Università degli Studi di Parma
Corso di Laurea in Informatica

Documentazione di progetto: UniCalendar

Esame di Ingegneria del Software Enrico Gabrielli

1. Introduzione

1.1. Panoramica dell’applicazione

L’applicazione sviluppata corrisponde al “Progetto 19 – Gestione del calendario esami”. La web application deve permettere al presidente del corso di studi, o al personale amministrativo, di visualizzare le date di appello scelte dai professori, preventivamente caricate nel database, e di ottimizzarne la calendarizzazione per mezzo di un algoritmo sviluppato ad hoc che rispetta le regole date dal corso di studi. Dovranno essere fornite diverse opzioni di ottimizzazione con uguale peso minimo tra cui scegliere e dovrà essere possibile anche modificarle manualmente.

2.Tecnologie utilizzate

In questa sezione vengono elencate e descritte le tecnologie utilizzate per lo sviluppo dell’applicazione, dall’ambiente di sviluppo ai linguaggi di programmazione utilizzati, dal database ai diversi plugin e librerie di cui ci si è serviti.

2.1. Visual Studio Code

Visual Studio Code (VSCode) è un editor di codice sorgente leggero e potente che supporta una vasta gamma di linguaggi di programmazione. Nel contesto di questo progetto, è stato utilizzato principalmente per lo sviluppo PHP e JavaScript e sono risultate particolarmente utili le seguenti estensioni:

- **GitGraph**: Consente di visualizzare la cronologia dei commit Git in modo grafico, facilitando il lavoro con il controllo versione.
- **Docker**: Permette di gestire facilmente contenitori Docker direttamente dall'editor.
- **PHP Intelephense**: Fornisce funzionalità avanzate di autocompletamento, navigazione del codice e diagnostica per PHP.
- **Auto Close Tag**: Una funzionalità che chiude automaticamente i tag HTML, migliorando l'efficienza nella scrittura del codice.
- **PHP IntelliSense**: Offre suggerimenti di completamento del codice e documentazione inline per migliorare la produttività durante la scrittura di codice PHP.

2.2. Backend

2.2.1.PHP

PHP è un linguaggio di scripting server-side ampiamente utilizzato per lo sviluppo web. È stato scelto per la gestione della logica di backend dell'applicazione. In questo progetto, PHP è utilizzato per gestire le operazioni CRUD relative agli appelli, agli utenti e ai corsi di laurea, interagendo con il database MySQL.

2.2.2. MySQL

MySQL è un sistema di gestione di database relazionali (RDBMS) che utilizza il linguaggio SQL per gestire i dati. È stato utilizzato per memorizzare le informazioni sugli appelli, sui corsi e sugli utenti. Nel progetto, la configurazione di MySQL è gestita tramite Docker, che crea e avvia il contenitore del database.

2.3. Frontend

2.3.1. HTML5

HTML5 è la versione più recente del linguaggio di markup per la strutturazione delle pagine web. In questo progetto, HTML5 è utilizzato per creare la struttura di base delle pagine web, inclusi moduli per l'inserimento degli appelli e la visualizzazione del calendario.

2.3.2. CSS3

CSS3 è la tecnologia per la stilizzazione delle pagine web, permettendo di definire colori, layout, tipografie e animazioni. È stato utilizzato per creare il design e la responsività delle pagine.

2.3.3. JavaScript

JavaScript è utilizzato per la gestione dinamica del comportamento delle pagine web, come la gestione degli eventi, l'interazione con il server tramite **API RESTful** e la manipolazione del DOM per la visualizzazione dei dati.

2.3.4. jQuery

jQuery è una libreria JavaScript che semplifica la manipolazione del DOM, la gestione degli eventi e le comunicazioni asincrone (**AJAX**). È stata utilizzata per interagire con le API del backend e per migliorare l'esperienza dell'utente.

2.3.5. Bootstrap

Bootstrap è un framework front-end open source che fornisce componenti e stili pronti per l'uso per costruire interfacce utente responsive e moderne. È stato utilizzato per migliorare la grafica e l'usabilità dell'applicazione, come le griglie responsivi e i componenti come pulsanti e form.

2.3.6. FullCalendar

FullCalendar è un plugin di **jQuery** utilizzato per la visualizzazione di calendari interattivi. È stato utilizzato per visualizzare gli appelli in un formato calendarizzato, consentendo agli utenti di visualizzare, aggiungere e modificare gli appelli in modo più chiaro e intuitivo.

2.3.7. Flatpickr

Flat è una libreria **JavaScript** per la selezione di date e orari, utilizzata per migliorare l'interazione con i campi di input data/ora nei form per la modifica delle date degli appelli.

2.4. Docker

Docker è una piattaforma di containerizzazione che permette di creare, distribuire e gestire applicazioni in contenitori leggeri e isolati. Nel progetto, Docker è utilizzato per creare i contenitori per il server **PHP**, il database **MySQL** e **phpMyAdmin**, facilitando l'ambiente di sviluppo e la distribuzione. Docker Compose è utilizzato per gestire più contenitori in modo semplice, definendo i servizi (come web-server-backend, php_docker) e le relative configurazioni (ad esempio, porte, variabili d'ambiente, volumi).

2.5. Apache

Apache è un web server open-source che gestisce la gestione delle richieste HTTP. In questo progetto, Apache è utilizzato per servire sia il frontend (HTML, CSS, JS) che le API RESTful, e per gestire la comunicazione tra il client e il server PHP.

3. Requisiti del sistema

3.1. Requisiti funzionali

- Gestione degli appelli:
  - Ogni insegnamento deve avere un numero minimo di date di appello.
  - Ogni appello deve avere almeno due possibili date (e orari).
  - Gli appelli scritti possono avere un appello orale correlato.
- Interfaccia di gestione:
  - Il presidente del corso di studi può selezionare tra più soluzioni proposte dal sistema e modificare manualmente il calendario.
- Autenticazione utenti:
  - Gli utenti (studenti, professori, presidenti di corso e amministrativi) devono autenticarsi tramite un sistema di login sicuro.

3.2. Requisiti non funzionali

- Regole di calendarizzazione:
  - Non è possibile calendarizzare due appelli dello stesso anno nello stesso giorno.
  - Il calendario ottimale deve massimizzare la distanza tra gli appelli.
- Gestione delle aule:
  - Il sistema non vincola la disponibilità delle aule.
- Sicurezza:
  - Il sistema deve garantire una gestione sicura dei dati degli utenti.
- Scalabilità:
  - Il sistema deve essere in grado di gestire facilmente un aumento del numero degli utenti e dei dati.
- Usabilità:
  - Le modalità di modifica delle date di appello devono essere chiare e intuitive.

4. Analisi dei requisiti

4.1. Requisiti funzionali dettagliati

- Appelli → Ogni insegnamento deve essere associato ad un numero minimo di appelli. I professori specificano almeno due date e orari per ogni appello scritto, e ogni appello scritto può essere associato a un appello orale.
- Ottimizzazione del calendario → Il sistema deve cercare di massimizzare la distanza tra gli appelli per evitare sovrapposizioni, tenendo conto delle preferenze dei professori e dei vincoli posti dal corso di studi, i quali avranno pesi diversi nel processo di scelta della calendarizzazione migliore.
- Ruolo del presidente del corso → Il presidente del corso, o un suo vicario, ha il potere di selezionare la soluzione ottimale per la calendarizzazione, di modificarla manualmente (se necessario) e di salvare l'ottimizzazione del database. Tutti gli altri utenti non devono disporre di questi poteri.

4.2. Requisiti non funzionali dettagliati

- Prestazioni → Il sistema deve garantire tempi di risposta rapidi per l'utente finale.

- Sicurezza → Le informazioni scambiate tra frontend e backend devono essere protette. Verranno, quindi, usati meccanismi di hash per la memorizzazione delle password e chiamate di tipo POST per le chiamate più critiche.
- Interfaccia utente → L'interfaccia deve essere user-friendly e responsive per essere utilizzata da diversi utenti e da diversi dispositivi.

5. Architettura del sistema

L'architettura del sistema è basata su un modello **Client-Server** con un'implementazione del pattern **MVP** (Model-View-Presenter) per separare la logica di business dalla presentazione. La comunicazione tra il client e il server avviene tramite **API RESTful**, gestite da un **Controller** che instrada le richieste verso i gateway appropriati.
Inoltre, l'applicazione è **dockerizzata**, il che consente di isolare l'ambiente di sviluppo e garantire una gestione semplice dei vari servizi come il backend PHP e il database MySQL.

5.1. Comunicazione Client-Server

L'architettura del sistema è costruita seguendo il modello **client-server**, dove il client (frontend) e il server (backend) sono separati e comunicano tramite richieste HTTP (A**PI RESTful**).

5.1.1. Client (frontend)

La parte client è responsabile della visualizzazione dei dati e dell'interazione con l'utente. In questo caso, è costruita utilizzando **HTML5**, **CSS3**, **Javascript**, **jQuery** e **Bootstrap**.
Il client invia richieste HTTP (via AJAX) al server per ottenere i dati necessari (ad esempio, gli appelli degli esami) e per inviare modifiche (ad esempio, aggiornare le date degli appelli).

5.1.2. Server (backend)

Il backend è costruito con **PHP** e gestisce la logica del sistema, le operazioni CRUD sul database (ad esempio, la gestione degli appelli) e la validazione delle azioni inviate dal client.

5.2. Architettura MVP (Model-View-Presenter)

L'architettura MVP è utilizzata per separare la logica di business (Model) dalla logica di presentazione (View), migliorando la manutenibilità del sistema.

5.2.1. Model

Rappresenta i dati e la logica del sistema. In questo caso, il Model è gestito da classi come DatabaseGateway e SessionGateway, che interagiscono con il database e gestiscono le operazioni CRUD.

