INDICE

# **Introduzione al progetto**

# **Tecnologie utilizzate**

## Spring Boot

### Descrizione di Spring Boot e come è stato utilizzato nel progetto

### Descrizione delle dipendenze utilizzate nel progetto

### Configurazione di Spring Boot per il progetto

## Angular

### Descrizione di Angular e come è stato utilizzato nel progetto

### Descrizione di Bootstrap e principali funzionalità

### Descrizione dei moduli utilizzati nel progetto

### Descrizione dei componenti utilizzati nel progetto

### Descrizione dei servizi utilizzati nel progetto

## Postman

### Descrizione di Postman e come è stato utilizzato nel progetto

### Esempi di richieste API con Postman

## Swagger

### Descrizione di Swagger e come è stato utilizzato nel progetto

## Database

### Descrizione del database e come è stato utilizzato nel progetto

### Modello di dati utilizzato nel progetto

### Descrizione delle tabelle utilizzate nel progetto

### Descrizione della migrazione tra Oracle e Postgres

# **Documentazione delle REST API**

## CourseRestController

## ExamRestController

## StudentCourseRestController

## StudentExamRestController

## StudentRestController

## TeacherRestController

## TokenRestController

## UserRestController

# **Casi d'uso**

## Descrizione dei casi d'uso principali

## Descrizione dei flussi principali

## Descrizione delle eccezioni e dei casi di errore

Introduzione al progetto

Questo progetto è stato sviluppato a scopo puramente didattico e si concentra sull'implementazione di un'applicazione che consenta ai docenti di tenere corsi e valutare gli esami degli studenti, mentre gli studenti possono iscriversi ai corsi e tenere un libretto dei risultati degli esami svolti.

L'applicazione è stata sviluppata utilizzando diverse tecnologie, tra cui Spring Boot per il back-end, Angular per il front-end, Bootstrap per la progettazione dell'interfaccia utente e Postman per la documentazione e il testing delle REST API. Inoltre, è stato utilizzato Swagger per generare la documentazione delle API e un database relazionale per la gestione dei dati.

Nella sezione successiva, verranno descritte le tecnologie utilizzate in maggior dettaglio, spiegando come sono state utilizzate nel progetto e come sono state configurate. Successivamente, verranno illustrate le REST API sviluppate e i relativi casi d'uso, con descrizione dei flussi principali e delle eccezioni.

Tecnologie Utilizzate

# Spring Boot

### Descrizione di Spring Boot e come è stato utilizzato nel progetto

Spring Boot è un framework per lo sviluppo di applicazioni web che si basa sul framework Spring. Si tratta di un progetto open source che permette di creare facilmente applicazioni stand-alone, in quanto include al suo interno un server web embedded, senza la necessità di configurazioni particolari.

Nel progetto in questione, Spring Boot è stato utilizzato come principale framework per la creazione di un'API RESTful. In particolare, è stato scelto per la sua flessibilità e per la facilità con cui si possono configurare i servizi web e le componenti della web application.

### Descrizione delle dipendenze utilizzate nel progetto

* **spring-security-crypto (versione 5.3.3.RELEASE)**: dipendenza di Spring Security che fornisce funzionalità di crittografia e hash.
* **spring-boot-starter-web**: dipendenza di Spring Boot che include le dipendenze necessarie per sviluppare un'applicazione web, tra cui Tomcat.
* **javax.servlet-api (versione 4.0.1)**: dipendenza che fornisce le API di base per la gestione delle richieste e delle risposte HTTP in una applicazione web basata su Java.
* **spring-boot-starter-data-jpa**: dipendenza di Spring Boot per la gestione delle operazioni di persistenza dei dati con JPA (Java Persistence API).
* **spring-boot-devtools**: dipendenza di Spring Boot che fornisce funzionalità di hot-reload dell'applicazione, consentendo di visualizzare immediatamente le modifiche apportate al codice durante lo sviluppo.
* **ojdbc8**: dipendenza per la connessione a database Oracle tramite JDBC (Java Database Connectivity).
* **lombok (versione 1.18.22)**: dipendenza che fornisce funzionalità di generazione automatica di codice boilerplate, come getter e setter per le proprietà delle classi.
* **spring-boot-starter-test**: dipendenza di Spring Boot che fornisce le dipendenze necessarie per scrivere test unitari per l'applicazione.
* **spring-boot-starter-tomcat**: dipendenza di Spring Boot che fornisce il server web Tomcat integrato per l'esecuzione dell'applicazione.
* **spring-boot-starter-mail (versione 3.0.0)**: dipendenza di Spring Boot che fornisce funzionalità per l'invio di e-mail dall'applicazione.
* **javax.mail (versione 1.6.2)**: dipendenza per la gestione delle operazioni di invio di e-mail in una applicazione Java.
* **springfox-swagger2 (versione 3.0.0)**: dipendenza per l'integrazione di Swagger 2 in una applicazione Spring Boot, utilizzato per la documentazione dell'API.
* **springfox-boot-starter (versione 3.0.0)**: dipendenza per l'integrazione di Swagger in una applicazione Spring Boot.
* **springfox-swagger-ui (versione 3.0.0)**: dipendenza per l'interfaccia utente di Swagger, utilizzata per visualizzare la documentazione dell'API generata.
* **spring-security-web (versione 5.5.5)**: dipendenza di Spring Security che fornisce funzionalità per l'autenticazione e l'autorizzazione nelle applicazioni web.
* **spring-security-config (versione 5.5.5)**: dipendenza di Spring Security che fornisce funzionalità di configurazione per la gestione dell'autenticazione e dell'autorizzazione.
* **java-jwt (versione 3.18.1)**: dipendenza che fornisce funzionalità per la creazione e la validazione di token JWT (JSON Web Token).
* **jjwt-api (versione 0.11.2)**: dipendenza per l'utilizzo delle API di creazione e validazione di token JWT.
* **jjwt-impl**: dipendenza per l'implementazione di JWT in Java.
* **jjwt-jackson**: dipendenza per la gestione delle funzionalità di serializzazione e deserializzazione dei JSON utilizzati da JWT.

Queste sono solo alcune delle dipendenze utilizzate nel progetto, ma coprono le principali funzionalità utilizzate.

### Configurazione di Spring Boot per il progetto

La configurazione di Spring Boot è stata personalizzata tramite il file **application.properties**. In particolare, il server è stato configurato per ascoltare sulla porta 8090. Per quanto riguarda il recupero della password, è stata utilizzata la mail di Google smtp.gmail.com sulla porta 587. Le credenziali della mail sono specificate all'interno del file **application.properties**.

