Pianificazione presentazioni progetti

Titolo del progetto	Pianificazione presentazioni progetti
Alunno/a	Michea Colautti
Classe	I4AA
Anno scolastico	2022/2023
Docente responsabile	Guido Montalbetti
Perita	Ioulia Zintchenko

1 Indice

2 Introduzione

2.1 Informazioni sul progetto

In questo capitolo raccogliere le informazioni relative al progetto, ad esempio:

- Progetto: LPI SAMT 2022/23
- Nome del progetto: Pianificazione presentazioni progetti
- Allievi coinvolti nel progetto: Michea Colautti
- Classe: I4AA - Scuola Arti e Mestieri Trevano
- Docenti responsabili: Guido Montalbetti
- Perita esame: Ioulia Zintchenko
- Data inizio: 02 maggio 2023.
- Data di fine: 26 maggio 2023
- Linguaggio *frontend*: JS con AJAX
- Linguaggio *backend*: MVC - PHP

2.2 Abstract

Lo sviluppo di progetti alla SAMT è un tema centrale della formazione degli studenti. A partire dalla terza si sviluppano 2 progetti all'anno; se si calcola LPI a fine quarta si arriva ad un totale di 5 progetti a studente. Quando si parla di pianificare le presentazioni, considerando che ci sono circa 20 studenti per ogni annata, si arriva ad avere 100 presentazioni all'anno. Con questa mole di dati è naturale che, in un contesto così informatizzato, sorga l'idea di delegare la programmazione delle presentazioni ad un programma. È proprio questo che si problema che il mio progetto mira a risolvere. Con il mio programma, inserendo dati importanti come gli orari delle classi, gli impegni dei singoli alunni e non da ultimo gli orari dei docenti, si può generare e consultare una griglia con le presentazioni dei progetti completa, e che tenga conto delle esigenze di tutti. Ma non solo questo: il mio progetto permetterà anche un'interazione diretta con tutti gli utenti coinvolti. Infatti, quando una nuova programmazione sarà disponibile, i formatori dei progetti avranno modo di collegarsi al sito e approvare o rifiutare la programmazione proposta. In seguito il gestore prenderà nota del problema e modificando i parametri di generazione sarà in grado di proporre una nuova programmazione

The development of projects at SAMT is a central theme of studens learngin. From the third year onwards, 2 projects are developed per year; if we calculate LPI at the end of the fourth year, we arrive at a total of 5 projects per student. When it comes to planning presentations, considering that there are about 20 students per year, we arrive at 100 presentations per year. With this amount of data, it is natural that, in such a computerised context, the idea of delegating the scheduling of presentations to a programme arises. It is precisely this problem that my project aims to solve. With my programme, by entering important data such as class timetables, individual pupils' schedules and, last but not least, teachers' timetables, a comprehensive project presentation grid can be generated and consulted. But not only this: my project will also allow direct interaction with all users involved. In fact, when a new schedule becomes available, project trainers will be able to log on to the site and approve or reject the proposed schedule. The manager will then take note of the problem and by modifying the generation parameters will be able to propose a new schedule.

2.3 Scopo

Il progetto riguarda la creazione di una piattaforma per la pianificazione delle presentazioni dei progetti e dell'LPI presso la SAMT. La piattaforma prevede un'interfaccia composta da più pagine protetta da autenticazione nella quale l'amministratore può recarsi e programmare una pianificazione. Per i docenti, invece, è prevista una pagina pubblica dedicata alla visualizzazione della pianificazione.

Le pagine a disposizione dell'amministratore sono quelle per:

- Docenti
- Studenti
- Classi
- Aule

Per creare una nuova pianificazione, la prima cosa che deve essere possibile fare e selezionare il periodo desiderato per le presentazioni e i dati del responsabile e del capo laboratorio. Successivamente, è necessario inserire i dati delle persone all'interno della scuola, iniziando dai docenti. Il processo di inserimento dei dati dei docenti prevede l'utilizzo di più pagina raggiungibili per esempio da una barra laterale, che permette di creare ed inserire i dati dei docenti coinvolti, tra cui i recapiti personali e le fasce orarie di disponibilità per i progetti.

Nel sito è possibile anche creare ed inserire i dati delle classi, associandole con il nome e le fasce orarie dei progetti. In seguito, è possibile inserire gli allievi e associarli ad un progetto. Una volta creato l'allievo con il suo progetto, è obbligatorio associarlo ad una classe per semplificare la gestione futura della pianificazione. Inoltre, è possibile inserire eventuali impedimenti degli allievi.

L'ultima parte del progetto prevede l'inserimento delle aule della scuola e della loro disponibilità, nonché la personalizzazione delle caratteristiche come l'antenna Wi-Fi. Infine, è possibile generare la tabella di pianificazione, tenendo conto dei vincoli sulle ore dei progetti.

Per i docenti, è previsto un meccanismo di approvazione della pianificazione, con un campo per giustificare eventuali problemi con la pianificazione. Una volta che tutti gli insegnanti hanno dato la propria approvazione, nella versione originale, era possibile inviare una email con il PDF della pianificazione a studenti e docenti. Tuttavia questa funzione è stata rimossa, dato che le politiche di sicurezza e le infrastrutture informatiche all'interno della SAMT non lo permettono.
Infine, è previsto il mantenimento delle vecchie pianificazioni. Infine, tutti i dati inseriti, come quelli delle aule, dei docenti e degli allievi, sono modificabili per generare una nuova pianificazione.

3 Analisi

3.1 Analisi del dominio

Per quanto esistano una moltitudine di applicazioni, come dei calendari, che permettono di gestire una serie di eventi penso che ci siano ancora relativamente ancora pochi programmi che sono invece capaci di generare dei calendari di pianificazione eventi: questo perché spesso risulta difficile se non impossibile creare un programma che abbia abbastanza funzionalità e che sia modulabile a sufficienza per adattarsi a tutti i casi d'uso che si presentano. Questo porta quindi alla creazione di più applicazioni distinte molto specifiche e diverse tra di loro. Pensando alla mia situazione per esempio, non so quante altre realtà avranno bisogno di gestire un calendario di presentazioni progetti informatici, tenendo conto di docenti aule ed esigenze speciali degli allievi. Tuttavia il fatto che le applicazioni siano molto specifiche ha anche dei vantaggi; ad esempio trattandosi di un programma di dimensioni limitate è possibile concentrarsi molto su quelle poche funzioni necessarie e svilupparle bene. Oppure, siccome verrà usato da pochi, si può soddisfare al meglio i casi d'uso per cui è stato pensato.
Il mio prodotto potrebbe essere utile nell'assistere colui che deve pianificare le presentazioni della SAMT, che ora non deve più farlo a mano e non ha più l'incombenza di organizzare tutte le fasce orarie preoccupandosi di tener conto delle esigenze e delle preferenze di tutti. Inoltre il mio programma sarà relativamente facile ed intuitivo da utilizzare, rendendolo pronto all'uso, senza alcun *traning*, o quasi.

3.2 Analisi e specifica dei requisiti

Requisito	Req-001	**Priorità**	1	**Versione**	1.0
Nome	Pagina autenticazione per generazione				
Note	Home page con la possibilità di autenticarsi e programmare una pianificazione				

Requisito	Req-002	**Priorità**	1	**Versione**	1.0
Nome	Visualizzazione pianificazione pubblica				
Note	Pagina pubblica per la visualizzazione della pianificazione				

Requisito	Req-003	**Priorità**	1	**Versione**	1.0
Nome	Inserimento dati pianificazione				
Note	Inserimento periodo desiderato per le presentazioni				
Sotto requisiti					
001	Inserimento dati responsabile e capo laboratorio				

Requisito	Req-004	**Priorità**	1	**Versione**	1.0
Nome	Inserimento dati docenti				
Note	Nel sito deve essere possibile creare ed inserire i dati dei docenti coinvolti: recapiti personali e fasce orarie di progetti				

Requisito	Req-005	**Priorità**	1	**Versione**	1.0
Nome	Inserimento dati classi				
Note	Nel sito deve essere possibile creare ed inserire dati delle classi: nome e fasce orarie progetti				

Requisito	Req-006	**Priorità**	1	**Versione**	1.0
Nome	Inserimento aule				
Note	Inserire aule di progetto, con la loro disponibilità.				
Sotto requisiti					
001	Personalizzare caratteristiche come antenna WiFi.				

Requisito	Req-007	**Priorità**	1	**Versione**	1.0
Nome	Inserimento allievi + progetto				
Note	Nel sito deve essere possibile creare ed inserire degli allievi, creando e associando loro un progetto.				
Sotto requisiti					
001	Associare un allievo ad una classe, questo semplificherà di molto la gestione futura della pianificazione.				
002	Inserire eventuali impedimenti allievi				

Requisito	Req-008	**Priorità**	1	**Versione**	1.0
Nome	Generazione tabella pianificazione				
Note	Tabella tenendo conto dei vincoli sulle ore di progetti				

Requisito	Req-009	**Priorità**	2	**Versione**	1.0
Nome	Meccanismo approvazione pianificazione				
Note					

Requisito	Req-010	**Priorità**	2	**Versione**	1.0
Nome	Mantenimento vecchie pianificazioni				
Note					

Requisito	Req-011	**Priorità**	2	**Versione**	1.0
Nome	Pagina di modifica dati (disponibilità, ecc) allievi, docenti e aule				
Note					

Requisito	Req-012	**Priorità**	2	**Versione**	1.0
Nome	Sistema di logging				
Note	Punto specifico 224				

3.3 Use case

Per il mio *use case* ho deciso, come in passato, di mostrare 2 utenti; uno eredita le azioni dall'altro.

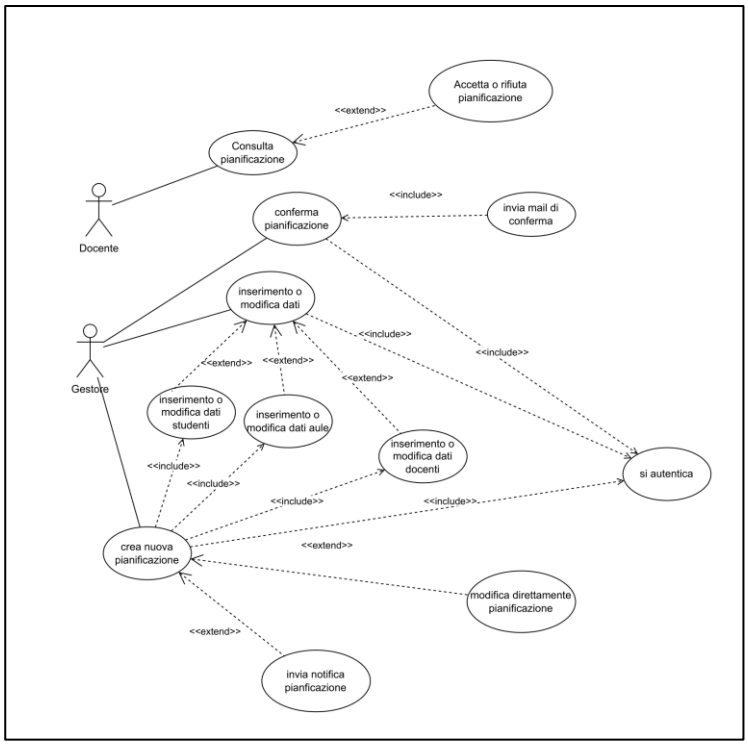


Figura 1 Use Case

Il primo utente è un **docente,** questo è l'utente classico nonché il più semplice: un docente può consultare la pianificazione sul sito, e può decidere di approvarla o rifiutarla.

Il secondo utente è un po' più complesso, ed è il **gestore** o **admin.**
Esso deve per forza autenticarsi per eseguire la maggior parte delle sue azioni: una volta compiuto questo importante passaggio il **gestore** può creare una nuova pianificazione, inviando una notifica di creazione. La creazione dipende però dall'inserimento dei dati necessari: senza di essi non può esserci alcuna pianificazione.
Una volta creata la pianificazione la può modificare direttamente oppure modificare i dati inseriti in precedenza.

Come detto prima il gestore può ovviamente consultare ed approvare la pianificazione, dato che non è detto che il gestore non sia anche un docente.

3.4 Pianificazione

Il modello concettuale che ho voluto adottare è il classico modello per i nostri progetti: quello *Waterfall.* Ho prodotto il seguente diagramma di Gantt.

	Modalità attività	WBS	Nome attività	Durata	Inizio	Fine	Predec
1		**1**	**Pianificazione presentazione progetti**	**82.5 h**	**mar 02.05.23**	**ven 26.05.23**	
2		**1.1**	**Analisi**	**8 h**	**mar 02.05.23**	**mer 03.05.23**	
3		1.1.1	Gantt	3 h	mar 02.05.23	mar 02.05.23	
4		1.1.2	Analisi requisiti	2.5 h	mar 02.05.23	mer 03.05.23	3
5		1.1.3	Analisi dei mezzi e teconlogie	1.5 h	mer 03.05.23	mer 03.05.23	4
6		1.1.4	Informazione mezzi	1 h	mer 03.05.23	mer 03.05.23	5
7		**1.2**	**Progettazione**	**8.5 h**	**mer 03.05.23**	**ven 05.05.23**	
8		1.2.1	Schema di flusso	2 h	mer 03.05.23	gio 04.05.23	2
9		1.2.2	Schema ER	2.5 h	gio 04.05.23	gio 04.05.23	8
10		1.2.3	Use Case	1 h	gio 04.05.23	ven 05.05.23	9
11		1.2.4	Design interfacce	3 h	ven 05.05.23	ven 05.05.23	10
12		**1.3**	**Implementazione**	**55 h**	**ven 05.05.23**	**mer 24.05.23**	
13		1.3.1	Implementazione DB	2 h	ven 05.05.23	lun 08.05.23	7
14		1.3.2	Wizzard inserimento dati	16 h	lun 08.05.23	gio 11.05.23	13
15		1.3.3	Motore creazione pianificazione	22 h	gio 11.05.23	mer 17.05.23	14
16		1.3.4	Invio Email e meccanismo conferma	8 h	mer 17.05.23	lun 22.05.23	15
17		1.3.5	Consultazione pianificazioni precedenti	7 h	lun 22.05.23	mer 24.05.23	16
18		**1.4**	**Test**	**6 h**	**mer 24.05.23**	**gio 25.05.23**	
19		1.4.1	Test	6 h	mer 24.05.23	gio 25.05.23	12
20		**1.5**	**Chiusura**	**2.5 h**	**ven 26.05.23**	**ven 26.05.23**	
21		1.5.1	Stampa, ritocchi doc, altro	2.5 h	ven 26.05.23	ven 26.05.23	18
22		1.6	Documentazione	22.5 h	mer 03.05.23	ven 26.05.23	