Esempio (SessionGateway.php):

```
class SessionGateway extends Gateway {
    public function handle_request($parts){
        // echo $parts;
        switch(count($parts)) {

        case 1:
            if ($_SERVER["REQUEST_METHOD"] === "OPTIONS") {
                http_response_code(204);
```

```
            exit();
        } else if ($_SERVER["REQUEST_METHOD"] === "GET") {
            // Verifica se la variabile di sessione 'user' è settata
            if (!isset($_SESSION['user'])) {
                echo json_encode([
                    'logged_in' => false,
                    'user' => "Not set"
                ]);
            } else {
                // Se la sessione è attiva, restituisci un messaggio di successo
                echo json_encode([
                    'logged_in' => true,
                    'user' => $_SESSION['user']['id'],
                    'role' => $_SESSION['user']['ruolo']
                ]);
            }
            exit();
        }
        break;

    case 2:
        if($parts[1] == "login") {
            if ($_SERVER["REQUEST_METHOD"] === "OPTIONS") {
                http_response_code(204);
                exit();
                // /users/login
            } else if ($_SERVER["REQUEST_METHOD"] === "POST") {


                if (!isset($_POST['email'], $_POST['password'])) {
                    echo json_encode([
                        'success' => false,
                        'logged_in' => false,
                        'error' => 'Dati mancanti'
                    ]);
                    exit();
                }

                $email = $_POST['email'];
                $password = $_POST['password'];

                $db = DBConnectionFactory::getFactory();
                $sql = "SELECT * FROM utenti WHERE email = ?";

                $data = $db->fetchAll($sql, [$email]);

                if(!empty($data)) {
                    $user = $data[0];
                    if(password_verify($password, $user["password"])) {
                        //Password corretta, salvo l'utente nella sessione
                        unset($user['password']);
                        $_SESSION['user'] = $user;
```

```
                        echo json_encode([
                            'success' => true,
                            'logged_in' => true,
                            'user' => $_SESSION['user']
                        ]);
                        exit();
                    }
                }

                echo json_encode([
                    'success'=> false,
                    'logged_in' => false,
                    'error' => 'Utente non trovato'
                ]);
                exit();
            }
        } else if($parts[1] == "logout") {
            if ($_SERVER["REQUEST_METHOD"] === "OPTIONS") {
                http_response_code(204);
                exit();
            } else if ($_SERVER["REQUEST_METHOD"] === "GET") {
                session_destroy();
                echo json_encode([
                    'success' => true,
                    'logged_in' => false
                ]);
            }

        }
        break;

    }
}
}
```

5.2.2. View

La View è la parte dell'interfaccia utente che si occupa di mostrare i dati all'utente. Nel contesto di questa applicazione, la View è costituita dai file index.html e script.js, che visualizzano i dati (ad esempio, il calendario) e gestiscono l'interazione con l'utente.

Esempio (frammento di codice di index.html):

```
<div id="calendar-section" class="col-lg-12" style="display: none;">
      <div id="calendar">
      </div>
</div>
```

5.2.3. Presenter

Il Presenter si occupa di interagire con il Model e aggiornare la View in base ai date ricevuti. All'interno di questa applicazione, il Presenter è rappresentato dai file userView.js e userPresenter.js. Il primo si occupa della gestione della logica per la visualizzazione

all’interno della View in base ai dati ricevuti dal Model, mentre il secondo si occupa della comunicazione vera e propria con il Model e del recupero dei dati dal backend tramite chiamate API.

Esempio (frammento di codice di userView.js):

```
showLogin(role) {
        this.role = role
        this.loginSection.style.display = "block"
        this.navbar.style.display = "none"
        this.calendarSection.style.display = "none"

        this.calendarSection.classList.remove("col-lg-8")
        this.calendarSection.classList.add("col-lg-12")
        this.infoSection.style.display = "none"
        this.optimizeSection.style.display ="none"
}

showCalendar(role) {
        this.role = role
        this.loginSection.style.display = "none"
        this.navbar.style.display = "block"
        this.calendarSection.style.display = "block"
        if(this.role === "admin")
            this.optimizeSection.style.display = "block"
        else
            this.optimizeSection.style.display = "none"
}
```

Esempio (frammento di codice di userPresenter.js):

```
checkSession() {
        $.ajax({
            url: "http://localhost:8080/session",
            method: "GET",
            dataType: "json",
            xhrFields: {
                withCredentials: true
            },
            success: function (response) {
                console.log(response);
                if (response.logged_in) {
                    this.view.showCalendar(response.role)
                    this.initializeCalendar()
                    //console.log(events)
                } else {
                    this.view.showLogin(response.role)
                }
            }.bind(this),
            error: function (response) {
                console.log(response);
            }
        });
}
```

5.3. Comunicazione tra Client e Server (API RESTful)

La comunicazione tra il Client e il Server avviene tramite API RESTful. Ogni operazione CRUD (Create, Read, Update, Delete) è esposta tramite endpoint specifici nel server PHP. Nel file Controller.php, vediamo come le richieste vengono gestite e instradate verso il corretto gateway:

```php
class Controller
{
    private $api = "";
    public function set_api($api)
    {
        $this->api = $api;
    }
    public function handle_request()
    {

        $uri = preg_replace("/^" . preg_quote($this->api, "/") . "/", "",
$_SERVER['REQUEST_URI']);
        # [user, login]
        $uri = preg_replace('/\\/$/', "", $uri);

        $parts = explode("/", $uri);

        switch ($parts[0]) {
            case "session":
                $gateway = new SessionGateway();
                break;
            case "db":
                $gateway = new DatabaseGateway();
                break;
            default:
            http_response_code(404);
            echo json_encode(array(
                "success" => false,
                "error" => array(
                    "code" => 404,
                    "message" => "Not found"
                )
            ));
            exit();
        }
        try {
            $gateway->handle_request($parts);
        } catch (Exception $e) {
            var_dump($e);
            return;
        }
    }
}
```

In questo esempio:

- Il Controller gestisce le richieste HTTP, determinando quale gateway deve essere utilizzato per elaborare la richiesta.
- La logica di ogni gateway è delegata a specifiche classi che gestiscono le operazioni CRUD sul database o la gestione della sessione.

5.4. Dockerizzazione dell'ambiente

Un aspetto importante dell'architettura del sistema è la dockerizzazione. L'intero sistema, comprensivo del backend (PHP) e del database (MySQL), è contenuto all'interno di container Docker.
Il file **docker-compose.yaml** definisce i vari servizi necessari per l'esecuzione dell'applicazione:

- **web-server-backend**: Servizio che esegue il backend PHP e le API RESTful.
- **php_docker**: Servizio che esegue il database MySQL.
- **web-server-phpmyadmin**: Servizio che fornisce un'interfaccia web per la gestione del database tramite phpMyAdmin.

```yaml
services:
    web-server-backend:
        container_name: web-server-backend
        hostname: web-server-backend
        image: web-server-backend
        # image: php:apache
        build:
            context: .
            dockerfile: ./Dockerfile-backend
        volumes:
            - ./frontend/:/var/www/html/
            - ./backend/:/var/www/api/
        ports:
            - "80:80"
            - "8080:8080"
        environment:
            - MYSQL_HOST=php_docker
            - MYSQL_USER=${MYSQL_USER}
            - MYSQL_PASSWORD=${MYSQL_PASSWORD}
            - MYSQL_DATABASE=${MYSQL_DATABASE}
        depends_on:
            - php_docker

    php_docker:
        container_name: php_docker
        hostname: php_docker
        image: php_docker
        build:
            context: .
            dockerfile: ./Dockerfile-db
        environment:
            MYSQL_DATABASE: ${MYSQL_DATABASE}
            MYSQL_USER: ${MYSQL_USER}
            MYSQL_PASSWORD: ${MYSQL_PASSWORD}
            MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
            MYSQL_ALLOW_AMPTY_PASSWORD: "yes"
        ports:
```

```
            - ${MYSQL_PORT}:3306
        volumes:
            - ./db_data/:/docker-entrypoint-initdb.d/

    web-server-phpmyadmin:
        image: phpmyadmin/phpmyadmin
        ports:
            - "8001:80"
        environment:
            - PMA_HOST=php_docker
            - PMA_PORT=3306
```

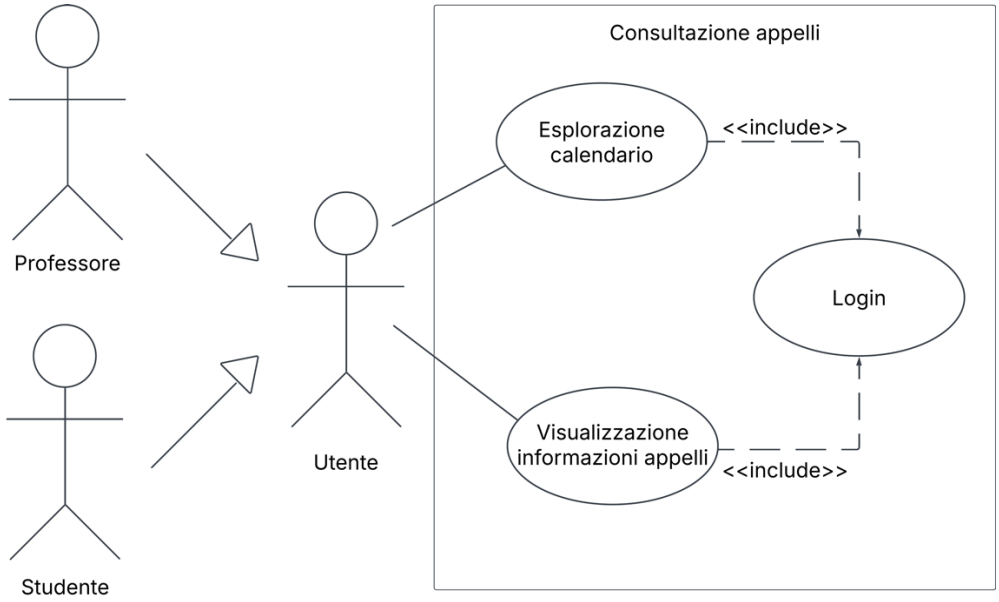
In questa configurazione, i servizi sono contenuti in contenitori separate, ma interagiscono tra loro tramite la rete Docker. La dockerizzazione permette di isolare i vari componenti, semplificando la gestione dell'ambiente di sviluppo e produzione.

