



UNIVERSITÀ DEGLI STUDI DI SALERNO

Software Engineering I-Z

A.A. 2020/2021

Prof. Giancarlo Nota

Relazione Project Work:

***Web Application per la gestione di attività di
manutenzione***

Team 5

Antonietta Napoli

Antonella Rossi

Simona Sorgente

Angelo Vistocco

1. User story e pianificazione

1.1 User Stories

In accordo con quanto stabilito dalla metodologia Agile SCRUM, in una prima fase di progettazione sono state ricavate le User Stories dal documento SRS fornito. Tali User Stories rappresentano una descrizione informale di quelli che sono i requisiti forniti dall'utente per la realizzazione della web application in questione.

USER STORY ID	PRIORITY	AS A <type of user>	I WANT TO <perform some task>	SO THAT I CAN <achieve some goal>
1	High	Planner	Manage maintenance activities of the factory. There are two types of maintenance activities: Planned and Un-planned(EWO and Extra activities).	Plan the maintenance activities of the factory.
2	High	Planner	For each maintenance activity of the factory record the relative data.	Track the maintenance activities of the factory.
3	High	Planner	View the list of scheduled maintenance activities ordered by week for each site.	Know what maintenance activities must be done on every week.
4	Medium	Planner	See general information about scheduled maintenance activities.	Plan the maintenance activities of a specific week.
5	High	Planner	Select a specific maintenance activity from the list of activities.	Assign the scheduled activity to a specific Maintainer, according to his availability.
6	High	Planner	Select a Maintainer according to his skills and his availability percentage to execute a planned activity.	Choose the best maintainer among all maintainers and assign him a maintenance activity.
7	High	Planner	Assign a maintenance activity to a Maintainer in a specific time slot.	Have a specific maintenance activity to be done on a specific time.
8	Medium	Planner	See detailed information about a selected maintenance activity.	Full knowledge of the maintenance activity.
9	High	Planner	Send notifications to a Maintainer when I assign him a maintenance activity and that the Production manager receives a copy of that notification by e-mail	I know they are aware of my decision
10	Medium	Planner	Add specific informations on an assigned maintenance activity.	Give useful info to the maintainer to facilitate him the work.
11	Low	Planner	Manage materials needed to run a maintenance activity (optional).	The Maintainer to understand what materials are needed for the maintenance activity, so that he can bring them with him.
12	Medium	Planner	See the list of assigned tickets (EWO) and specific information for each one.	Know the current EWO assigned.
13	High	System Administrator	Manage system users, assign them username, password and a specific role (Maintainer or Planner).	Manage all the employees.
14	High	System Administrator	Assign specific competencies to a Maintainer.	Know the competencies of the Maintainer.
15	High	System Administrator	Associate to a Maintainer the procedure to be performed according to each maintenance activity.	Know the Maintainer will understand what he has to do.
16	Medium	System Administrator	Associate to a Maintenance Procedure a Standard Maintenance Procedure (SMP) file in PDF format.	Make sure the Maintainer can follow a standard procedure reading the SMP file.
17	Low	System Administrator	Manage Workspace notes (Optional) that can be associated to a specific place or a set of places in my factory.	Make sure the Maintainer can read that before the maintenance procedure.
18	High	System Administrator	Make sure that all of the access to the application will be recorded.	Track the actions of the employees.
19	High	Repository Manager	Manage a list of competences related to a specific task.	Allow the Planner to assign Maintainer a maintenance that he can do.
20	Medium	Repository Manager	Manage sites, which are composed of factory site (branch offices) and area (or department) inside the factory	Make sure that the structure of the factory is well defined.
21	High	Repository Manager	Manage a list of materials	Allow the Planner to use them during the maintenance activity.
22	High	Repository Manager	Manage a list of Maintenance procedures	Choose one of them before assigning a procedure to a maintainer.
23	High	Repository Manager	Assign the specific competencies required to perform the maintenance activity	Make sure the activity will be done in the best way.
24	Medium	Repository Manager	Manage maintenance typologies as Electrical, electronic, hydraulic, mechanical	Have more information about the maintenance activity.
25	High	Planner	Select a specific Maintainer for an EWO activity after I have received an EWO notification	Know the EWO is immediately resolved and the broken machine can continue to work
26	High	Planner	Send notifications to a Maintainer when I assign him an EWO activity and that the Production manager receives a copy of that notification by e-mail	Know they are aware of my decision and that the Maintainer resolves the EWO in the shortest time

Figura 1: User Stories

Dopo la stesura di tali User Stories, a cui sono state assegnate le rispettive tipologie di ruolo a cui si riferiscono (Planner, System Administrator, Repository Manager) e le relative priorità (Alta, Media, Bassa), sono stati pianificati tre Sprint per la realizzazione delle richieste dell'utente.

1.2 Sprint 1

TASK ID	TASK NAME	SPRINT #	ASSIGNED TO	START	FINISH	STORY	SPRINT READY	PRIORITY	STATUS	STORY POINTS	ASSIGNED TO SPRINT
		Sprint 1	Angelo Vistocco	19/11/2020	2/12/2020		Yes	High	Complete	38	
US1	Manage maintenance activities	" "	" "	23/11/2020	26/11/2020	Yes	Yes	High	Complete	8	Yes
US2	Record maintenance data	" "	" "	19/11/2020	20/11/2020	Yes	Yes	High	Complete	2	Yes
US3	View scheduled maintenance activities list	" "	" "	26/11/2020	1/12/2020	Yes	Yes	High	Complete	4	Yes
US4	View scheduled maintenance activities informations	" "	" "	26/11/2020	1/12/2020	Yes	Yes	Medium	Complete	4	Yes
US5	Create the link between the previous views	" "	" "	2/12/2020	2/12/2020	Yes	Yes	High	Complete	2	Yes
US19-24	Manage procedure maintenance data	" "	" "	19/11/2020	26/11/2020	No	Yes	High	Complete	16	Yes
US10	Make possible to insert workspace notes	" "	" "	26/11/2020	1/12/2020	Yes	Yes	Medium	Complete	2	Yes

Figura 2: Product Backlog Sprint 1

Il primo sprint, schedulato a partire dal 19 novembre 2020 fino al 2 dicembre 2020, prevede la realizzazione di una serie di task, ognuno dei quali è stato assegnato ad un membro del team, ricavati dalle User Stories selezionate per implementare alcune delle principali funzionalità dell'applicazione (*Figura 2*).

Dal Product Backlog, inoltre, è stato ricavato lo Sprint Backlog in di *Figura 3*.

A partire da questo documento, dopo aver inserito l'ammontare di ore stabilito con relative ore effettivamente consumate per completare un certo task, è stato ricavato il Burndown Chart di *Figura 4*.

BACKLOG TASK & ID	STORY POINTS	ASSIGNED TO	STATUS	ORIGINAL ESTIMATE	DAY 1 19	DAY 2 20	DAY 3 21	DAY 4 23	DAY 5 24	DAY 6 25	DAY 7 26	DAY 8 28	DAY 9 29	DAY 10 30	DAY 11 1	DAY 12 2	SPRINT REVIEW
User Story 1																	
Database: maintenance activity and EWO tables	2	Antonietta Napoli	Completed	2	2	2	2	0	0	0	0	0	0	0	0	0	0
Implement html interface to add, remove or modify and activity	2	Antonella Rossi	Completed	3	3	3	3	1	2	0	0	0	0	0	0	0	0
Implement php module to read maintenance data from db and create json file	2	Angelo Vistocco	Completed	3	3	3	3	3	3	3	3	3	3	3	2	1	0
Implement js class to manage data in the html file	2	Simona Sorgente	Completed	3	3	3	3	3	1	1	0	0	0	0	0	0	0
User Story 2																	
Database: data about maintenance activities	2	Simona Sorgente	Completed	2	1	1	0	0	0	0	0	0	0	0	0	0	0
User Story 3																	
Implement an html interface to list all scheduled maintenance activities ordered by week	1	Antonella Rossi	Completed	2	2	2	2	2	2	2	2	1	2	1	0	0	0
Implement a php module to get data from DB	2	Angelo Vistocco	Completed	3	3	3	3	3	3	3	3	3	2	2	1	0	0
Implement a js module to show maintenance informations in the html	1	Antonietta Napoli	Completed	3	3	3	3	3	3	3	3	3	2	1	1	0	0
User Story 4																	
HTML: show all the general information about scheduled maintenance activities	1	Antonella Rossi	Completed	2	2	2	2	2	2	2	2	2	2	1	2	0	0
Implement a php module to get data from DB	2	Angelo Vistocco	Completed	3	3	3	3	3	3	3	2	1	1	0	0	0	0
Implement a js module to show data in the html	1	Simona Sorgente	Completed	2	2	2	2	2	2	2	2	2	1	1	0	0	0
User Story 5																	
Implement js function to change the html view by pressing a button	2	Simona Sorgente	Completed	2	2	2	2	2	2	2	2	2	2	2	0	1	0
User Story 10																	
HTML: provide a blank workspace note to write information about the activity	1	Antonella Rossi	Completed	1	1	1	1	1	1	1	1	1	1	1	0.5	0	0
Database: store information about the maintenance activity	1	Simona Sorgente	Completed	2	2	2	2	2	2	2	1	1	0	0	0	0	0
User Story 19-24																	
Database: list of competences related to a specific task	2	Antonietta Napoli	Completed	2	0	1	0	0	0	0	0	0	0	0	0	0	0
Database: manage sites, which are composed of factory site (branch offices) and area (or department) inside the factory	1	Simona Sorgente	Completed	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Database: list of materials	1	Antonietta Napoli	Completed	1	1	1	0	0	0	0	0	0	0	0	0	0	0
Database: list of Maintenance procedures	1	Simona Sorgente	Completed	1	1	1	0	0	0	0	0	0	0	0	0	0	0
PHP module to get data	4	Angelo Vistocco	Completed	4	3	2	1	1	0	0	0	0	0	0	1	1	0
JS module to interact with HTML	2	Antonietta Napoli	Completed	2	2	2	2	1	1	1	0	0	0	0	0	0	0
Implement an HTML view for system admin to manage data in the DB	4	Antonella Rossi	Completed	4	2	1	1	0	0	0	0	0	0	0	0.5	0	0
Database: maintenance typologies as Electrical, electronic, hydraulic, mechanical	1	Antonietta Napoli	Completed	1	1	1	1	1	1	0	1	0	0	0	0	0	0
TOTAL	38			49	43	42	37	30	28	25	22	19	16	12	7.5	3	0