Figura 2 Gantt preventivo - ore

	Modalità attività	WBS	Nome attività	Durata	Inizio	Fine	Predec
1		**1**	**Pianificazione presentazione progetti**	**82.5 h**	**mar 02.05.23**	**ven 26.05.23**	
2		**1.1**	**Analisi**	**8 h**	**mar 02.05.23**	**mer 03.05.23**	
3		1.1.1	Gantt	3 h	mar 02.05.23	mar 02.05.23	
4		1.1.2	Analisi requisiti	2.5 h	mar 02.05.23	mer 03.05.23	3
5		1.1.3	Analisi dei mezzi e teconlogie	1.5 h	mer 03.05.23	mer 03.05.23	4
6		1.1.4	Informazione mezzi	1 h	mer 03.05.23	mer 03.05.23	5
7		**1.2**	**Progettazione**	**8.5 h**	**mer 03.05.23**	**ven 05.05.23**	
8		1.2.1	Schema di flusso	2 h	mer 03.05.23	gio 04.05.23	2
9		1.2.2	Schema ER	2.5 h	gio 04.05.23	gio 04.05.23	8
10		1.2.3	Use Case	1 h	gio 04.05.23	ven 05.05.23	9
11		1.2.4	Design interfacce	3 h	ven 05.05.23	ven 05.05.23	10
12		**1.3**	**Implementazione**	**55 h**	**ven 05.05.23**	**mer 24.05.23**	
13		1.3.1	Implementazione DB	2 h	ven 05.05.23	lun 08.05.23	7
14		1.3.2	Wizzard inserimento dati	16 h	lun 08.05.23	gio 11.05.23	13
15		1.3.3	Motore creazione pianificazione	22 h	gio 11.05.23	mer 17.05.23	14
16		1.3.4	Invio Email e meccanismo conferma	8 h	mer 17.05.23	lun 22.05.23	15
17		1.3.5	Consultazione pianificazioni precedenti	7 h	lun 22.05.23	mer 24.05.23	16
18		**1.4**	**Test**	**6 h**	**mer 24.05.23**	**gio 25.05.23**	
19		1.4.1	Test	6 h	mer 24.05.23	gio 25.05.23	12
20		**1.5**	**Chiusura**	**2.5 h**	**ven 26.05.23**	**ven 26.05.23**	
21		1.5.1	Stampa, ritocchi doc, altro	2.5 h	ven 26.05.23	ven 26.05.23	18
22		1.6	Documentazione	22.5 h	mer 03.05.23	ven 26.05.23	

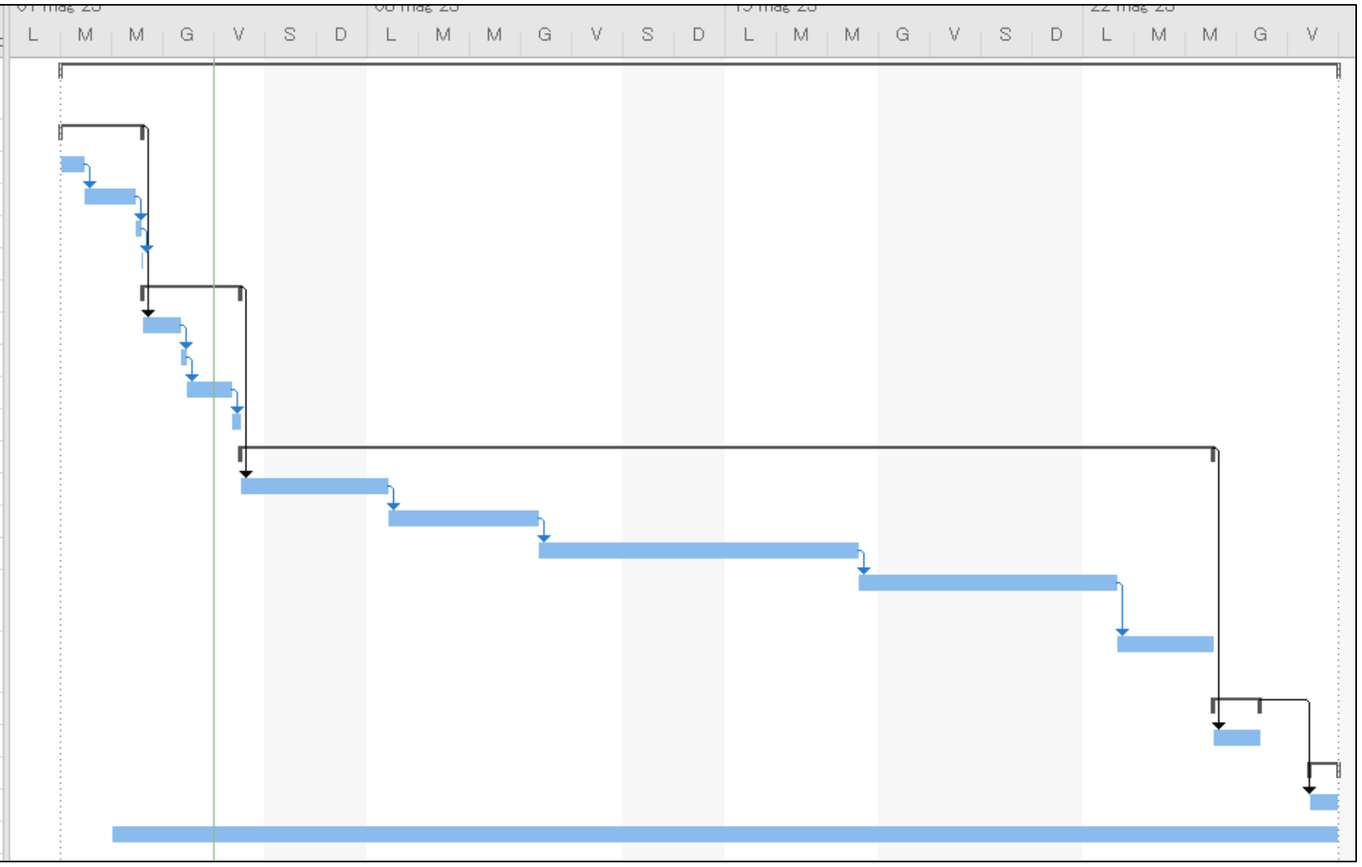


Figura 3 Gantt preventivo - completo

3.5 Analisi dei mezzi

3.5.1 Software

- Draw.io
- GitHub Desktop 2.9.12
- Gitlab
- Google Chrome
- Microsoft Office
- MySQL Workbench
- VS Code 1.7
- Windows 10 Enterprise
- XAMPP

3.5.2 Hardware

- PC scolastico
    - Processore: Intel I7 9700 – 3GhZ
    - RAM: 32GB
    - Scheda grafica: NVIDIA GeForce RTX 2060
    - SW: Windows 10 Enterprise 21H2

4 Progettazione

Per la progettazione, rispetto ad altri progetti, è stata più breve ma non per questo meno completa o utile.

4.1 Design dell'architettura del sistema

Per la struttura del sito ho riflettuto a lungo su cosa utilizzare. Per quanto la maggior parte della logica sarà gestita in JS il *backend* deve per forza essere implementato con un altro linguaggio. La scelta più ovvia è anche quella che ho deciso di applicare, ovvero un pattern di sviluppo noto: MVC.

4.2 Design dei dati e database

Come primo passo ho creato una bozza di schema ER con le convezioni che ci sono state insegnate a scuola, ovvero con la convenzione Chenn. Trovo che questo modo di rappresentare gli schemi sia migliore per quanto riguarda la parte concettuale e di progettazione iniziale. Questo schema mi ha anche permesso di identificare dei problemi, come ad esempio il concetto di progetti singoli/di gruppo. Inizialmente ho pensato che potessero esserci dei progetti di gruppo, quindi la relazione tra progetto e studente era di tipo molti-molti. Tuttavia discutendo con il formatore abbiamo concluso di dedicarci solo ai progetti di 4a, dove si svolgono solo progetti singoli. Essendo una bozza, e contenendo qualche errore, non la inserirò in questa documentazione, ma rimarrà comunque visibile nel mio repository.

Una volta imbastita la bozza, servendomi del *tool* grafico di MySQL Workbench, ho creato lo schema ER finale con le convenzioni volute dal formatore, chiamate Martin.
Nella prossima pagina lo documenterò in dettaglio.

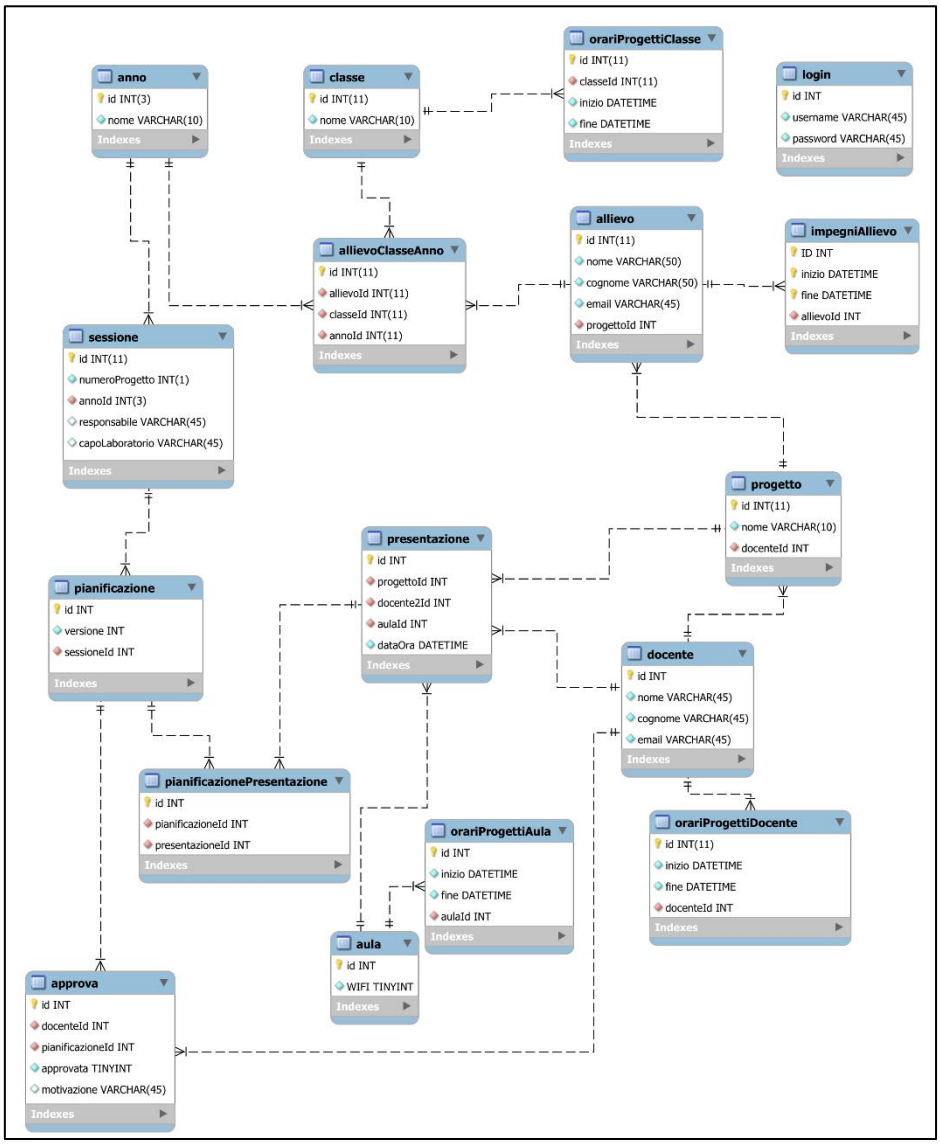


Figura 4 Diagramma ER

La prima entità è **anno**: in questa tabella sono memorizzati tutti gli anni, con un nome, ad esempio: 2022/2023.

Dall'entità anno parte l'entità **sessione**; essa rappresenta il semestre in cui viene svolto il progetto, tuttavia siccome il LPI viene svolto al secondo semestre ho deciso di generalizzare il concetto e chiamarla **sessione.** All'interno dell'entità sono contenuti dei dati amministrativi, come il nome del responsabile, e il riferimento all'anno in cui si svolge il progetto.

Da QdC doveva essere possibile generare più di un calendario di presentazioni per sessione di progetti, ed è per questo che ho creato l'entità **pianificazione,** sarà qui che saranno contenute le informazioni di base per le varie versioni di pianificazioni.
Per la stessa logica, una presentazione potrà essere inserita in più pianificazioni: per questo ho creato la tabella ponte **pianificazionePresentazione**.

Ho anche previsto l'entità **approva**, che servirà a raccogliere le approvazioni, o i rifiuti, della pianificazione dei progetti da parte dei docenti.

Per quanto riguarda l'entità principale dello schema ER, **presentazione,** l'ho pensata come un insieme di riferimenti (FK) ai dati che vengono usati per pianificarla:

- Una presentazione è strettamente legata ad un **progetto**. Oltre a sapere il nome del progetto e l'allievo a cui è assegnato, posso ottenere anche il docente che vi è assegnato.
- Oltra al **docente** principale è memorizzato anche un secondo docente di supporto

- Una presentazione si svolgerà in un'aula specifica, da qui la relazione con **aula.**

Nell'entità **progetto**, sono memorizzati, come accennato, l'allievo e il docente responsabile.
Ho creato la relazione **allievoClasseAnno** per motivi organizzatavi. Dato che mi è indispensabile sapere quando uno studente ha le ore di progetti nel suo orario per pianificare le presentazioni, e per evitare di assegnare una ventina di volte l'orario ad ogni studente ho pensato di creare l'entità **classe,** in modo da assegnare più allievi ad una classe, e poi assegnare gli orari di progetti alla classe tramite l'entità **orariProgettiClasse.** Grazie a questa architettura posso fare si che una determinata classe possa avere più fasce orarie di disponibilità per le presentazioni. Ho aggiunto il riferimento all'anno in quanto bisognava mantenere uno storico delle classi, e la classe chiamata, ad esempio, I4AA cambia di anno in anno.

La stessa logica usata in questa entità l'ho applicata anche per l'entità **docente** e **aula** con rispettivamente **orariProgettiDocente** e **orariProgettiAula.**
L'unica differenza è che lo studente può avere delle eccezioni nel suo orario, causate da congedi o assenze, per questo ho aggiunto l'entità **impegniAllievo** al DB.

Infine l'entità **login** permette la memorizzazione delle credenziali dell'amministratore.

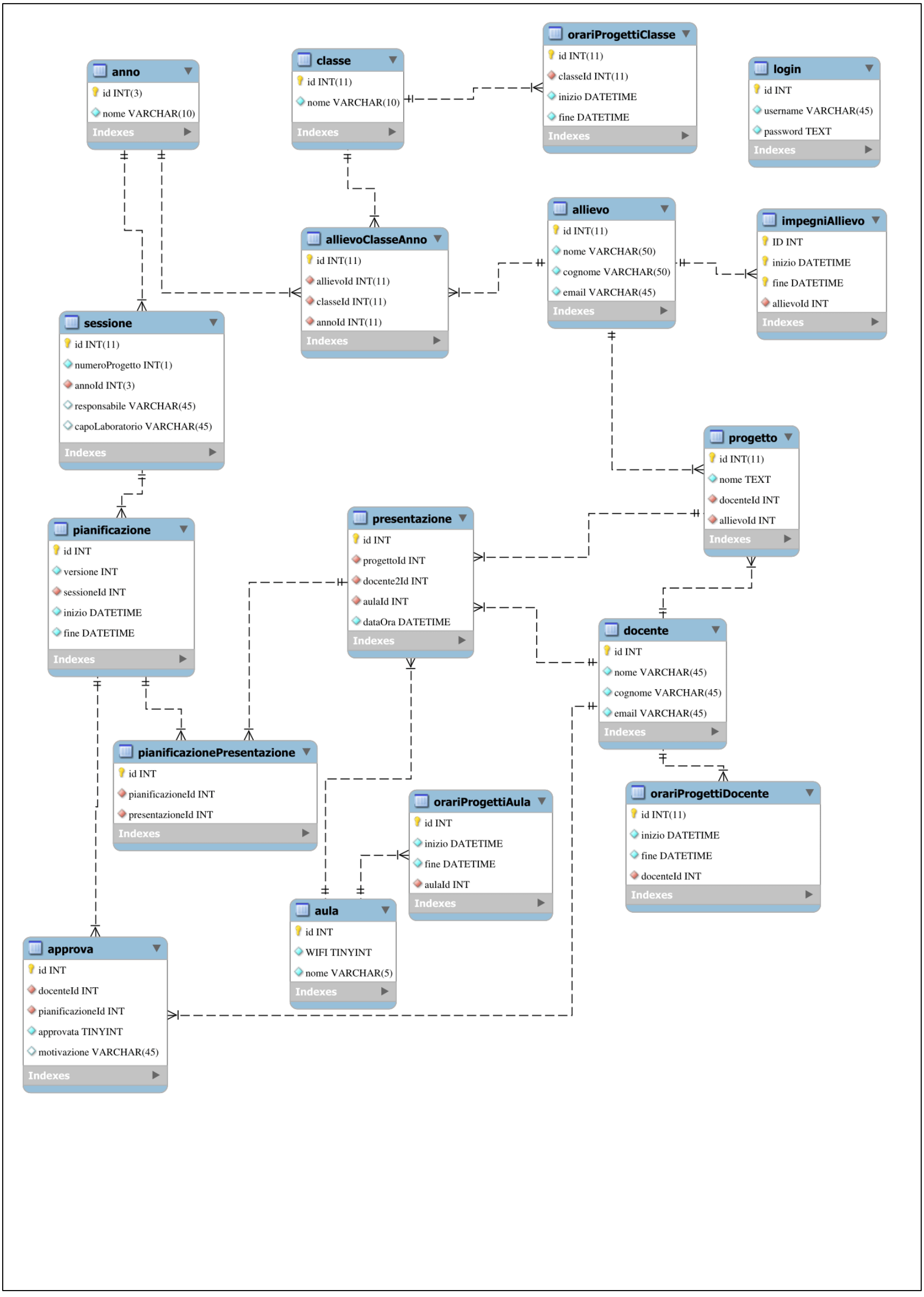


Figura 5 Diagramma ER - A3

4.3 Design delle interfacce

Rispetto ad altri progetti ho realizzato poche interfacce, dato che prevedo di mantenere lo stesso stile per tutte le pagine. Si noti che ho basato il mio stile su un template Bootstrap (vedi diario 03.05.2023).