Per quanto riguarda il database, è stato utilizzato Oracle Database. La configurazione del database è stata specificata all'interno del file **application.properties**. In particolare, l'URL di connessione al database è **jdbc:oracle:thin:@localhost:1521:xe**, e le credenziali di accesso al database sono specificate tramite i parametri username e password. È stata inoltre specificata la classe del driver Oracle con il parametro **driver-class-name**.

Sono state inoltre specificate alcune impostazioni di sicurezza per la gestione degli utenti, specificando le credenziali dell'utente admin tramite i parametri **spring.security.user.name** e **spring.security.user.password**.

Infine, per la gestione delle query e delle operazioni sul database, sono state utilizzate le funzionalità fornite da Spring Data JPA, con la configurazione specificata tramite i parametri **spring.jpa.show-sql**, **spring.jpa.properties.hibernate.format\_sql** e **spring.jpa.properties.hibernate.dialect**. In particolare, è stato configurato Hibernate per generare le tabelle e le colonne automaticamente utilizzando **spring.jpa.hibernate.ddl-auto**.

# Angular

### **Descrizione di Angular e come è stato utilizzato nel progetto**

Angular è un framework open source per lo sviluppo di applicazioni web. Nel nostro progetto, abbiamo utilizzato Angular per la parte client-side dell'applicazione, in particolare per la creazione dell'interfaccia utente e la gestione delle richieste API verso il backend.

Abbiamo utilizzato la versione 12 di Angular, la quale è stata scelta per la sua flessibilità e per la vasta gamma di funzionalità offerte dal framework. Abbiamo inoltre utilizzato diverse librerie di terze parti per aumentare la produttività e semplificare lo sviluppo, tra cui Bootstrap per la creazione dell'interfaccia utente e RxJS per la gestione delle richieste HTTP.

Il progetto è stato suddiviso in moduli per facilitare la gestione e la manutenzione del codice. In particolare, abbiamo definito un modulo per la gestione dell'autenticazione e dell'autorizzazione, un modulo per la gestione delle classi e un modulo per la gestione delle materie.

Abbiamo utilizzato i componenti di Angular per creare le diverse schermate dell'applicazione e abbiamo definito diversi servizi per la gestione delle richieste API verso il backend. Inoltre, abbiamo utilizzato il router di Angular per la navigazione tra le diverse pagine dell'applicazione.

Angular ci ha permesso di creare un'interfaccia utente moderna e user-friendly, offrendo al contempo una struttura solida e facile da mantenere.

### **Descrizione di Bootstrap e principali funzionalità**

Bootstrap è un framework front-end open-source sviluppato da Twitter che facilita lo sviluppo di siti web e applicazioni web. Il framework fornisce una vasta gamma di componenti e stili predefiniti che consentono di creare interfacce utente moderne e responsive con facilità.

Le principali funzionalità di Bootstrap includono:

* Responsive design: Bootstrap è progettato per adattarsi automaticamente a diverse dimensioni di schermo, consentendo di creare interfacce utente che funzionano su desktop, tablet e dispositivi mobili.
* Griglia flessibile: Bootstrap utilizza un sistema di griglia flessibile che consente di organizzare i contenuti in modo chiaro e coerente, adattandosi alle dimensioni dello schermo.
* Componenti UI predefiniti: il framework include numerosi componenti UI predefiniti, come pulsanti, modali, alert, form, navbar e molto altro ancora, che possono essere utilizzati per creare rapidamente interfacce utente funzionali e accattivanti.
* Temi personalizzabili: Bootstrap include anche una serie di temi predefiniti che consentono di personalizzare l'aspetto del proprio sito web o applicazione web. Inoltre, è possibile personalizzare il design del framework utilizzando vari strumenti di personalizzazione, come SASS e LESS.
* Supporto cross-browser: Bootstrap è progettato per funzionare su tutti i principali browser web, inclusi Chrome, Firefox, Safari, Edge e Internet Explorer.
* Documentazione completa: il framework viene fornito con una documentazione completa e dettagliata, che include esempi di codice, tutorial e linee guida per l'utilizzo delle funzionalità del framework.

In sintesi, Bootstrap è un framework front-end flessibile, potente e facile da usare, che consente di creare rapidamente interfacce utente moderne e responsive.

### **Descrizione dei moduli utilizzati nel progetto**

* **BrowserModule**: modulo fornito da Angular che viene importato nell'app.module.ts e fornisce i servizi di base per l'applicazione. Esso contiene tutte le direttive di base, come \*ngIf e \*ngFor, ed è necessario per far funzionare l'applicazione.
* **AppRoutingModule**: modulo che gestisce il routing dell'applicazione. Esso definisce le rotte e le associa ai componenti corrispondenti. Inoltre, è possibile definire delle rotte parametriche e delle rotte figlie.
* **HttpClientModule**: modulo che permette di effettuare chiamate HTTP all'interno dell'applicazione. Esso contiene il servizio HttpClient, che è utilizzato per effettuare richieste RESTful al server.
* **FormsModule**: modulo che permette di utilizzare i template-driven forms di Angular. Esso fornisce la direttiva ngModel, che viene utilizzata per associare i dati del form al modello.
* **ReactiveFormsModule**: modulo che permette di utilizzare i reactive forms di Angular. Esso permette di gestire il form come un oggetto, permettendo di effettuare validazioni e controllare lo stato del form in modo reattivo.
* **CommonModule**: modulo fornito da Angular che contiene le direttive comuni utilizzate nei componenti. Esso viene importato nei moduli in cui sono necessarie le direttive comuni, come \*ngIf e \*ngFor.
* **CKEditorModule**: modulo che permette di utilizzare l'editor di testo CKEditor all'interno dell'applicazione. Esso fornisce la direttiva ckeditor, che viene utilizzata per renderizzare l'editor.
* **NgMultiSelectDropDownModule**: modulo che permette di utilizzare il componente di selezione multipla all'interno dell'applicazione. Esso fornisce la direttiva ng-multiselect-dropdown, che viene utilizzata per renderizzare il componente.
* **BrowserAnimationsModule**: modulo che permette di utilizzare le animazioni all'interno dell'applicazione. Esso contiene le direttive necessarie per gestire le animazioni, come \*ngIf e \*ngFor.
* **MatDialogModule**: modulo che permette di utilizzare il componente MatDialog all'interno dell'applicazione. Esso fornisce la direttiva MatDialog, che viene utilizzata per renderizzare il componente dialog.