Figura 3: Product Backlog Sprint 1

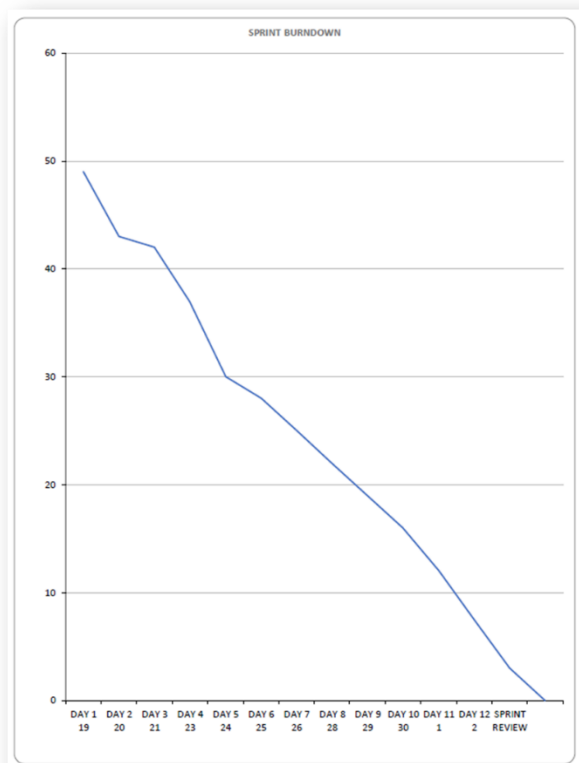


Figura 4: Burndown Chart Sprint 1

Abbiamo deciso di implementare prima di tutto il database attraverso PostgreSQL, dopo aver realizzato in una prima fase di progettazione concettuale e logica, i seguenti diagrammi di *Figura 5* e *Figura 6*.

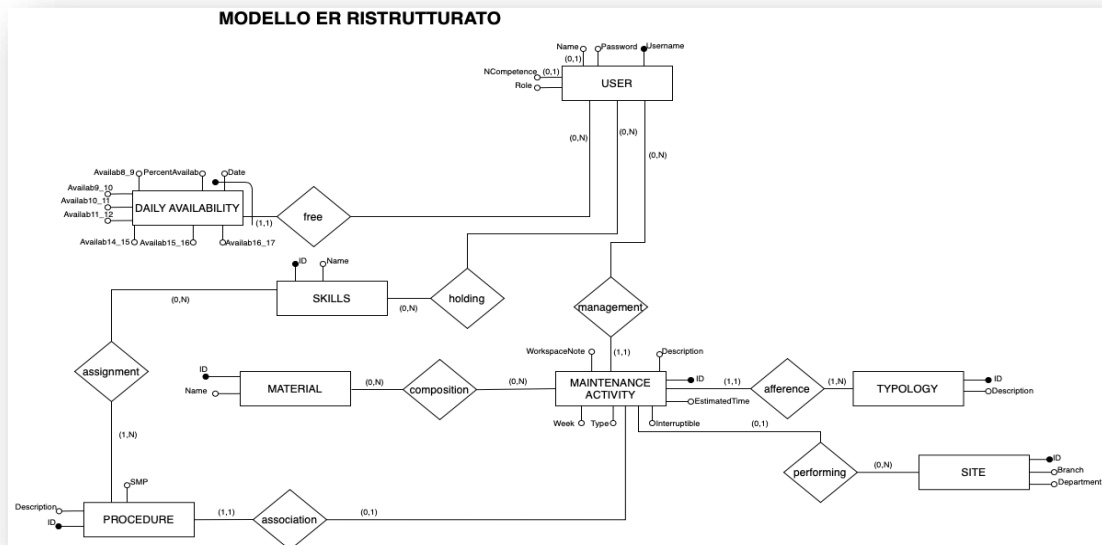


Figura 5: diagramma ER

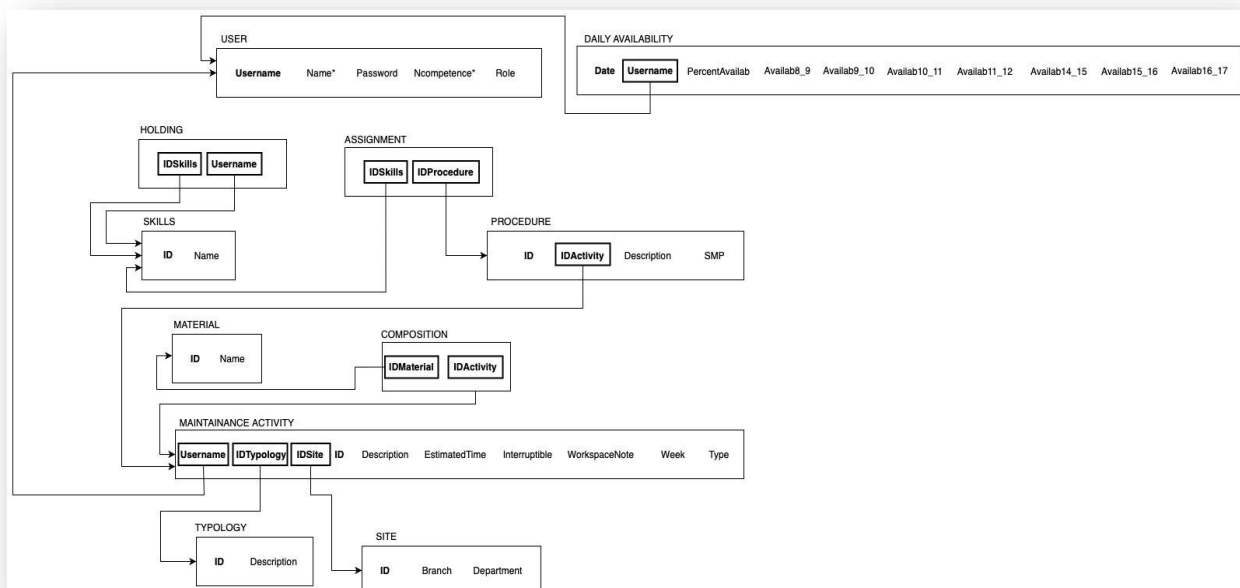


Figura 6: Diagramma logico

Successivamente, siamo passati alla realizzazione di alcune GUI attraverso l'utilizzo del linguaggio di mark-up HTML ed il CSS per definirne lo stile. In questo primo sprint, non avendo realizzato le funzionalità di login di un utente, è stato reso possibile accedere alle due distinte viste, una per il System Administrator ed una per il Planner, cliccando su degli appositi link nella home page.

Attraverso PHP e JavaScript sono poi state implementate funzioni in grado di permettere all'utente di interagire con il database.