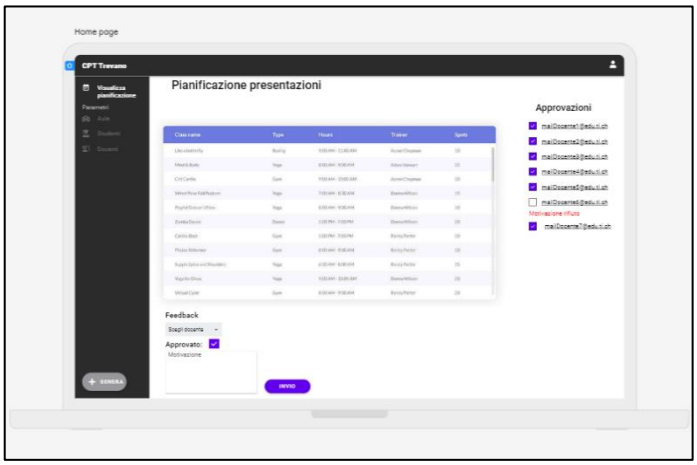


Figura 6 GUI - Home page

La prima interfaccia proposta è la home page. Questa in immagine è la versione **dell'amministratore** di sistema. Essenzialmente è possibile vedere la tabella di pianificazione e le conferme degli admin. Inoltre sotto la tabella è presente l'interfaccia di conferma della pianificazione.
Nella toolbar è presente il pulsante per la creazione di una pianificazione, visibile solo al gestore. Inoltre anche le conferme sono visibili solo ad un admin.

Questa a fianco è un esempio di pagina di creazione, c'è una tabella ed è possibile aggiungere/visualizzare i docenti. Cliccandoci sopra verrà aperto un *modal* che mostrerà la sua disponibilità.
In realtà oltre alla tabella dei dati sarà presente, sotto, anche un'altra tabella che servirà all'inserimento dei dati della disponibilità.

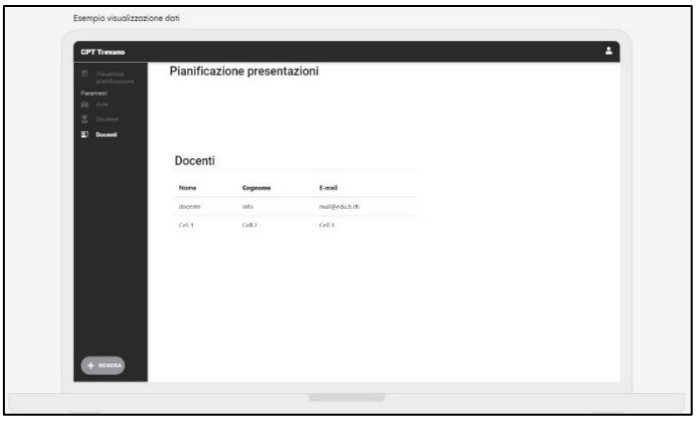


Figura 7 GUI - Pagina di esempio

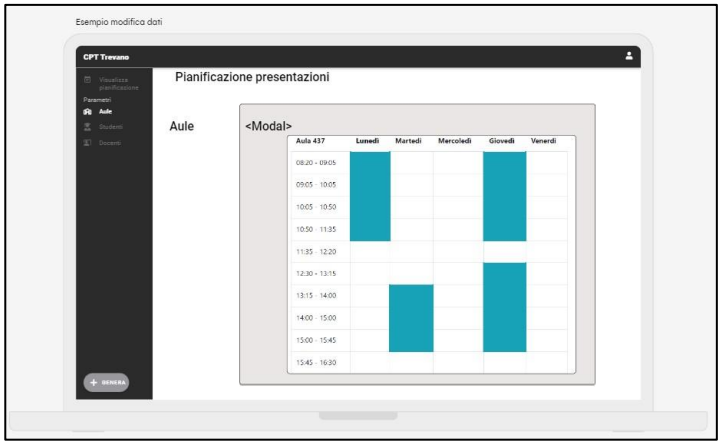


Figura 8 GUI - Modal di esempio

Quest'ultima GUI mostra un esempio di tabella di programmazione delle ore. IN questo caso si rifà ad un'aula e è prevista in un *modal*, ma si può pensare anche come sempre fissa sulla pagina.

4.4 Design procedurale

Per capire meglio come avrebbe funzionato il mio sito ho creato due diversi diagrammi di flusso, uno per il docente ed uno per l'amministratore del sistema. In generale, ma soprattuto per questo progetto, ritengo che i diagrammi di flusso siano una parte chiave di qualunque progettazione. Per questo ho impiegato qualche momento in più per il pensare e sviluppare questi diagrammi, nel corso dello sviluppo ho pure fatto qualche cambiamento per adattarmi a problemi e modifiche sopraggiunti. Con questi spero anche di aver soddisfatto sia il punto specifico 114 che il 164. Infatti la gestione degli errori l'ho sviluppata, su base teorica, grazie a questi grafici.

Il primo che voglio presentare è quello più complesso, ovvero quello dell'amministratore.

4.4.1 Flusso di un amministratore

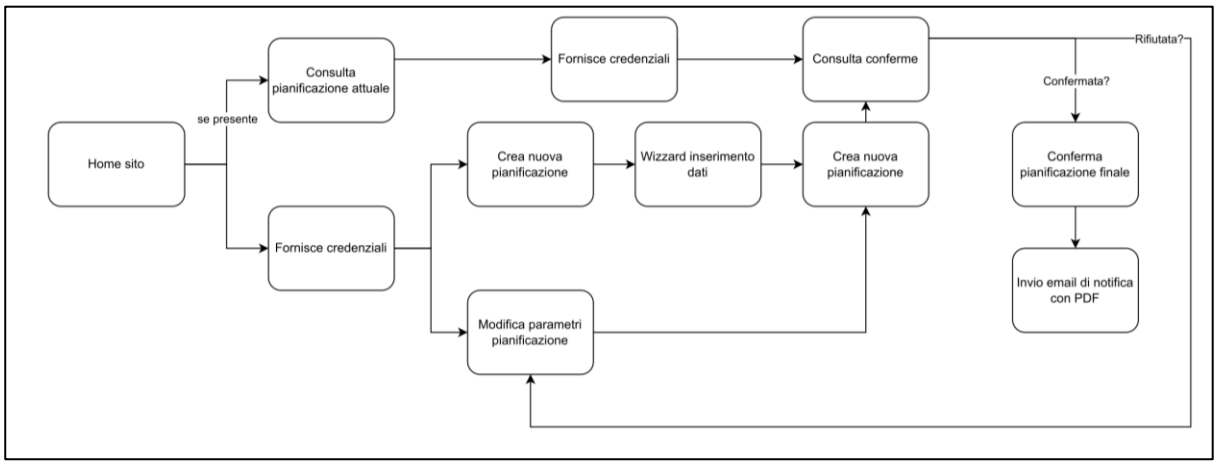


Figura 9 Diagramma di flusso - admin

Un **amministratore** può, una volta connesso al sito, gestire le presentazioni. Può semplicemente consultare la pianificazione del sito. Se si autentica può visualizzare le conferma dei docenti e, se lo ritiene opportuno, vidimare la conferma finale. Questo, almeno nella versione originale, avrebbe azionato il meccanismo per l'invio delle email di conferma. Se invece la pianificazione viene rifiutata l'amministratore può tornare alla fase di modifica, alterando i parametri forniti o agendo direttamente sulla pianificazione.
Inoltre un amministratore può da subito autenticarsi: ora può creare una nuova pianificazione inserendo i dati necessari, oppure può modificare i dati della pianificazione già creata.

4.4.2 Flusso di un docente

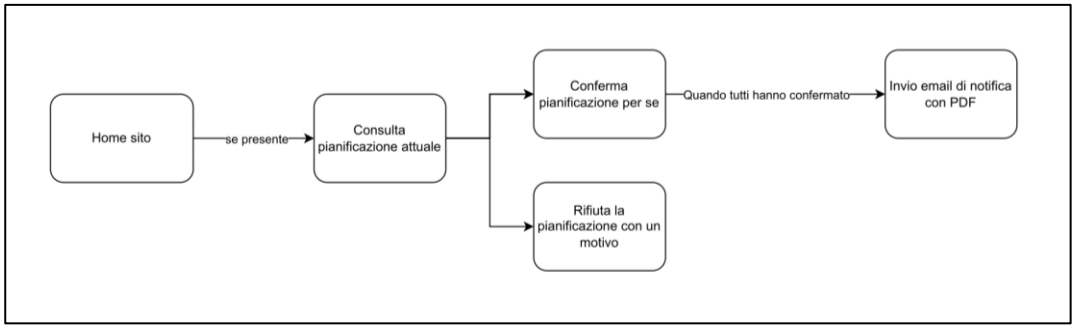


Figura 10 Diagramma di flusso - docente

Un **docente** può eseguire molte meno operazioni. Una volta connesso la sito può consultare la pianificazione corrente, se presente. Se è d'accordo la può approvare e, quando sarà stata approvata da tutti e vidimata dall'amministratore, sarebbe dovuta partire l'email di conferma. Quest'ultima azione viene eseguita dal sito, non dal docente. Se non è invece d'accordo con la pianificazione proposta la può rifiutare e può inviare il rifiuto con una nota che lo giustifichi.

5 Implementazione

Come detto per l'implementazione ho deciso di appoggiarmi su un template Bootstrap e di gestire il *frontend* con JavaScript. Dapprima documenterò tutto il *frontend*, poi il *backend*.

5.1 Database

In questo progetto per l'implementazione del database, mi sono affidato all'applicazione MySQLWorkBench. Una volta fatto lo schema non ho dovuto fare altro che andare sul menù "Database → Forward Engenieer" e creare così lo script in automatico. Ho spuntato, nel *wizzard*, le opzioni per la creazione delle istruzioni di drop prima di ogni *insert*.
Inoltre dallo script finale ho dovuto rimuovere tutte le definizioni degli indici e dei *constraint,* in quanto portavano ad errori di sintassi nella generazione.

5.2 Frontend

Per l'aspetto del sito, come già detto, ho deciso di utilizzare un template di Bootstrap. Ho scelto uno dei più semplici ma credo che per la mia applicazione calzi molto bene.

5.2.1 Introduzione

5.2.1.1 Bootsrap

Per questo progetto ho deciso, come detto, di utilizzare Bootstrap. Si tratta di un framework CSS molto potente e utilizzato in moltissimi siti che utilizzammo tutti i giorni. È così famoso da potersi considerare uno standard per lo sviluppo di interfacce Web, soprattutto quando si parla di pagine *responsive.* L'ho implementato tramite l'uso di file CSS scaricati nel template e anche, in alcuni casi, tramite CDN.
Il sito si compone di una pagina home e di una toolbar, che contiene invece i collegamenti alle pagine per l'apparato di gestione.

5.2.1.2 Ajax

Per il passaggio di dati da JS a PHP, ovvero dal *frontend* al *backend* ho deciso di utilizzare AJAX.
Grazie a questo insieme di tecnologie, tramite l'uso delle chiamate GET e POST, e tramite XML è possibile creare un pacchetto di dati da passare al codice sul server.

5.2.2 Script.js

Script.js è il file JavaScript che contiene tutte le mie funzioni di gestione per la parte *frontend* e per il passaggio di dati.

5.2.2.1 colorBtn

Questa semplice funzione mi permette, tramite il passaggio di un id, di colorare una specifica cella delle tabelle nelle pagine di gestione.
In queste pagine infatti è presente una tabella che permette all'utente di indicare al sito quando una persona o una classe sono disponibili.
Agendo sullo stile della pagina questa tabella colora quindi le celle, che contengono un elemento *span* che occupa tutto lo spazio, di colorarsi.
Inoltre il nome di tutti questi elementi viene impostato a "selected".

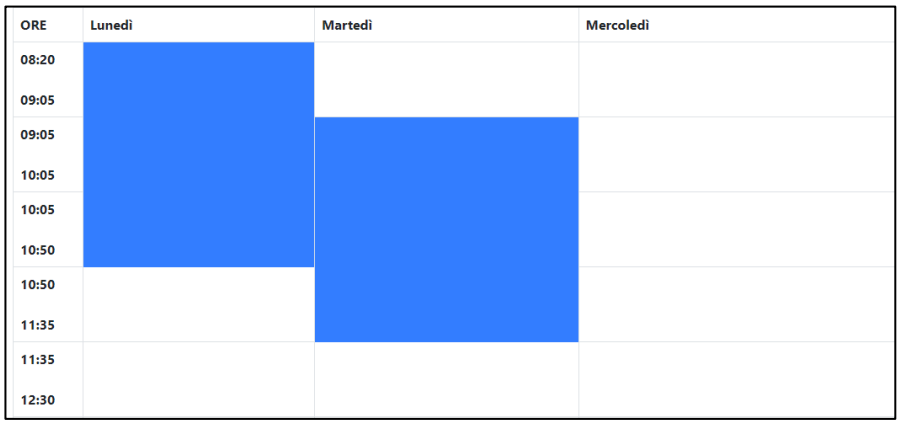


Figura 11 Tabella oraria

La logica degli id nella tabella è la seguente: "giorno.ora".
Il giorno è un numero da 1 a 5: per lunedì e 5 per venerdì. Mentre il secondo numero corrisponde alla fascia oraria scolastica, si va da 1 a 10.