### **Descrizione dei componenti utilizzati nel progetto**

* **Account**: componente che permette all'utente di modificare le informazioni del proprio account, come la password e l'indirizzo email.
* **Add-course**: componente utilizzato dal docente per aggiungere un nuovo corso.
* **Add-exam**: componente utilizzato dal docente per aggiungere un nuovo esame.
* **Contact-us**: componente utilizzato dagli utenti per contattare gli amministratori dell'applicazione.
* **Courses**: componente che mostra la lista dei corsi disponibili per gli studenti, con la possibilità di iscriversi a uno o più corsi.
* **Exam-details**: componente che mostra i dettagli di un esame, come la data, l'aula, il docente e la lista degli studenti iscritti.
* **Exam-results**: componente che mostra i risultati degli esami di un determinato studente.
* **Home-student**: componente che rappresenta la home page per gli studenti, con la lista dei corsi a cui sono iscritti e i prossimi esami.
* **Home-teache**r: componente che rappresenta la home page per i docenti, con la lista dei corsi e degli esami da loro tenuti.
* **Homepage**: componente padre delle home personalizzate per insegnante e studente.
* **Inside-app**: componente che gestisce tutte le rotte figlie accessibili dopo il login, come Home-student, Home-teacher, My-courses, My-exams, etc.
* **Login**: componente utilizzato per accedere all'applicazione, richiedendo user ID e password.
* **Manage-exams**: componente utilizzato dal docente per gestire gli esami di un corso, con la possibilità di modificarli o cancellarli.
* **My-courses**: componente che mostra la lista dei corsi a cui l'utente è iscritto, con la possibilità di accedere ai dettagli di ciascun corso.
* **My-courses-student**: componente che mostra la lista dei corsi a cui lo studente è iscritto.
* **My-courses-teacher**: componente che mostra la lista dei corsi gestiti dal docente, con la possibilità di accedere ai dettagli di ciascun corso e di gestire gli esami.
* **My-exams**: componente che mostra la lista degli esami a cui l'utente è iscritto, con la possibilità di accedere ai dettagli di ciascun esame e di visualizzare i propri voti.
* **My-exams-student**: componente che mostra la lista degli esami a cui lo studente è iscritto, con la possibilità di accedere ai dettagli di ciascun esame e di visualizzare i propri voti.
* **My-exams-teacher**: componente che mostra la lista degli esami gestiti dal docente, con la possibilità di accedere ai dettagli di ciascun esame e di inserire i voti degli studenti.
* **Navbar**: componente che rappresenta la barra di navigazione dell'applicazione, con la possibilità di accedere alle varie sezioni dell'applicazione.
* **Reset-password**: componente utilizzato per reimpostare la password, richiedendo l'email e il codice di verifica.
* **Reset-password-request**: componente utilizzato per richiedere la mail per il reindirizzamento alla modifica della password.
* **Sign-up**: componente utilizzato per permettere agli utenti di registrarsi alla piattaforma inserendo le proprie informazioni personali e di login.
* **Vote-exam**: componente utilizzato dal docente per inserire i voti degli esami svolti dagli studenti e gestire la valutazione finale di ciascun esame.

### **Descrizione dei servizi utilizzati nel progetto**

* **AuthService**: servizio per la gestione dell'autenticazione degli utenti.
* **CourseService**: servizio per la gestione dei corsi, con la possibilità di creare, modificare ed eliminare un corso.
* **ExamService**: servizio per la gestione degli esami, con la possibilità di creare, modificare ed eliminare un esame.
* **JwtService**: servizio per la gestione del JSON Web Token utilizzato per l'autenticazione degli utenti.
* **StudentCourseService**: servizio per la gestione dei corsi a cui uno studente è iscritto, con la possibilità di recuperare la lista dei corsi e di iscriversi ad un corso.
* **StudentExamService**: servizio per la gestione degli esami a cui uno studente è iscritto, con la possibilità di recuperare la lista degli esami e di visualizzare i propri voti.
* **TokenService**: servizio per la gestione del token utilizzato per la gestione della password dimenticata.
* **UserService**: servizio per la gestione degli utenti, con la possibilità di recuperare la lista degli utenti e di creare un nuovo utente.

# Postman

### Descrizione di Postman e come è stato utilizzato nel progetto

Postman è un'applicazione che consente di testare le API REST. Nel nostro progetto, abbiamo utilizzato Postman per testare le chiamate HTTP alle nostre API e per verificare il corretto funzionamento dei servizi REST.

In particolare, abbiamo utilizzato Postman per inviare richieste HTTP alle nostre API e per verificare la corretta risposta. In questo modo, abbiamo potuto testare la corretta integrazione tra il frontend e il backend e verificare che i dati vengano scambiati correttamente.