La vista del System Administrator è stata realizzata in modo da permettere di selezionare da una navigation bar nella pagina di Management Area una particolare sezione da gestire: *tipologie di manutenzione, materiali, skill, siti e procedure di manutenzione*. Per ognuno di questi elementi, il System Administrator può aggiungerne uno nuovo, modificarne o rimuoverne uno già esistente.

La vista del Planner, invece, è stata realizzata in modo da permettere di visualizzare le attività di manutenzione pianificate ordinate per settimana, di poterle modificare o cancellare, oppure di aggiungerne una nuova. Selezionando una particolare attività, si può poi accedere ad una pagina in cui vengono visualizzate le principali informazioni di quell'attività: *numero della settimana, attività da assegnare (activity ID, area, typology, estimated intervention time), workspace notes, descrizione dell'attività e skill richieste*.

1.3 Sprint 2

		Sprint 2	Antonietta Napoli	3/12/2020	16/12/2020		Yes	High	Complete	22	
US13-14	Manage users and Maintainers competencies	" "	" "	3/12/2020	9/12/2020	No	Yes	High	Complete	8	Yes
US6	Select a Maintainer from a list of maintainers	" "	" "	9/12/2020	14/12/2020	Yes	Yes	High	Complete	4	Yes
US7	Select a time slot for the maintainer	" "	" "	15/12/2020	15/12/2020	Yes	Yes	High	Complete	2	Yes
US9	Send a notification to the selected Maintainer and to the Production manager	" "	" "	15/12/2020	16/12/2020	Yes	Yes	High	Complete	4	Yes
US8	View for selected activity and selected maintainer	" "	" "	9/12/2020	14/12/2020	Yes	Yes	Medium	Complete	4	Yes

Figura 7: Product Backlog Sprint 2

In questo secondo sprint, schedato a partire dal 5 dicembre 2020 al 16 dicembre 2020, sono state implementate una serie di funzionalità suddivise in task e assegnate ad ognuno dei membri del team a partire da una serie di User Stories selezionate (Figura 7).

Da questo Product Backlog è stato ricavato lo Sprint Backlog di Figura 8.

Come nel primo Sprint, stimando le ore di lavoro necessarie rispetto a quelle effettivamente consumate, abbiamo ricavato il Burndown Chart di Figura 9.

BACKLOG TASK & ID	STORY POINTS	ASSIGNED TO	STATUS	ORIGINAL ESTIMATE	DAY 1: 5	DAY 2: 7	DAY 3: 9	DAY 4: 10	DAY 5: 11	DAY 6: 12	DAY 7: 14	DAY 8: 15	DAY 9: 16	SPRINT REVIEW
User Story #13-14	8													
Create system admin interface in html to assign skills to maintainers	1	Antonella Rossi	completed	2	2	1	0	0	0	0	0	0	0	0
Create system admin interface in html to manage users	1	Antonella Rossi	completed	2	2	1	0	0	0	0	0	0	0	0
Create html login interface for users like planner or maintainer	1	Antonella Rossi	completed	2	2	0	0	0	0	0	0	0	0	0
Implement php module to assign skills to maintainers and its test cases	1	Angelo Vistocco	completed	2	1	0	0	0	0	0	0	0	0	0
Implement php module to manage users and its test cases	2	Angelo Vistocco	completed	3	2	1	0	0	0	0	0	0	0	0
Implement js module to populate html assigning skills interface	1	Antonieta Napoli	completed	2	2	2	0	0	0	0	0	0	0	0
Implement js module to populate html managing users	1	Antonieta Napoli	completed	2	2	2	0	0	0	0	0	0	0	0
User Story #6	4													
Create an html interface to select a maintainer among all the maintainers	2	Simona Sorgente	completed	3	3	3	2	1	0	0	0	0	0	0
Implement a php module to load all maintainers and its test cases	1	Angelo Vistocco	completed	2	2	2	2	1	1	1	0	0	0	0
Implement a js module to populate html with maintainers loaded with php module	1	Antonieta Napoli	completed	2	2	2	2	2	2	1	1	0	0	0
User Story #7	4													
Create an html interface to show maintainer availability and to select a specific time slot for him	1,5	Antonella Rossi	completed	3	3	3	3	3	2	2	1	0	0	0
Implement a php module to load time slots for a specific maintainer and its test cases	1	Angelo Vistocco	completed	2	2	2	2	2	2	2	1	0	0	0
Implement a js module to populate html with maintainer time slots loaded with php module	1	Simona Sorgente	completed	2	2	2	2	2	2	0	0	0	0	0
Get selected time slot in html with js	0,5	Antonieta Napoli	completed	1	1	1	1	1	1	1	2	1	0	0
User Story #8	4													
Create an html view to show the selected maintainer and the selected activity	1,5	Antonella Rossi	completed	2	2	2	2	1	1	0	0	0	0	0
Implement a php module to load the selected maintainer and the selected activity	1,5	Angelo Vistocco	completed	2	2	2	2	2	2	2	2	1	0	0
Implement a js module to populate html with selected maintainer and activity	1	Antonieta Napoli	completed	2	2	2	2	2	2	2	0	0	0	0
User Story #9	4													
Implement js function to send notification to a maintainer	1,5	Simona Sorgente	completed	2	2	2	2	2	2	2	2	0	0	0
Modify db in order to let a user add an e-mail address	1	Antonieta Napoli	completed	1	1	1	1	1	1	1	1	0	0	0
Implement js function to send notification to the production manager	1,5	Simona Sorgente	completed	2	2	2	2	2	2	2	2	2	0	0
Integration test														
Test all the classes together		TEAM 5	completed	3	3	3	3	3	3	3	3	3	0	0
TOTAL				41	39	33	25	22	20	16	12	4	0	0

Figura 8: Sprint Backlog Sprint 2

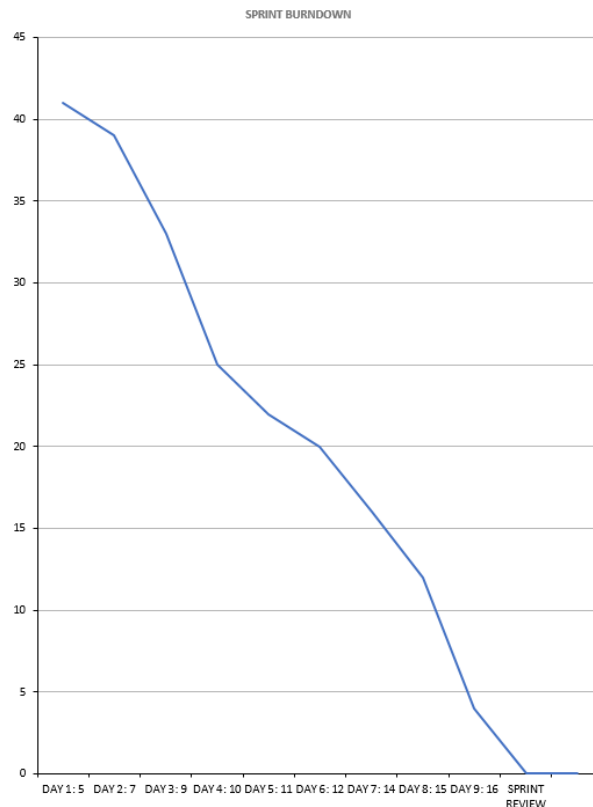


Figura 9: Burndown Chart Sprint 2

Durante questo Sprint è stata realizzata la funzionalità di login per cui un utente, System Administrator o Planner, può accedere alla propria vista riservata inserendo le proprie credenziali (username e password) in un apposito form, dalla home page dell'applicazione.

Sono state poi aggiunte funzionalità che permettono al System Administrator di gestire tutti gli utenti dell'applicazione, per cui può creare un account con relative informazioni (username, password, email e ruolo), a seconda che si tratti di un Planner o di un Maintainer. Un System Administrator può, inoltre, modificare o cancellare un utente selezionato.

Per quanto riguarda la vista del Planner, sono state realizzate funzionalità che permettono a tale utente, dopo aver effettuato il login, di poter selezionare un'attività precedentemente inserita, di assegnarvi uno specifico Maintainer a seconda delle sue disponibilità giornaliere e orarie in una specifica settimana. Dopo aver completato l'operazione di assegnazione, il Planner può inviare una e-mail di notifica al Maintainer selezionato ed al Production Manager riguardo l'attività che è stata assegnata.