5.2.2.2 wrapData

Questa funzione viene richiamata quando, in una delle pagine di gestione, viene completato il form e si desidera quindi salvare i dati. La funzione accetta un parametro, che specifica quale form si è appena riempito. Questo parametro viene letto e viene eseguito il blocco di codice giusto.
All'interno del suddetto blocco di codice vengono letti i dati dai campi della pagina, se sono degli input *select* viene utilizzata la funzione *getValueFromDropD*, viene letta la disponibilità tramite la funzione getSelectedHours, e poi il tutto viene unito in un array e mandato al metodo *sendData*.
Il codice sottostante ne è un esempio.

```
[...]
if ($el == "general") {

    var year = document.getElementById("year").value;
    [...]
    var startDate = date[0];
    var endDate = date[1];

    var data = {
        year: year,
        [...]
        startDate: startDate,
        endDate: endDate,
    };
    sendData("GeneralController/addInfo", data);
}
```

5.2.2.3 getSelectedHours

Questo metodo mi permette, grazie al cambio di nomi precedenti, di identificare tutte le celle selezionate e prelevare tutti gli id degli elementi. Poi la funzione richiama *getTimeSpanFromId*.

5.2.2.4 getTimeSpanFromId

Questa funzione mi permette di trasformare un id in una stringa contenente l'orario di disponibilità. Per esempio, se viene selezionato l'id 1.1, viene generata un array di due elementi: il primo elemento contiene il giorno [1], mentre il secondo elemento contiene il periodo di tempo corrispondente ["08:20 - 09:05"].

5.2.2.5 getValueFromDropD

Questa semplice funzione permette di selezionare il valore dell'elemento selezionato delle *select*. È relativamente semplice e non necessita di particolari spiegazioni, degli input *select* parlerò meglio nella parte di view, in quanto sono particolari.

5.2.2.6 sendData

Questo è forse la funzione più importante di tutto il file, permette di impacchettare i dati e mandarli a PHP. La funzione riceve due parametri in entrata: il percorso del controller di destinazione e l'array di dati da inviare. Tramite XHR e una chiamata post viene modificato l'URL e inviata una richiesta POST al *backend* con i dati desiderati.

```
function sendData(controllerUrl, data) {
    var xhr = new XMLHttpRequest();
    xhr.open("POST", appUrl + controllerUrl, true);
    xhr.setRequestHeader('Content-Type', 'application/json');
    xhr.onreadystatechange = function () {
        if (response.includes("<td>")) {
                localStorage.removeItem("table");
                localStorage.setItem("table", this.responseText)
            } else {
                location.reload();

            }
    };
    xhr.send(JSON.stringify(data));
}
```

La risposta con dei dati è gestita solo nel caso si richiami il controller per generare una pianificazione. In questo caso verrà ritornata una tabella HTML generata con PHP. Per questo ho messo un *if* che, nel caso in cui la risposta sia una tabella, inserisca il codice della tessa nella in una variabile che poi verrà stampata nella home.
Se invece non viene restituita una tabella la pagina viene ricaricata. Questo comportamento potrebbe essere utile in uno sviluppo futuro, in cui le tabelle con i dati degli studenti, delle classi, ecc. vengono generate da PHP e non sono *hard coded*.

Siccome ho voluto rendere questo codice una funzione, al posto di copiarlo/incollarlo 5 volte, ho dovuto inserire la scelta del controller dinamica. Tuttavia, come visibile nel codice, per raggiungere il controller devo specificare anche l'URL dell'app. Per farlo ho creato una costante:

```
const appUrl = "http://localhost:8080/PianificazionePresentazioniProgetti/"
```

A questa costante ogni volta viene aggiunto il percorso del controller e del metodo dello stesso che si desidera eseguire, ad esempio:

```
GeneralController/addInfo
```

Questo permette di avere una funzione per l'invio di dati dinamica ed efficiente.

5.3 Backend

Come già accennato per il *backend* ho deciso di adottare PHP e un pattern di sviluppo MVC.

5.3.1 View

Ho deciso, come già accennato, di suddividere le mie *view* in più sezioni. Questo perché essendo delle *view* si tratta, essenzialmente, di pagine HTML. Dato che, per motivi logici e di continuità, tutte le pagine avranno lo stesso *header* e lo stesso *footer*, ho creato una cartella *templates* che contiene questi due elementi. L'*header* conterrà il titolo del sito e tutti i collegamenti al CSS e al JS della pagina. Il *footer* invece contiene anche esso delle chiamate a JS e poi le informazioni sul creatore e sul copyright. Tramite i controller, ogni volta che viene richiamata una pagina, vengono richiamati questi elementi fondamentali.
Trovo che questa logica di riutilizzabilità e parametrizzazione vada a soddisfare, in parte, anche nel punto specifico 147

5.3.1.1 Home

Questa è la view dove tutti i docenti avranno accesso. Qui è possibile consultare la pianificazione ed accettarla o rifiutarla.
Le conferme e i rifiuti verranno mostrati a schermo.
Qui è presente anche la barra laterale con il *dropdown* per accedere alla parte amministrativa.
La tabella con la pianificazione viene invece stampata con una semplice istruzione JS tramite *document.write*.

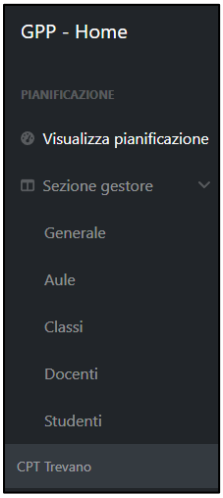


Figura 12 Side bar - home

Pianificazione presentazioni

10 entries per page

Search...

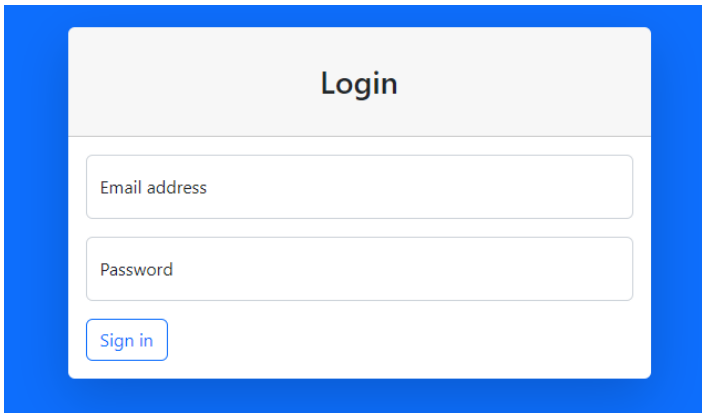
Data	Orario Inizio	Orario Fine	Aula	Allievo	Classe	Docente	Docente	Progetto
2023-06-09	12:30:00	13:15:00	426	Michea Colautti	I4AA	Guido Montalbetti	Geo Petrini	Gestione Presentazioni Progetti
2023-06-13	08:20:00	09:05:00	426	Nadia Fasani	I4AA	Ingrid Cered	Guido Montalbetti	Roba Con Firebase
2023-06-16	12:30:00	13:15:00	426	Julian Cummaudo	I4AA	Luca Muggiasca	Fabrizio Valsangiacomo	Rover lunaere
2023-06-09	09:05:00	10:05:00	426	Andrea Masciocchi	I4AA	Fabio Piccioni	Ingrid Cered	Gestione note
2023-06-12	09:05:00	10:05:00	426	Xavier Horisberg	I4AA	Massimo Sartori	Fabio Piccioni	Gestione tuttecose
2023-06-14	09:05:00	10:05:00	426	Nathan Ferrari	I4AA	Micheal Palucci	Fabio Piccioni	Caccia al Tesoro
2023-06-08	09:05:00	10:05:00	426	Erik Pelloni	I4AC	Fabrizio Valsangiacomo	Luca Muggiasca	Boh
2023-06-16	09:05:00	10:05:00	426	Nicola Anghileri	I4AC	Geo Petrini	Ingrid Cered	Boh parte2
2023-06-08	08:20:00	09:05:00	426	Pinco Pallino	I4AC	Fabio Piccioni	Guido Montalbetti	Scanner 3D
2023-06-12	13:15:00	14:00:00	426	Marco Marconi	I4AC	Ingrid Cered	Geo Petrini	MarconiSA

Showing 1 to 10 of 14 entries

1 2

Figura 13 Tabella pianificazione

5.3.1.2 LoginView



La view di login è una semplice pagina con una maschera che richiede l'inserimento di email e password.
Il codice dietro non è altro che un classico from PHP con una chiamata POST al controller e alla funzione atta al login. Qualcosa del tipo:

Figura 14 View Login

```
<form method="post" action="<?php echo URL ?>login/authenticate">
```

Tuttavia questa funzione non è completamente implementata ed è disabilitata per questa versione del progetto. Per farla funzionare dovrei aggiungere dei controlli al controller *openPages* con delle variabili di sessione. Quando l'utente fa il login, viene settato un valore che permette l'accesso alle pagine di gestione.

5.3.1.3 GeneralView

Questa è la prima *view* della sezione amministrativa: c'è un form che permette l'inserimento dei dati generali sulla pianificazione e sulle persone coinvolte.
Inoltre è presente un calendario che permette di decidere una data di inizio e una di fine.
Il pulsante sottostate non è un vero pulsante, ma un elemento *span* con lo stile Bootstrap di un pulsante.
Questo pulsante richiamerà la funzione JavaScript *wrapData* ed userà come parametro "general".

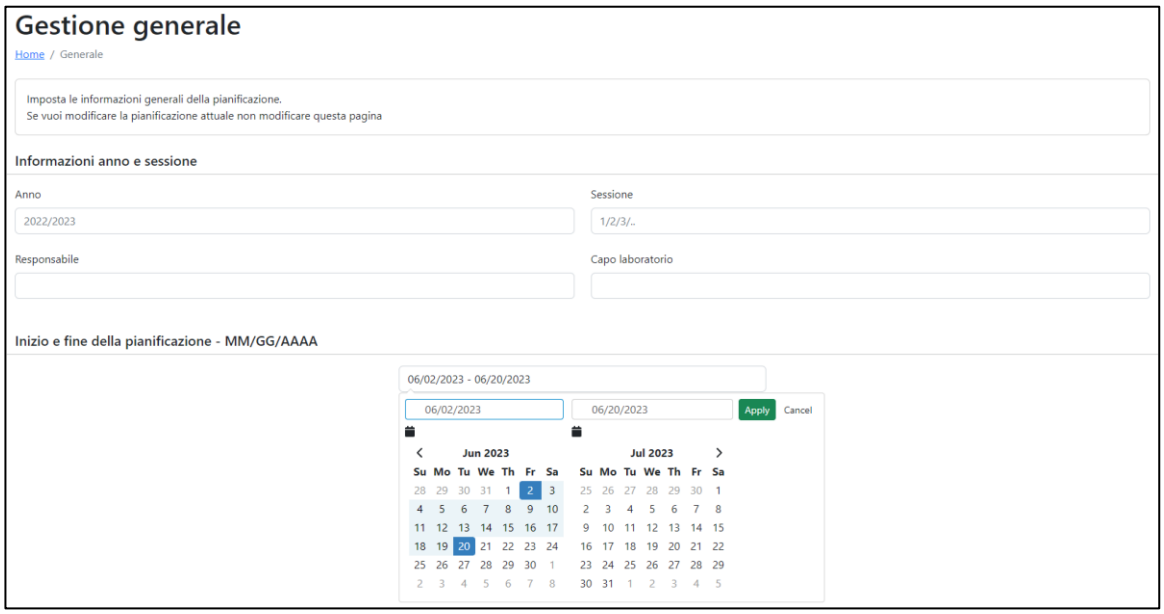


Figura 15 Form studenti

Degno di nota è il modo in cui ho disposto i campi nel form. In riferimento al punto specifico 193 ho deciso di organizzare specificatamente i campi in questo modo. Ovvero raggruppare a sezioni ben distinte in una stessa pagina i vari campi di inserimento. Nella pagina qui sopra per esempio ho messo i campi più "generici" e di testo nella prima sezione. Mentre nella seconda ho messo solamente il calendario, in modo da dargli l'importanza che merita.

Figura 16 Side bar - gestione

Un'altra particolarità di questa view, e di tutte quelle successive, è che nella *toolbar* è presente un pulsante per generare una nuova pianificazione.
Questo pulsante è quindi accessibile solo una volta che si è entrati nella sezione amministrativa. Per entravi, almeno teoricamente, è necessario dapprima eseguire un login.
Da questa toolbar si può anche vedere l'ordine cronologico in cui andrebbero consultate e/o riempite le pagine. In realtà non è così importante, l'unica pagina che è meglio venga utilizzata per ultima è quella degli studenti, dato che carica dei dati dalle altre pagine.

5.3.1.4 ClassroomView

Questa è la *view* atta a gestire le aule per le presentazioni. È relativamente semplice come pagina: contiene una tabella che, come spiegato nel capitolo riguardo alla funzione *colorBtn,* permette di dare una disponibilità dell'aula. È inoltre presente un form per fornire i dati necessari per l'aggiunta di un'aula: nome e se è munita di un segnale Wi-Fi potente.
Sotto il form è presente una tabella che dovrebbe, in futuro, accogliere la lista di aule. Cliccando su una di esse le informazioni vengono caricate nel form e sono modificabili.

5.3.1.5 SchoolClassView

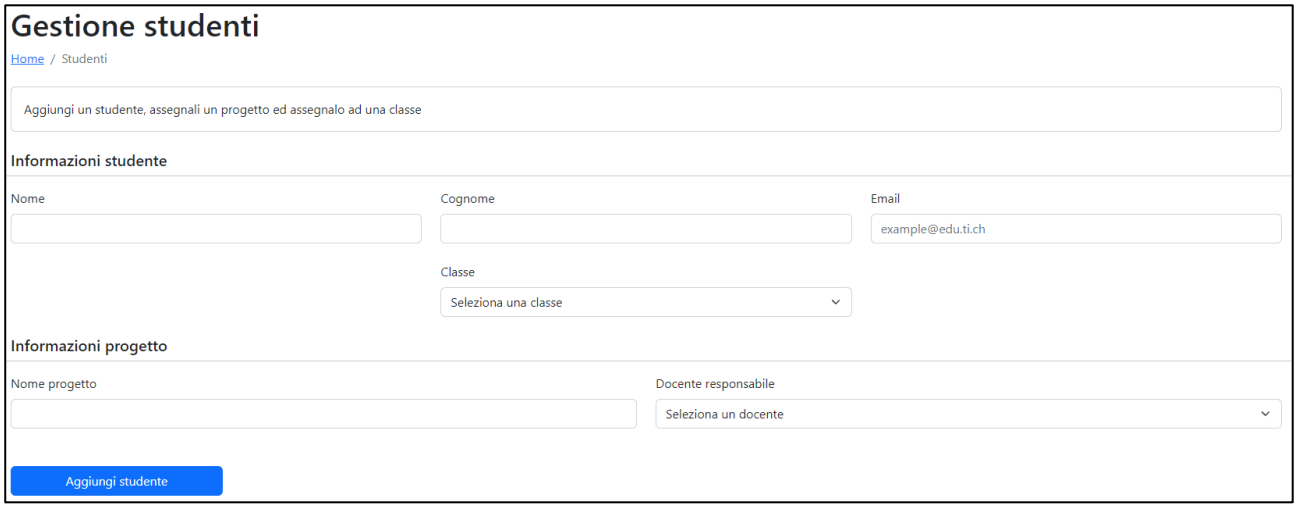
Questa seconda view permette invece di aggiungere delle classi. Anche per questo elemento è necessario impostare un nome ed una disponibilità, ancora tramite la tabella sopracitata. Si noti che nel mio codice ho chiamato l'entità classe, *SchoolCalass.* Questo perché, logicamente, il nome *Class* è riservato è non possibile assegnarlo come nome ad una classe di PHP.

5.3.1.6 TeacherView

Questa view è invece dedicata ai docenti. Qui, oltre ad esserci la solita la tabella per la disponibilità, è presente un form in cui è possibile inserire il nome, cognome e email del docente. Anche qui dovrebbe esserci una tabella con tutti i docenti inseriti.

