserimento duplicato nel database	OK
TAAT5	Controllo messaggio di avvenuta conferma della operazione	form delle rispettive pagine di inserimento e modifica	SE LA MODIFICA O L'INSERIMENTO ha contenuto già presente nel database, il sistema segnala l'errore e non inserirà o modificherà nel database	Dopo la MODIFICA e L'INSERIMENTO il sistema NON segnala errore e inserisce e modifica le entry nel database	OK
TAAT6	Controllo aggiunta tamponi	form di inserimento sintomi, malattie e informazioni COVID19	Avvenuto caricamento sintomi, malattie e informazioni COVID19 nel dataset.csv	Non è presente la funzionalità	NO
TAAT7	Controllo aggiornamento sintomi	dataset.csv opportunamente trattato nelle fasi di Machine Learning	Avvenuto aggiornamento del dataset.csv	Non è presente la funzionalità	NO

Tabella 7.10: TAAT - ALFA TEST 2

7.2 Test non funzionale

I test non funzionali verificano le prestazioni, l'affidabilità, la scalabilità e altri aspetti non funzionali del sistema software.

7.2.1 Test di carico

Dunque, come anticipato, oltre ai test sulle funzionalità richieste al software in questo paragrafo approfondiamo gli aspetti relativi al test delle prestazioni; si tratta di un test approfondito che esamina le prestazioni del software in diversi scenari. Vengono raccolte informazioni sulla stabilità e reattività del software. In particolare, faremo riferimento a due parametri per valutare la bontà o meno delle prestazioni ottenute, tali parametri sono **Richieste per secondo (RPS)** e **Tempi di risposta (RT)**.

Per i test di carico andiamo a considerare il software ristretto alle funzioni considerate a priorità più alta, ovvero quelle specificate già nel paragrafo 4.4. Per simulare il software sotto carico si fa uso del tool **Locust**, tale strumento è un framework di test di carico scalabile scritto in Python, che permette di definire il comportamento degli utenti del sistema attraverso un normale codice Python, invece di interagire con una UI. Questo rende Locust infinitamente espandibile e molto intuitivo per gli sviluppatori.

Quindi, tornando al nostro test, si è scelto di mettere sotto carico le seguenti funzioni implementate all'interno della Web Application:

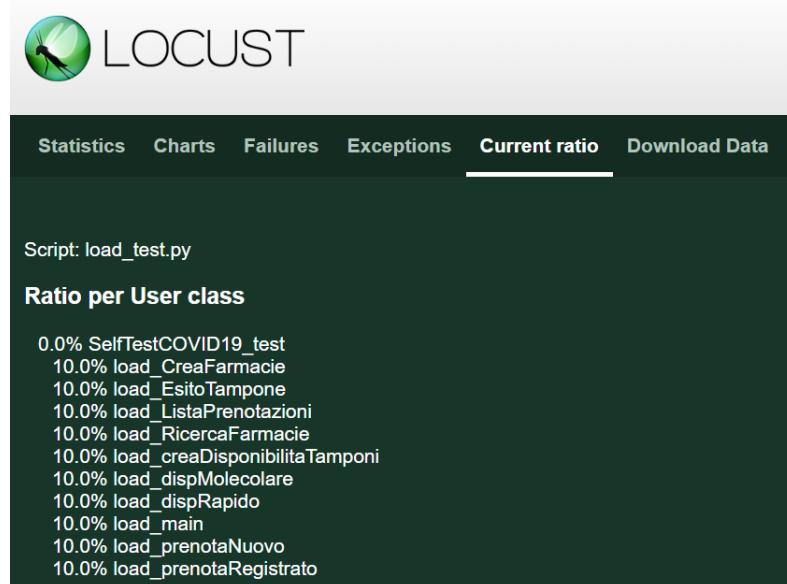


Figura 7.5: Copertura dei Test di carico

Dalla Figura 7.5, si evince che il tool ripartisce il carico di lavoro sul software in misura uguale per ogni funzione testata.

Per simulare condizioni di funzionamento del sistema più possibilmente realistiche si è deciso di simulare 3 tipi di carico differenti, il primo è un **basso carico** infatti si è ipotizzato che ci siano solo 10 utenti che usano il software ed i risultati ottenuti sono i seguenti:

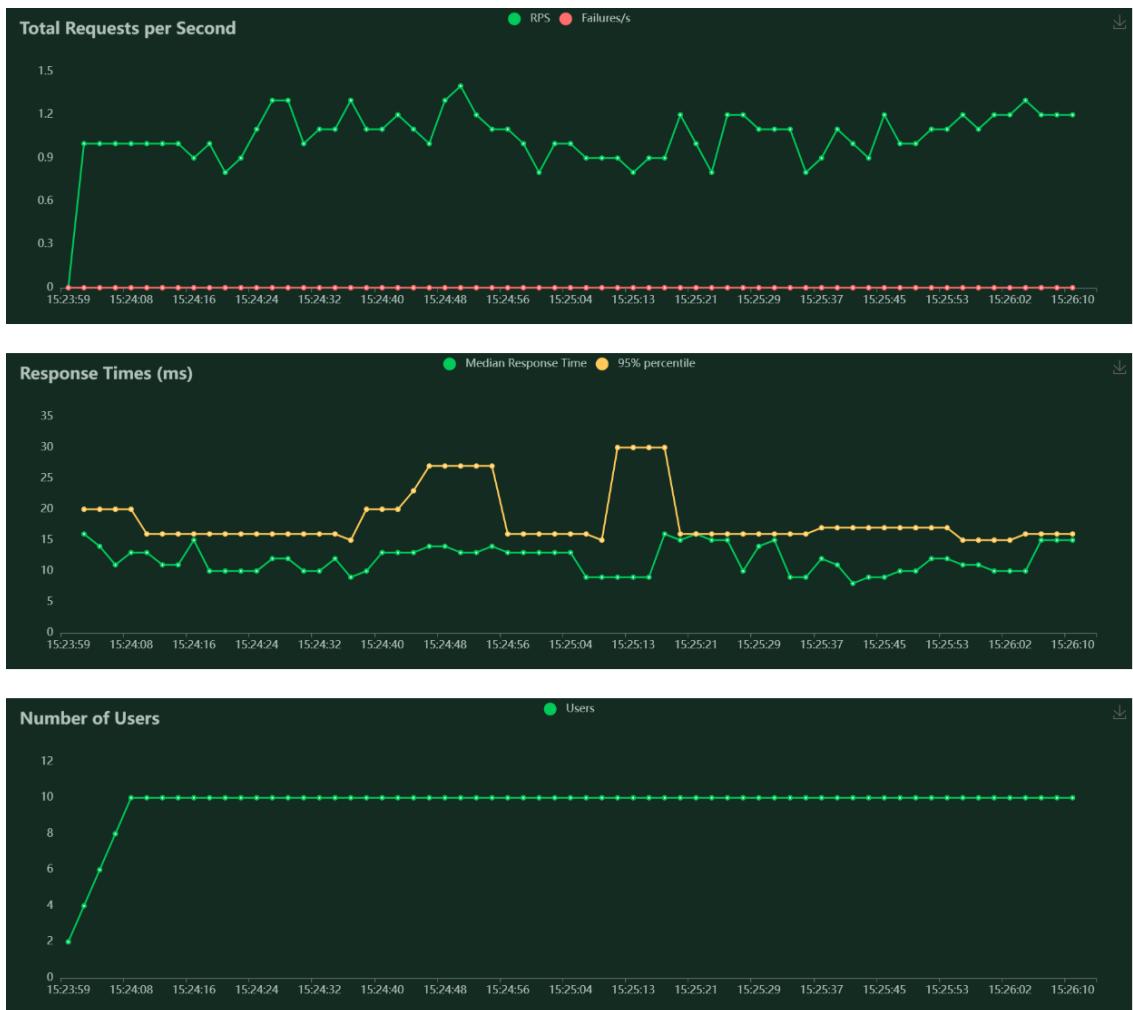


Figura 7.6: Test di carico con 10 utenti

Il tempo di esecuzione del test è di 2 minuti, in cui si è ottenuto che le **RPS** sono mediamente intorno all'1 con un picco di 1.4, e dunque, il **RT** è dell'ordine di 15-18ms il che ci permette di dire che il software si comporta bene in queste condizioni. Continuiamo la simulazione con un **carico medio** di 100 utenti per un arco temporale di poco più di 2 minuti:

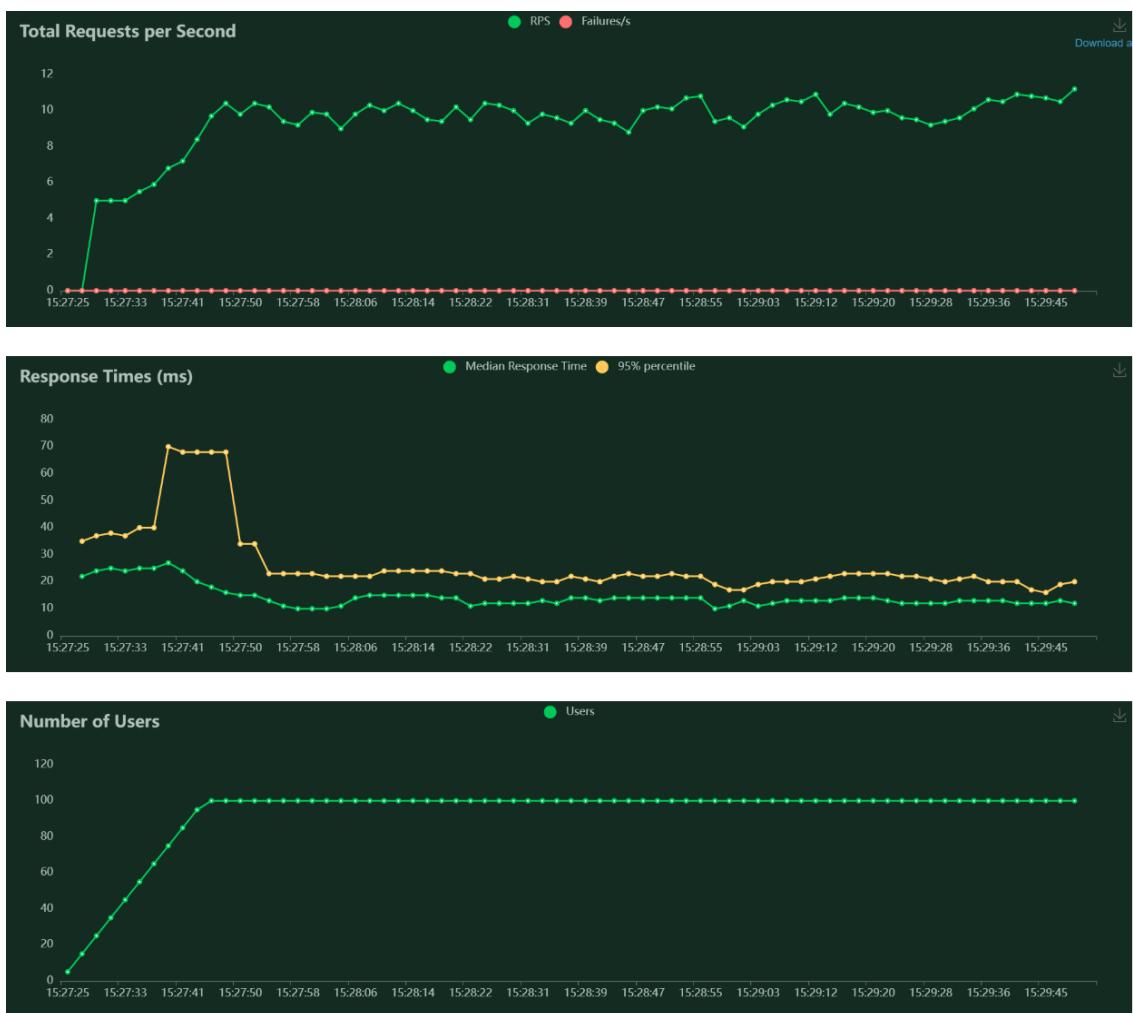


Figura 7.7: Test di carico con 100 utenti

In questo caso, le **RPS** sono mediamente di 10 e dunque anche il **RT** è salito rispetto alla simulazione precedente ad un valore medio di circa 24-25ms ed un valore di picco nella fase iniziale di addirittura 70ms. Le prestazioni, tuttavia, risultano ancora accettabili e non influiscono sull'usabilità finale del software da parte dell'utente. L'ultima simulazione è quella relativa ad un **alto carico** dove si hanno 1000 utenti ed un tempo di simulazione, come nei casi precedenti superiore di poco ai 2 minuti:

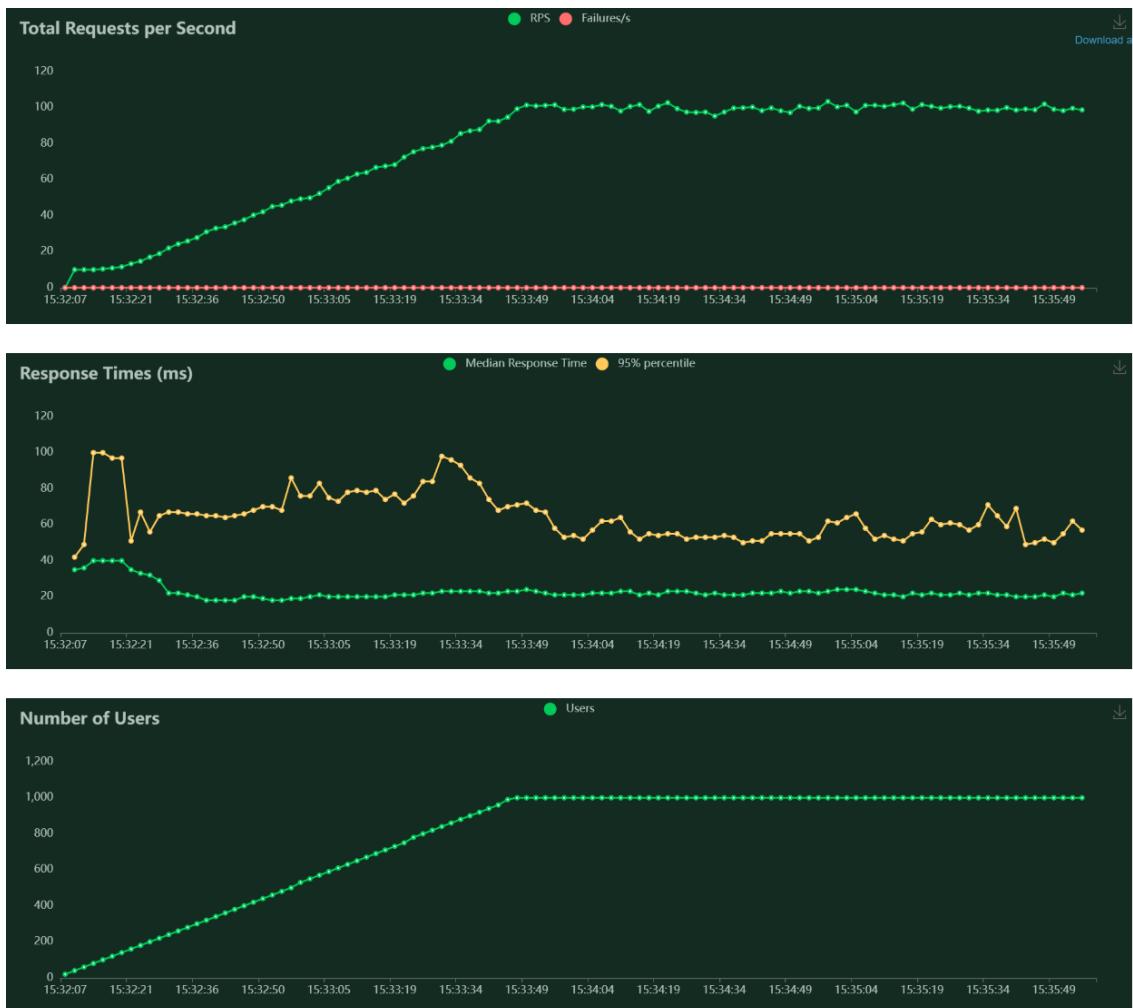


Figura 7.8: Test di carico con 1000 utenti

Nell'ultima simulazione si sono ottenuti un **RPS** di all'incirca 100, il che ci fa capire che il software viene posto sotto stress in maniera più pesante rispetto alle simulazioni precedenti, tuttavia, il **RT** non aumenta in maniera sconsiderata rimanendo su un valore medio di 70ms e con un valore di picco di 100ms.

In definitiva, anche in quest'ultima simulazione si può essere soddisfatti delle prestazioni ottenute dal software che anche con un carico maggiore riesce a soddisfare le richieste degli utenti in un tempo ragionevole e soprattutto senza intaccare la fruibilità generale del sistema.

Codice Python per il test di carico

https://github.com/giuseppericcio/SelfTestCovid19/blob/main/tests/load_test.py

8 Rilascio del software

La fase in cui viene rilasciato quanto fatto durante la realizzazione del software e reso disponibile, anche a livello prototipale, il progetto

Contenuti

8.1 Rilascio - PythonAnywhere	114
8.1.1 Gestione delle collisioni e degli errori di esecuzione	114
8.1.2 Potenza di calcolo	116
8.1.3 Il funzionamento generale	116
8.2 Performance Test - post Rilascio	117
8.3 Test di accettazione - Beta Test - post Rilascio	119
8.4 Deployment Diagram - Remoto	120

8.1 Rilascio - PythonAnywhere

Per rilasciare e distribuire il prodotto software realizzato si è fatto uso di un *ambiente di sviluppo integrato online* e di un servizio di web hosting (*Platform as a service*) basato su WSGI, come nel caso in esame FLASK. Il piano di distribuzione mette a disposizione sia la potenza di calcolo di esecuzione dei componenti di cui la web application è composta, sia la gestione dei possibili conflitti che vi possono essere nell'accedere alla web application. Lo spazio disponibile è di 1 GB.

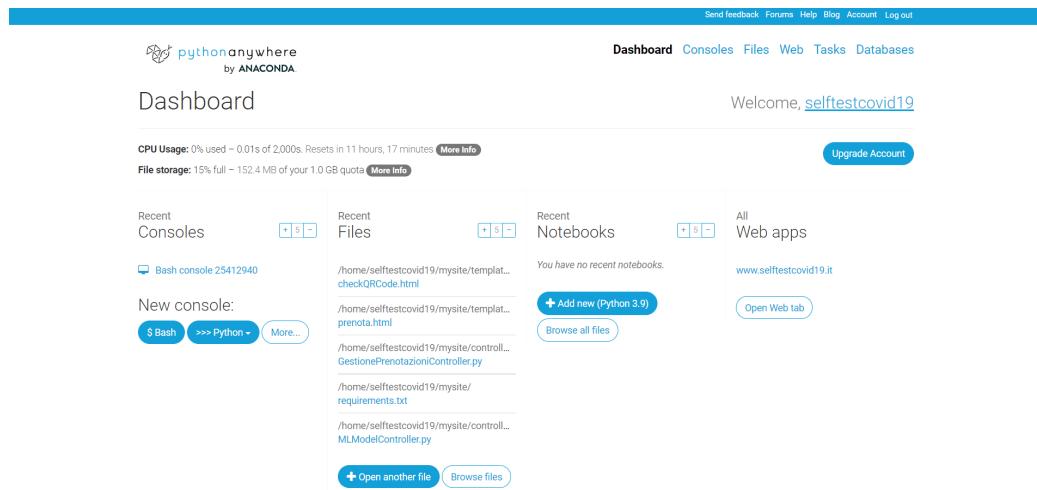


Figura 8.1: Dashboard principale - PythonAnywhere

8.1.1 Gestione delle collisioni e degli errori di esecuzione

Per la gestione delle collisioni e dunque del traffico di rete si fa uso dei **Web Worker** messi a disposizione da *PythonAnywhere*. Ogni web worker è un processo indipendente in grado di servire l'intera applicazione, quindi se ci sono più richieste, mentre uno è impegnato a gestire una richiesta, un altro può impegnarsi per l'altra richiesta. Se si dispone di un solo Web Worker (o se tutti i Web Worker sono occupati), le richieste vengono accodate e