1.4 Sprint 3

		Sprint 3	Antonella Rossi	17/12/2020			No		Not Started	32	
US12	View list of assigned tickets (EWO)	" "	" "	17/12/2020	22/12/2020	Yes	No	Medium	Not Started	4	Yes
US15	Associate to a Maintainer the procedure to be performed according to each maintenance activity.	" "	" "	17/12/2020	22/12/2020	Yes	No	High	Not Started	4	Yes
US16	Associate to a Maintenance Procedure a Standard Maintenance Procedure (SMP) file in PDF format.	" "	" "	17/12/2020	22/12/2020	Yes	No	Medium	Not Started	4	Yes
US18	Record of the application accesses	" "	" "	17/12/2020	29/12/2020	Yes	No	High	Not Started	8	Yes
US25	Select a Maintainer for an EWO activity	" "	" "	28/12/2020	30/12/2020	Yes	No	High	Not Started	4	Yes
US26	Send a notification to the Maintainer who has an EWO activity and to the Production manager	" "	" "	30/12/2020	30/12/2020	Yes	No	High	Not Started	4	Yes
U11	Manage materials needed to run a maintenance activity.	" "	" "	17/12/2020	22/12/2020	Yes	No	Medium	Not Started	4	Yes

Figura 10: Product Backlog Sprint 3

Lo Sprint 3 è stato schedulato a partire dal 17 dicembre e comprende tutti quei task e User Stories non ancora implementati nei due Sprint precedenti. In questa fase, in particolare, ci si concentra sulla realizzazione di funzionalità di assegnazione di un'attività EWO ad uno specifico Maintainer. Per ogni attività EWO sarà possibile effettuare modifiche oppure rimuoverla, oltre al fatto di poterne aggiungere una nuova.

Come è possibile vedere dalla tabella di *Figura 10*, questo Sprint non è stato portato a termine in quanto il tempo a disposizione per la realizzazione dell'applicazione è stato sufficiente solo per completare i primi due Sprint pianificati.

2. Architettura

2.1 System design model e service layer

La definizione dell'architettura è un processo con cui vengono colti i bisogni e gli interessi delle parti interessate per soddisfare i bisogni del cliente.

Tra i vari modelli esistenti, sicuramente il Model-View-Controller è il modello più efficiente e più diffuso per le applicazioni web. Infatti, questo tipo di modello 3-tier è spesso usato per organizzare i programmi che devono visualizzare diverse viste di dati come nel caso dell'applicazione realizzata. L'idea principale è separare i dati dal display. In questo modo, un programmatore può concentrarsi su un problema specifico, senza doversi preoccupare che i cambiamenti apportati al codice vadano ad influire su altre parti del programma.

MVC, come suggerisce il nome, si divide l'applicazione in tre parti interconnesse: Model, View e Controller.

- **Model:** è "l'oggetto dell'applicazione", che è solo un altro modo di dire che viene usato per gestire la persistenza e l'accesso ai dati. Egli è il ponte tra le altre due componenti (View e Controller). Inoltre, ha la responsabilità di notificare la componente View ogni volta che i dati cambiano.
- **View:** è quello che l'utente vede quando si utilizza l'applicazione. Prende i dati che gli vengono passati dal Model e rende il risultato finale visibile sullo schermo. Nel caso di una web app, si può immaginare la vista finale del documento HTML come il rendering tramite il browser web (user-agent). Visto che questo componente è strettamente collegato con il Model, può modificare quest'ultimo e rispondere alle notifiche che riceve andando a ridisegnare sé stesso.
- **Controller:** definisce il modo con cui l'utente interagisce con l'applicazione. Esso gestisce le interazioni dell'utente e lo scambio di dati tra la View e il Model.

Attraverso il System Design Model in *Figura 11*, è stata definita l'architettura MVC relativa all'applicazione realizzata. In particolare, quanto realizzato prevede tre viste, una per ogni ruolo identificato nelle User Story:

- Una prima vista permette al Planner di eseguire le sue attività di pianificazione.
- Una seconda consente al Maintainer di consultare quali attività di manutenzione deve svolgere e comunicare informazioni sul lavoro svolto. Inoltre, questa vista consente, al ruolo in questione, di segnalare eventuali EWO.
- Una terza permette al System Administrator di interagire con il database aziendale per gestire i vari parametri da lui richiesti.

Le tre viste comunicano con un Controller che interagisce con il modello in tecnologia PHP attraverso un'interfaccia JSON. In questo modo, i dati richiesti dalle interfacce saranno prima raccolti da un database attraverso il Model e poi passati al Controller per essere mostrati.

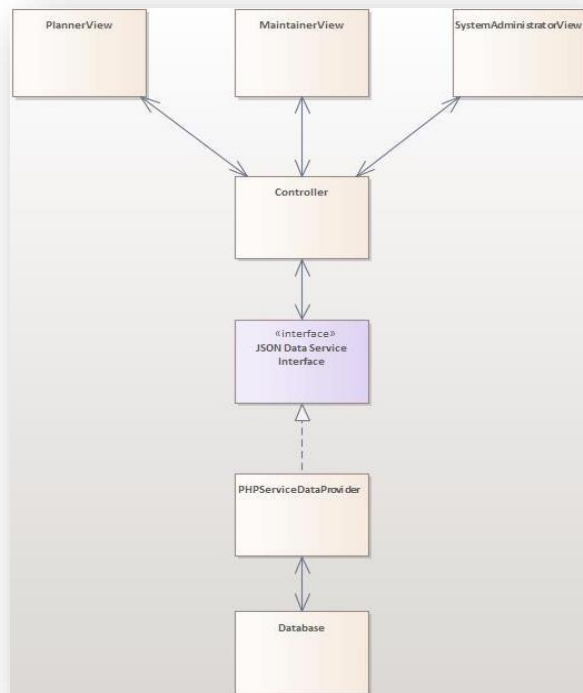


Figura 11: Model-View-Controller

In una seconda fase, il System Design Model è stato modificato con l'aggiunta di classi e moduli utilizzati. Il Service Layer realizzato è il seguente:

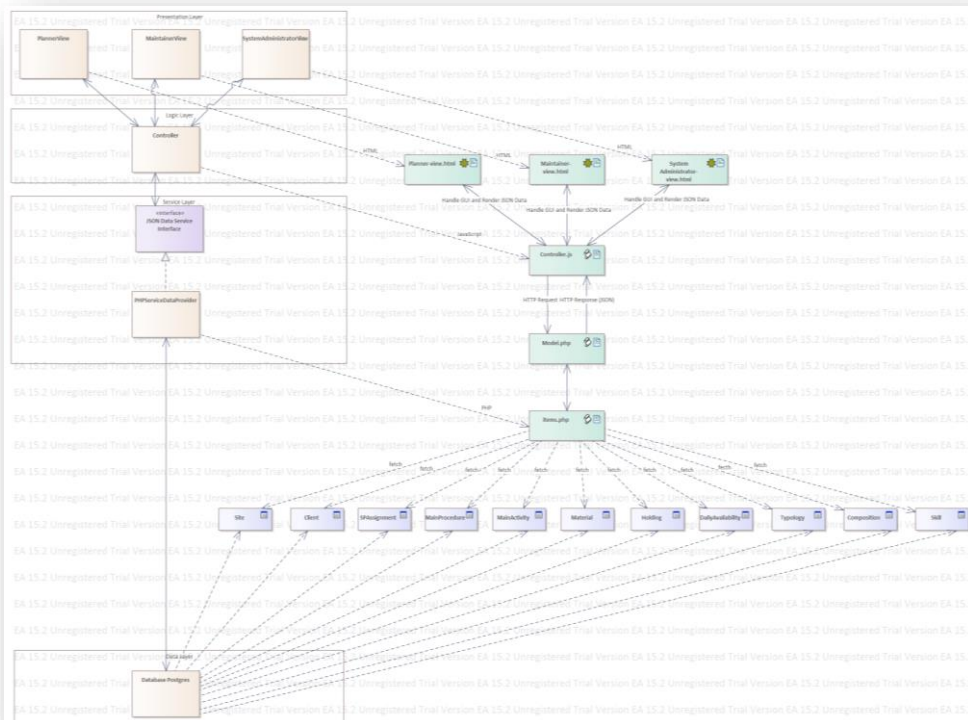


Figura 12: Service Layer

Il primo layer ossia il Presentation Layer mostra le tre pagine html che sono state usate per realizzare i tre ruoli descritti all'interno del documento SRS. Queste tre pagine html sono collegate, in modo bidirezionale, con il controller.js in modo tale da poter gestire le Graphical User Interface e interpretare i dati JSON.

Il Controller comunica con il Model attraverso uno scambio continuo di richieste e risposte HTTP. Il Model, inoltre, è collegato al Service Layer, all'interno del quale è presente il PHPServiceDataProvider che, nel progetto, è rappresentato da items.php.

Infine, il servizio fornito da PHP comunica con il Database Postgres per poter accedere alle tabelle e, eventualmente, poter modificare i dati con operazioni di remove, update o add.