6. Progettazione del sistema

6.1. Diagrammi dei casi d'uso

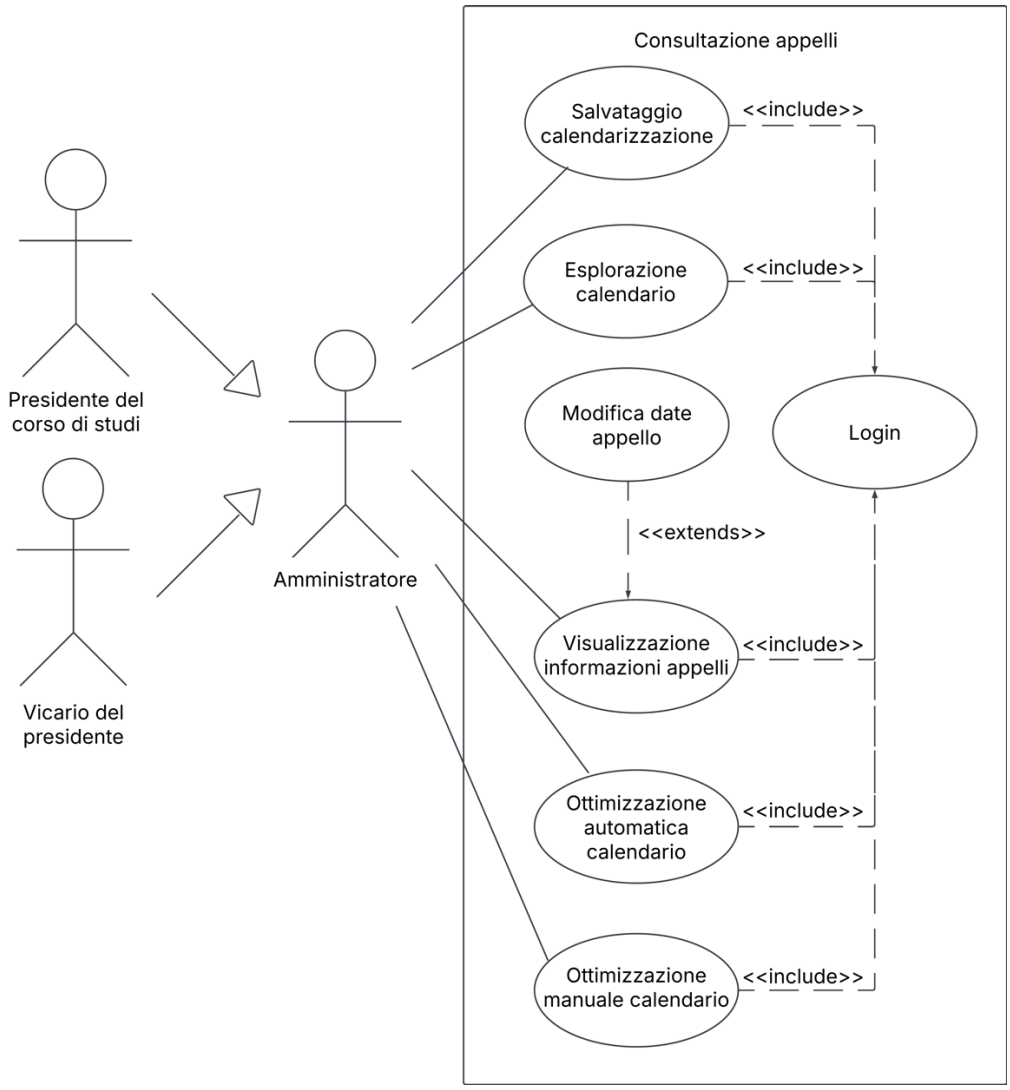
6.1.1. Consultazione appelli

Gli utenti (in particolare gli studenti o i professori) accedono al sistema con le proprie credenziali e possono consultare la calendarizzazione attuali degli appelli del corso di laurea.

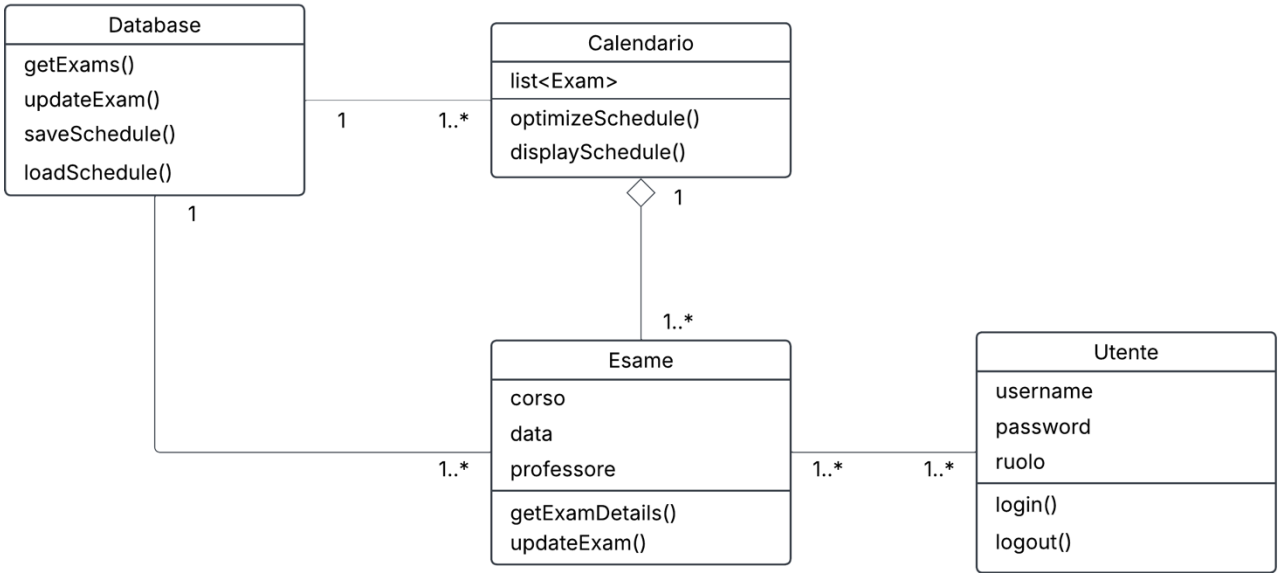


6.1.2. Calendarizzazione appelli

Il presidente del corso di studi, o un suo vicario, possono visualizzare, approvare e modificare la calendarizzazione degli appelli del corso di laurea.

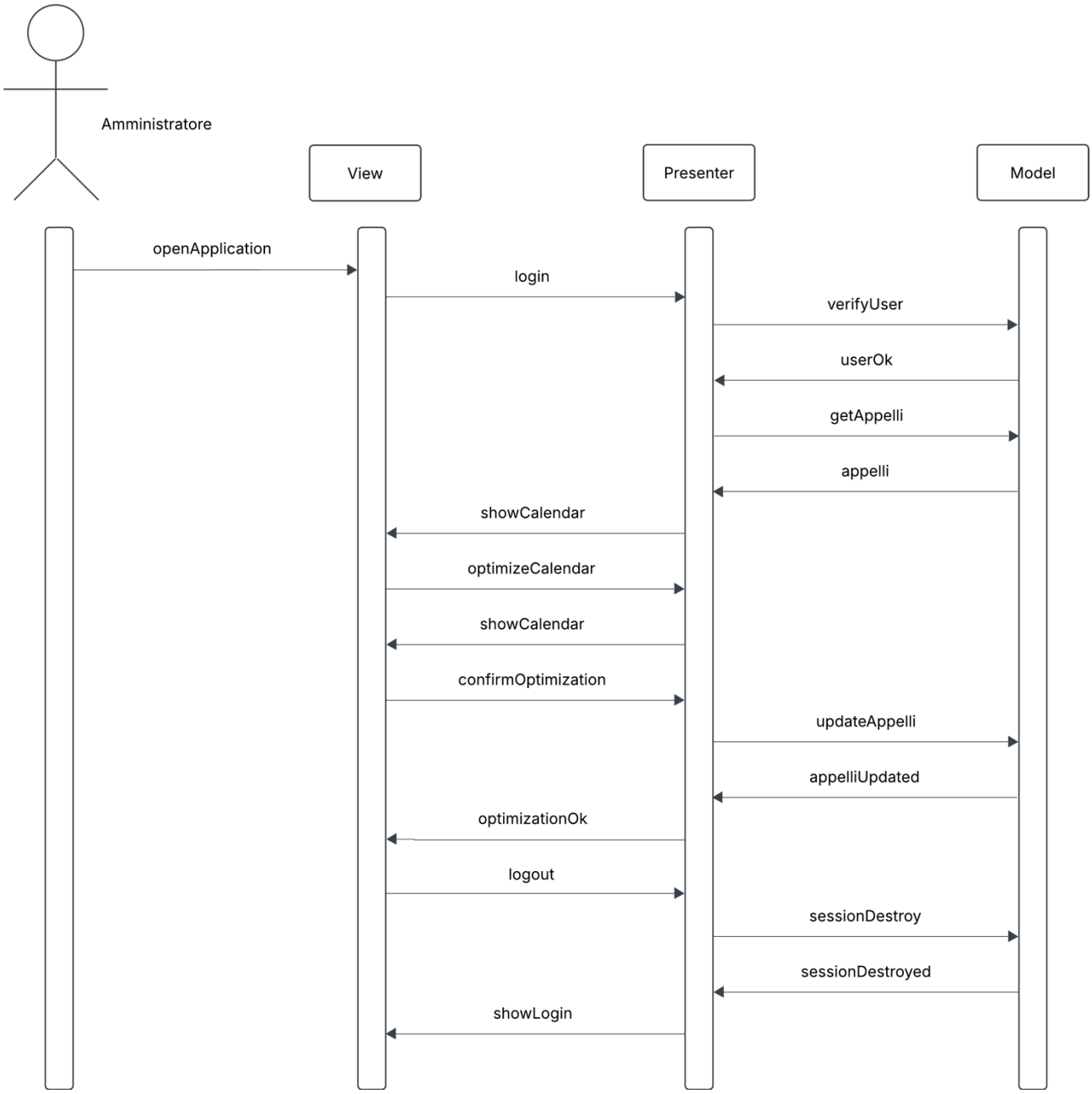


6.2. Diagramma delle classi



6.3. Diagramma di sequenza

Scenario: ottimizzazione e aggiornamento della calendarizzazione da parte dell'amministratore.



7. Dettagli implementativi

7.1. Frontend (HTML, CSS, JavaScript)

Per l'interfaccia utente, si è scelto di implementare una visualizzazione grafica che potesse risultare intuitiva per l'utente finale. A tale scopo, è stato utilizzato il plugin **FullCalendar** di jQuery, il quale permette di visualizzare una serie di eventi in diverse modalità, come quella mensile, settimanale o ad elenco. Tramite personalizzazioni grafiche e funzioni integrate di FullCalendar, gli eventi (in questo caso, gli appelli) verranno visualizzati con uno sfondo di colore verde, giallo o rosso, in base alle condizioni in cui l'evento si trova. Per esempio, il **rosso** rappresenta un appello calendarizzato con un altro

appello appartenente allo stesso anno di corso, il **giallo** rappresenta due appelli che si trovano in conflitto orario ma che non appartengono allo stesso anno di corso, e il **verde** rappresenta gli appello che non presentano problematiche. Selezionando un appello, poi, verrà visualizzata una schermata relativa ai **dettagli dell'evento**, come l'anno di corso, il professore responsabile dell'insegnamento, l'email del professore, l'orario di inizio e l'orario di fine. Se l'utente loggato all'interno dell'applicazione è un amministratore, sarà anche possibile modificare data, orario di inizio e orario di fine per mezzo della libreria **Flatpickr** di Javascript e confermare la modifica attraverso un apposito bottone. In ogni caso, le date degli appelli sono modificabili all'interno del calendario anche effettuando un "drag and drop" dell'evento da un giorno all'altro. In questo caso, la logica dell'applicazione provvederà ad aggiornare tutti gli appelli corrispondenti al giorno di partenza del drag e al giorno di arrivo del drop, in modo che siano rispettate le regole cromatiche per ogni appello. Per gli utenti amministratori, poi, saranno visualizzabili un bottone per l'avvio dell'**algoritmo di ottimizzazione**, un menu a tendina contenenti tutte le possibili ottimizzazioni con eguale **peso minimo** trovate dopo l'elaborazione e un bottone per il salvataggio della calendarizzazione all'interno del database (disabilitato finché non viene modificato almeno un appello all'interno del calendario.
L'algoritmo di ottimizzazione è stato creato sulla logica di base dell'algoritmo di Dijkstra; in particolare, si andranno a ricercare le combinazioni di appelli che generano un calendario con il peso totale minimo, con i singoli **pesi** calcolati con le seguenti regole:

- **8** → appelli dello **stesso anno di corso** calendarizzati nello **stesso giorno**
- **4** → appelli dello **stesso insegnamento** calendarizzati a **meno di 14 giorni di distanza** l'uno dall'altro
- **3** → appelli di **anni di corso diversi** calendarizzati lo **stesso giorno** e in **orari sovrapposti**
- **1** → appelli di **anni di corso diversi** calendarizzati lo **stesso giorno** ma in orari non sovrapposti.