5.3.1.7 StudentView

Infine la *view Student* è la più interessante. Al contrario di quello che il nome suggerisce in questa pagina è possibile aggiungere, oltre ai dati dello studente, i dati progetto.
Come accennato questa dovrebbe essere l'ultima ad essere compilata, questo perché al suo interno sono presenti due input *select* che permettono di scegliere la classe di cui fa parte lo studente e il docente responsabile del progetto. Questi dati sono presi dal database. La *view* si presenta in questo modo:



Sotto il form è presente, la tabella con tutti gli studenti. Al contrario delle altre pagine in questa non è invece presente la tabella con le disponibilità, dato che esse vengono associate all'allievo tramite la classe.

I due input select sono la parte più interessante della pagina.

```php
<?php
    require_once 'application/controller/StudentController.php';
    $studController = new StudentController();
    $select = $studController->populateSlectTeacher();
    echo $select;
?>
```

In questa porzione di codice viene richiamato il controller *Student* e viene dato l'ordine di popolare la *select*. Il risultato di questa funzione sarà il codice dell'input in questione, che verrà stampato.
In questo caso la select che viene popolata è quella relativa ai docenti, come si evince dalla chiamata a populateSlect**Teacher**.

Anche in questa view si può notare che ho diviso la pagina in sezioni e raggruppato i campi. Ho cercato di rendere il from quanto più chiaro possibile, sempre rispettando il punto specifico 193.

5.3.2 Controller

5.3.2.1 OpenPages

Questo controller mi permette di navigare nella struttura MVC. Invocando per esempio la funzione "openClassroom()" verrà aperta, tramite dei *require_once,* la pagina per la gestione delle classi. Anche questa centralizzazione delle chiamate può in un qualche modo rientrare sotto il cappello del punto specifico 147.

```
public function openClassroom(){
    require_once 'application/views/templates/header.php';
    require_once 'application/views/pages/ClassroomView.php';
    require_once 'application/views/templates/footer.php';
}
```

5.3.2.2 Login

Il controller di login permette, logicamente, di effettuare il login. Quello che fa è semplicemente prendere gli input dalla home tramite il form preposto, poi sanitizza i dati e li invia al modello di login. Come detto anche prima questa funzione non è completa, manca una gestione vera e propria dell'autenticazione tramite, ad esempio, delle variabili di sessione.

5.3.2.3 Home

Il controller home è il più semplice di tutti, permette solamente di caricare la pagina home, tramite la stessa tecnica usata nel controller *OpenPages*.

5.3.2.4 GeneralController

Questo è il controller per la pagina *GeneralView.* Ho cercato di scrivere i miei controller seguendo sempre la stessa struttura. Dapprima vengono presi i dati da AJAX:

```
[...]
$data = json_decode(file_get_contents('php://input'), true);
$data = array_values($data);

Poi vengono codificati i campi di testo e, come vedremo dopo, le ore di disponibilità:
$year = $model->filterField($data[0]);
$session = $model->filterField($data[1]);
$projManager = $model->filterField($data[2]);
[...]
```

Infine vengono invocati i metodi della model per il salvataggio dei dati e delle informazioni. In questo caso viene inserita anche la pianificazione nella tabella apposita.

```
//Require
require 'application/models/General.php';
$generalModel = new General();

$lastId = $generalModel->newInfo($year, $session, $projManager, $chief);
$generalModel->insertAvailability($lastId, $startDate, $endDate);
```

In questo estratto di codice si può vedere che il metodo per inserire i dati ritorna l'ultimo id inserito, che viene usato per inserire la disponibilità nel DB.
In questo caso l'id farà da FK per la relazione sessione/pianificazione.

5.3.2.5 ClassroomController

In questo controller vengono, come precedentemente, letti i dati da AJAX. In questo caso vengono prese anche le ore di disponibilità sotto forma di array:

```
$hours = array_values($data[2]);
```

Poi come prima vengono salvate i dati dell'aula e viene inserita la disponibilità:

```
$classroomModel->insertAvailability($classroomId, $startDate, $endDate, $hours);
```

Anche se il metodo è sempre *insertAvailability* si noti che viene eseguito partendo da una classe diversa, in questo caso il model *Classroom.*

5.3.2.6 SchoolClassController

Questo controller si occupa di gestire l'inserimento dei dati per una classe. Come per il controller precedente dapprima vengono inclusi i modelli *General* e *SchoolClass,* poi vengono letti i dati in arrivo da AJAX e vengono inseriti nel DB tramite i metodi appositi dei *model.*

5.3.2.7 TeacherController

Questo controller si occupa di gestire l'inserimento dei dati per un docente. È essenzialmente uguale ai controller precedenti, comprende le stesse tre fasi già citate.

5.3.2.8 StudentController

Questo controller è leggermente diverso da quelli già documentati. Benché si occupi di gestire l'inserimento dei dati per uno studente ha in aggiunta anche l'inserimento dei dati per un progetto. Per questo i dati presi sono di più, ma i metodi per l'inserimento dei dati riguardanti il

progetto si trovano comunque nel modello dello studente. Inoltre questo controller non comprende nessuna chiamata al metodo *insertAvailability,* in quanto è l'entità classe a gestire la disponibilità a presentare di uno studente.

```
//1step insert student and associate it to class and retrive student id
$studentId = $studentModel->newStudent($name, $lastname, $email, $studClass);

//2step insert project data
$studentModel->newProject($projName, $projTeach, $studentId);
```

In questo estratto di codice si può vedere che l'id dello studente appena inserito viene usato per inserire le informazioni relative al progetto.

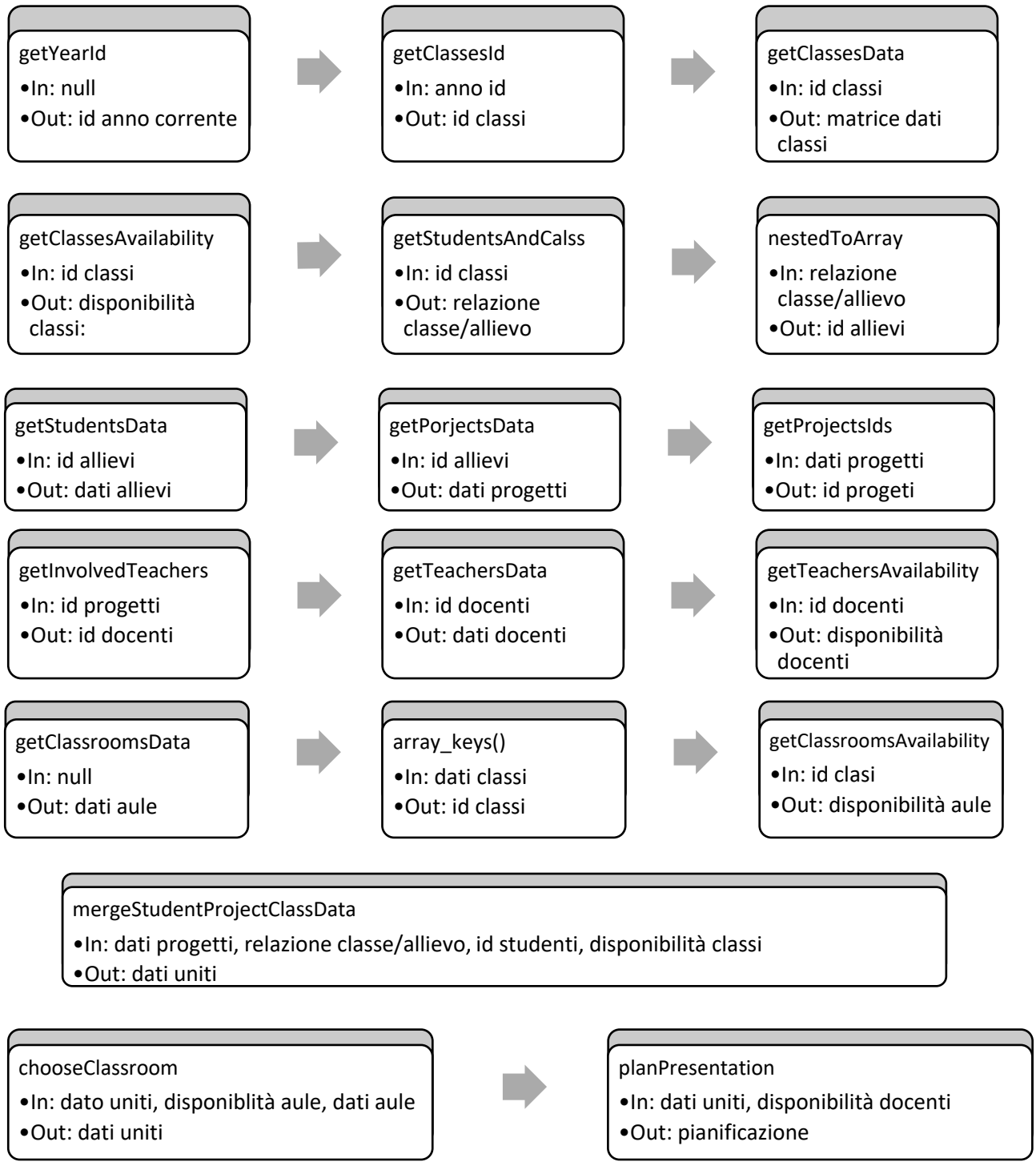
Tuttavia il controller ha altri due metodi molto importanti, per altro già citati, quelli dedicati alla gestione dell'input *select.*
Questi metodi, richiamati direttamente dalla *view,* permettono di invocare dei metodi nel modello *Model* che permettono a loro volta di fare delle *query* nel DB.
Il metodo, passando al metodo apposito come primo parametro i campi del DB che si desiderano e come secondo la tabella che si vuole interrogare, riceve i dati e li inserisce in una stringa che crea l'input model.

```
public function populateSlectTeacher()
{
    require_once 'application/models/Model.php';
    $model = new Model();
    $selectData = $model->getData("id,nome,cognome", "docente");
    $select = '<select id="projTeach" class="form-select" aria-label="Default select
      example"> <option selected>Seleziona un docente</option>';
      foreach ($selectData as $el) {
         $select .= '<option value="' . $el['id'] . '">' . $el['nome'] . " " .
            $el["cognome"] . '</option>';
      }
      $select .= '</select>';
    return $select;
}
```

Questa porzione di codice mostra il metodo per riempire la select dei docenti. Si noti che nell'input *select*, come valore, viene inserito un *id.* Questo è proprio l'*id* del docente nella tabella **docente.**
Questo passaggio è fondamentale per, in fase di inserimento dei dati nei modelli, avere il riferimento corretto al docente scelto.
Il metodo atto alla popolazione della *select* delle classi differisce di poco da quello presentato. Cambia leggermente la prima parte della dichiarazione della *select* e i parametri passati al metodo del modello.

5.3.2.9 PlanningController

Questo è infine il controller più lungo di tutto il progetto, ma non per questo il più complesso. Tutto quello che fa è invocare una lunga serie di metodi nel modello *Planning* e ottenere in risposta dei dati, che saranno utilizzati a loro volta per invocare altri metodi ed ottenere altri dati. Riassumo a punti quello che viene fatto dal controller.



toPrintableTable

- In: pianificazione, dati studenti, relazione classe/allievo, dati docenti, dati classi, dati aule
- Out: tabella di pianificazione

5.3.3 Model

5.3.3.1 Database

La classe rappresenta un modello per l'accesso e la gestione di un database ed estende PDO

La classe ha diverse variabili private, tra cui “host” per l'host del database, "user” per l'utente del database, “password” per la password del database, “dbname” per il nome del database e “port” per la porta del database. Inoltre, ha una variabile “connection” che sarà utilizzata per memorizzare l'istanza della connessione al database.

La classe ha un costruttore vuoto e un metodo chiamato “getConnection” che restituisce la connessione al database come oggetto PDO. Se la connessione è già stata stabilita precedentemente, viene restituita direttamente. Altrimenti, viene creata una nuova istanza PDO utilizzando le informazioni di configurazione del database fornite e vengono impostati gli attributi di gestione degli errori. Se si verifica un'eccezione durante la creazione della connessione, viene restituito un messaggio di errore. Ecco la stringa di connessione:

```
$this->connection = new PDO("mysql:host = " . $this->host . "; dbname=" .
    $this->dbname . "; port=" . $this->port, $this->user, $this->password );
```

Questa classe viene usata da tutti i modelli, o quasi, per l’accesso e per le operazioni sul mio DB.

5.3.3.2 Login

Il modello di login contiene tutte le funzioni per gestire l’accesso alla parte di admin.

5.3.3.3 Model

Il modello *Model* è il modello generico per tutto il sito. Qui sono contenute delle funzioni che più pagine utilizzano e che sarebbe stato ridondante e rischioso copiare/incollare in tutti i modelli separati. In questo modo se dovesse venire fatta una modifica ad un metodo generale, lo si deve modificare un'unica volta, senza preoccuparsi di dover riportare le modifiche in tutte le pagine.
Oltre al costruttore, che richiama e definisce un’istanza di *Database,* in questo file è presente un *dictonray* che contiene la traduzione da numero a giorno: 1 corrisponde a lunedì, 2 a martedì e così via fino a venerdì.

Altra funzione in questo file è quella per sanitizzare i dati: grazie a delle funzioni di PHP vengono fatte operazioni di pulizia come *trim, htmlspecialchars,* ecc.
C'è poi una combo molto importante di funzione, *getAvailability e getSpecifiedDayInRange*. Queste funzioni permettono la traduzione delle date di disponibilità segnate nella *view*.

Il primo metodo accetta in input un array con tutte le ore di disponibilità, le date di inizio e fine della programmazione e l'id dell'elemento. Poi, per ognuna di queste date, chiama il secondo metodo

```
public function getAvailability($hours, $startDate, $endDate, $id){
    $availability = array();
    foreach ($hours as $el) {
        $dates = $this->getSpecifiedDayInRange($startDate, $endDate, $el[0]);
        foreach ($dates as $date) {
            array_push($availability, $this->createArrayTime($id, $date, $el));
        }
    }
        return $availability;
```

Il secondo invece si occupa di trovare tutti, ad esempio, i lunedì nell'range di date fornito dall'utente. Come output verrà quindi prodotta una matrice che conterrà, per tutti i giorni nell'range di date specificato, le date e ore di inizio e fine della disponibilità. Logicamente la disponibilità marcata per il lunedì, ad esempio, sarà sempre uguale per tutto il range di date.

```
public function getSpecifiedDayInRange($dateFromString, $dateToString, $dayNumber){
    $dayName = $this->daysFromNumber[$dayNumber];
    $dateFrom = new DateTime($dateFromString);
    […]

    if ($dayNumber != $dateFrom->format('N')) {
        $dateFrom->modify('next ' . $dayName);
    }
    while ($dateFrom <= $dateTo) {
        $dates[] = $dateFrom->format('Y-m-d');
        $dateFrom->modify('+1 week');
    }
    return $dates;
}
```

Infine c'è un'ultima funzione molto importante, quella per gestire gli input *select*. Tramite una semplice *query* i dati vengono presi e restituiti al controller che gli ha richiesti.
Credo che questo model, assieme forse alla classe *Database* siano gli esempi più lampanti per soddisfare il punto specifico 147.

5.3.3.4 General

General è il primo dei modelli atto alla gestione della parte amministrativa. Come per i controller ho cercato di renderli il più possibile uno uguale all'altro, in modo da semplificare la gestione e la comprensione.
Ecco il modello nelle sue parti più importanti:

1. **__construct()**

Il primo metodo è il costruttore della classe, esso viene eseguito automaticamente quando viene istanziata un'istanza della classe. Al suo interno viene inclusa la classe *Database* e, con i metodi appositi viene stabilita la connessione al DB.