2.2 Implementation Model

Dopo aver definito l'architettura e mentre l'applicazione prendeva forma, è stato realizzato un altro modello noto come Implementation Model.

Questo non è altro che una raccolta di componenti e dei sottosistemi di implementazione che li contengono. I componenti includono sia componenti deliverable, come eseguibili, sia componenti da cui vengono prodotti i deliverable, come file di codice sorgente.

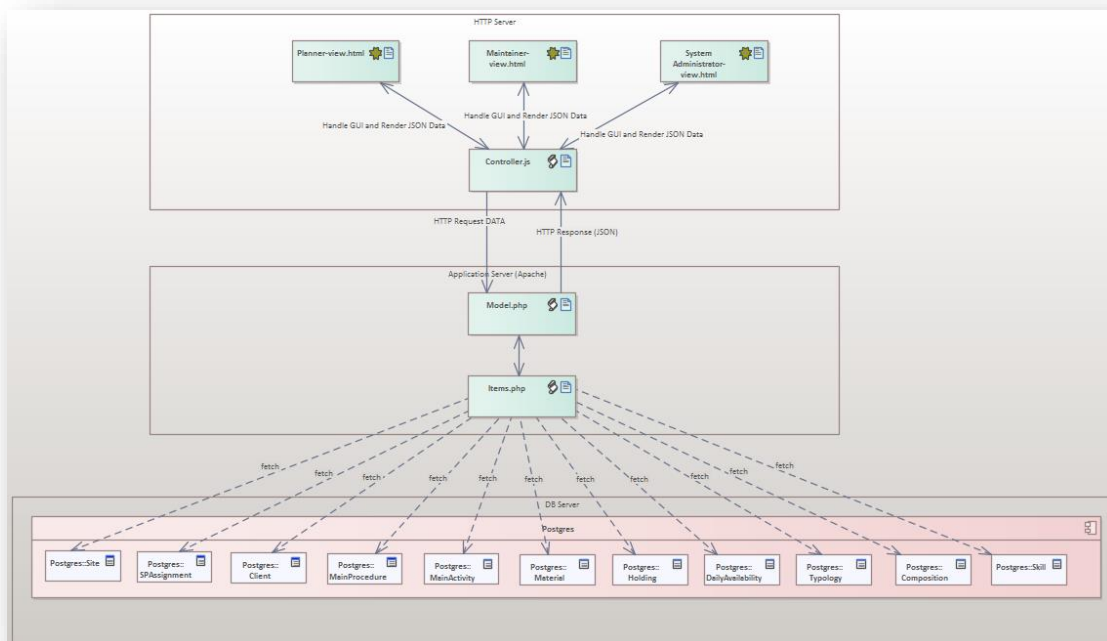


Figura 13: Implementation Model

L'HTTP Server e l'Application Server utilizzati comunicano tra di loro attraverso uno scambio di messaggi HTTP: l'Application Server rimane in ascolto di request sulla rete e produce la response adeguata usando il linguaggio indicato (php).

Infine, abbiamo items.php ossia il package che è collegato con tutte le tabelle presenti nel database realizzato.

2.3 Execution model

L'Execution Model specifica il comportamento degli elementi del linguaggio. In particolare, in *Figura 14* il modello è stato usato per legare insieme tutti gli aspetti del modello di programmazione di una pagina web. A partire dall'applicazione web realizzata in php, affinché possa essere invocata attraverso chiamate web, deve essere prima di tutto istanziata. Per fare ciò, è necessario mettere sul web tutte le implementazioni php sviluppate in modo tale da ottenere un'interazione di tipo client-server grazie all'utilizzo del protocollo HTTP. Per rendere visibile l'applicazione realizzata è stato necessario sfruttare i servizi forniti da Apache per rendere possibile l'esecuzione delle classi php e l'interazione con il database Postgres.

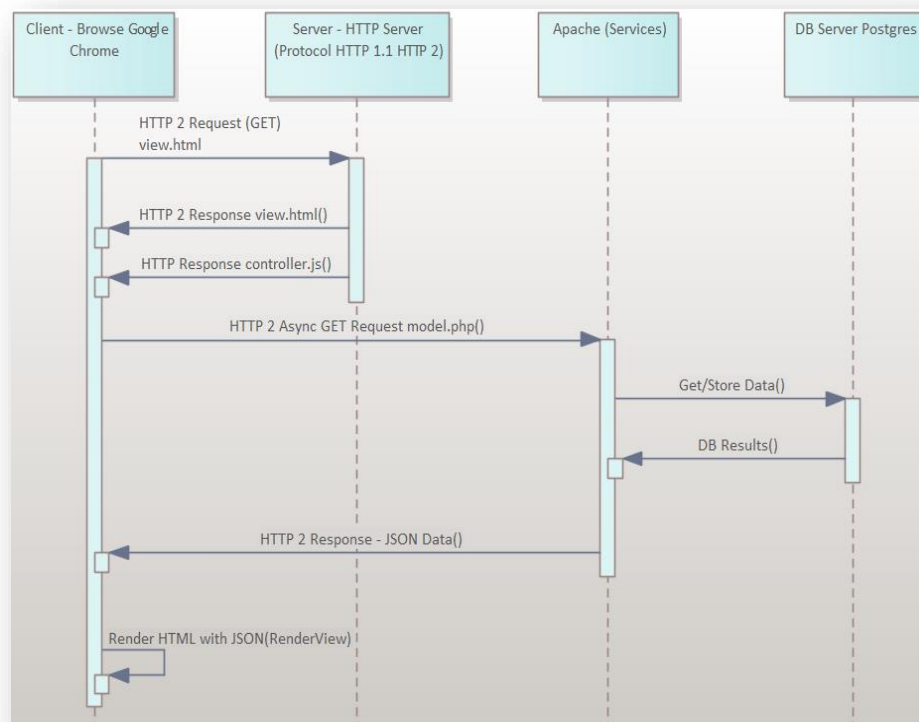


Figura 14: Execution Model

2.4 WEB Model

Il Web Model realizzato con Enterprise Architect permette di specificare il comportamento degli elementi del linguaggio relativamente allo sviluppo web realizzato.

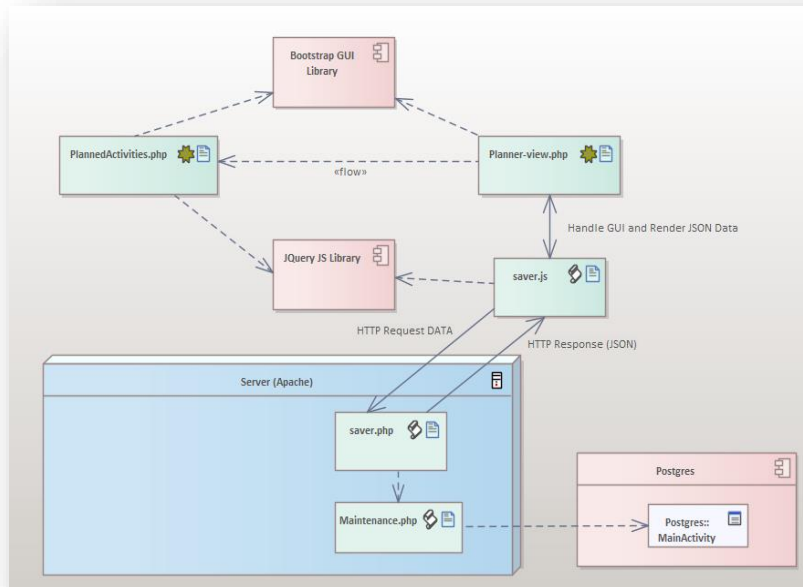


Figura 15: Web Model

Per la creazione dell'applicazione web è stata utilizzata Bootstrap, una libreria contenente modelli di progettazione basati su HTML e CSS così come alcune estensioni di JavaScript.

Il modello in *Figura 15* è stato realizzato prendendo in considerazione un solo caso specifico dell'applicazione ossia l'interazione tra la vista del ruolo Planner e la pagina **PlannedActivities.php**. Come specificato nel documento SRS, il Planner può visualizzare le attività pianificate in modo da poterle assegnare ad uno specifico Maintainer.

Inoltre, sia l'interfaccia **PlannedActivities.php** che il modello **saver.js** interagiscono con la libreria JQuery che permette di selezionare, manipolare, gestire eventi in pagine HTML e CSS. In particolare, **saver.js** rappresenta uno dei quattro controller che consentono di gestire i dati presenti nell'applicazione attraverso operazioni di aggiunta, rimozione, caricamento o aggiornamento.