servite quando un lavoratore diventa disponibile. Sarà poi possibile gestire gli errori e visualizzare i log facilmente come la seguente Figura 8.2. In particolare è possibile gestire direttamente in console come se fosse in locale

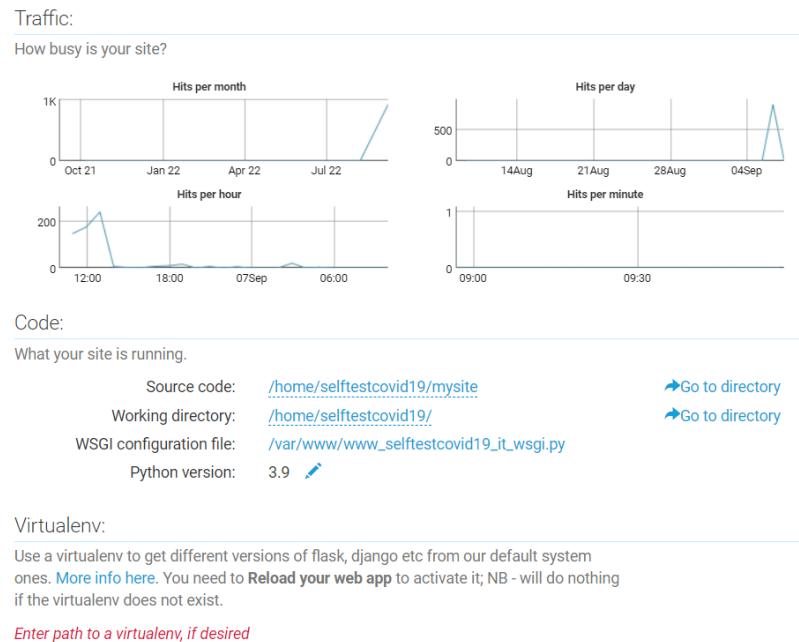


Figura 8.2: Gestione Traffico ed errori - PythonAnywhere

(Figura 8.3)

```

Bash console SELFTESTCOVID19
Share with others
15:39 ~/mysite $ ls
pycache config.py controllers data main.py model.pkl models myTrain
07:55 ~/mysite $ pip list
Package           Version
-----
absl-py           0.11.0
aggdraw           1.3.12
alabaster         0.7.12
alembic            1.4.3
aniso8601         8.1.0
appdirs             1.4.4
arabic-reshaper   2.1.3
argcomplete        1.12.2
argon2-cffi       20.1.0
arrow              0.17.0
arviz              0.11.2
asgiref            3.3.4
asn1crypto          1.4.0
astor              0.8.1
astropy             4.2
astunparse          1.6.3
async               0.6.2
async-generator    1.10
atomicwrites        1.4.0

```

Figura 8.3: Console - PythonAnywhere

8.1.2 Potenza di calcolo

Il piano di rilascio scelto per il prodotto software in esame prevede l'utilizzo di **numeri di secondi CPU al giorno**. Un secondo CPU è un secondo di utilizzo a piena potenza su un processore Intel Xeon E5-2670 v2 (Ivy Bridge) ad alta frequenza (un core CPU su un'istanza *Amazon AWS*). Se l'operazione che si sta eseguendo non utilizza alcuna potenza della CPU (forse non è in esecuzione, o è in attesa di input o di una richiesta web da restituire), allora non sta usando alcun secondo della CPU. Tuttavia, anche se si esauriscono i secondi CPU inclusi nel piano, PythonAnywhere mette a disposizione un processo con priorità inferiore detto **Tarpit**, il risultato sarà il medesimo ma più lento.

8.1.3 Il funzionamento generale

Da *locale*, si carica tutto il *package: SelfTestCOVID19* in remoto. Verrà effettuato l'upload della Web Application nei Server di PythonAnywhere (Amazon EC2, macchina virtuale di Amazon configurata dal servizio in questione per eseguire le applicazioni Web). Si lancia l'ambiente (reload nomewebapp.domino) e verrà pubblicato. Sarà dunque poi aggiornabile attraverso l'ambiente di sviluppo messo sempre a disposizione da PythonAnywhere e si ripeterà il ciclo. Il funzionamento che si è seguito è raffigurato in Figura 8.4. Dal locale

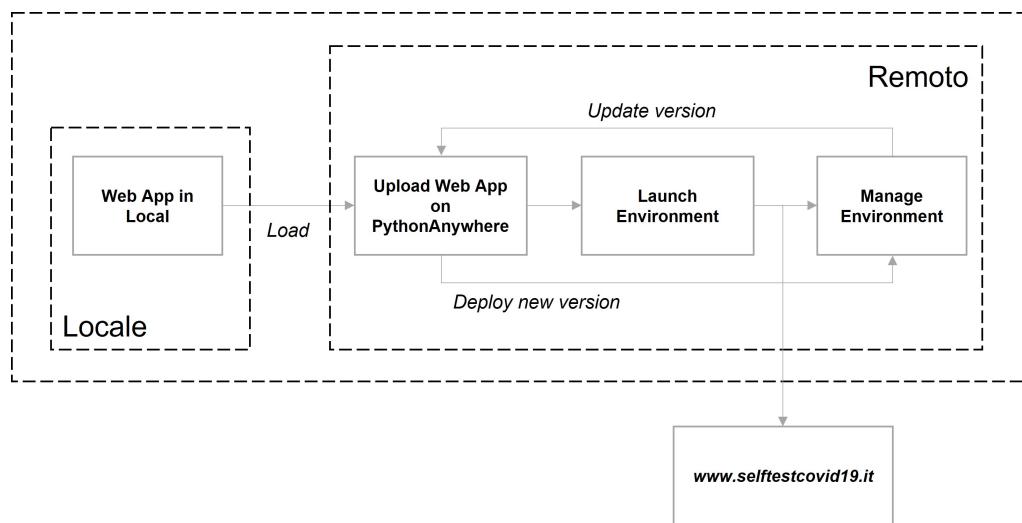


Figura 8.4: Funzionamento - PythonAnywhere

8.2 Performance Test - post Rilascio

Si valutano le prestazioni complessive della web application post rilascio. Riflette, sia, quanto velocemente la web app è stata caricata per gli utilizzatori, sia quanto bene è costruita a livello di prestazioni. La valutazione è stata fatta grazie il tool: **GTmetrix** con la configurazione del Server situato in Canada e browser Chrome/Desktop) vers.103.