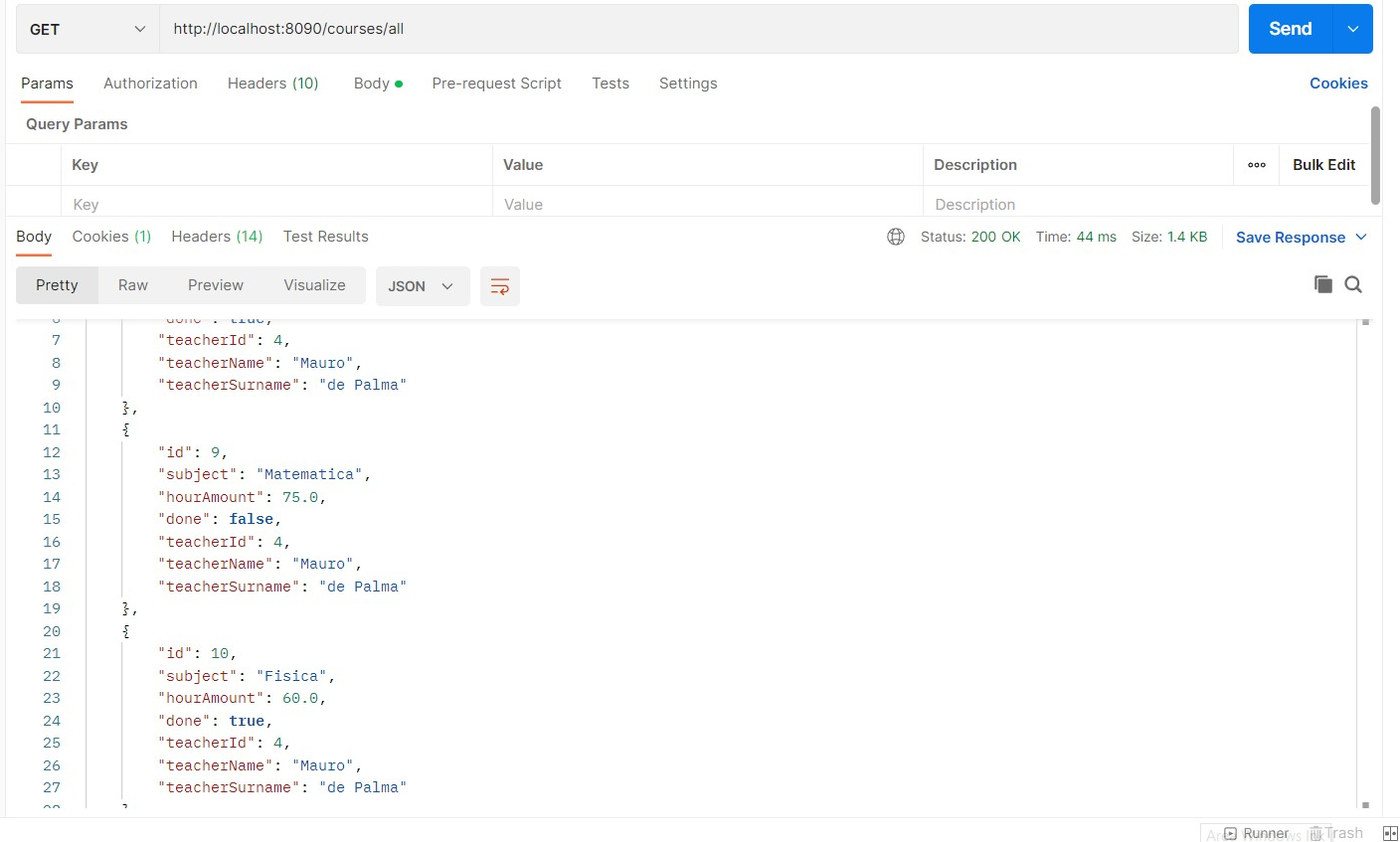
Inoltre, Postman ci ha permesso di verificare la corretta configurazione dei parametri di autenticazione, come ad esempio il token JWT.

In sintesi, Postman è stato uno strumento fondamentale per lo sviluppo del nostro progetto, poiché ci ha permesso di testare e verificare il corretto funzionamento delle nostre API REST.

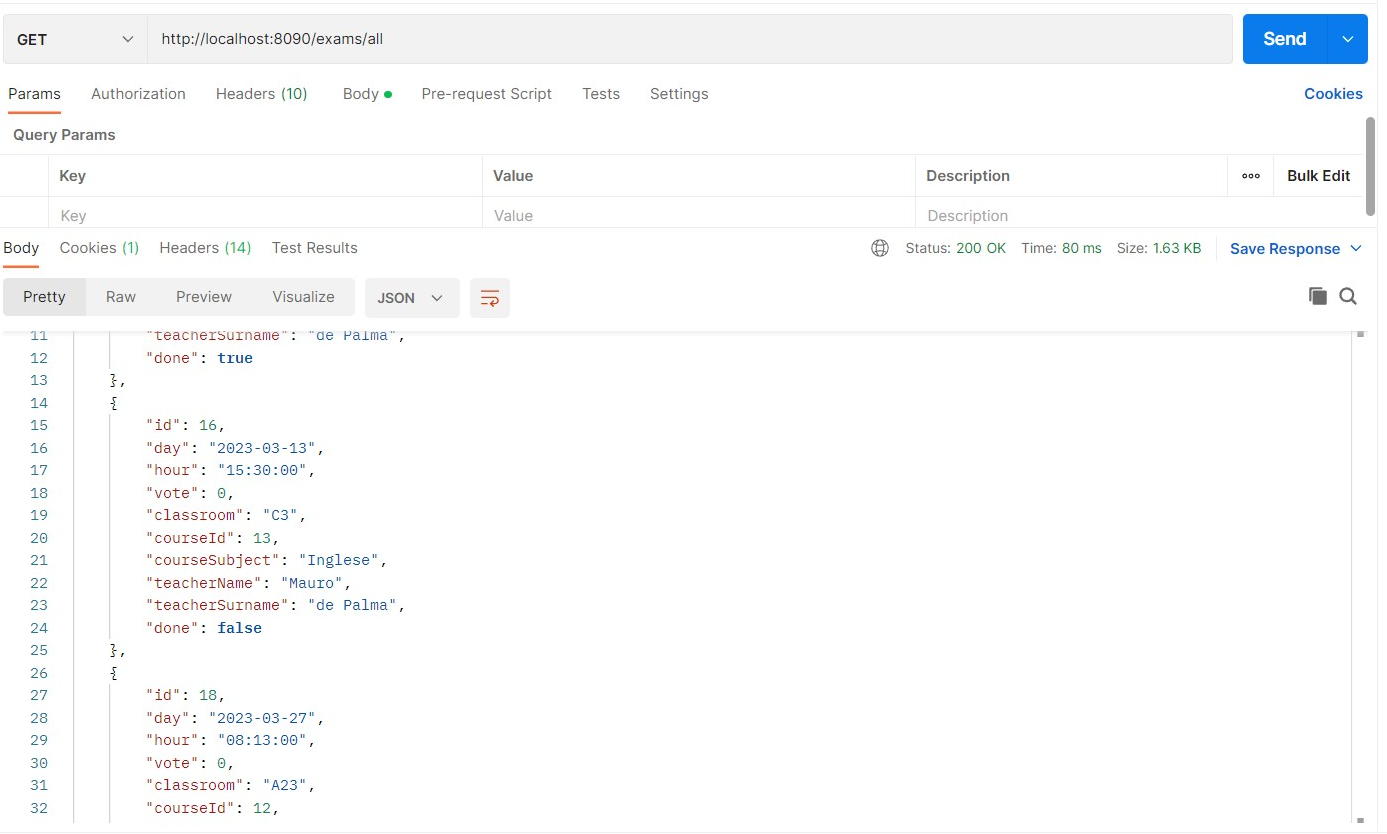
### Esempi di richieste API con Postman

In questa sezione presenteremo alcuni esempi di richieste API utilizzate nel progetto, accompagnate dai relativi screenshot di Postman per una migliore comprensione. Le richieste API copriranno le principali funzionalità del progetto, come l'autenticazione, la gestione dei corsi, la gestione degli esami e così via.