Come già descritto nell'Execution Model, affinché l'applicazione web funzioni correttamente, il controller **saver.js** deve interagire con il **saver.php** presente all'interno del server Apache attraverso una serie di http response e request.

Infine, all'interno del server, le funzioni presenti nel file **saver.php** interagiranno con la pagina **Maintenance.php** la quale, a sua volta, salverà i dati passategli nella tabella **MainActivity** creata nel database.

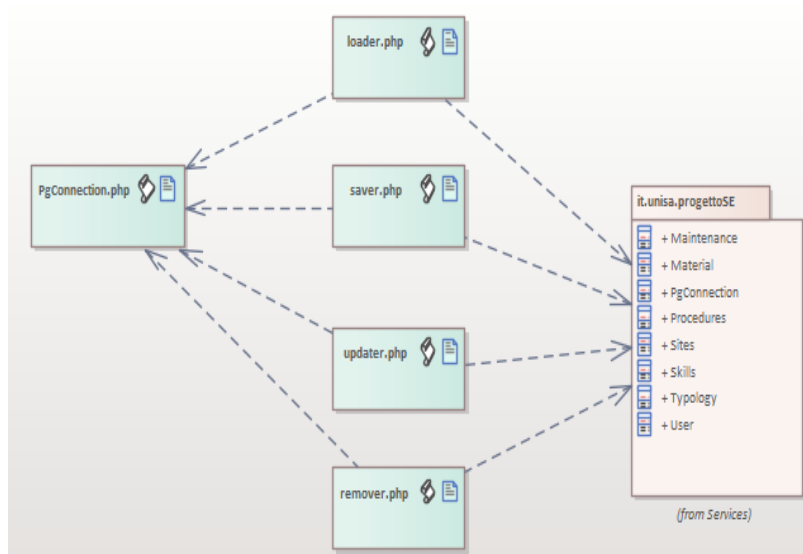
2.5 Class Model

Per avere una migliore comprensione di quanto realizzato è stato realizzato anche il Class Model in UML. In *Figura 16* sono rappresentate tutte le classi che vanno ad interagire con la classe PgConnection che consente di connettersi al database realizzato in Postgres.



Figura 16: Class Model

Possiamo vedere l'interazione delle classi anche nel seguente modo:



I quattro moduli loader.php, saver.php, updater.php e remover.php consentono, rispettivamente, di caricare, salvare, aggiornare e rimuovere i dati presenti sul database.

3. Object Design

L'approccio di progettazione migliore tra tutti è sicuramente quello ad oggetti. La progettazione ad oggetti garantisce proprietà quali:

- *Riusabilità*: singoli componenti identificati nella progettazione del sistema possono essere tradotti in classi con funzionalità e caratteristiche ben specifiche, utilizzabili anche in contesti diversi da quello di partenza.
- *Manutenibilità*: le modifiche sono limitate alle singole classi, permettendo di cambiarne anche l'intera implementazione se mantenuta l'interfaccia. Anche l'individuazione e la correzione di errori risulta più semplice.
- *Disaccoppiamento*: ogni classe è altamente coesa e gestisce un insieme di dati omogenei e le operazioni che agiscono su di essi, limitando quanto più possibile la dipendenza da elementi esterni alla classe stessa.

Per questo durante la progettazione è stato individuato un insieme di classi sulla base dei requisiti funzionali. Nello specifico, è stata creata una classe per ogni entità del database data la necessità di caricare, salvare o modificare i dati presenti nel db. In questo modo le azioni su ogni specifica tabella sono state individuate e racchiuse all'interno di specifiche classi, riducendo al minimo l'accoppiamento e garantendo una massima coesione.

In aggiunta è stata pensata una classe che facesse da interfaccia per il database, in modo da garantire l'indipendenza dallo specifico database utilizzato (in questo caso PostgreSQL). In questo modo risulterà possibile in futuro cambiare tecnologia con un impatto piccolissimo sul software (basterà agire su una sola classe).

3.1 Design Pattern

Nella fase di progettazione ad oggetti è buona norma tenere ben a mente i design pattern individuati nel corso del tempo dagli sviluppatori che ci hanno preceduto. Un design pattern è una soluzione progettuale generale ad un problema ricorrente e permette di affrontare i problemi di progettazione rifacendosi a modelli ben definiti e testati nel corso degli anni.

Un design pattern è caratterizzato da quattro elementi:

- Un *nome* che identifica univocamente il pattern.
- Una *descrizione del problema* che descrive le situazioni in cui il pattern può essere usato.
- Una *soluzione* definita come un insieme di classi e interfacce collegate.
- Un insieme di *conseguenze* che spiegano i trade-off e le alternative da considerare rispetto ai propri obiettivi.

3.2 Command

Il primo design pattern individuato è il *command*, un pattern comportamentale. Viene utilizzato quando si ha la necessità di disaccoppiare l'invocazione di un comando dai suoi dettagli implementativi, separando colui che invoca il comando da colui che esegue l'operazione.

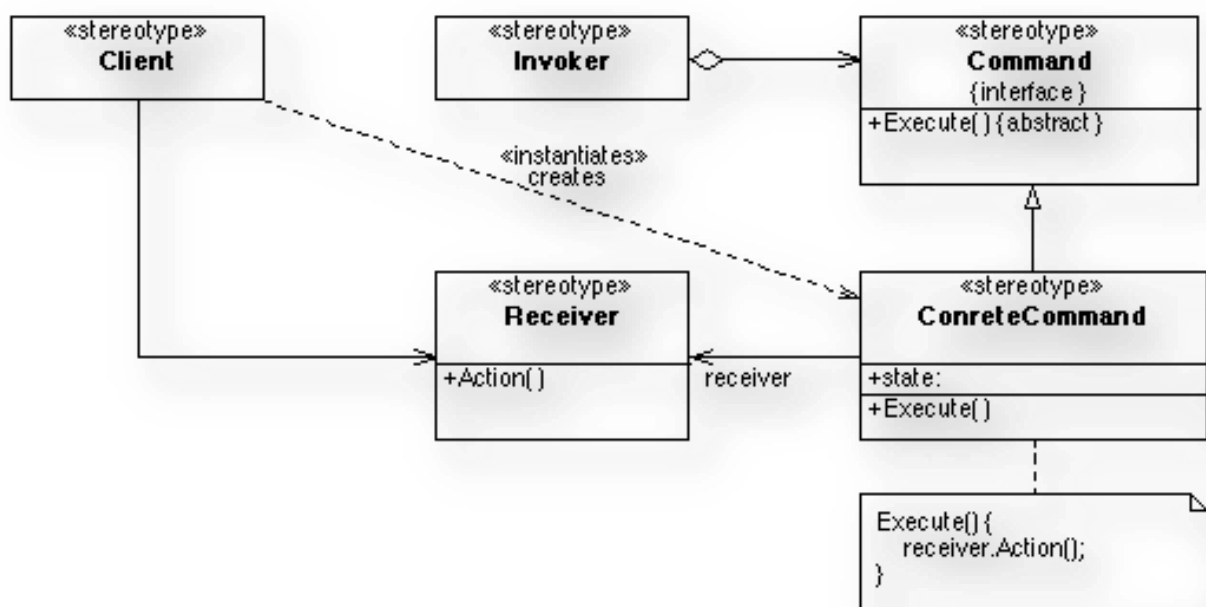
Tale operazione viene realizzata attraverso la catena *Client->Invocatore->Ricevitore*

1. Il *Client* non è tenuto a conoscere i dettagli del comando ma il suo compito è solo quello di chiamare il metodo dell'Invocatore che si occuperà di intermediare l'operazione.
2. L'*Invocatore* ha l'obiettivo di incapsulare, nascondere i dettagli della chiamata come nome del metodo e parametri.
3. Il *Ricevitore* utilizza i parametri ricevuti per eseguire l'operazione

Ma tra l'Invocatore ed il Ricevitore viene posto il *Command* ossia il comando da eseguire. Il *Command* è una semplice interfaccia che viene implementata da una o più classi concrete che invocano il *Ricevitore*.

Nell'architettura definita per il progetto, di tipo MVC, il ruolo del client è occupato dalle interfacce html (view), che richiedono servizi ad un controller (invocatore). Il controller per recuperare i dati richiesti dal client comunica con un'interfaccia intermedia (es. loader.php) che assume il ruolo di command, invocando a seconda della richiesta uno dei metodi presenti nelle classi che assumono il ruolo di model.

In questo modo è stato possibile disaccoppiare l'invocatore dal ricevitore, dando la possibilità di lavorare in parallelo su due parti del software una volta definita l'interfaccia. Inoltre è stato possibile aggiungere man mano nuove funzionalità attraverso i due sprint ampliando l'interfaccia command.