File:

- **index.html**: La pagina principale dell'applicazione, che mostra la UI con il login, il calendario e i dettagli dell'appello. L'applicazione è stata sviluppata come SPA (Single Page Application), perciò non sono presenti altri file .html ma, al contrario, la visualizzazione della pagina index verrà aggiornata dinamicamente con il contenuto necessario.
- **script.js**: Crea le istanze di UserView e UserPresenter, inizializza i listener dei vari oggetti e gestisce il primo controllo di sessione.

```
import { UserView } from "./js/userView.js";
import { UserPresenter } from "./js/userPresenter.js";

$(document).ready(function () {
    const view = new UserView();
    const presenter = new UserPresenter(view);
    view.initializeEvents(presenter);

    presenter.checkSession();
})
```

- **userPresenter.js** e **userView.js:** Gestiscono la connessione con il backend, le operazioni relative all'ottimizzazione e l'aggiornamento della UI.

- **style.css**: Contiene le personalizzazioni grafiche inerenti alla UI.

7.2. Backend (PHP)

Per il backend (raggiungibile attraverso la **porta 8080** del container), il file **index.php** cede il controllo della richiesta a un oggetto di tipo Controller, che si occuperà dello smistamento verso un SessionGateway (nel caso la richiesta da gestire sia inerente a un controllo di sessione, a un login o ad un logout) o verso un DatabaseGateway (nel caso ci si debba interfacciare direttamente con il database, come per il popolamento del calendario a partire dagli appelli memorizzati, per la modifica delle date di un appello o per il salvataggio dell'ottimizzazione).
File:

- **Controller.php**: Gestisce la logica delle richieste API e la comunicazione tra il client e il database. Questa classe rappresenta l'implementazione del design pattern ***Front Controller***, in cui una singola classe o funzione si occupa di gestire tutte le richieste HTTP provenienti dall'utente e di instradarle verso il giusto handler. In più, insieme a index.php e il gateway designato, può essere vista come un'implementazione del pattern ***Chain of Responsibility,*** poiché il controllo delle diverse possibili richieste (session e db) è un esempio di meccanismo in cui la responsabilità della gestione della richiesta viene assegnata a diversi handler.

```php
class Controller
{
    private $api = "";
    public function set_api($api)
    {
        $this->api = $api;
    }
    public function handle_request()
    {

        $uri = preg_replace("/^" . preg_quote($this->api, "/") . "/", "", $_SERVER['REQUEST_URI']);
        $uri = preg_replace('/\\/$/', "", $uri);

        $parts = explode("/", $uri);

        switch ($parts[0]) {
            case "session":
                $gateway = new SessionGateway();
                break;
            case "db":
                $gateway = new DatabaseGateway();
                break;
            default:
            http_response_code(404);
            echo json_encode(array(
                "success" => false,
                "error" => array(
                    "code" => 404,
                    "message" => "Not found"
                )
            ));
            exit();
        }
```

```
        try {
            $gateway->handle_request($parts);
        } catch (Exception $e) {
            var_dump($e);
            return;
        }
    }
}
```

- **Gateway.php**: Definisce una classe astratta con le caratteristiche comuni a tutti i Gateway.
- **DatabaseGateway.php**: Interagisce con il database per eseguire operazioni CRUD su appelli, corsi e utenti.
- **SessionGateway.php**: Gestisce la sessione utente, garantendo la sicurezza e la persistenza delle credenziali.
- **DBConnectionFactory.php**: Crea la connessione al database MySQL. A questo scopo, è stato utilizzato un design pattern di tipo ***Singleton*** per garantire che solo una connessione al database venga creata e utilizzata. In più, sebbene questa classe non sia una Factory in senso stretto (ad esempio, non crea una varietà di oggetti complessi), l'uso di una funzione che incapsula la logica di creazione della connessione si avvicina a un'applicazione del pattern ***Factory***.

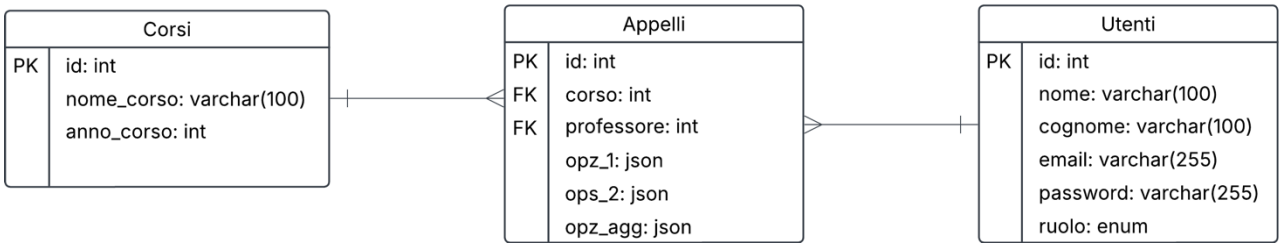
```
class DBConnectionFactory {

    private $host;
    private $username;
    private $password;
    private $dbname;
    private $connection;

    private static $factory;
    public static function getFactory() {
        if(!self::$factory) {
            self::$factory = new DBConnectionFactory();
            self::$factory->host = getenv("MYSQL_HOST");
            self::$factory->username = getenv("MYSQL_USER");
            self::$factory->password = getenv("MYSQL_PASSWORD");
            self::$factory->dbname = getenv("MYSQL_DATABASE");
            self::$factory->connection = null;
        }
        return self::$factory;
    }

    private function connect() {
        if($this->connection === null)
            $this->connection = new mysqli($this->host, $this->username, $this->password, $this->dbname);

        if($this->connection->connect_error) {
            die("Connection failed: " . $this->connection->connect_error);
        }

        return $this->connection;
```

```
    }

    public function close() {
        if($this->connection !== null) {
            $this->connection->close();
            $this->connection = null;
        }
    }

    public function query($sql, $params = []) {
        $this->connect();
        $stmt = $this->connection->prepare($sql);

        if($params) {
            $types = str_repeat('s', count($params));
            $stmt->bind_param($types, ...$params);
        }

        $stmt->execute();

        return $stmt;
    }

    public function update($sql, $params = []) {
        $stmt = $this->query($sql, $params);
        $affectedRows = $stmt->affected_rows;
        $stmt->close();
        return $affectedRows;
    }

    public function fetchAll($sql, $params = []) {
        $stmt = $this->query($sql, $params);
        $result = $stmt->get_result();
        $rows = $result->fetch_all(MYSQLI_ASSOC);
        $stmt->close();
        return $rows;
    }

}
```

7.3. Database

Per la popolazione iniziale del calendario, si è scelto di inserire gli appelli appartenenti alla **sessione invernale** appena conclusasi, garantendo che fossero presenti **almeno due appelli per insegnamento**. In più, per ogni appello, si è inserita una seconda data (accompagnata da orario di inizio e di fine), che rappresenta una ipotetica **seconda opzione** fornita dal professore corrispondente.
Il database è, quindi, composto da tre tabelle: **corsi**, **utenti** e **appelli**. La prima contiene gli attributi *id* (chiave primaria con auto increment), il *nome del corso* e l'*anno del corso*. La seconda contiene gli attributi *id* (chiave primaria con auto increment), *nome*, *cognome*, *email*, *password* (salvata in formato cifrato) e *ruolo* (che può essere admin, student o prof). L'ultima, invece, contiene gli attributi *id* (chiave primaria con auto increment), *corso* (chiave

esterna che si riferisce all'id di corsi), *professore* (chiave esterna che si riferisce all'id di utenti), *opz_1* (un json contenente la data, l'ora di inizio e l'ora di fine della prima opzione per l'appello), *opz_2* (un json contenente la data, l'ora di inzio e l'ora di fine della seconda opzione per l'appello) e *opz_agg* (un json contenente un attributo *agg* di tipo booleano che indica se per l'appello è presente una prova aggiuntiva, come può essere una prova orale, e un attributo *info* contenente le informazione sulla relativa prova aggiuntiva).



File:

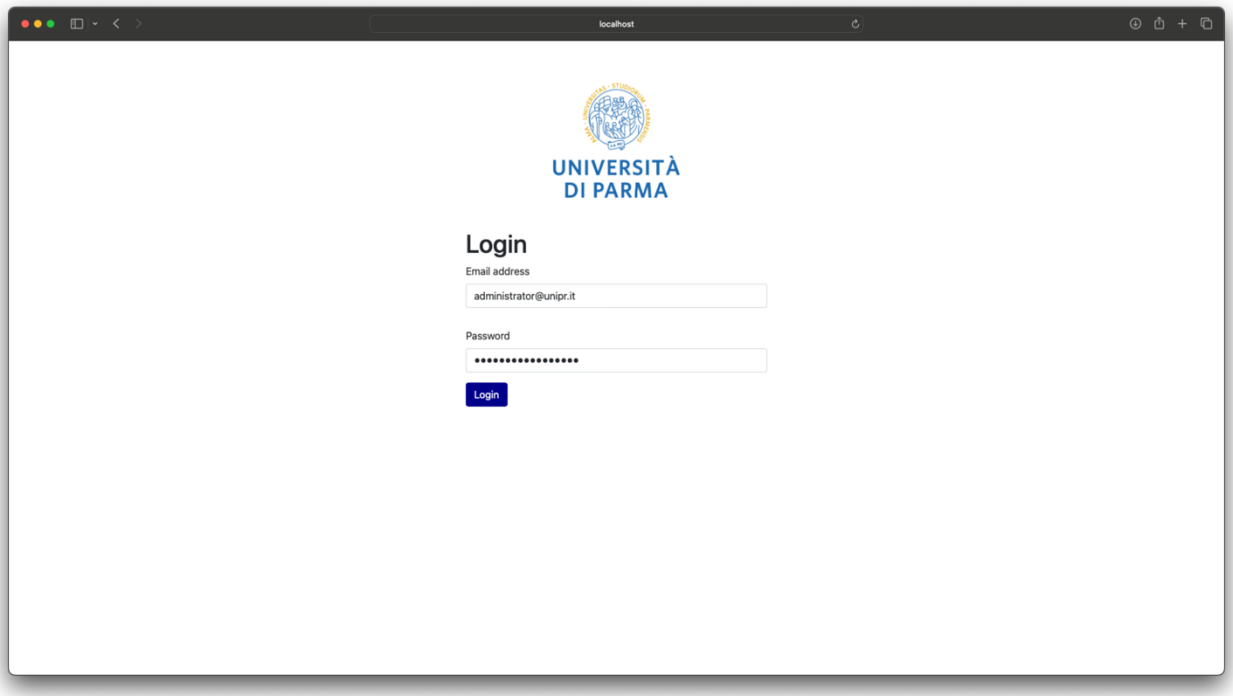
- **mycalendar.sql**: File di iniziazione che viene eseguito nel momento dell'avvio del container, permette di caricare nel database tutte le informazioni relative agli utenti, ai corsi e agli appelli.