**Recupero di tutti i corsi**:



**Recupero di tutti gli esami**:



Abbiamo mostrato come le richieste API siano state utilizzate per comunicare con il server e gestire i dati all'interno dell'applicazione. Postman si è rivelato uno strumento molto utile per testare le richieste API e per assicurarsi che tutto funzioni correttamente. La comprensione delle API utilizzate e la capacità di testarle correttamente sono aspetti fondamentali per assicurare un funzionamento fluido e affidabile dell'applicazione.

# Swagger

### Descrizione di Swagger e come è stato utilizzato nel progetto

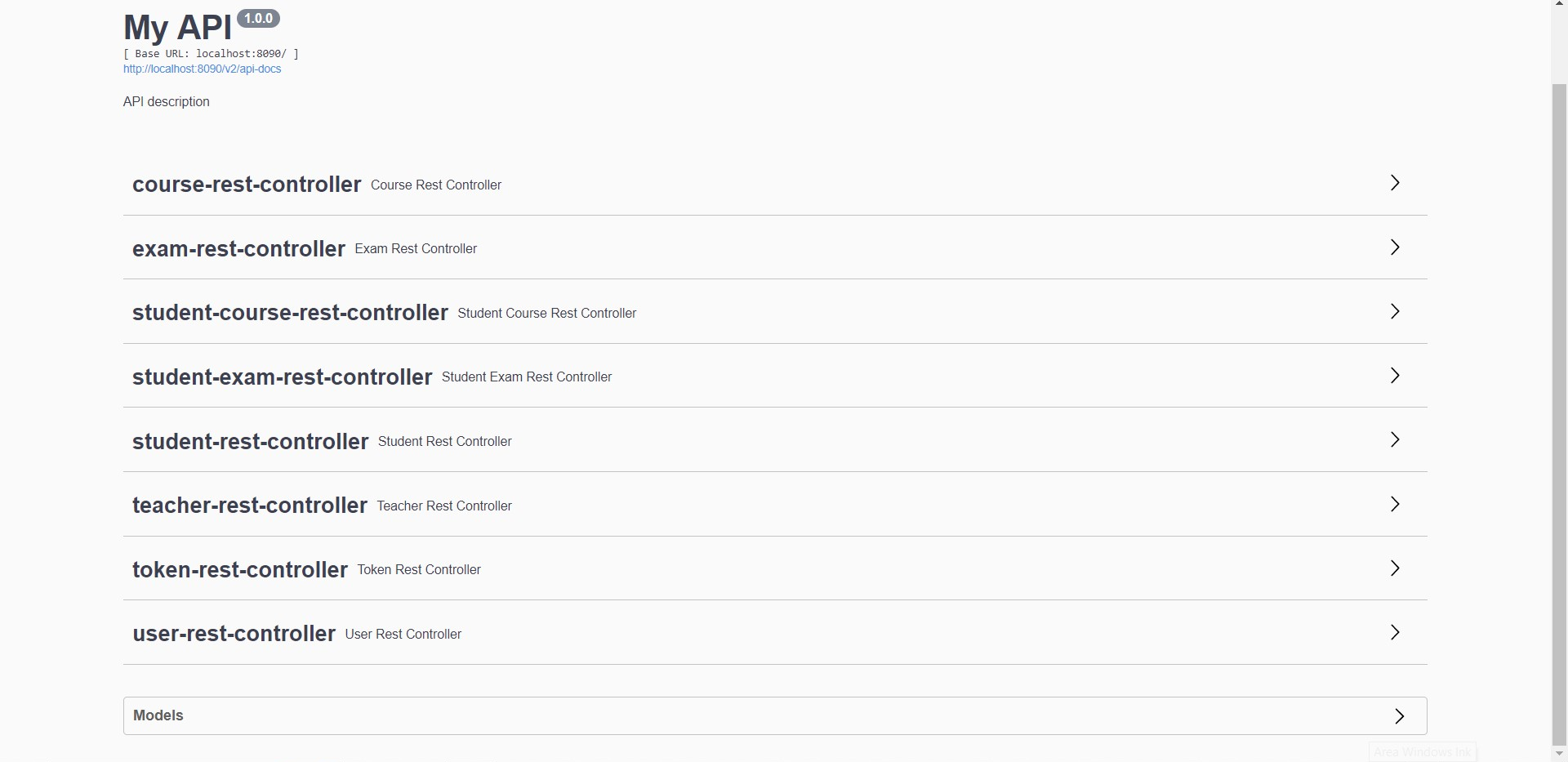
Swagger è uno strumento open-source per la documentazione di API RESTful. È stato utilizzato nel progetto per creare una documentazione interattiva delle API disponibili, in modo da rendere più facile la comprensione e l'utilizzo delle stesse da parte degli sviluppatori esterni o dei membri del team.

Per generare la documentazione, è stato utilizzato lo strumento Swagger UI, che consente di visualizzare e testare le API direttamente dal browser.

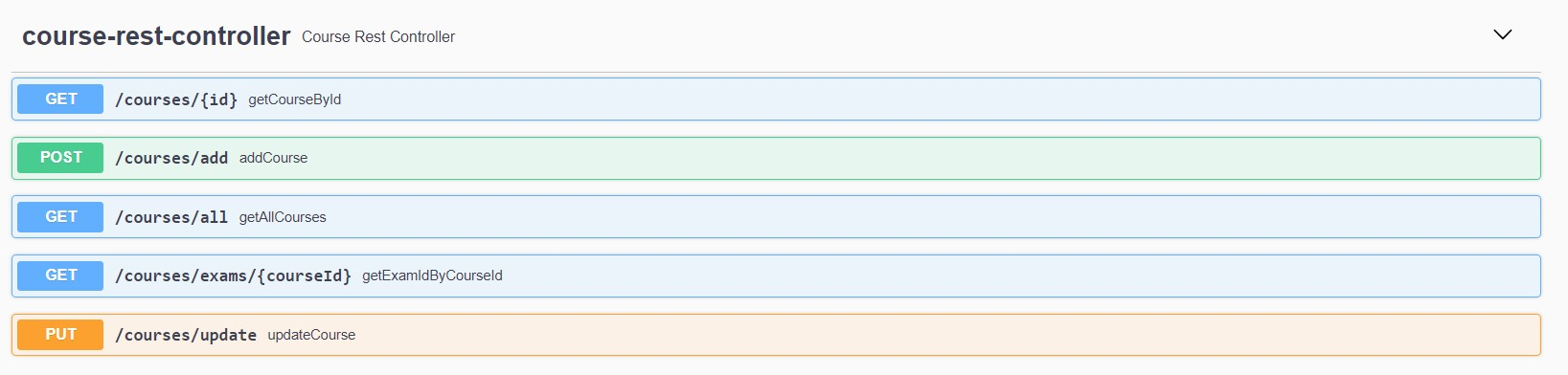
L'utilizzo di Swagger ha permesso di creare una documentazione sempre aggiornata e facilmente consultabile, senza dover scrivere manualmente la documentazione per ogni endpoint dell'API. Inoltre, Swagger UI consente di testare le API direttamente dal browser, semplificando il processo di debugging e di sviluppo.