3.3 Singleton

Il singleton è uno dei pattern più semplici. È un pattern creazionale basato su una singola classe responsabile di creare un'istanza di un oggetto facendo in modo che questa sia l'unica nell'intero programma.

Si creerà quindi una classe *SingleObject* con un costruttore privato, in modo tale da non permettere l'istanziamento all'esterno. La classe potrà essere utilizzata richiamando il metodo statico *getInstance*, che restituirà un'istanza dell'oggetto salvata in un attributo privato e statico della classe.



Una critica al Singleton potrebbe essere la possibilità di allocare una variabile globale a sostituzione del pattern. Questa scelta è però meno preferibile per:

- Evitare di sporcare lo spazio globale dei nomi con variabili non necessarie.
- Permettere l'allocazione e l'inizializzazione dell'oggetto solo quando necessario e non all'inizio del programma.

Il pattern è stato utilizzato nell'implementazione delle classi nella parte architetturale del model (quali User.php, Skill.php, ecc.). Poiché queste classi hanno il solo scopo di interagire con il database, e quindi il comportamento dei metodi al loro interno è caratterizzato solo dai parametri dati in input, è stato deciso di permettere di avere una sola istanza all'interno del software.

4. Testing

Uno degli obiettivi principali dello sviluppo software è rilasciare al cliente un prodotto di alta qualità, ovvero un prodotto che soddisfi le sue specifiche e le sue aspettative. Affinché ciò accada, è necessario trovare ed eliminare il maggior numero di errori presenti nel software. Questa attività è svolta attraverso varie tecniche di error detection, fra cui il testing. Quest'ultimo è il processo di scoperta delle eventuali differenze fra il comportamento atteso del sistema e il comportamento osservato del sistema implementato. È importante notare che il testing non può provare che un prodotto funzioni ma può solo trovare difetti e mostrare come si comporta quando viene sollecitato dai casi di test forniti e non dà alcuna garanzia per tutti gli altri casi in cui non viene testato, dunque è fondamentale invalidare quanto più possibile il software.

Poiché l'attività in esame richiede l'impegno di tempo e risorse, sia umane che materiali, è necessario pianificarla all'inizio, così come avviene per tutte le altre azioni che conducono alla realizzazione del prodotto. Successivamente, il tester procede con l'unit testing, ovvero la ricerca di differenze fra le specifiche degli oggetti e le singole unità funzionali (metodi, classi, sottosistemi). Si passa poi al functional testing, cioè alla ricerca di differenze fra i requisiti funzionali scritti nelle user stories e il sistema software. A differenza dello unit test, dove i test vengono automatizzati per facilitare il testing di regressione, in questa fase è necessario l'intervento dell'uomo. Infine, si giunge all'integration testing, dove vengono testati i componenti in combinazione coinvolgendo l'intero sistema, per verificare il giusto funzionamento della comunicazione fra i moduli.

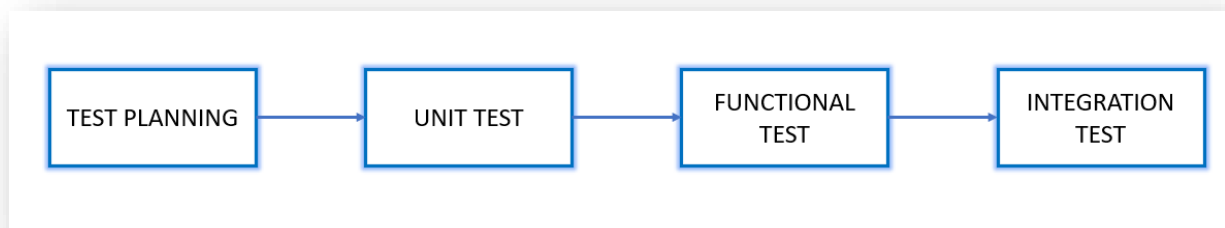


Figura 17: Fasi del testing

4.1 Test Planning

Inizialmente, per ogni classe della web application è stato prodotto il "Class test design specification" che contiene le seguenti informazioni: l'identificativo del documento, la versione, la data in cui è stato scritto, l'identificativo e il nome della classe da testare, il nome del programmatore che ha sviluppato la classe, a quali requisiti dell'SRS fa riferimento, qual è l'obiettivo del test, quali sono i metodi da testare, la data di inizio e di fine del test, l'approccio scelto, la tecnica e il tool usati e infine qual è il criterio di successo o insuccesso del test.

Segue un esempio di documento Class test design specification compilato in ogni sua parte, in particolare si riporta quello della classe Maintenance, usata come riferimento in tutte le argomentazioni successive.

CLASS TEST DESIGN SPECIFICATION		
ID: CTD006	Versione: 1.0	Data: 05/12/20
INFORMAZIONI SULLA CLASSE		
ID Classe: C6	Nome: Maintenance	Versione: 1.0
Programmatore: Angelo Vistocco		
RIFERIMENTI		
Specifica dei requisiti:	Documenti: SRS	
	Requisiti collegati: P1, P2, P3, P4, P6, P7, FRP8	
Obiettivi del test:		
Il test dovrà verificare il corretto funzionamento dei metodi contenuti nella classe Maintenance		
METODI DA TESTARE		
Nome	Rischi	Nuovo/Modificato
save		nuovo
getByWeek		nuovo
loadActivity		nuovo
loadWeeks		nuovo
updateActivity		nuovo
removeActivity		nuovo
Data inizio: 13/12/2020		Data fine: 13/12/2020
APPROCCIO		
<input type="checkbox"/> Debug <input type="checkbox"/> Ispezione <input checked="" type="checkbox"/> Testing		
Tecnica:	Blackbox	
Tool: PHPUnit		
Criteri Pass/Fail:		
Il test ha successo quando viene trovato almeno un errore nei casi di test pianificati		

Figura 18: Class test design specification della classe Maintenance

4.2 Unit Test

Dopo la pianificazione dell'attività di test, uno dei compiti principali dello sviluppatore è generare i casi di test. Infatti, per l'applicazione prodotta, per ogni classe è stata scritta la corrispondente classe di test e per ogni metodo della classe da testare è stato prodotto il rispettivo metodo di test. È stata inoltre utilizzata la tecnica black-box, dove metodi e classi vengono visti come black box, cioè non si accede alla loro implementazione ma si determina il loro comportamento osservando se gli input forniti producono l'output desiderato. Poiché i possibili input sono potenzialmente infiniti, per minimizzare il numero di test case si fa riferimento alle classi di equivalenza, ovvero la tecnica

specification-based più comune. Dunque, gli input sono stati divisi in classi e per ogni classe è stato scelto un rappresentante da dare al metodo di test. Sono stati considerati anche i casi limite e quelli di errore, conducendo quindi una boundary-value analysis e quando un metodo prevede più di un parametro in input, attraverso la tecnica di combinazione di condizioni sono stati scelte e testate alcune combinazioni.

Di seguito, a titolo di esempio, si riportano le classi di equivalenza costruite per il metodo save della classe Maintenance ma a corredo di questa relazione vengono fornite tutte le classi prodotte.

- METODO SAVE		
VARIABLE	CLASS	REPRESENTATIVE
IDSITE	ID maggiore di 0	1, 2, 3, ..., +∞ (2)
	ID minore o uguale a 0	-∞, ..., -3, -2, -1, 0 (-3)
DESCRIPTION	descrizione già presente nel database	replacement of robot 23 welding cables
	descrizione non presente nel database o vuota	testDesc
ESTIMATEDTIME	durata >0	1:00:00
	durata <=0	-1:00:00
WEEK	1<=settimana <=52	12
	settimane negative	-3
	settimane >52	60
INTERRUPTIBLE	yes/no	yes
IDTYPOLOGY	ID maggiore di 0	1, 2, 3, ..., +∞ (2)
	ID minore o uguale a 0	-∞, ..., -3, -2, -1, 0 (-3)
MTYPE	planned activity/EWO	planned activity
	stringa vuota	""