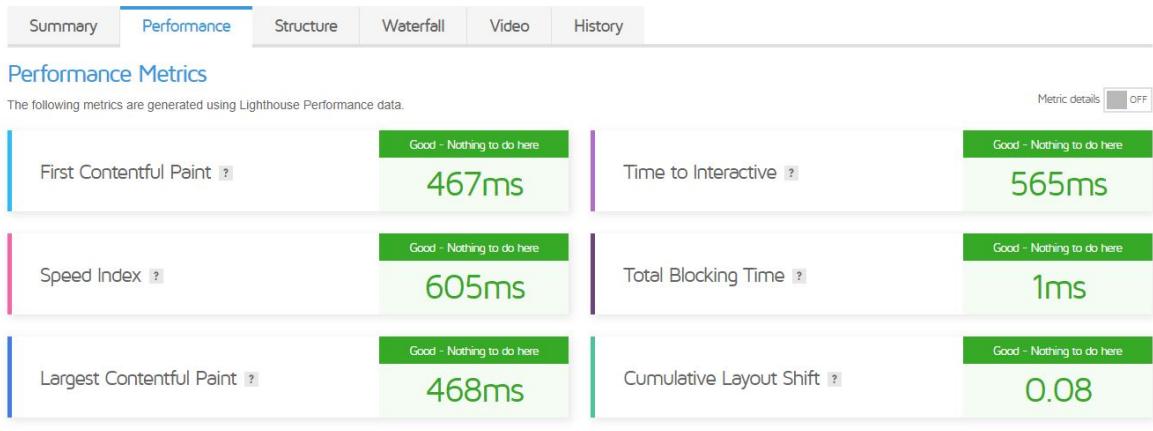
Figura 8.5: Valutazione prestazioni complessive della web application - post rilascio

Dalla Figura 8.5 si nota in breve come le prestazioni complessive della web application siano **ottime** risultando di grado A. Risultano inoltre ottime le performance e ottima struttura (rappresenta quanto bene la pagina è costruita per prestazioni ottimali).

Si mostrano, inoltre, le metriche valutate:

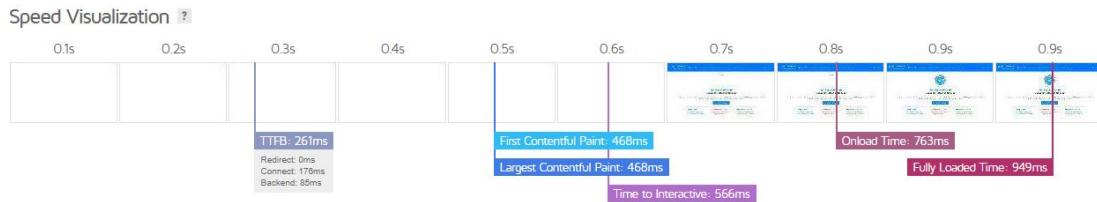
- **First Contentful Paint:** Quanto velocemente i contenuti come testo o immagini vengono mostrati dal browser. (Buona esperienza utente sotto i 0.9 s)
- **Time to interactive:** Quanto tempo ci vuole purchè la pagina diventi completamente interattiva. (Buona esperienza utente sotto i 2.5 s)
- **Speed Index:** Quanto velocemente TUTTI i contenuti della pagina sono visibilmente mostrati dal browser (Buona esperienza utente sotto i 1.3s)
- **Total blocking time:** In quanto tempo vengono eseguiti gli script durante il processo di caricamento della pagina. (Buona esperienza utente sotto i 150ms)
- **Largest Contentful Paint:** Quanto tempo ci vuole purchè l'elemento più grande del contenuto (ad esempio un'immagine) venga mostrato sulla pagina.(Buona esperienza utente sotto i 1.2s)
- **Cumulative Layout Shift:** Quanto cambia il layout della pagina durante il caricamento. (Buona esperienza utente sotto i 0.1s)

Anche in questo caso, il tester ha dato risultati importanti e tutti nella norma (Figura 8.6). Si mostra anche un'acquisizione intervallata del caricamento della pagina. Complessivamente la pagina viene ricaricata in un secondo (Figura 8.7).

**Browser Timings**

These timings are milestones reported by the browser.

Redirect Duration	0ms	Connection Duration	176ms	Backend Duration	85ms
Time to First Byte (TTFB)	261ms	First Paint	468ms	DOM Interactive Time	491ms
DOM Content Loaded Time	491ms	Onload Time	763ms	Fully Loaded Time	949ms

Figura 8.6: Performance Metrics della web application post rilascio**Figura 8.7:** Speed Visualization della web application post rilascio

8.3 Test di accettazione - Beta Test - post Rilascio

Si è data l'opportunità di testare il software tramite una versione beta agli interessati. Quest'ultimi rilasciano i seguenti feedback utili per possibili miglioramenti. I seguenti feedback saranno presi in carico in futuro per un aggiornamento del prodotto software in esame.

Feedback	Descrizione del problema	Dispositivo
Feedback 1	Nella fase di prenotazione del tampono deve esserci anche la possibilità di modificare la data della prenotazione.	Qualsiasi dispositivo
Feedback 2	Nella fase di prenotazione del tampono non si riesce a visualizzare il box in sovraimpressione per prenotare	Iphone, IOS, Safari
Feedback 3	Nella fase di prenotazione del tampono non c'è nessun controllo sul codice fiscale	Qualsiasi dispositivo
Feedback 4	Nella dashboard farmacia, c'è un disallineamento del testo "Disponibilità tamponi"	Dispositivi mobili
Feedback 5	Nella dashboard il mio profilo, non c'è la possibilità di cancellare il profilo	Qualsiasi dispositivo
Feedback 6	Nel QRCodeChecker, se la prenotazione ha già un risultato, non deve dare più esito positivo ma indicare che la prenotazione ha già effettuato il test	Qualsiasi dispositivo
Feedback 7	Ci sono nuovi sintomi da COVID19, si necessita l'aggiornamento del modello di Machine Learning	Qualsiasi dispositivo
Feedback 8	Nel mio profilo, in assenza di prenotazioni, quando si entra in "Modifica prenotazioni" e "Rimuovi prenotazioni" non compare il messaggio di assenza prenotazioni	Qualsiasi dispositivo
Feedback 9	Nel mio profilo, Dopo aver svolto il tampone e il risultato è disponibile, è possibile comunque modificare e rimuovere la prenotazione. Bisogna rimuovere la prenotazione nella lista dopo essere stato verificato in farmacia.	Qualsiasi dispositivo

Tabella 8.1: Feedback - Beta test

8.4 Deployment Diagram - Remoto

Si utilizza questo diagramma per mostrare come le componenti software siano distribuite sull'ambiente di sviluppo e servizio di web hosting **PythonAnywhere**.