Di seguito, verranno mostrati alcuni screenshot di Swagger UI con la documentazione delle API del progetto.

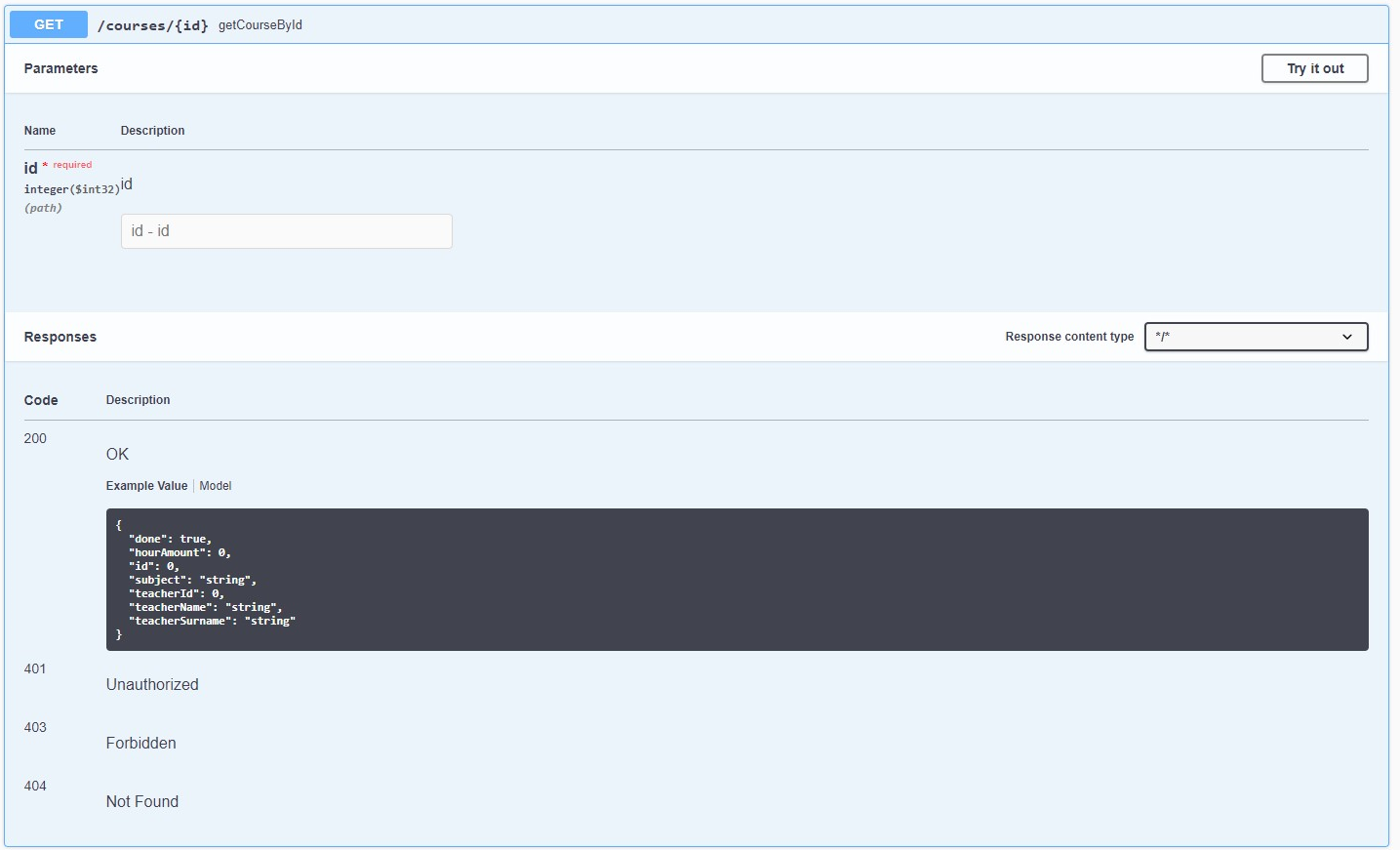
**Lista di tutti i RestController:**



**Lista delle restApi all’interno di un RestController:**

****

**Esempio di una restAPI su Swagger-UI:**

****

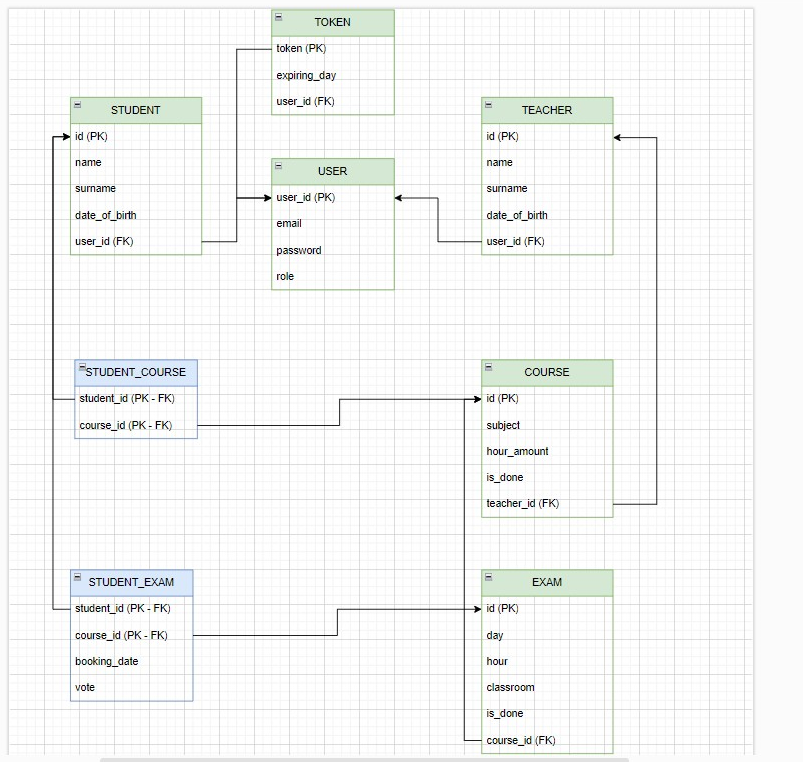
# Database

### Descrizione del database e come è stato utilizzato nel progetto

Oracle DB è un sistema di gestione di database relazionali ampiamente utilizzato nell'industria. Nel nostro progetto abbiamo scelto di utilizzare Oracle DB come nostro database principale. Abbiamo deciso di utilizzare Oracle DB perché è un sistema stabile e affidabile, che offre funzionalità avanzate come la scalabilità, la sicurezza e la disponibilità ad alta velocità. Inoltre, abbiamo utilizzato il linguaggio SQL per interagire con il database, che ci ha permesso di creare, leggere, aggiornare e cancellare i dati in modo efficace. Grazie all'uso di Oracle DB, abbiamo potuto progettare e sviluppare un sistema altamente performante e scalabile per gestire i dati del nostro progetto.

### Modello di dati utilizzato nel progetto

In questa sezione, verrà presentato il modello dei dati utilizzato nel progetto per la memorizzazione delle informazioni all'interno del database. Il modello dei dati è stato progettato per soddisfare le esigenze funzionali del sistema, consentendo la memorizzazione e l'elaborazione delle informazioni relative agli utenti, ai corsi, agli esami e alle altre entità coinvolte nel processo didattico.



### 

### Descrizione delle tabelle utilizzate nel progetto

| **Tabella** | **Descrizione** |
| --- | --- |
| User | Tabella che contiene i dati degli utenti registrati nell'applicazione |
| Token | Tabella che contiene i token utilizzati per l'autenticazione degli utenti durante il recupero password |
| Student | Tabella che contiene i dati degli studenti |
| Teacher | Tabella che contiene i dati dei docenti |
| Course | Tabella che contiene i dati dei corsi |
| StudentCourse | Tabella che contiene le associazioni tra i corsi e gli studenti iscritti |
| Exam | Tabella che contiene i dati degli esami |
| StudentExam | Tabella che contiene le associazioni tra gli esami e gli studenti iscritti |

In questo paragrafo abbiamo presentato le tabelle del database utilizzato nel progetto, ognuna delle quali svolge un ruolo specifico nell'applicazione. Grazie alla struttura del database, è possibile organizzare e gestire in modo efficace i dati degli studenti, degli insegnanti, dei corsi e degli esami. Il modello di dati è stato progettato per garantire la corretta archiviazione e gestione dei dati, rispettando le regole di integrità referenziale e le buone pratiche di progettazione del database.