8. Test e verifica

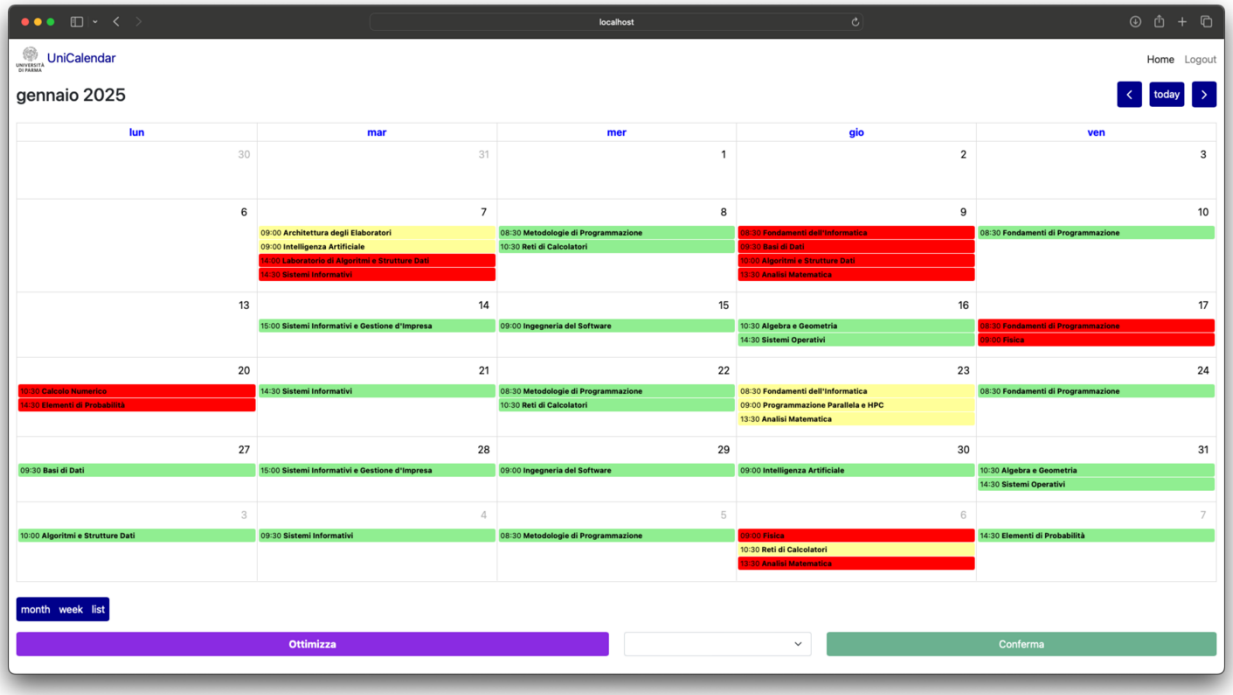
8.1. Test funzionali

8.1.1. Test di login e gestione sessioni

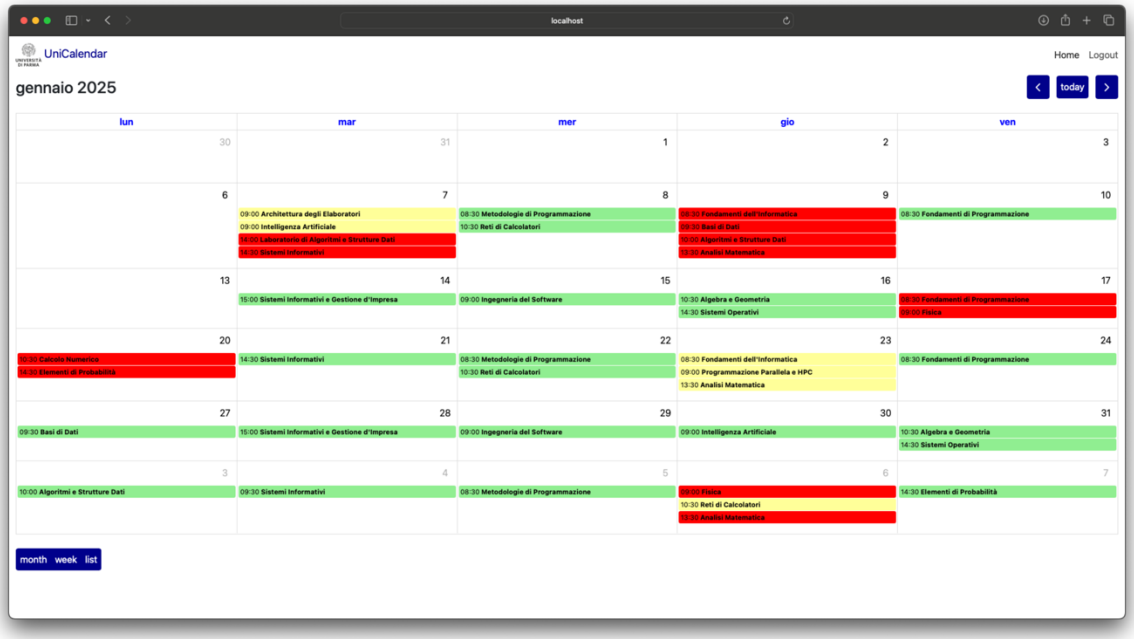
Per testare il corretto funzionamento della sezione di login e delle sessioni utente, si sono effettuati diversi test minori. Per prima cosa, si è provato ad effettuare l'accesso con ognuna delle tre tipologie di utente definite nel database, per controllare, in primo luogo, che l'autenticazione attraverso il database funzionasse correttamente, e poi per verificare che gli amministratori avessero i poteri aspettati.



Sezione di Login.

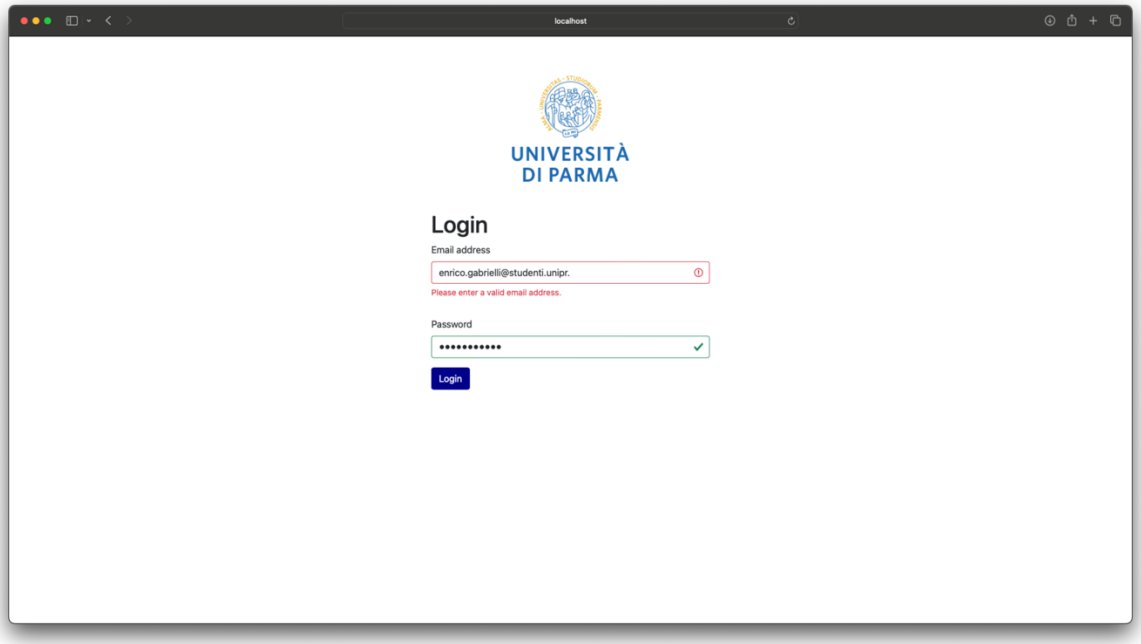


Homepage di un amministratore.

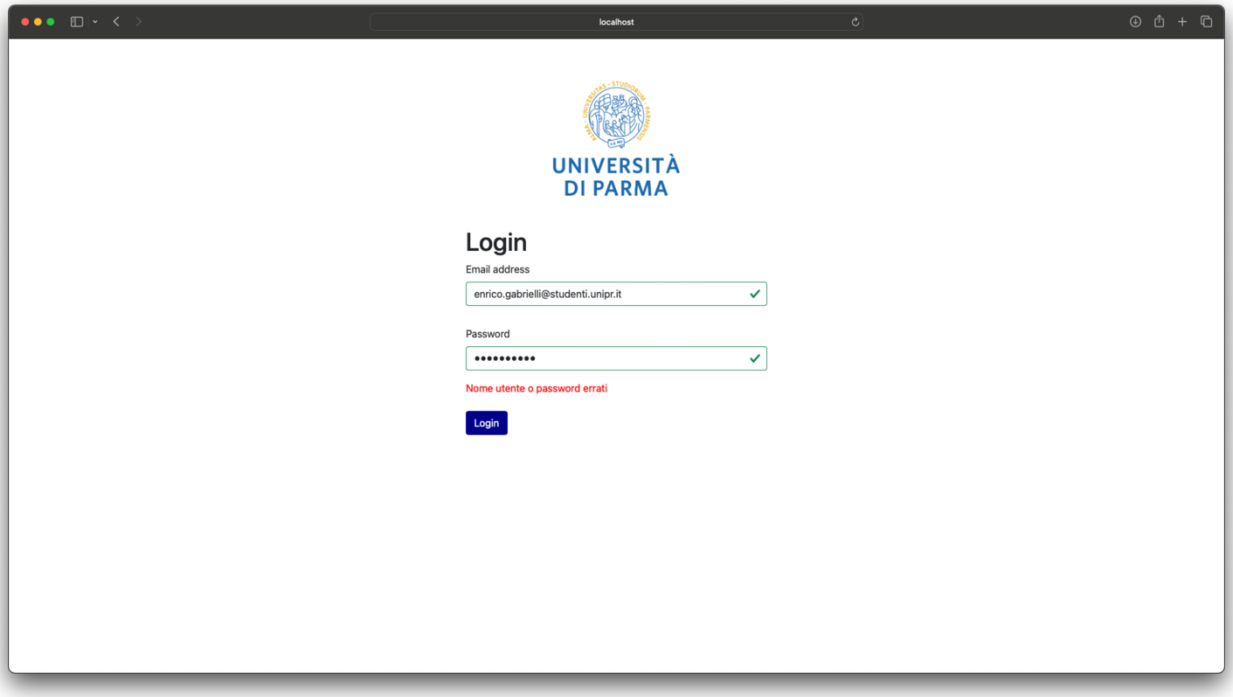


Homepage di uno studente (equivalente a quella di un professore).