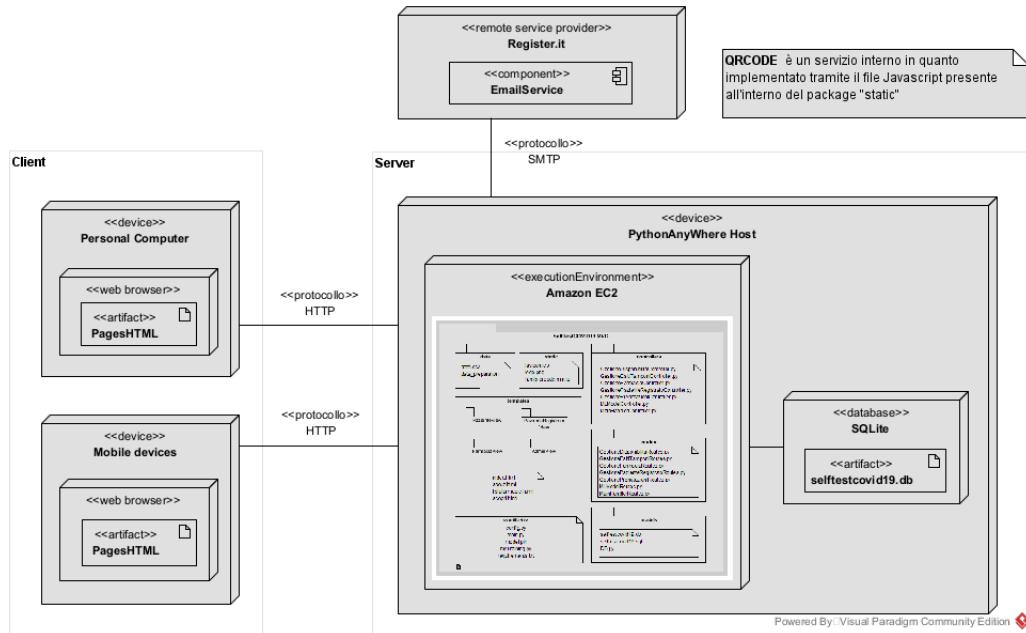


Figura 8.8: Deployment Diagram - Remoto

Dalla Figura 8.8 si nota il **Client** che a livello hardware può essere sia un *Personal Computer* e sia un *Mobile Devices* (smartphone, tablet) su cui è presente un web browser. Questi ultimi interagiscono con il **Server** attraverso il protocollo HTTP. Il **Server** esegue sulla piattaforma di **PythonAnywhere** tramite l'ambiente di esecuzione "Amazon EC2", al cui interno sono istanziate i package come visto già in Figura 6.4. L'ambiente di esecuzione è collegato al database "SQLite" ed al servizio di invio Mail di Register.it tramite protocollo SMTP.

9 Uso del prodotto software

Si mostra il risultato ottenuto a livello descrittivo

Contenuti

9.1 Manuale d'utilizzo	121
9.1.1 Dashboard Admin	121
9.1.2 Home Page e inserimento sintomi, malattie e informazioni COVID19	121
9.1.3 Dashboard Farmacia	123
9.1.4 Il mio profilo	124
9.1.5 La prenotazione	125
9.2 Video dimostrativo e link alla Web Application SelfTestCOVID19	126
9.3 Scopi futuri	127

9.1 Manuale d'utilizzo

Si mostra dunque la *beta* del prodotto software in questione mostrando il manuale d'utilizzo e alcune importanti funzionalità che la web application presenta.

9.1.1 Dashboard Admin

Si avrà il controllo delle farmacie che acquisteranno e utilizzeranno il prodotto software. Dunque l'Admin ha la possibilità di creare, modificare, ricercare e rimuovere le farmacie (Figura 9.1).

The screenshot shows the 'Dashboard Admin' page of the Self Test COVID-19 application. At the top, there's a header with the logo, navigation links ('Dashboard', 'Torna alla Home'), and a timestamp ('2022-9-7 12:7:2 Day Mercoledì'). Below the header, a message says 'Benvenuto nella Dashboard dell'Admin. Puoi effettuare le classiche operazioni CRUD per la gestione delle farmacie.' A button labeled 'Vai alla lista delle farmacie disponibili' is present. The main area contains four cards for managing pharmacies:

- Gestione farmacie** + Creazione Farmacia: 'Creazione delle farmacie che aderiscono alla piattaforma.' with a link to '→ Accedi'.
- Gestione farmacie** ⚡ Modifica Farmacia: 'Modifica le informazioni relative ad una farmacia presente nel sistema.' with a link to '→ Accedi'.
- Gestione farmacie** ✗ Rimuovi Farmacia: 'Rimuovi una farmacia dal sistema.' with a link to '→ Accedi'.
- Gestione farmacie** ❓ Ricerca Farmacia: 'Ricerca le farmacie presenti nel sistema per Nome e Città (e/o CAP).' with a link to '→ Accedi'.

Figura 9.1: Dashboard Admin

9.1.2 Home Page e inserimento sintomi, malattie e informazioni COVID19

L'index della web application è presentata nella Figura 9.2, da qui sarà possibile effettuare il **SelfTestCOVID19** (sempre in Figura 9.2) e accedere alla **Farmacia** e **Admin** nel footer della pagina. Perché nel footer? Perché sono

funzionalità che **soltanto** gli attori suddetti possono accedere e dunque per "nascondere" si è deciso di posizionarli lì.

Figura 9.2: HomePage e inserimento sintomi

9.1.3 Dashboard Farmacia

Nella sezione **Farmacia** è possibile effettuare le operazioni definite nei paragrafi precedenti (Figura 9.3).