### Descrizione della migrazione tra Oracle e Postgres

Per la migrazione del database dal sistema di gestione di database Oracle a Postgres, abbiamo utilizzato un software chiamato SQLIE. Questo software ci ha permesso di esportare le tabelle dal database Oracle in formato CSV e successivamente importarle nel database Postgres. Abbiamo scelto il formato CSV perché è un formato accettato sia da Oracle che da Postgres per l'importazione dei dati. Il processo di migrazione è stato fluido e senza alcun problema. Dopo la migrazione, abbiamo testato tutte le funzionalità dell'applicazione per garantire che il database funzionasse correttamente.

Documentazione delle REST API

# CourseRestController

1. GET /courses/all
   * Descrizione: restituisce una lista di tutti i corsi presenti nel sistema con l'aggiunta del nome del relativo docente
   * Request parameters: nessuno
   * Request body: nessuno
   * Response body: lista di oggetti CourseJoinTeacherDto
   * Possibili status code restituiti:
     + 200 (OK): lista restituita correttamente
     + 500 (Internal Server Error): errore interno del server
2. GET /courses/{id}
   * Descrizione: restituisce il corso identificato dall'id passato come parametro con l'aggiunta del nome del relativo docente
   * Request parameters: id del corso da cercare (intero)
   * Request body: nessuno
   * Response body: oggetto CourseJoinTeacherDto
   * Possibili status code restituiti:
     + 200 (OK): corso trovato e restituito correttamente
     + 404 (Not Found): il corso cercato non è stato trovato
     + 500 (Internal Server Error): errore interno del server
3. PUT /courses/update
   * Descrizione: questo endpoint permette di aggiornare le informazioni di un corso esistente nel database.
   * Request parameters: nessuno
   * Request body: un oggetto CourseDto contenente le informazioni del corso da aggiornare.
   * Response body: oggetto Course contenente le informazioni del corso appena aggiornato
   * Possibili status code restituiti:
     + HTTP status code 200 se l'aggiornamento è stato effettuato con successo.
     + HTTP status code 404 se il corso non esiste nel database.
     + HTTP status code 400 se i parametri passati sono invalidi.
     + HTTP status code 500 in caso di errori interni del server.
4. POST /courses/add
   * Descrizione: aggiunge un nuovo corso al sistema
   * Request parameters: nessuno
   * Request body: oggetto CourseDto contenente le informazioni del corso da aggiungere
   * Response body: oggetto Course contenente le informazioni del corso appena creato
   * Possibili status code restituiti:
     + 201 (Created): corso creato correttamente
     + 404 (Not Found): il corso cercato non è stato trovato
     + 500 (Internal Server Error): errore interno del server