Successivamente, si è controllato il funzionamento delle sessioni utente effettuando un refresh del sito a partire dalla visualizzazione del calendario e verificando che l'utente non venisse (correttamente) reindirizzato alla sezione di login. Poi, si è effettuato il logout e si è effettuato un nuovo refresh. In questo caso, l'utente visualizza (correttamente) la sezione di login.
Infine, si è testato il comportamento del form di login a fronte di input indesiderati, provando a immettere un indirizzo email non valido e, successivamente, una password che non corrisponde a quella dell'utente.



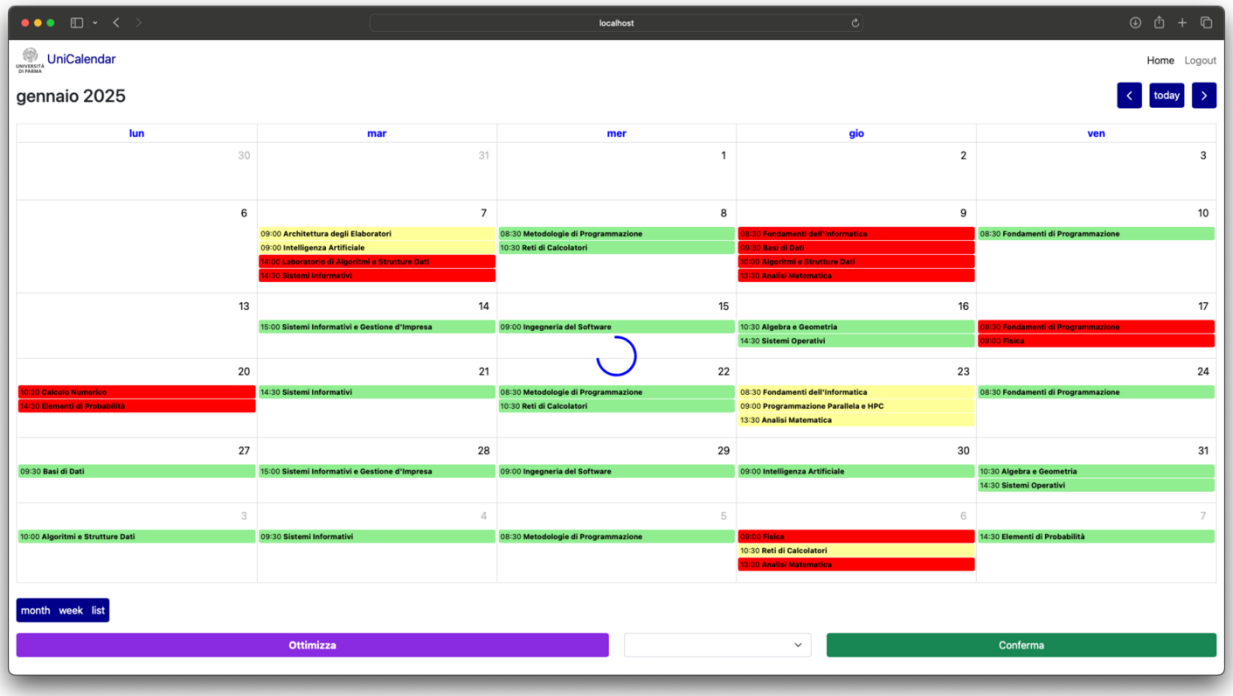
Errore relativo ad un indirizzo email non valido.



Errore relativo a nome utente o password errati.

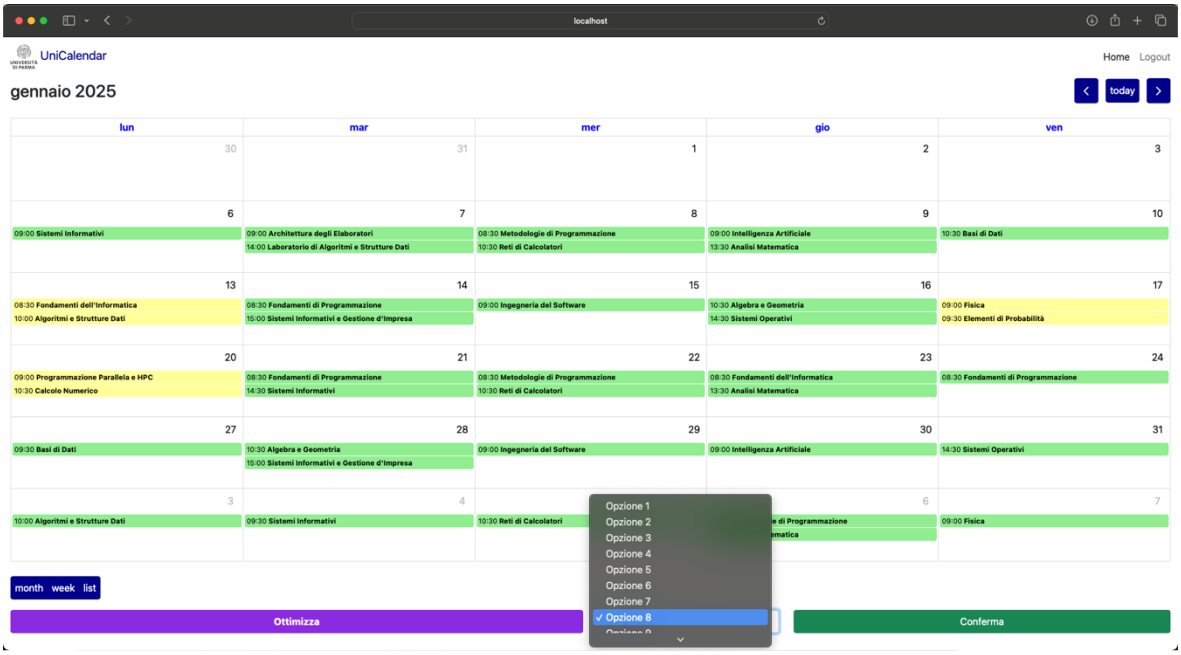
8.1.2. Test di ottimizzazione calendarizzazione

In seguito, dopo aver effettuato l'accesso con un account amministratore, si è testato il funzionamento dell'algoritmo di ottimizzazione. Si è, dunque, proceduto a cliccare sul bottone "Ottimizza", visualizzando istantaneamente un indicatore di loading.



Indicatore di loading durante l'ottimizzazione.

Dopo qualche secondo, il calendario si è aggiornato con l'ottimizzazione migliore, il menu a tendina si è popolato con tutte le opzioni con il medesimo peso minimo e il pulsante "Conferma" si è abilitato.



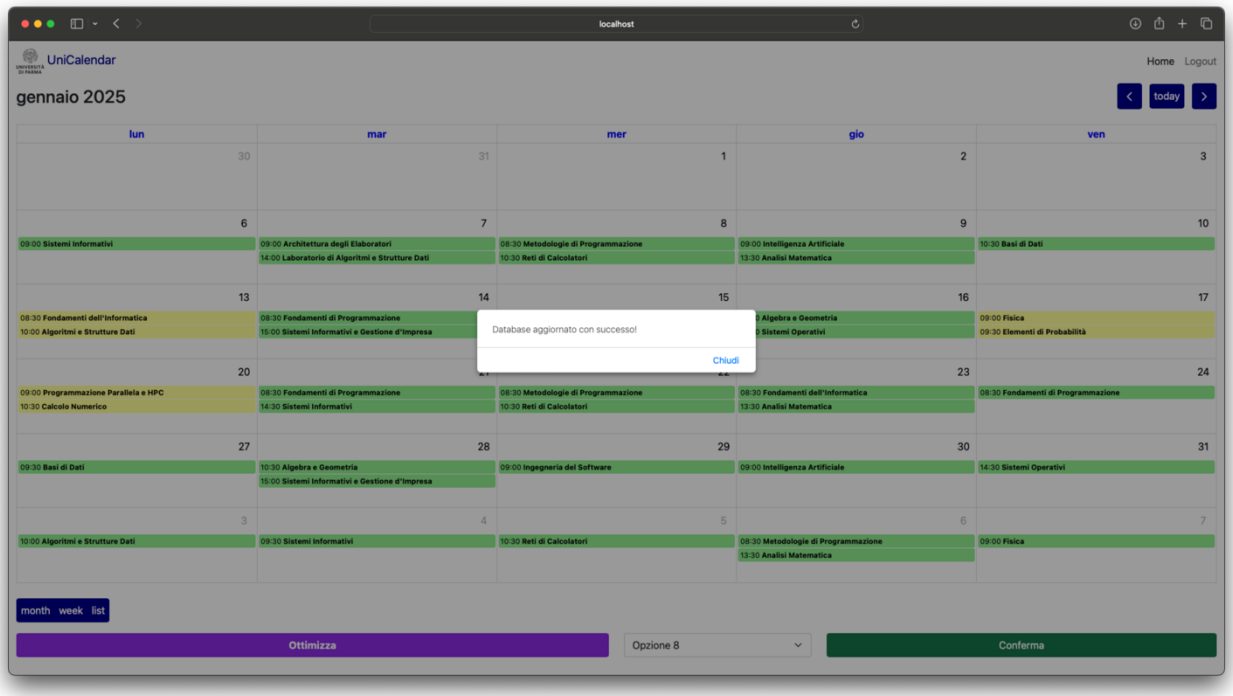
Calendario aggiornato dopo l'ottimizzazione.

8.2. Test di integrazione

8.2.1. Test dell'integrazione tra il frontend e il backend tramite le API RESTful

Nonostante il funzionamento delle API RESTful sia già stato in parte provato dalla visualizzazione stessa degli appelli all'interno del calendario (dal momento che essi vengono ottenuti proprio da una chiamata API verso il backend), si è proceduto a testare le chiamate rimanenti.

Per prima cosa, si è cliccato sul pulsante "Conferma" per eseguire un update e salvare all'interno del database la calendarizzazione aggiornata, ricevendo immediatamente un messaggio di conferma.