2. **newInfo**

Metodo "newInfo": questo metodo prende in input le variabili lette dal controller, in questo caso parliamo di anno, sessione, responsabile progetti e capo. Questi valori vengono inseriti nel DB. Per una maggiore sicurezza ho usato *i prepared statement* e PDO.

Dopo il primo inserimento ne viene fatto un secondo, questa volta la tabella interessata è quella della **sessione**. In questo modo posso ottenere l'id che poi userò come FK per la pianificazione.
Di seguito una delle mie *query* di esempio.

```
$sql = "INSERT INTO sessione(numeroProgetto,annoId,responsabile,capoLaboratorio) VALUES
(?,?,?,?)";
$query = $this->connection->prepare($sql);
$query->bindParam(1, $session, PDO::PARAM_STR);
$query->bindParam(2, $yeardId, PDO::PARAM_STR);
$query->bindParam(3, $projManager, PDO::PARAM_STR);
$query->bindParam(4, $chief, PDO::PARAM_STR);
$query->execute();

//getSessionID
return $this->connection->lastInsertId();
```

3. **insertAvailability**

Questo metodo prende in input l'ID della sessione, una data di inizio e una data di fine. Vengono effettuate alcune operazioni di formattazione delle date, in modo da poterle inserire nel DB e poi vengono inserite nello stesso.

```
$startDate = DateTime::createFromFormat('m/d/Y', $startDate);
$endDate = DateTime::createFromFormat('m/d/Y', $endDate);
$startDate = $startDate->format('Y-m-d');
$endDate = $endDate->format('Y-m-d');
[...]
$sql = "INSERT INTO pianificazione(versione,sessioneId,inizio,fine) VALUES (?,?,?,?)";
$query = $this->connection->prepare($sql);
$query->bindParam(1, $version, PDO::PARAM_STR);
[...]
$query->execute();
```

4. **getYearId**

Questo metodo esegue una *query* per selezionare l'ID dell'ultimo anno presente nella tabella **anno** e lo restituisce come risultato.

5.3.3.5 Student

Questo modello permette l'inserimento di dati relativi allo studente. La maggior parte del codice è molto simile al modello *General.* Cambiano solo i nomi dei parametri e le eventuali tabelle del DB.

1. **newStudent**

Questo metodo permette di aggiungere uno studente al DB. Prende in input nome, cognome, email dello studente e l'id della sua classe. Viene ritornato l'id dello studente appena inserito.

2. **newProject**:

Partendo dall'id dello studente ritornato vengono inserite le informazioni del progetto.

5.3.3.6 Teacher

Questo modello permette l'inserimento di dati relativi al docente. Contiene sia un metodo per i dati, simile a *newStudent*, sia un metodo per l'inserimento della disponibilità. La logica è la stessa usata nei due controller precedenti. L'inserimento dei dati del docente produce un id. Questo id viene usato nel metodo *insertAvailability* per inserire la disponibilità. Attenzione che il metodo *insertAvailability* in questione, pur essendo praticamente uguale a quello già documentato, differisce nella tabella su cui agisce. In questo caso i dati vengono inseriti in **orariprogettidocente.**

5.3.3.7 SchoolClass

Questo modello permette la gestione delle classi. Al suo interno contiene i metodi *newSchoolClass* e *insertAvailability*, che sono molto simili a quelli già documentati. Per evitare ridondanze non mi dilungherò sul loro funzionamento

5.3.3.8 Classroom

Questo modello permette la gestione delle aule. Al suo interno contiene i metodi *newcClassroom* e *insertAvailability*, che sono molto simili a quelli già documentati. Per evitare ridondanze non mi dilungherò sul loro funzionamento.

5.3.3.9 Planning

Questo codice è diviso in sezioni, una per ogni parte del progetto. Documenterò quindi sezione per sezione.

Sezione generale

1. **getYearId**

Questa funzione peremtte, tramite una query che selezione l'ultimo elemento del DB nella tabella anno, di ottenre l'id dell'anno corrente.

2. **getClassesId**

Questa funzione restituisce gli ID delle classi per l'anno specificato. Prende come parametro l'ID dell'anno ed esegue una query per recuperare gli ID delle classi dalla tabella **allievoClasseAnno**. Restituisce un array contenente gli ID delle classi trovate.

```
$sql = "SELECT DISTINCT classeId FROM allievoClasseAnno WHERE annoId=?";
```


```
$sql = "SELECT DISTINCT classeId FROM allievoClasseAnno WHERE annoId=?";
```


3. **getClassesData**

Questa funzione crea una matrice con i dati delle classi. Prende come parametro l'array di id delle classi restituito dalla funzione precedente e, per ogni ID, richiama la funzione *getThisClassData* per ottenere i dati della classe. I dati ottenuti vengono poi inseriti in una matrice che utilizza come chiave l'id della classe.

```
$classesData = array();
foreach ($classes as $classId) {
    $classesData[$classId] = $this->getThisClassData($classId);
}
return $classesData;
/**
 * Matrix of:
 * classId{
 *     name
 * };
*/
```

4. **getThisClassData**

Questa funzione recupera i dati di una classe dal database. Prende come parametro l'ID della classe e esegue una query per ottenere il nome della classe dalla tabella classe. I dati vengono memorizzati in un array e restituiti.

5. getClassesAvailability

Questa funzione crea una matrice con la disponibilità delle classi Prende come parametro l'array di id delle classi restituito dalla funzione *getClassesData* e, per ogni ID, richiama la funzione *getAvailability* (docuementata di seguito) per ottenere tenere la disponibilità della classe. La disponibilità viene inserita in una matrice con l'ID della classe come chiave, similmente a prima.

```
foreach ($classes as $classId) {
    $classAvailability[$classId] =
    $this->getAvailability("orariProgettiClasse", "classeId", $classId);
}
/**
 * Matrix of:
 * classId{
 *     Availability,
 *     Availability,
 */
return $classAvailability;
```

Si noti la chiamata a *getAvailability,* dopo tornerà utile.

Sezione studenti

1. getStudentsAndCalss

Questa funzione permette di ottenere la relazione tra allievi e la loro classe di appartenenza. Prende come parametro il solito array di ID delle classi e, per ogni ID, richiama la funzione *getStudentsFromClass* per ottenere la lsita di ID degli studenti appartenenti alla classe specificata. Questi dati vegono poi inseriti in una con l'ID della classe come chiave.

```
foreach ($classes as $classId) {
    $students[$classId] = $this->getStudentsFromClass($classId);
}
/**
 * Matrix of:
 * classId{
 *    studId,
 *    studId,
 *};
 */
return $students;
```

2. getStudentsFromClass

Questa funzione permette di ottenere gli studenti di una classe partedno dall'id della classe stessa Prende come parametro l'ID della classe e esegue una query per ottenere gli ID degli studenti dalla tabella **allievoClasseAnno**. Gli ID degli studenti vengono memorizzati in un array e restituiti.

```
$sql = "SELECT allievoId FROM allievoClasseAnno WHERE classeId=?";
[…]
foreach ($row as $el) {
    array_push($students, $el["allievoId"]);
}
```

3. getStudentsData

Questa funzione crea una matrice con i dati degli studenti. Prende come parametro l'array di ID di studenti restituito dalla funzione precedente e, per ogni ID, richiama la funzione *getThisStudentData* per ottenere i dati dello studente. Questi dati vengono poi inseriti in una matrice, che l'ID dello studente come chiave. Alla fine restituisce la matrice contenente i dati degli studenti. La logica usata è quella adottata fino ad ora

4. getThisStudentData

Questa funzione recupera i dati di uno studente dal database. Prende come parametro l'ID dello studente, passato dalla funzione precedente, ed esegue una *query* per ottenere il nome, il cognome e l'email dello studente dalla tabella allievo. I dati vengono memorizzati in un array e restituiti

Sezione progetti

1. getPorjectsData

Questa funzione crea una matrice con i dati dei progetti. Prende come parametro l'array di ID degli student citato in precedenz e, per ciasucuno di essi, richiama la funzione *getLastProjectOfStudent* per ottenere i dati dell'ultimo progetto sviluppato dallo studente, che sarà logicamente quello correntem, e li inserisce nella matrice con l'ID dello studente come chiave. Alla fine restituisce la matrice contenente i dati dei progetti.

```
foreach ($studentsId as $studId) {
    $studentsProject[$studId] = $this->getLastProjectOfStudent($studId);
}
/**
 * Matrix of:
 * Return array of these:
 * StudentId{
 *    projId,
 *    projName,
 *    projTeacher,
 * };
 */
return $studentsProject;
```

2. getLastProjectOfStudent

Questa funzione recupera i dati dell'ultimo progetto sviluppato da uno studente dal database. Prende come parametro l'ID dello studente e esegue una query per ottenere l'ID, il nome e l'ID del docente responsabile dell'ultimo progetto dalla tabella progetto. I dati vengono memorizzati in un array e restituiti.

```
foreach ($studentsId as $studId) {
    $studentsProject[$studId] = $this->getLastProjectOfStudent($studId);
}
/**
 * Matrix of:
 * Return array of these:
 * StudentId{
 *    projId,
 *    projName,
 *    projTeacher,
 * };
 */
return $studentsProject;
```

3. getProjectsIds

Questa funzione isola gli ID dei progetti dalla matrice restituita dalla funzione *getPorjectsData*. Prende come parametro la matrice in questione e, per ogni elemento, ne estrae l'ID e lo inserisce in un array. Alla fine viene restituito l'array contenente gli ID dei progetti.

```
foreach ($projetcs as $proj) {
    $ids[] = $proj[0];
}
```

Sezione docenti

1. getInvolvedTeachers

Questa funzione crea un array con gli ID dei docenti coinvolti nei progetti. Prende come parametro l'array di ID dei progetti citato precedentementem e, per ciascuono di essi, richiama la funzione *getTeacherId* per ottenere l'ID del docente responsabile del progetto. I dati vengono inseriti poi in un array di id dei docenti coinvolti.

2. getTeacherId

Questa funzione recupera l'ID del docente dal database. Prende come parametro l'ID del progetto dalla funzione precedente ed esegue una query per ottenere l'ID del docente dalla tabella progetto.

```
$sql = "SELECT docenteId FROM progetto WHERE id=?";
```

3. getTeachersData

Questa funzione crea una matrice con i dati dei docenti. Prende come parametro l'array di ID dei docenti generato dalla funzione *getInvolvedTeachers* e, per ciascuno di essi, chiama la funzione *getThisTeacherData* per ottenere i dati del docente. Questi dati vengono poi inseriti in una matrice con l'ID del docente come chiave.

```
foreach ($teacherIds as $teacherId) {
    $teacherData[$teacherId] = $this->getThisTeacherData($teacherId);
}
return $teacherData;
/**
 * Matrix of:
 * teacherId{
 *    name,
 *    lastname,
 *    email
 * };
 */
```

4. getThisTeacherData

Questa funzione recupera i dati di un docente dal database. Prende come parametro l'ID del docente e esegue una query per ottenere il nome, il cognome e l'email. I dati vengono memorizzati in un array e restituiti.

```
$sql = "SELECT nome,cognome,email FROM docente WHERE id=?";
[...]
foreach ($row as $el) {
    $teacher = array($el["nome"], $el["cognome"], $el["email"]);
}
//array with the teacher name,lastname and email
return $teacher;
```

5. getTeachersAvailability

Questa funzione crea una matrice con la disponibilità dei docenti. Prende come parametro il solito array di ID dei docenti e, per ciasucno di essi, chiama la funzione *getAvailability* per ottenere la disponibilità del docente. Alla fine restituisce la matrice contenente la disponibilità dei docenti.
Come si può notare anche in queste sezioni viene utilizzato uno schema più o meno simile.

Sezione aule

1. getClassroomsData

Questa funzione recupera i dati di tutte le aule dal database. Non prende alcun elemento come input. Il metodo esegue una query SQL per selezionare l'ID, il nome e lo stato della connessione WiFi di ogni aula.
Poi per ogni riusltato estrae nome e stato del wifi e li inserisce in un array temporaneo.
Questo array temporaneo viene poi inserito in una matrice che ha come chiave l'ID dell'aula.

```
$sql = "SELECT id,nome,WIFI FROM aula";
[...]
foreach ($row as $el) {
    $dum = array($el["nome"], $el["WIFI"]);
    $classrooms[$el["id"]] = $dum;
}
/**
 * Return array of these:
 * classroomId{
 *    classroomName,
 *    wifi,
 *};
 */
return $classrooms;
```

2. getClassroomsAvailability

Questa funzione crea una matrice con la disponibilità delle aule. È molto simiel alle funzioni precedenti riguardo la disponibilità. Prende come parametro l'array di ID delle aule appena generato e per ognuno di essi viene chiamata la funzione *getAvailability* per ottenere la disponibilità dell'aula dalla tabella specificata e la inserisce nella matrice con l'ID dell'aula come chiave.

Sezione metodi di supporto

Questa è una sezione un po' particolare, non si rifà ad un elemento nel mio progetto ma è una raccolta di metodi utili e che ho creato per accorciare il codice ed evitare i copia/incolla.

1. nestedToArray

Questa funzione consente di "appiattire" un array nidificato, ovvero un array che contiene altri array, anche detto *matrice*. Prende come parametro l'array da "appiattire". D'apprima viene dichiarato un array di risultato per accogliere gli elementi interni all'array; poi la funzione inizia.
Se il parametro passato non è un array, la funzione restituirà il valore booleano false.
Altrimenti, la funzione procede ad iterare sugli elementi dell'array. Se un elemento è a sua volta un array, la funzione richiama se stessa in modo ricorsivo passando come parametro l'array appena trovato. In questo modo anche questo array "appiattito". Se un elemento non è un array, viene aggiunto al risultato finale con la sua chiave corrispondente. Alla fine, la funzione restituisce l'array appiattito.

```
$result = array();
foreach ($array as $key => $value) {
    if (is_array($value)) {
        $result = array_merge($result, $this->nestedToArray($value));
    } else {
        $result[$key] = $value;
    }
}
return $result;
```

2. getAvailability

Questa funzione consente di estrarre la disponibilità da una tabella del database. In precendza l'ho citata più volte in più contesti quinid mi sembra opportuno docuemntarla bene.
La funzione prende come parametri:

- Il nome della tabella da interrogare
- Il nome della FK che si desidera
- Il valore della FK che si desidera.

La funzione esegue una query SQL per selezionare gli elementi "inizio" e "fine" dalla tabella specificata, usando un WHERE per selezionare solo gli elementi dove il valore della FK corrisponde a quello desiderato.

```
function getAvailability($tableName, $elementName, $elementId){

    $sql = "SELECT inizio,fine FROM " . $tableName . " WHERE " . $elementName . " = ?";
    $query = $this->connection->prepare($sql);
    $query->bindParam(1, $elementId, PDO::PARAM_STR);
    [...]
```

I valori restituiti vengono estratti dall'array risultante dalla query e inseriti in un array temporaneo. Questo array viene inserito poi in un array finle di risultato. L'approccio mi permette di avere una matrice che abbia tanti piccoli array conteneti l'ora di inizo e quella di fine.

```
foreach ($row as $el) {
    $dum = array($el["inizio"], $el["fine"]);
    array_push($availability, $dum);
}
return $availability;
```

Sezione pianificazione

1. mergeStudentProjectClassDat

Questa funzione unisce i dati del progetto, dello studente, e della classe con la relativa disponibilità in un unico array di dati. Prende in input:

- L'array di progetti generato in precedenza, di progetti.
- L'array con la relazione tra studenti e classi
- L'array di ID degli studenti
- L'array di disponibilità delle classi.