Figura 19: Classi di equivalenza e input selezionati per testare il metodo save

CLASS	REPR.	REPR.	REPR.	REPR.	REPR.	REPR.	REPR.
IdSite>0 – descrizione non presente – durata >0 – 1<=settimana<=52 – yes/no – idTypology >0 – planned activity/EWO	2	testDesc	1:00:00	12	yes	2	planned activity
IdSite>0 – descrizione già presente – durata >0 – 1<=settimana<=52 – yes/no – idTypology >0 – planned activity/EWO	2	replacement of robot 23 welding cables	1:00:00	12	yes	2	planned activity
IdSite<=0 – descrizione non presente – durata >0 – 1<=settimana<=52 – yes/no – idTypology >0 – planned activity/EWO	-3	testDesc	1:00:00	12	yes	2	planned activity
IdSite>0 – descrizione non presente – durata <=0 – 1<=settimana<=52 – yes/no – idTypology >0 – planned activity/EWO	2	testDesc	-1:00:00	12	yes	2	planned activity
IdSite>0 – descrizione non presente – durata >0 – settimane negative – yes/no – idTypology >0 – planned activity/EWO	2	testDesc	1:00:00	-3	yes	2	planned activity
IdSite>0 – descrizione non presente – durata >0 – settimane >52 – yes/no – idTypology >0 – planned activity/EWO	2	testDesc	1:00:00	60	yes	2	planned activity
IdSite>0 – descrizione non presente – durata >0 – 1<=settimana<=52 – yes/no – idTypology <=0 – planned activity/EWO	2	testDesc	1:00:00	12	yes	-3	planned activity
IdSite>0 – descrizione non presente – durata >0 – 1<=settimana<=52 – yes/no – idTypology <=0 – stringa vuota	2	testDesc	1:00:00	12	yes	-3	""

Figura 20: Casi di test scelti

Una volta stabilite le classi di equivalenza, si procede con la definizione dei casi di test, compilando per ognuno il "Class test case specification". Questo documento oltre ad indicare il suo identificativo, la versione, la data in cui è stato scritto, il nome del metodo da testare, qual è l'obiettivo di questo test, qual è il percorso del file da testare, è fondamentale in quanto permette

di visualizzare per ogni attributo del metodo su quale input viene testato e in corrispondenza di tutti gli input viene specificato il risultato atteso. L'esempio riportato sotto fa riferimento al documento prodotto per il metodo save della classe Maintenance.

CLASS TEST CASE SPECIFICATION			
TC ID: TCS021	TDS ID: CTD021	Versione: 1.0	Data: 06/12/20
Metodo da testare: save			
Obiettivo: Verificare che, dati in input id del luogo, la descrizione dell'attività, il tempo stimato, la settimana in cui si svolge l'attività, la possibilità di interruzione, l'id della tipologia e il tipo di attività (attività pianificata o EWO), viene inserita la nuova riga corrispondente nel database			
Percorso File: C:\Users\simon\Desktop\PROGETTO SE\esame\progetto_se_gruppo5-main\src\Items\Maintenance.php			
DATI DI TEST			
ATTRIBUTO	INPUT	RISULTATO ATTESO	OUTPUT
idsite	2	true	
description	testDesc		
estimatedtime	1:00:00		
week	12		
interruptible	true		
idtypology	2		
mtype	planned activity		
idsite	2	false	
description	replacement of robot 23 welding cables		
estimatedtime	1:00:00		
week	12		
interruptible	true		
idtypology	2		
mtype	planned activity		
idsite	-3	false	
description	testDesc		
estimatedtime	1:00:00		
week	12		
interruptible	true		
idtypology	2		
mtype	planned activity		
idsite	2	false	
description	testDesc		
estimatedtime	-1:00:00		
week	12		
interruptible	true		
idtypology	2		
mtype	planned activity		
idsite	2	false	
description	testDesc		
estimatedtime	1:00:00		
week	-3		
interruptible	true		
idtypology	2		
mtype	planned activity		
idsite	2	false	
description	testDesc		
estimatedtime	1:00:00		
week	60		
interruptible	true		
idtypology	2		
mtype	planned activity		
idsite	2	false	
description	testDesc		
estimatedtime	1:00:00		
week	12		
interruptible	true		
idtypology	-3		
mtype	planned activity		
idsite	2	false	
description	testDesc		
estimatedtime	1:00:00		
week	12		
interruptible	true		
idtypology	2		
mtype	---		

Figura 21: Class test case specification del metodo save della classe Maintenance

Infine, al momento dell'esecuzione dei test è stato compilato il "Class test incident report" che oltre alle informazioni quali identificativo del documento, versione, data in cui viene scritto, metodo da testare, obiettivo del test, il nome del tester e l'ambiente di test, permette di confrontare l'output osservato rispetto a quello atteso per ogni combinazione di input fornita. L'esempio seguente è il documento inerente al metodo save della classe Maintenance.

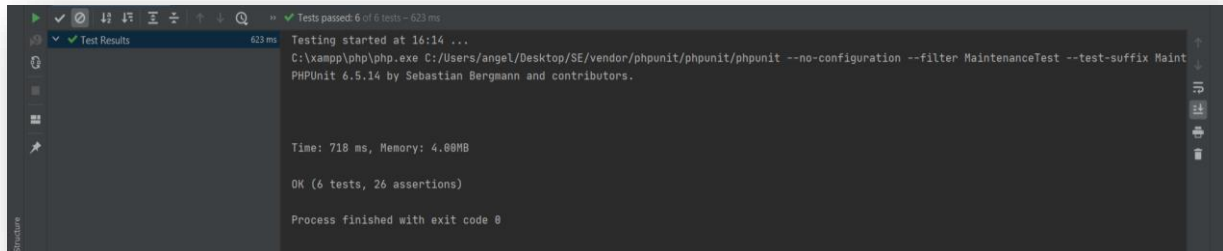
CLASS TEST INCIDENT REPORT			
TIR ID: TIR022	TCS ID: TCS022	Versione: 1.0	Data: 13/12/20
Metodo da testare: save			
Obiettivo: Eseguire i test case sul metodo save			
Tester: Angelo Vistocco			
Ambiente di test: Postazione PC del programmatore			
DATI DI TEST			
ATTRIBUTO	INPUT	RISULTATO ATTESO	OUTPUT
idsite	2	true	true
description	testDesc		
estimatedtime	1:00:00		
week	12		
interruptible	true		
idtypology	2		
mtype	planned activity	false	false
idsite	2		
description	replacement of robot 23 welding cables		
estimatedtime	1:00:00		
week	12		
interruptible	true		
idtypology	2	false	false
mtype	planned activity		
idsite	-3		
description	testDesc		
estimatedtime	1:00:00		
week	12		
interruptible	true	false	false
idtypology	2		
mtype	planned activity		
idsite	2		
description	testDesc		
estimatedtime	-1:00:00		
week	12	false	false
interruptible	true		
idtypology	2		
mtype	planned activity		
idsite	2		
description	testDesc		
estimatedtime	1:00:00	false	false
week	-3		
interruptible	true		
idtypology	2		
mtype	planned activity		
idsite	2		

mtype	planned activity	false	false
idsite	2		
description	testDesc		
estimatedtime	1:00:00		
week	60		
interruptible	true		
idtypology	2	false	false
mtype	planned activity		
idsite	2		
description	testDesc		
estimatedtime	1:00:00		
week	12		
interruptible	true	false	false
idtypology	-3		
mtype	planned activity		
idsite	2		
description	testDesc		
estimatedtime	1:00:00		
week	12	false	false
interruptible	true		
idtypology	2		
mtype	planned activity		
idsite	2		
description	testDesc		
estimatedtime	1:00:00	false	false
week	12		
interruptible	true		
idtypology	2		
mtype	planned activity		
idsite	2		

Impatto: ☐ alto ☐ medio ☒ basso

Figura 22: Class test incident report del metodo save della classe Maintenance

Infine, dopo aver eseguito i casi di test stabiliti per ogni metodo di ogni classe e dopo aver corretto eventuali difetti nel codice, il tool PHPUnit adoperato fornisce le seguenti informazioni di riepilogo per garantire che tutti i test sono stati superati con successo. Di seguito si riporta quanto ottenuto dall'esecuzione dei casi di test sulla classe Maintenance.



```
Tests passed: 6 of 6 tests - 623 ms
Testing started at 16:14 ...
C:\xampp\php\php.exe C:/Users/angel/Desktop/SE/vendor/phpunit/phpunit/phpunit --no-configuration --filter MaintenanceTest --test-suffix Maint
PHPUnit 6.5.14 by Sebastian Bergmann and contributors.

Time: 718 ms, Memory: 4.88MB

OK (6 tests, 26 assertions)

Process finished with exit code 0
```

Figura 23: Test eseguiti con successo sulla classe Maintenance

4.3 Integration Test

Dopo aver verificato la corretta corrispondenza fra le specifiche richieste dal cliente e il sistema software realizzato, si giunge alla fase finale del testing ovvero all'integrazione dei vari moduli per trovare errori riguardanti la loro comunicazione. Fra le varie possibili strategie, è stata usata la strategia bottom up che testa prima ogni componente dello strato più basso individualmente e poi lo integra con componenti degli strati superiori. Quando necessario sono stati usati dei driver allo scopo di testare l'interfaccia ma successivamente sono stati sostituiti con l'implementazione completa ed è stato ripetuto il test.