The screenshot shows the Farmacia Don Bosco dashboard with the following interface elements:

- Header:** Self Test COVID-19 Probability Infection Checker, Dashboard, Torna alla Home, Date (2022-9-7 11:53:25), Day (Mercoledì), Logout.
- Title:** Farmacia Don Bosco
- Welcome Message:** Benvenuto nella [Dashboard](#) della tua farmacia. Puoi effettuare le classiche operazioni CRUD per le prenotazioni, per i tamponi e per gli esiti dei tuoi pazienti.
- Buttons:** Vai alla lista prenotati della giornata, Check QR Code.
- Information Bar:** Disponibilità tampone rapido (28) Disponibilità tampone molecolare (12)
- Card 1 (Red):** Gestione esito tamponi, + Aggiornare esito tamponi, Aggiornamento esito tamponi rapidi e/o molecolari ai prenotati, → Accedi.
- Card 2 (Blue):** Gestione prenotazioni, ↗ Modifica prenotazioni, Modifica prenotati per il tampone qualcosa qualcosa qualcosa qualcosa, → Accedi.
- Card 3 (Blue):** Gestione prenotazioni, ✘ Rimuovi prenotati, Rimuovi prenotati per il tampone per qualcosa qualcosa qualcosa qualcosa, → Accedi.
- Card 4 (Green):** Gestione disponibilità tamponi, + Creazione disponibilità tamponi, Creazione disponibilità tamponi esiti molecolari esistenti in...
- Card 5 (Green):** Gestione disponibilità tamponi, ↗ Modifica disponibilità tamponi, Modifica la disponibilità dei tamponi per l'esaminazione della...
- Card 6 (Green):** Gestione disponibilità tamponi, ✘ Rimuovi disponibilità tamponi, Rimuovi la disponibilità dei tamponi per l'esaminazione della...

Figura 9.3: Dashboard Farmacia

È possibile fare il check del prenotato una volta in sede attraverso la funzionalità **Check QR Code** (Figura 9.4).

The screenshot shows the QR code checker feature with the following interface elements:

- Header:** Self Test COVID-19 Probability Infection Checker, Dashboard, Torna alla Home, Logout.
- Title:** QR Code checker
- Text:** Verifica prenotazione del paziente, posiziona il QR code nel riquadro e il sistema riconoscerà la prenotazione.
- Scanning Area:** Scansione il QR code della prenotazione IN ATTESA, with options to Access camera or Select image from device.
- Scanned QR Code:** A QR code is shown with the text "Accedi alla videocamera" and "Seleziona un'immagine dal dispositivo". A red arrow points from the scanning area to this QR code.
- Result Area:** Risultato della scansione, showing the recognized appointment details: Prenotazione nr. 15 riconosciuta, Nome: Mario, Cognome: Rossi.

Figura 9.4: QR code checker

Ovviamente, dopo aver svolto il tampone e dopo aver effettuato l'analisi, la farmacia dovrà aggiornare l'esito del tampone. A valle di questo verrà inviata **una mail** al paziente che gli solleciterà di accedere nel suo profilo e visualizzare l'esito del tampone (Figura 9.5).



Figura 9.5: Email di sollecito esito tamponi

9.1.4 Il mio profilo

Nella sezione **il mio profilo** è possibile accedere per visionare l'**esito del tampone** e **scaricare il certificato** soltanto se è disponibile (sarà la farmacia aggiornare l'esito). Inoltre è possibile **modificare** e **rimuovere** le prenotazioni fatte (Figura 9.7).

ID Prenotazione	Nome Farmacia	Città	Indirizzo	Giorno	Ora	Tipo Tampono	Esito	Certificato
15	Don Bosco	Napoli	Via Roma 45	2022-09-07	13:00	Molecolare	Negativo	Certificato disponibile

Figura 9.6: Dashboard paziente - Certificato disponibile

Self Test COVID-19 Probability Infection Checker

Il mio profilo Torna alla Home 2022-9-7 12:7:43 Day Mercoledì Logout

Benvenuto Mario, gestisci le prenotazioni

Puoi gestire le tue prenotazioni effettuate, accedi nelle aree sottostanti per effettuare le modifiche.

[Vai al referto](#)

Gestione prenotazione

Modifica prenotazione
Modifica le informazioni relative alla tua prenotazione
[→ Accedi](#)

Gestione prenotazione

Rimuovi prenotazione
Rimuovi la prenotazione da te effettuata
[→ Accedi](#)

Referto tamponi
Visualizza gli esiti dei tamponi effettuati nelle farmacie scelte.

Figura 9.7: Dashboard paziente

9.1.5 La prenotazione

La prenotazione, come dai requisiti, sarà possibile farla soltanto dopo il test e con esito maggiore del 50% (Figura 9.8).



Il risultato

Attenzione!

La probabilità di infezione del paziente è **99.0%**. Pertanto la tua situazione suscita preoccupazione e per il test sei stato infettato dal SARS-CoV-2 (COVID-19). Tuttavia, recati SOLO in farmacia per la prenotazione di un tampone MOLECOLARE e/o chiama al telefono il tuo medico di famiglia, il tuo pediatra o la guardia medica. Oppure chiama il Numero Verde regionale oppure ancora al Numero di Pubblica utilità: 1500.

Attenzione recati in farmacia senza frequentare luoghi affollati in quanto potenzialmente positivo!

Numero Verde Regionale: [Clicca qui per l'elenco](#)

Numero Pubblica Utilità: 1500

Link utile: [Tutte le informazioni a come comportarsi in caso di sintomi](#)

[Prenota un tampone molecolare](#)

SELF TEST COVID-19

POLICY

ACCESSO AL SISTEMA

SOURCE CODE

Prenotazione interattiva in farmacia
per la somministrazione e acquisto

Privacy Policy

Farmacia

Scopri il codice sorgente del
progetto SelfTestCOVID19, è [open](#)

Figura 9.8: Risultato SelfTestCOVID19

Inizialmente però, si deve prima effettuare la **ricerca della farmacia disponibile** inserendo Nome farmacia, città o CAP, verrà poi mostrata la lista delle farmacie (Figura 9.9).