Restituisce una matrice contenente i dati combinati.
La funzione itera sui dati degli studenti e delle classi, controllando, con *in_array,* se un dato studente si trova in una classe.

```
foreach ($studentsAndClass as $key => $class) {
            foreach ($studentsIds as $stud) {
                if (in_array($stud, $class)) {
```

Per ogni risultato trovato, viene creato un set di dati temporaneo che include il progetto, la disponibilità della classe e l'ID dello studente. Questo array di dati viene quindi aggiunto a un array finale chiamato *mergedData.*

```
//IN foreach
$ava = $classAvailability[$key];
$proj = $projects[$stud];
$dum = array($proj, $ava, $stud);
array_push($mergedData, $dum);
//OUT foreach
[...]

 /**
 * Return matrix of
 * {
 *  projData
 *  availability
 *  studentId
 * }
 */
return $mergedData;
```

2. chooseClassroom

Questa funzione combina ulteriormente i dati provenienti da *mergeStudentProjectClassData* con le informazioni sulla disponibilità delle aule. Una volta unti la funzione restituisce un set di dati che include l'ID dell'aula selezionata. La funzione prende come parametri l'array *mergedData*, l'array delle disponibilità delle aule e l'array dei dati delle aule. Per ogni dato dell'array di dati uniti viene presa la disponibilità, che si trova alla posizione 0, poi con la funzione *getEqualElementse* vengono trovate le corrispondenze di disponibilità.
La funzione ritorna un array contente la disponibilità dell'aula e la chiave della stessa. I dati della disponibilità vanno a sostituire la disponibilità nell'array *mergedData.*
La chiave dell'aula scelta viene aggiunto all'array *mergedData*

```
foreach ($mergedData as $key => $data) {
    $dataAva = $data[1];
    $possibleDatas = $this->getEqualElements($dataAva, $classroomsAvailability);
    $mergedData[$key][1] = $possibleDatas[0];
    array_push($mergedData[$key], $possibleDatas[1]);
    }
return $mergedData;
/**
 * projData
 * avalability
 * student
 * roomId
 */
```

3. getEqualElements

Questa funzione cerca le corrispondenze tra la disponibilità di una classe e la disponibilità delle aule, restituendo un array contenente la disponibilità combinata e la chiave dell'aula corrispondente.
Si può dividere la funzione in due parti. Una prima parte di formattazione ed una seconda di ricerca. Nel corso della prima parte gli array vengono formattati. Infatti gli array di disponibilità sono composti da matrici a più livelli. Per permettere una comparazione i dati di inizio e fine della disponibilità vengono uniti in un unico elemento, in modo da appiattire l'array di un livello.
Una volta formattati viene presa la prima disponibilità possibile e viene ritornata.

4. planPresentation

Questa funzione è forse la più importante di tutte:

D'dapprima iene inizializzato un insieme di array vuoti per memorizzare i dati delle presentazioni pianificate:

- projects,
- timestamps
- supportTeacher
- students
- classroomsId

Poi viene creato un nuovo array chiamato *newTeachersAvailability*, per rielaborare i dati di disponibilità degli insegnanti. Questo passaggio comprime i dati di disponibilità degli insegnanti in un formato specifico, in maniera simile alla funzione getEqualElements.

```
$newTeachersAvailability = array();
foreach ($teachersAvailability as $key => $class) {
    foreach ($class as $el) {
        $compressedAva = $el[0] . " | " . $el[1];
        $newTeachersAvailability[$key][] = $compressedAva;
    }
 }
```

L'array risultante viene aggiunto *newTeachersAvailability*, mantenendo la chiave della classe come indice di array.

L'array delle disponibilità degli insegnanti, viene sovrascritto con il nuovo array che contiene i dati compressi.

Poi viene eseguito un *foreach* su ogni elemento all'interno di *mergedData.* E i dati vengono "trasportati" dall'array di dati uniti a gli array separati creati in precedenza.
L'unica particolarità avviene per la disponibilità. Infatti viene chiamata la funzione *mergeArrays* per combinare i dati di disponibilità nell'array *mergedData* con i dati di disponibilità degli insegnanti. Poi viene chiamata la funzione *chooseAvailability* per selezionare un orario di presentazione disponibile e un insegnante di supporto.

```
$projTeach = $data[0][2];
$timestampsAndTeacher = $this->mergeArrays($data[1], $teachersAvailability);
$timestampsAndTeacher = $this->chooseAvailability($timestampsAndTeacher,
$timestamps, $projTeach);
array_push($timestamps, $timestampsAndTeacher[0]);
array_push($supportTeacher, $timestampsAndTeacher[1]);
array_push($students, $data[2]);
```

Infine viene creato un nuovo array chiamato *planning*, che contiene tutti gli array appena riempiti. Alla fine si otterrà quindi qualcosa del tipo:

```
/**
 * Matrix of:
 * {
 *   projectData
 *   timeStamp
 *   supportTeacherId
 *   studentId
 *   classroomId
 * }
 */
return $planning;
```

5. chooseAvailability

Questa funzione seleziona un orario per la presentazione in base alla disponibilità degli insegnanti. Prende in input l'array contenente la disponibilità degli insegnanti, l'array contenente gli orari già scelti per le presentazioni e l'ID dell'insegnante associato al progetto. Ciò viene fatto per evitare di pianificare più presentazioni nello stesso momento.
Il docente di supporto viene scelto con un numero randomico. Questo numero rappresenterà l'id del docente scelto. Viene fatto un controllo in modo da non scegliere un docente di supporto uguale al docente responsabile del progetto.

6. toPrintableTable

Questa funzione prende in input la matrice *planning* e si occupa, con una lunga sequenza di cicli di prendere i dati da *planning* e ottenere i dati effettivi di tutti gli elementi.
Per esempio viene preso da planning l'id di uno studente e vengono presi, grazie ai dati presi precedentemente, i dati dello studente.

```
$classes = array();
    foreach ($studentId as $stud) {
    array_push($students, $studentsData[$stud]);
    $classId = $this->getClassOfStudent($stud, $studentsAndClass);
    array_push($classes, $classesData[$classId]);
}
```

In questo esempio non solo vengono presi i dati di uno studente ma viene presa anche la sua classe con la funzione *getClassOfStudent.*

7. getClassOfStudent

Questa funzione permette semplicemente di prendere la classe di uno studente. Accetta come parametro l'id di uno studente e restituisce il nome della classe.

8. mergedArrays

La funzione prende due array, *array1* e *array2*, e restituisce un nuovo array che contiene gli elementi comuni presenti in entrambi gli array.
Inizialmente viene inizializzato un array vuoto chiamato che conterrà gli elementi comuni dei due array.
Viene eseguito un ciclo su ogni elemento del secondo array, array2, utilizzando il suo indice come chiave. Al suo interno, viene inizializzato un altro array vuoto chiamato *mergedSubArray* che conterrà gli elementi comuni tra l'elemento corrente di array2 e $array1.

Viene poi fatto un altro ciclo *foreach*, stavolta l'array iterato è l'elemento corrente dell'array originale *array2*. Se l'elemento è presente anche in array1 viene aggiunto all'array *mergedSubArray*.

```
foreach ($subArray as $element) {
    if (in_array($element, $array1)) {
        $mergedSubArray[] = $element;
    }
}
```

6 Test

6.1 Test

Per i test in questo progetto ho adottato il classico approccio che si adotta nei nostri progetti: stilare almeno un test per ogni requisito e definire un protocollo ben definito. Questi test sono stati svolti sia alla fine che nel corso dello sviluppo.

6.1.1 Protocollo di test

I miei protocolli di test, con il riferimento al requisito che viene provato.

Test Case **Riferimento**	TC-001 REQ-001	**Nome**	Pagina autenticazione per generazione
Descrizione	La home page offre la possibilità di autenticarsi per generare pianificazioni		
Prerequisiti	-		
Procedura	1. Collegarsi al sito 2. Cliccare su: Sezione gestore → Generale 3. Autenticarsi con: • Username: root • Password: Password&1		
Risultati attesi	Si viene autenticati e viene mostrata la sezione per la gestione		

Test Case **Riferimento**	TC-002 REQ-002	**Nome**	Pagina pubblica di visualizzazione
Descrizione	La home page mostra la pianificazione		
Prerequisiti	Una pianificazione generata		
Procedura	1. Collegarsi al sito 2. Consultare la pianificazione		
Risultati attesi	Nella pagina home è presente la pianificazione		

Test Case **Riferimento**	TC-003 REQ-003	**Nome**	Inserimento dati pianificazione
Descrizione	Inserimento dei dati della pianificazione		
Prerequisiti	-		
Procedura	1. Collegarsi al sito 2. Cliccare su: Sezione gestore → Generale 3. Autenticarsi 4. Registrare dei dati 5. Consultare la tabella sottostante		
Risultati attesi	I dati dell'anno vengono correttamente inseriti e mostrati nella tabella		

Test Case **Riferimento**	TC-004 REQ-004	**Nome**	Inserimento dati docenti
Descrizione	Inserimento dei dati dei docenti		
Prerequisiti	-		
Procedura	1. Collegarsi al sito 2. Cliccare su: Sezione gestore → Docenti 3. Autenticarsi 4. Registrare dei dati 5. Consultare la tabella sottostante		
Risultati attesi	I dati dei docenti vengono correttamente inseriti e mostrati nella tabella		

Test Case **Riferimento**	TC-005 REQ-005	**Nome**	Inserimento dati classi
Descrizione	Inserimento dei dati delle classi		
Prerequisiti	-		
Procedura	1. Collegarsi al sito 2. Cliccare su: Sezione gestore → Classe 3. Autenticarsi 4. Registrare dei dati 5. Consultare la tabella sottostante		
Risultati attesi	I dati delle classi vengono correttamente inseriti e mostrati nella tabella		

Test Case **Riferimento**	TC-006 REQ-006	**Nome**	Inserimento dati aule
Descrizione	Inserimento dei dati delle aule		
Prerequisiti	-		
Procedura	1. Collegarsi al sito 2. Cliccare su: Sezione gestore → Aule 3. Autenticarsi 4. Registrare dei dati 5. Consultare la tabella sottostante		
Risultati attesi	I dati delle aule vengono correttamente inseriti e mostrati nella tabella		

Test Case **Riferimento**	TC-007 REQ-007	**Nome**	Inserimento dati allievi e progetti
Descrizione	Assegnazione di un allievo ad una classe e ad un progetto e inserimento dei dati dell'allievo e del suo progetto.		
Prerequisiti	Dati di docenti e classi inseriti		
Procedura	1. Collegarsi al sito 2. Cliccare su: Sezione gestore → Studenti 3. Autenticarsi 4. Registrare dei dati a. Selezionare una classe b. Selezionare un docente responsabile 5. Inserire un impedimento di un allievo		
Risultati attesi	I dati dei docenti e delle classi sono visibili e selezionabili dalla pagina I dati dello studente vengono correttamente inseriti e mostrati nella tabella.		

Test Case **Riferimento**	TC-008 REQ-008	**Nome**	Generazione tabella di pianificazione
Descrizione	Viene generata una tabella con la pianificazione per i progetti		
Prerequisiti	Tutti i dati di docenti, classi aule ed allievi salvati sul sito.		
Procedura	1. Collegarsi al sito 2. Cliccare su: Sezione gestore → Generale 3. Autenticarsi 4. Premere su "Nuova pianificazione" 5. Recarsi alla home page		
Risultati attesi	Viene visualizzata una nuova pianificazione, che tenga conto delle ore di progetti impostate e degli impedimenti degli studenti.		

Test Case **Riferimento**	TC-009 REQ-009	**Nome**	Meccanismo approvazione pianificazione
Descrizione	La pianificazione può essere approvata		
Prerequisiti	Una pianificazione generata		
Procedura	1. Collegarsi al sito 2. Recarsi alla home page 3. Approvare una pianificazione selezionando un docente		
Risultati attesi	La pianificazione viene approvata da quel docente. *Refreshando* la pagina è possibile vedere il docente elencato tra quelli che hanno dato la conferma.		

Test Case **Riferimento**	TC-010 REQ-010	**Nome**	Pianificazione vecchie consultabili
Descrizione	È possibile consultare le vecchie pianificazioni.		
Prerequisiti	Una pianificazione generata		
Procedura	1. Collegarsi al sito 2. Recarsi alla home page 3. Premere su invia conferma e autenticarsi 4. Selezionare una pianificazione da quelle presenti sulla pagina Home		

Test Case **Riferimento**	TC-011 REQ-011	**Nome**	Pagina modifica dati
Descrizione	È possibile modificare i dati inseriti		
Prerequisiti	Dei dati di docenti o aule o classi o studenti inseriti nel sito.		
Procedura	1. Collegarsi al sito 2. Cliccare su una pagina qualunque nella sezione gestore 3. Autenticarsi 4. Selezionare un dato dalla tabella in basso 5. Modificare qualche informazione 6. Salvare le modifiche		
Risultati attesi	Le modifiche vengono registrate e salvate sulla pagina		

Test Case **Riferimento**	TC-012 REQ-012	**Nome**	Logging
Descrizione	Sistema di log		
Prerequisiti	-		
Procedura	1. Collegarsi al sito 2. Cliccare su una pagina qualunque nella sezione gestore 3. Autenticarsi 4. Aggiungere qualche dato 5. Generare una pianificazione 6. Consultare il log		
Risultati attesi	Nel file di log sono presenti degli avvisi che documentino tutte le azioni svolte.		

6.1.2 Tabella riassuntiva dei risultati

Test Case	Esito	Data ultimo test	Risultato	Commenti
TC-001	Fallito	25.05.2023	Nessuno	Funzione non implementata
TC-002	Passato	25.05.2023	La pianificazione è presente	-
TC-003	Parzialmente passato	25.05.2023	Tabella non generata	-
TC-004	Parzialmente passato	25.05.2023	Tabella non generata	
TC-005	Parzialmente passato	25.05.2023	Tabella non generata	-
TC-006	Parzialmente passato	25.05.2023	Tabella non generata	-
TC-007	Parzialmente passato	25.05.2023	Tabella non generata	
TC-008	Passato	25.05.2023	La tabella viene generata	Pianificazione delle aule non perfetta. Ma tabella funzionante.
TC-009	Fallito	-	Nessuno	Funzione non implementata
TC-010	Fallito	-	Nessuno	Funzione non implementata
TC-011	Fallito	-	Nessuno	Funzione non implementata
TC-012	Parzialmente passato	25.05.2023	Nessun log di informazione	Funzione di log presente solo in fase di sviluppo

6.2 Mancanze/limitazioni conosciute

In questo progetto ci sono svariate mancanze. Oltre a non essere concluso le funzioni che lo sono hanno comunque dei bug e non sono perfette. Penso che la mancanza maggiore non sia tanto un elemento, quando l'incompletezza del progetto in generale.
Per esempio gli inserimenti dei dati, per quanto mi piacciano, non sono completi dato che manca una conferma di inserimento.
Il motore di pianificazione ha molti limiti e può essere che a volte si incarti. Oltre a questo il motore è, a parer mio, poco efficiente e il codice potrebbe essere ottimizzato.
Svilupperò l'argomento sulle mancanze ulteriormente nelle conclusioni; ma mi sento di aggiungere che un'altra mancanza, che può rientrane sotto il cappello del motore di generazione, è il mantenimento delle pianificazioni vecchie, che non ho avuto il tempo di sviluppare. Infine un'altra mancanza importante che ho identificato è il meccanismo di conferma da parte dei docenti, che non ho purtroppo completato.

7 Consuntivo

Per il Gantt consuntivo ho rimosso le attività non fatte, o quelle non più previste, e sistemato le tempistiche.