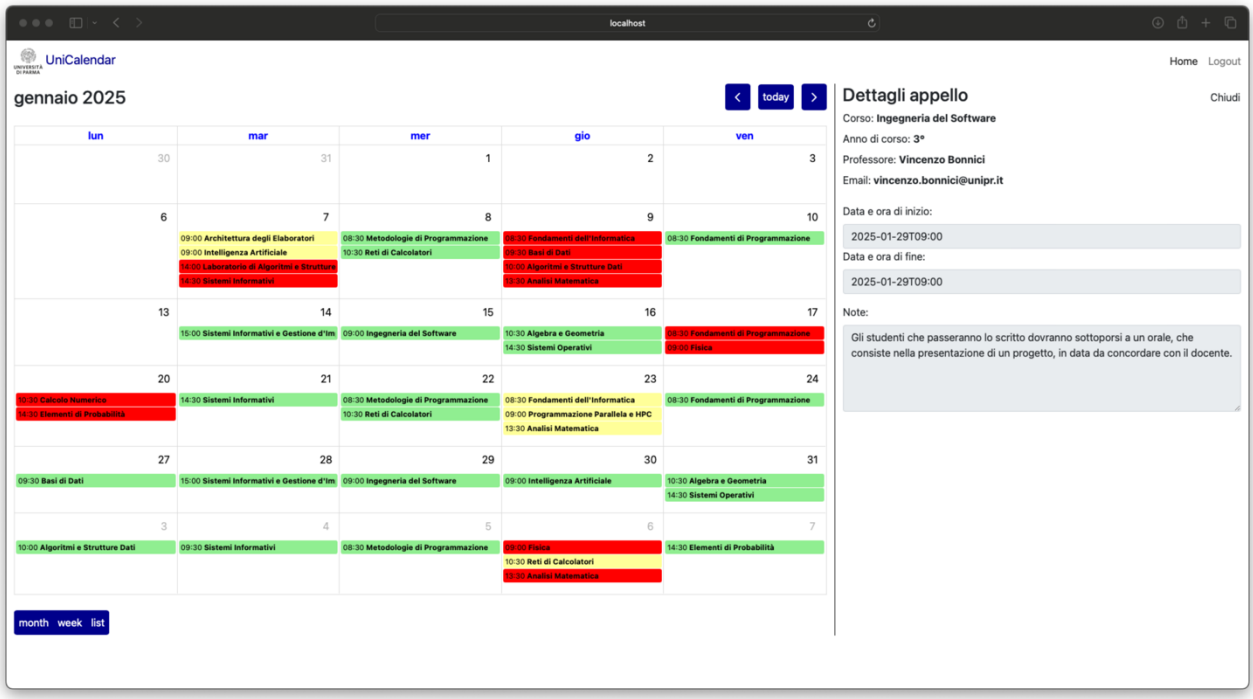
Conferma dell’avvenuto salvataggio della calendarizzazione sul database.

Per verificare effettivamente che gli appelli aggiornati siano stati salvati, si è effettuato il logout e poi, di nuovo, il login, in modo da constatare se la memorizzazione e la conseguente estrazione delle nuove date sia avvenuta con successo, ricevendo esito positivo.

8.3. Test di usabilità

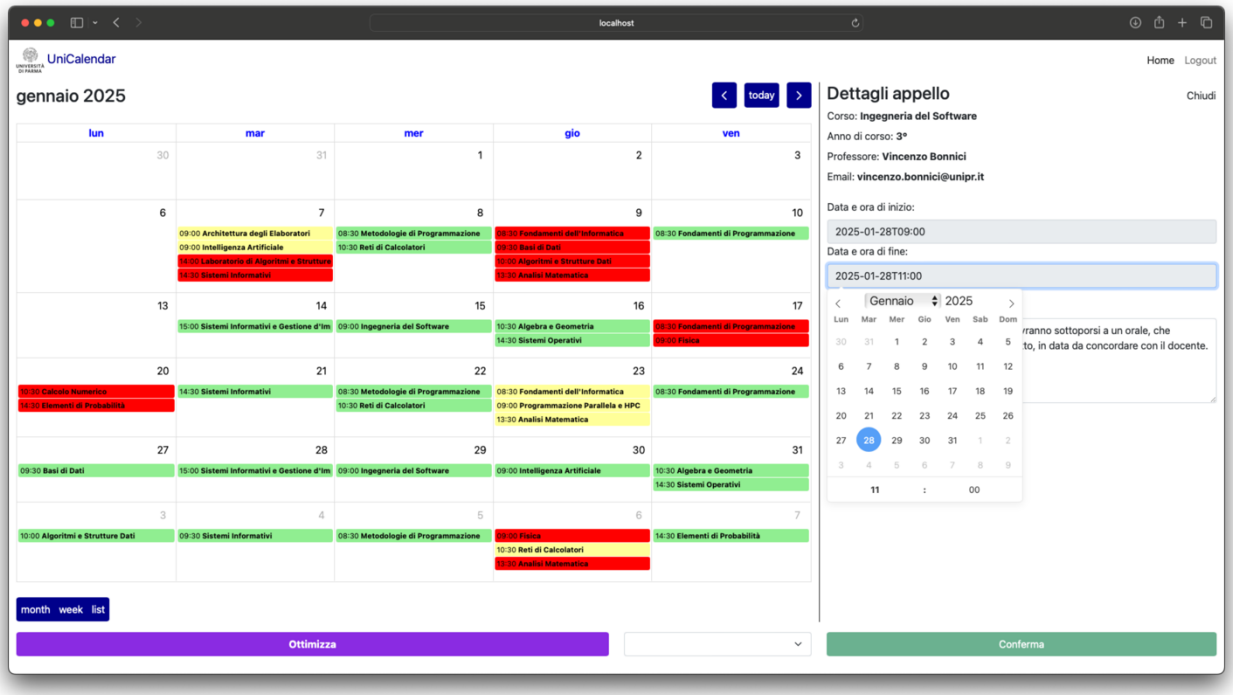
8.3.1. Verifica dell’intuitività dell’interfaccia e dell'accessibilità delle funzionalità

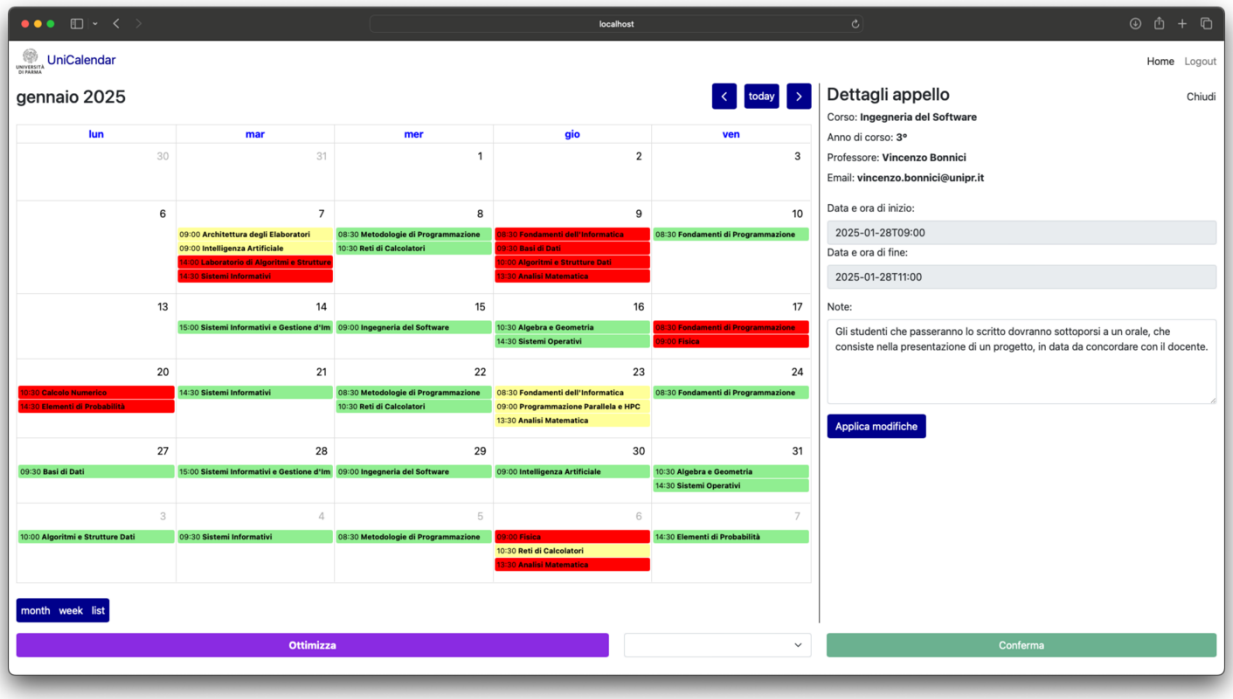
Per verificare l’usabilità dell’interfaccia utente, si sono effettuati alcuni test relativi alle funzionalità messe a disposizione. Si è, quindi, passati alla verifica della sezione relativa alle informazioni degli appelli. A tale scopo, si è cliccato su un appello presente nel calendario, visualizzando correttamente la sezione.



Dettagli appello in una sessione studente (equivalente a quella di un professore).

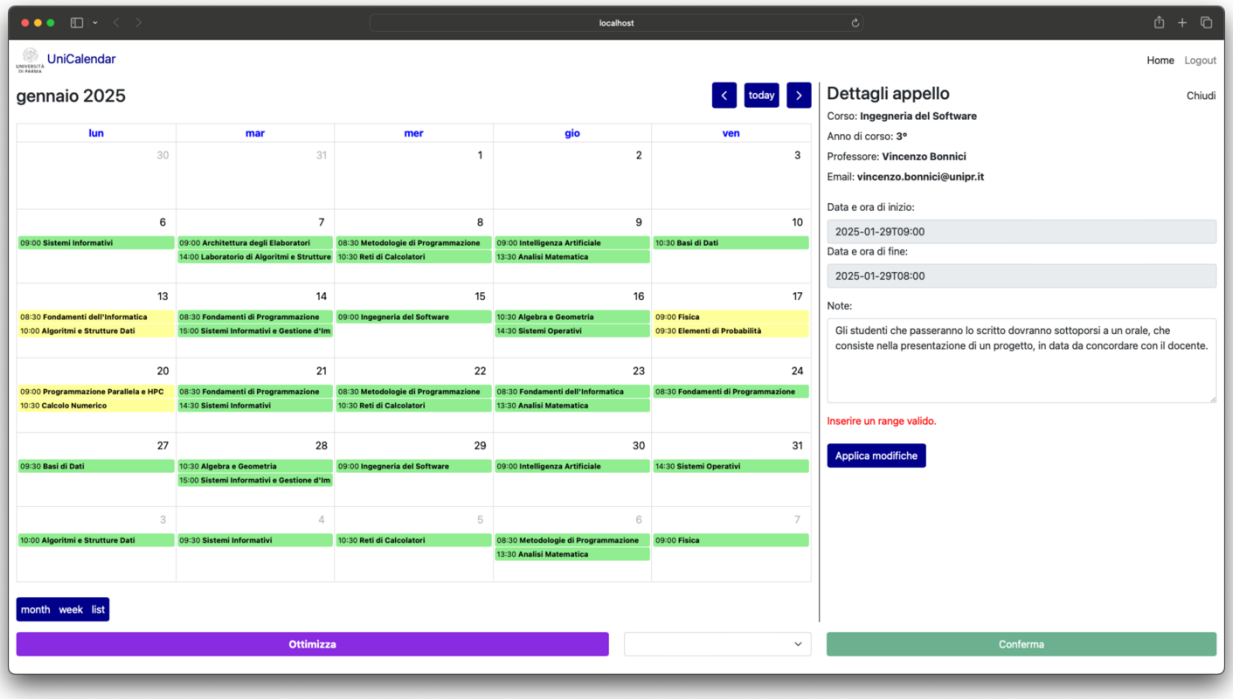
In questo caso, gli input relativi agli orari di inizio e di fine e alle note sono disabilitati, ma effettuando l'accesso con un account da amministratore saranno modificabili e confermabili attraverso l'apposito bottone (il quale, anch'esso, si abilita solo dopo aver modificato uno dei campi).