Prenota tampone molecolare nella farmacia selezionata

Nome Farmacia	Città	Nome Tampone	Giorno	Ora Inizio	Ora Fine	Prezzo	Azione
Don Bosco	Napoli	DazTamp	2022-09-07	10:00	17:00	35.99	Prenota tampone molecolare

SELF TEST COVID-19

POLICY

ACCESSO AL SISTEMA

SOURCE CODE

Prenotazione interattiva in farmacia
per la somministrazione e acquisto
dei tamponi **rapidi/molecolari**

Privacy Policy

Farmacia

Scopri il codice sorgente del
progetto SelfTestCOVID19, è [open](#)
[source](#)

Cookie Policy

Admin

[Visita il nostro repository](#)

Condizioni d'uso

© 2022 Copyright: Antonio Romano, Giuseppe Riccio
Il progetto è stato realizzato per soli scopi dimostrativi per l'esame del Corso Magistrale di **Software Architecture Design** all' Università della Federico II di Napoli

Figura 9.9: Lista farmacie disponibili

La prenotazione potrà essere fatta da un nuovo utente (**nuovo paziente**) oppure da un **paziente registrato** (la registrazione si ottiene solo dopo aver effettuato una prenotazione in precedenza). In Figura 9.10

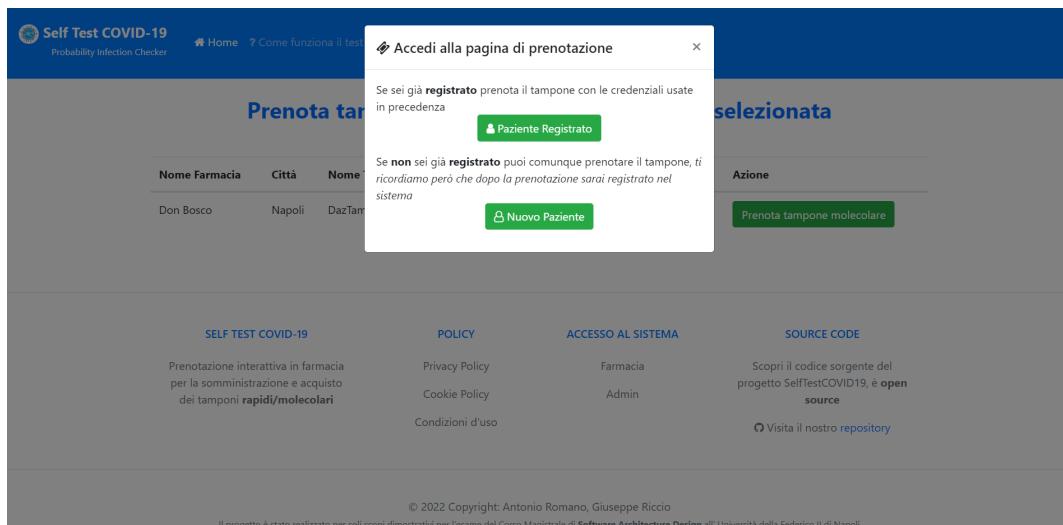


Figura 9.10: Prenotazione nuovo paziente o paziente registrato

9.2 Video dimostrativo e link alla Web Application SelfTestCOVID19

Si mostrano tutte le funzionalità implementate e le interazioni di tutti gli attori del prodotto software SelfTestCOVID19 attraverso un video demo:

Youtube

<https://youtu.be/SI5Z3io0Atc>

La web application è pubblica al seguente sito:

SelfTestCOVID19

<https://www.selftestcovid19.it/>

Il repository con tutto il package completo:

GitHub

<https://github.com/giuseppericcio/SelfTestCovid19>

9.3 Scopi futuri

Il nostro obiettivo è stato quello di gettare delle basi per un prodotto software che può essere esteso in tante altre tematiche e non solo. Questo grazie alla flessibilità degli strumenti utilizzati per sviluppare questo progetto. Si immagina al caso scelto che è di notevolissima importanza, soprattutto in questo periodo di pandemia mondiale, ovvero in ambito Healthcare (sanità). Grazie alla quantità di dati medici generati oggigiorno, le organizzazioni sanitarie possono sfruttare questa mole di conoscenza per fornire risultati migliori con maggiore precisione. Infatti, la realizzazione di software che ingloba ML che tratta questi dati, può aiutare le organizzazioni sanitarie a fornire le previsioni e i risultati più semplici per i pazienti, riducendo i costi e personalizzando in modo significativo l'assistenza ai pazienti.

Pertanto sorge spontaneo l'esigenza di poter estendere il nostro software non solo per il controllo e la gestione del COVID19 e della gestione e controllo di affluenza nelle farmacie. Si fanno degli esempi:

- Dall'analisi del sangue per la predizione di possibili allarmi e cura da fare
- Diagnostica ecografica per la predizione di masse tumorali
- Diagnostica elettroencefalografica per la predizione di possibili presenze di alterazioni che possono indurre il neurologo a fare delle analisi più approfondite
- Analisi della pressione per la predizione di una diagnosi di ipertensione per verificare se sia necessaria una terapia antipertensiva
- Analisi per il colesterolo, colesterolo HDL e LDL, trigliceridi per la predizione del rischio di svilupparli, se già presente, di andare incontro a complicazioni e dunque guidare i pazienti.
- Analisi Moc preventivo per la predizione del livello di osteoporosi utile per verificare in maniera rapida come stanno le proprie ossa.
- Tante altre possibili idee

Bibliografia

- (1) Lavecchia, V. Project Management: I vincoli di progetto (tempo, costi e scopo), <https://vitolavecchia.altervista.org/project-management-i-vincoli-di-progetto-tempo-costi-scopo/>, 2019.
- (2) Gabriel, C. Glossary of Website Terms and Definitions, <https://carolinegabriel.com/glossary-website-terms/>, 2022.
- (3) Pastore, D. What exactly is Werkzeug, <https://stackoverflow.com/questions/37004983/what-exactly-is-werkzeug>, 2019.
- (4) PythonGeeks Python Flask Introduzione, <https://www.pythongeeks.org/python-flask-introduction/>, 2021.
- (5) Dugar, D. Jinj2, <https://codeburst.io/jinja-2-explained-in-5-minutes-88548486834e>, 2018.
- (6) Scrum Guide, <https://www.scrumguides.org/scrum-guide.html>., 2022.
- (7) Bootstrap, <https://getbootstrap.com/>, 2022.
- (8) Hill, M. UML2 Unified Process, , 2005.
- (9) Materiale del corso di Software Architecture Design, , 2022.
- (10) Testing Flask, <https://medium.com/analytics-vidhya/how-to-test-flask-applications-aef12ae5181c>.
- (11) Rest Explanation, <https://www.freecodecamp.org/news/rest-api-tutorial-rest-client-rest-service-and-api-calls-explained-with-code-examples/>.
- (12) Integrazione sistema documentale a microservizi: Realizzazione di due applicazioni Web, <https://webthesis.biblio.polito.it/14434/1/tesi.pdf>, 2022.
- (13) Microservizi, <https://www.sergentelorusso.it/architettura-microservizi>, 2022.