	WBS	Nome attività	Durata	Inizio	Fine	Predeces
1	**1**	**Pianificazione presentazione progetti**	**82.5 h**	**mar 02.05.23**	**ven 26.05.23**	
2	**1.1**	**Analisi**	**8 h**	**mar 02.05.23**	**mer 03.05.23**	
3	1.1.1	Gantt	3 h	mar 02.05.23	mar 02.05.23	
4	1.1.2	Analisi requisiti	2.5 h	mar 02.05.23	mer 03.05.23	3
5	1.1.3	Analisi dei mezzi e teconlogie	1.5 h	mer 03.05.23	mer 03.05.23	4
6	1.1.4	Informazione mezzi	1 h	mer 03.05.23	mer 03.05.23	5
7	**1.2**	**Progettazione**	**8.5 h**	**mer 03.05.23**	**ven 05.05.23**	
8	1.2.1	Schema di flusso	2 h	mer 03.05.23	gio 04.05.23	2
9	1.2.2	Schema ER	2.5 h	gio 04.05.23	gio 04.05.23	8
10	1.2.3	Use Case	1 h	gio 04.05.23	ven 05.05.23	9
11	1.2.4	Design interfacce	3 h	ven 05.05.23	ven 05.05.23	10
12	**1.3**	**Implementazione**	**57.5 h**	**ven 05.05.23**	**mer 24.05.23**	
13	1.3.1	Implementazione DB	2 h	ven 05.05.23	lun 08.05.23	7
14	1.3.2	Wizzard inserimento dati	24 h	lun 08.05.23	lun 15.05.23	13
15	1.3.3	Motore creazione pianificazione	32 h	lun 15.05.23	mer 24.05.23	14
16	**1.4**	**Test**	**2 h**	**ven 26.05.23**	**ven 26.05.23**	
17	1.4.1	Test	2 h	ven 26.05.23	ven 26.05.23	15
18	**1.5**	**Chiusura**	**2.5 h**	**ven 26.05.23**	**ven 26.05.23**	
19	1.5.1	Stampa, ritocchi doc, altro	2.5 h	ven 26.05.23	ven 26.05.23	16
20	1.6	Documentazione	22.5 h	mer 03.05.23	ven 26.05.23	

Figura 17 Gantt consuntivo - ore

	WBS	Nome attività	Durata	Inizio	Fine	Predeces
1	**1**	**Pianificazione presentazione progetti**	**82.5 h**	**mar 02.05.23**	**ven 26.05.23**	
2	**1.1**	**Analisi**	**8 h**	**mar 02.05.23**	**mer 03.05.23**	
3	1.1.1	Gantt	3 h	mar 02.05.23	mar 02.05.23	
4	1.1.2	Analisi requisiti	2.5 h	mar 02.05.23	mer 03.05.23	3
5	1.1.3	Analisi dei mezzi e teconlogie	1.5 h	mer 03.05.23	mer 03.05.23	4
6	1.1.4	Informazione mezzi	1 h	mer 03.05.23	mer 03.05.23	5
7	**1.2**	**Progettazione**	**8.5 h**	**mer 03.05.23**	**ven 05.05.23**	
8	1.2.1	Schema di flusso	2 h	mer 03.05.23	gio 04.05.23	2
9	1.2.2	Schema ER	2.5 h	gio 04.05.23	gio 04.05.23	8
10	1.2.3	Use Case	1 h	gio 04.05.23	ven 05.05.23	9
11	1.2.4	Design interfacce	3 h	ven 05.05.23	ven 05.05.23	10
12	**1.3**	**Implementazione**	**57.5 h**	**ven 05.05.23**	**mer 24.05.23**	
13	1.3.1	Implementazione DB	2 h	ven 05.05.23	lun 08.05.23	7
14	1.3.2	Wizzard inserimento dati	24 h	lun 08.05.23	lun 15.05.23	13
15	1.3.3	Motore creazione pianificazione	32 h	lun 15.05.23	mer 24.05.23	14
16	**1.4**	**Test**	**2 h**	**ven 26.05.23**	**ven 26.05.23**	
17	1.4.1	Test	2 h	ven 26.05.23	ven 26.05.23	15
18	**1.5**	**Chiusura**	**2.5 h**	**ven 26.05.23**	**ven 26.05.23**	
19	1.5.1	Stampa, ritocchi doc, altro	2.5 h	ven 26.05.23	ven 26.05.23	16
20	1.6	Documentazione	22.5 h	mer 03.05.23	ven 26.05.23	

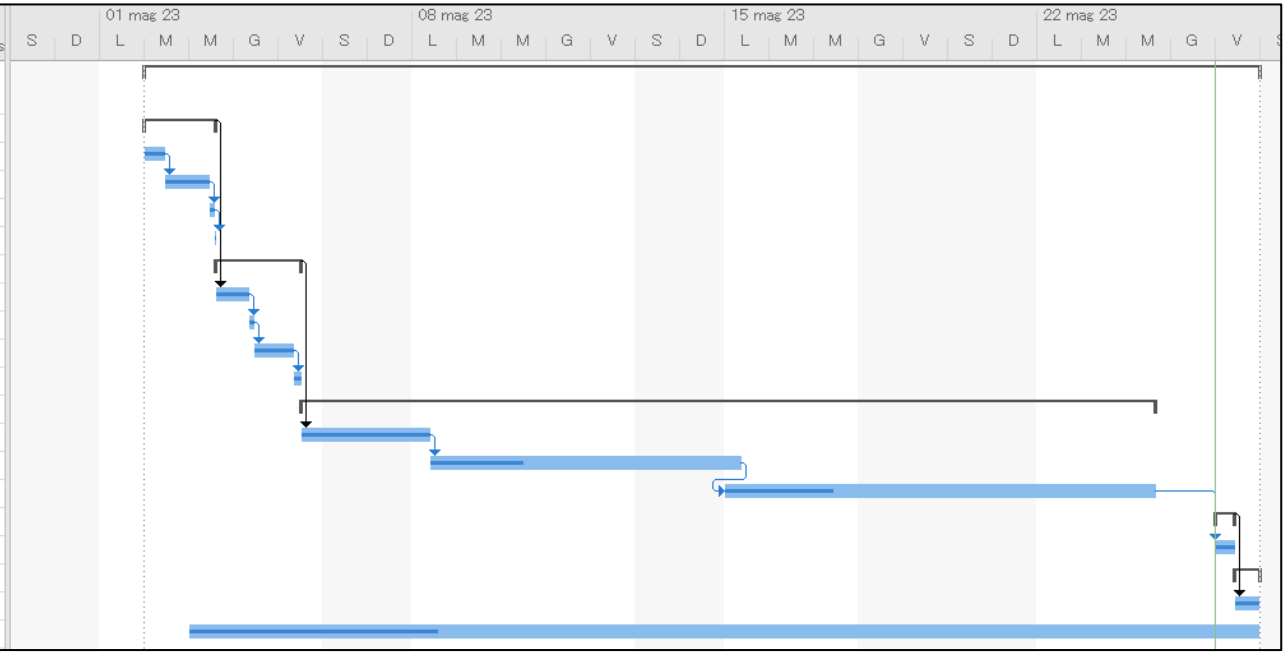


Figura 18 Gantt consuntivo - completo

8 Conclusioni

Credo che la mia soluzione non sia ancora pronta all'uso. Infatti le sue diverse mancanze non ne permettono un vero utilizzo in un vero caso d'uso. Mi dispiace non aver portato a termine il compito assegnatomi, dato che l'idea generale era intrigante: non mi è ami capitato infatti di lavorare su un motore di generazione di questo tipo, e mi sarebbe piaciuto portare più avanti il progetto.
Tuttavia penso di aver sviluppato una base decente per uno sviluppo futuro. Soprattutto la fase di inserimento dati è quasi completa, ed il codice che già c'è trovo sia un buon codice. Mi sono impegnato per renderlo comprensibile e riutilizzabile. Sempre per il lato tecnico devo dire che speravo di utilizzare un altro linguaggio ed affrontare un altro tipo di progetto per il mio LPI, ma di questo parlerò dopo. Credo che la scelta di utilizzare PHP senza framework sia stato un beneficio. Per gli scorsi due semestri ho utilizzato Laravel, ma non l'ho ne capito né sfruttato a pieno, sono sempre rimasto legato al PHP "convenzionale". Per questo ho deciso di sviluppare tutto in *plain PHP*, in quanto pensare e sviluppare un codice con questa modalità mi risulta più semplice.

Parlando invece del lato più generico del progetto: comunque sia andata sono contento di essere arrivato alla fine di questo LPI. Lavorare per più ore consecutive e avere una scadenza così stretta non è sempre stato semplice, ma non posso dire che non sia stato stimolante. Ho apprezzato anche il clima in classe e il supporto che non è certamente mancato da parte di molti docenti.

8.1 Sviluppi futuri

Il primo sviluppo futuro che vedo per questo progetto è senza dubbio il completamento del motore di generazione: come detto ora come ora non lo reputo utilizzabile, visto che ho scritto molto codice alla fine pur di raggiungere un obbiettivo decente non ho potuto curare il codice come avrei voluto. Inoltre, un altro aspetto che si potrebbe aggiungere al capitolo delle mancanze e delle limitazioni conosciute, è l'aspetto della sicurezza dei dati. Pur avendo imparato molte tecniche e *best practice* in questi anni alla SAMT non ho implementato praticamente nessun filtro per quanto riguarda gli input o l'autenticazione al sito, credo che anche questo sia figlio della mancanza di tempo per completare il progetto. Tuttavia, in questo contesto, credo che sia una "buona cosa" aver fatto passare la sicurezza dei dati in secondo piano. Ho valutato cosa fare quando mi sono accorto che non avrei finito, e parlando con il formatore abbiamo concluso che sarebbe stato meglio avere almeno qualcosa da mostrare del motore di generazione.
Oltre quindi al completamento del motore di generazione e la sicurezza dei dati sarebbe bello veder completato anche la funzione di inserimento dati. Per quanto funzioni infatti manca una vera e propria conferma, che in delle applicazioni che prevedono degli input credo siano fondamentali. Per questo un altro sviluppo di medio/grande importanza starebbe nell'implementare le tabelle dei dati dinamiche e che mostrino gli elementi inseriti sotto ai form.
Invece per una parte più di *backend* sarebbe opportuno implementare un sistema di logging. In fase di sviluppo ho fatto uso della classe error_log, e credo che potrebbe essere una buona soluzione.

8.2 Considerazioni personali

Alla fine di questo progetto posso dire di ritenermi mediamente soddisfatto di quanto ho sviluppato. Pur non avendo completato il progetto credo di non aver lavorato male in queste settimane. Rispetto ad altri progetti la parte di analisi e di progettazione è stata molto breve, ma credo comunque di aver sviluppato e prodotto tutto quello che serviva.
Basti pensare che solo sviluppando il DB ho trovato molti punti, non per forza problematici, che facendo l'analisi dei requisiti non avevo notato. Inoltre mi ha fatto rendere conto delle dimensioni del progetto. Inizialmente avevo subito pensato che sarebbe stato difficile completare il progetto, vedendo il DB e i requisiti espandersi questo sentimento mi è stato confermato. Credo che uno dei motivi per non aver finito il progetto stia proprio nella sua complessità e in quanto si è rivelato grande analizzando bene i requisiti.
Per esempio inizialmente non avevo pensato a cosa comportasse davvero il mantenimento delle presentazioni, poi ho realizzato che mi sarebbero servite più di 3-4 tabelle per implementarlo. Stessa cosa per gli orari di progetti, viste le molte entità (docenti, studenti, allievi e classi) ho dovuto fare molte tabelle ed implementare molte *view.*
Forse però parte del non completamento del progetto è da cercare nella modalità di lavoro e nel processo di sviluppo. Io ho ragionato in maniera "logica", sviluppando il progetto dall'inizio alla fine. Ho sviluppato prima le pagine di inserimento dei dati e dopo il motore. Guardando le priorità dei requisiti e l'obbiettivo del progetto forse avrei potuto iniziare da un'altra parte, ma non credo che il risultato sarebbe stato più completo di quello che è ora. Su questo punto mi sembra corretto dire ho vagliato la possibilità di inserire i dati per la generazione manualmente nel DB; tuttavia considerando le mie esperienze passate nei progetti ed in generale il modo in cui sviluppo ho pensato che fosse meglio, soprattutto per come affronto io i problemi, affrontare il problema dalla testa alla coda, e non bruciare tappe in partenza.
Rimango convinto che lo sviluppare prima le maschere di inserimento dati mi ha permesso di identificare problemi anche nel DB: ad esempio inizialmente non avevo previsto i campi di data inizio e data di fine per la pianificazione.
Come ho detto prima però sono comunque contento di essere arrivato alla fine di questo percorso e di aver imparato nuove nozioni su PHP, anche se le mie conoscenze derivano comunque dai due progetti precedenti.

9 Bibliografia

9.1 Sitografia

- https://cloudconvert.com/pdf-converter, cloudconvert.com
- https://datatables.net/, datatable.net
- https://getbootstrap.com/docs/5.3/helpers/position/, getbootstrap.com
- https://getbootstrap.com/docs/5.3/layout/breakpoints/#between-breakpoints, getbootstrap.com
- https://getbootstrap.com/docs/5.3/utilities/vertical-align/#sass, getbootstrap.com
- https://mockflow.com/, mockflow.com
- https://onlinepngtools.com/create-transparent-png, onlinepngtools.com
- https://stackoverflow.com/questions/10725967/run-a-php-function-when-click-on-a-button, stackoverflow.com
- https://stackoverflow.com/questions/12551166/getting-data-from-elements-from-a-php-response, stackoverflow.com
- https://stackoverflow.com/questions/15221371/best-method-of-including-views-within-views-in-codeigniter, stackoverflow.com
- https://stackoverflow.com/questions/15221371/best-method-of-including-views-within-views-in-codeigniter, stackoverflow.com
- https://stackoverflow.com/questions/19323010/execute-php-function-with-onclick, stackoverflow.com
- https://stackoverflow.com/questions/32778845/send-span-value-to-php-script-using-post, stackoverflow.com
- https://stackoverflow.com/questions/32778845/send-span-value-to-php-script-using-post, stackoverflow.com
- https://stackoverflow.com/questions/55249036/fatal-error-require-once-failed-opening-required-header-php, stackoverflow.com
- https://stackoverflow.com/questions/6782230/ajax-passing-data-to-php-script, stackoverflow.com
- https://www.html.it/guide/guida-bootstrap/, html.it
- https://www.php.net/manual/en/control-structures.foreach.php, php.net
- https://www.php.net/manual/en/function.explode.php, php.net
- https://www.w3schools.com/jsref/jsref_split.asp, w3schools.com
- https://www.w3schools.com/jsref/prop_style_border.asp, w3schools.com
- https://www.w3schools.com/php/func_array_intersect.asp, w3schools.com
- https://www.w3schools.com/php/php_file.asp, w3schools.com
- https://www.w3schools.com/php/php_file_create.asp, w3schools.com
- https://www.w3schools.com/php/php_forms.asp, w3schools.com
- https://www.w3schools.com/php/php_looping_foreach.asp, w3schools.com
- https://www.w3schools.com/php/php_mysql_insert_lastid.asp, w3schools.com
- https://www.w3schools.com/php/php_oop_constructor.asp, w3schools.com
- https://www.w3schools.com/xml/ajax_xmlfile.asp, w3schools.com

10 Glossario

Termine	Significato
MVC	Modello di sviluppo **M**odel **V**iew **C**ontrol
AJAX	Asynchronous JavaScript and XML

11 Indice delle figure

12 Allegati

- Diari di lavoro
- Mandato e QdC
- Prodotto
- Script SQL