Dettagli appello in una sessione amministratore.

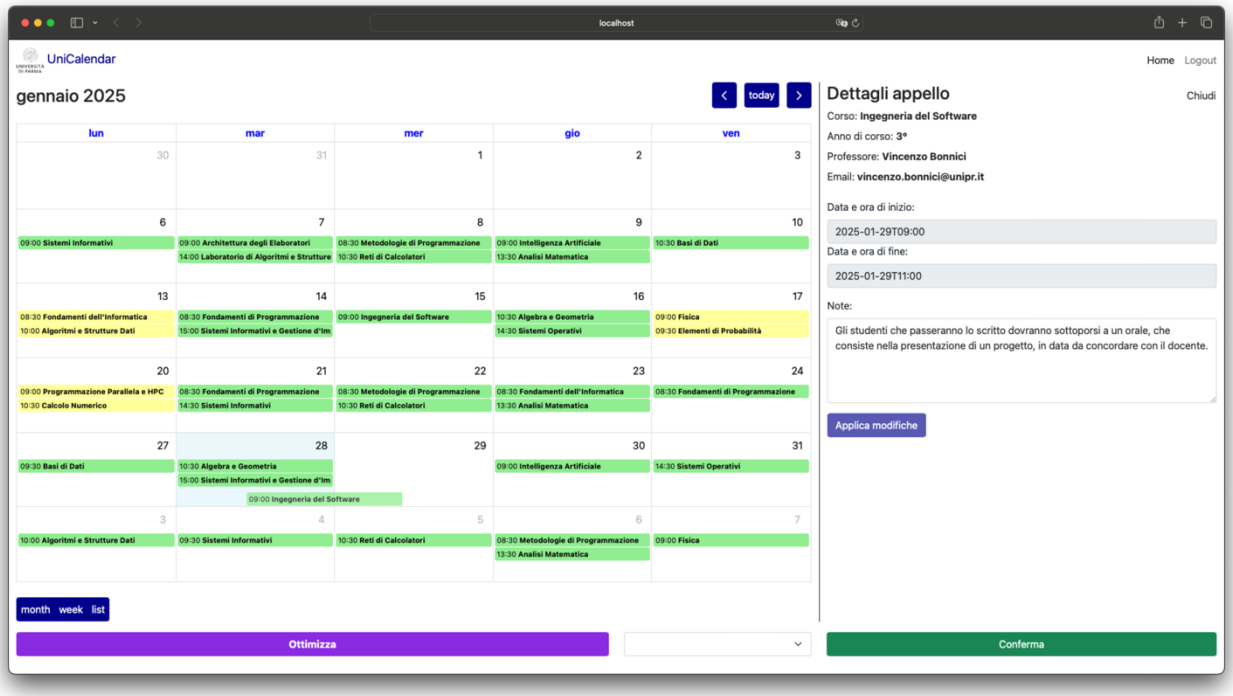
Proseguendo, si è modificato una data per mezzo dei corrispondenti picker e si è cliccato sul pulsante “Applica modifiche”. Inizialmente, si è inserito un orario di inizio successivo ad un orario di fine, per testare il comportamento dell’applicazione a fronte di input indesiderati, ricevendo un messaggio di errore.



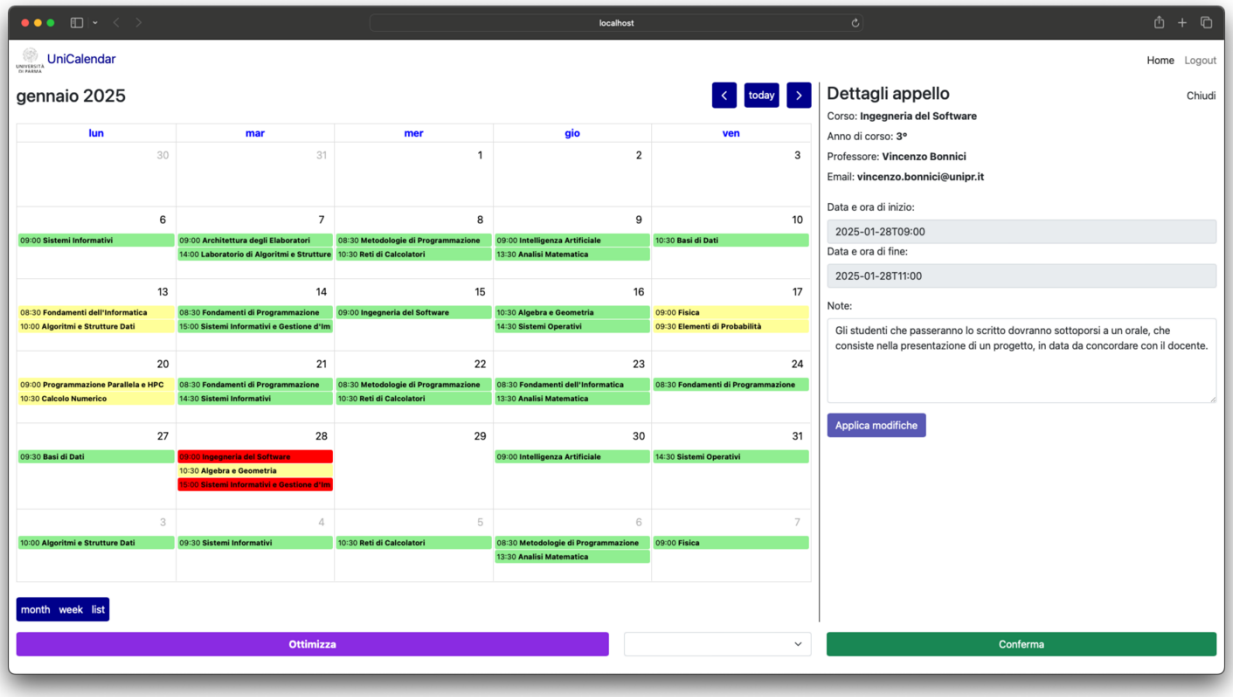
Errore relativo a un range di date e orari incorretto.

In seguito, si è testato il funzionamento a fronte di un input corretto, riscontrando il risultato aspettato, ovvero lo spostamento dell'appello alla data inserita e l'aggiornamento della UI con i colori corretti.

Infine, si è testato il corretto funzionamento della funzione *drag and drop*, provando a spostare un appello in un'altra data e verificando che l'interfaccia grafica si aggiornasse a dovere.

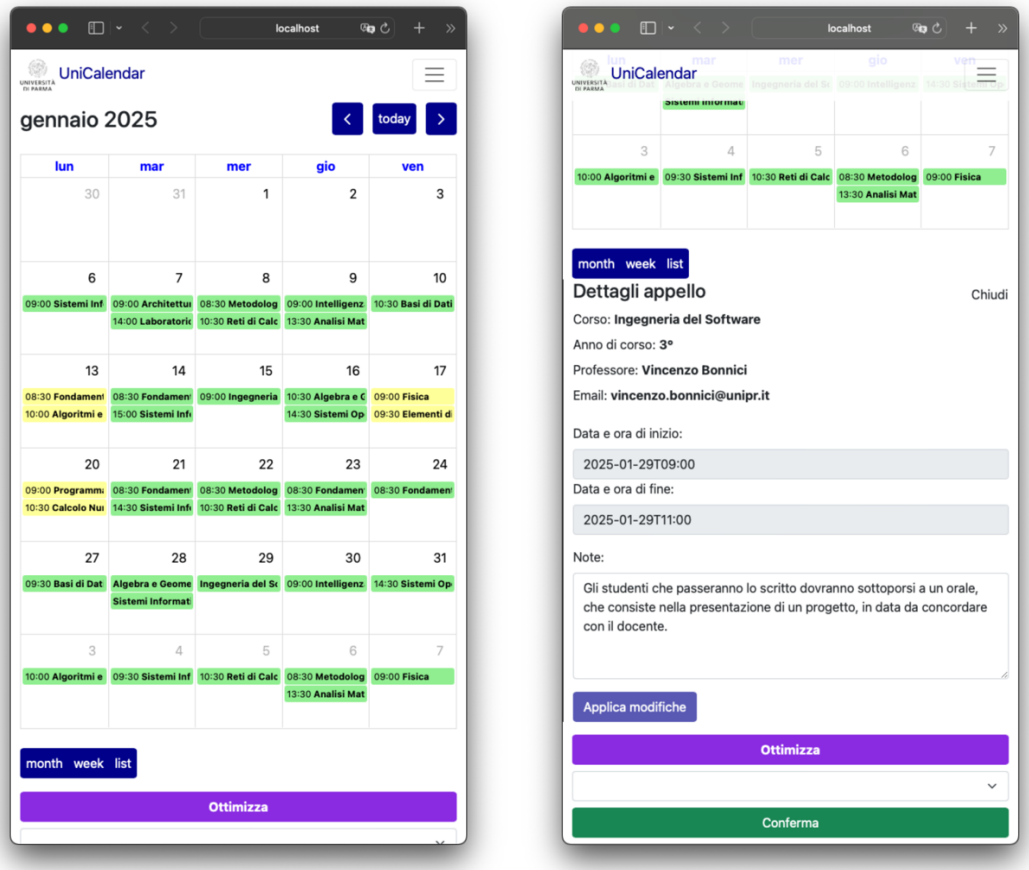


Interfaccia grafica durante lo spostamento.



Interfaccia grafica dopo lo spostamento.

Come ultima cosa, si è verificato che l'applicazione fosse facilmente utilizzabile anche su dispositivi di risoluzione e forma diversa, come cellulari e tablet. In questo caso, nella barra di navigazione i pulsanti "Home" e "Logout" vengono sostituiti da un toggler, la visualizzazione del calendario si adatterà alla larghezza dello schermo, i dettagli degli appelli verranno visualizzati sotto al calendario invece che affianco e gli elementi della sezione relativa all'ottimizzazione verranno disposti in verticale invece che in orizzontale.



Interfaccia grafica su dispositivi mobile.

9. Sviluppi futuri

Nella progettazione e nello sviluppo dell'applicazione, si è lasciato spazio anche a una riflessione su quelli che potrebbero essere gli sviluppi futuri. In particolare, si sono individuati i seguiti campi di azione:

- possibilità per i professori di poter inserire le preferenze di appello
- possibilità per gli amministratori di assegnare ai professori un nuovo insegnamento.