# ExamRestController

1. GET /exams/all
   * Descrizione: Recupera la lista di tutti gli esami presenti nel sistema con le informazioni del corso associato.
   * Request parameters: nessuno
   * Request body: nessuno
   * Response body: lista di oggetti ExamJoinCourseDto, ognuno contenente le informazioni di un esame e del corso associato.
   * Possibili status code restituiti:
     + 200 (OK): lista degli esami recuperati con successo.
     + 500 (Internal Server Error): errore interno del server.
2. GET /exams/{id}
   * Descrizione: Recupera le informazioni di un esame con un determinato ID.
   * Request parameters:
     + id (obbligatorio): ID dell'esame.
   * Request body: nessuno
   * Response body: oggetto ExamJoinCourseDto contenente le informazioni dell'esame e del corso associato.
   * Possibili status code restituiti:
     + 200 (OK): esame recuperato con successo.
     + 404 (Not Found): esame non trovato.
     + 500 (Internal Server Error): errore interno del server.
3. POST /exams/add
   * Descrizione: Crea un nuovo esame.
   * Request parameters: nessuno
   * Request body: oggetto ExamDto contenente le informazioni del nuovo esame.
   * Response body: oggetto Exam contenente le informazioni dell'esame creato.
   * Possibili status code restituiti:
     + 201 (Created): esame creato con successo.
     + 404 (Not Found): corso non trovato.
     + 500 (Internal Server Error): errore interno del server.
4. PUT /exams/update
   * Descrizione: Aggiorna le informazioni di un esame esistente.
   * Request parameters: nessuno
   * Request body: oggetto ExamDto contenente le informazioni aggiornate dell'esame.
   * Response body: oggetto Exam contenente le informazioni dell'esame aggiornato.
   * Possibili status code restituiti:
     + 200 (OK): esame aggiornato con successo.
     + 400 (Bad Request): corso non trovato.
     + 404 (Not Found): esame non trovato.
     + 500 (Internal Server Error): errore interno del server.
5. DELETE /exams/delete/{id}
   * Descrizione: Elimina un esame con un determinato ID.
   * Request parameters:
     + id (obbligatorio): ID dell'esame da eliminare.
   * Request body: nessuno
   * Response body: nessuno
   * Possibili status code restituiti:
     + 200 (OK): esame eliminato con successo.
     + 404 (Not Found): esame non trovato.
     + 500 (Internal Server Error): errore interno del server.

# 

# StudentCourseRestController

1. POST /studentcourses/iscription
   * Descrizione: iscrive uno studente ad un corso
   * Request parameters: nessuno
   * Request body: oggetto StudentCourseDto contenente le informazioni dell’iscrizione dello studente
   * Response body: oggetto StudentCourse contenente le informazioni dell’iscrizione appena creata
   * Possibili status code restituiti:
     + 201 (Created): iscrizione creata correttamente
     + 409 (Conflict): l'iscrizione è già presente
2. GET /studentcourses/{id}/students
   * Descrizione: recupera la lista di tutti gli studenti iscritti ad un corso
   * Request parameters:
     + id: id del corso
   * Request body: nessuno
   * Response body: lista di oggetti StudentDto contenente le informazioni degli studenti iscritti al corso
   * Possibili status code restituiti:
     + 200 (OK): lista di studenti restituita correttamente
     + 404 (Not Found): il corso cercato non è stato trovato
3. GET /studentcourses/{id}/courses
   * Descrizione: recupera la lista di tutti i corsi a cui uno studente è iscritto
   * Request parameters:
     + id: id dello studente
   * Request body: nessuno
   * Response body: lista di oggetti CourseJoinTeacherDto contenente le informazioni dei corsi a cui lo studente è iscritto
   * Possibili status code restituiti:
     + 200 (OK): lista di corsi restituita correttamente
     + 404 (Not Found): lo studente cercato non è stato trovato

# StudentExamRestController

1. POST /studentexams/booking
   * Descrizione: prenota un esame per uno studente
   * Request parameters: nessuno
   * Request body: oggetto StudentExamDto contenente le informazioni dell'esame da prenotare
   * Response body: oggetto StudentExam contenente le informazioni dell'esame appena prenotato
   * Possibili status code restituiti:
     + 201 (Created): esame prenotato correttamente
     + 404 (Not Found): lo studente cercato non è stato trovato
     + 400 (Bad Request): l'esame cercato non è stato trovato
     + 409 (Conflict): l'esame è già stato prenotato dallo studente
     + 500 (Internal Server Error): errore interno del server
2. PUT /studentexams/updatevote
   * Descrizione: aggiorna il voto di uno studente in un esame
   * Request parameters: nessuno
   * Request body: oggetto StudentExamDto contenente le informazioni dell'esame e il voto da aggiornare
   * Response body: oggetto StudentExam contenente le informazioni dell'esame appena aggiornato
   * Possibili status code restituiti:
     + 200 (OK): voto aggiornato correttamente
     + 404 (Not Found): lo studente o l'esame cercato non sono stati trovati
     + 400 (Bad Request): l'esame cercato non è stato trovato
     + 409 (Conflict): lo studente non ha ancora prenotato l'esame
     + 500 (Internal Server Error): errore interno del server
3. GET /studentexams/{id}/examstodo
   * Descrizione: ottiene la lista degli esami da sostenere per uno studente
   * Request parameters: id dello studente
   * Request body: nessuno
   * Response body: lista di oggetti ExamJoinCourseDto contenenti le informazioni degli esami da sostenere per lo studente
   * Possibili status code restituiti:
     + 200 (OK): lista di esami restituita correttamente
     + 404 (Not Found): lo studente cercato non è stato trovato
     + 500 (Internal Server Error): errore interno del server
4. GET /studentexams/{id}/examsdone
   * Descrizione: ottiene la lista degli esami sostenuti da uno studente
   * Request parameters: id dello studente
   * Request body: nessuno
   * Response body: lista di oggetti ExamJoinCourseDto contenenti le informazioni degli esami sostenuti dallo studente
   * Possibili status code restituiti:
     + 200 (OK): lista di esami restituita correttamente
     + 404 (Not Found): lo studente cercato non è stato trovato
     + 500 (Internal Server Error): errore interno del server
5. GET /exams/{id}/students
   * Descrizione: recupera la lista degli studenti registrati a un esame
   * Request parameters: id dell'esame del quale recuperare la lista degli studenti registrati
   * Request body: nessuno
   * Response body: lista di oggetti StudentDto, ognuno contenente le informazioni di uno studente registrato all'esame
   * Possibili status code restituiti:
     + 200 (OK): lista degli studenti registrati all'esame recuperata correttamente
     + 404 (Not Found): l'esame richiesto non è stato trovato
     + 500 (Internal Server Error): errore interno del server
6. GET /studentexams/{id}/all
   * Descrizione: restituisce la lista di tutti gli esami effettuati dagli studenti per un dato esame, comprensiva di voto assegnato e data di prenotazione.
   * Request parameters: id dell'esame per cui si vuole recuperare la lista degli esami effettuati.
   * Request body: nessuno.
   * Response body: una lista di oggetti StudentExamDto
   * Possibili status code restituiti:
     + 200 (OK): la lista degli esami effettuati dagli studenti è stata restituita correttamente.
     + 404 (Not Found): l'esame per cui si vuole recuperare la lista degli esami effettuati non esiste.
     + 500 (Internal Server Error): errore interno del server.

# StudentRestController

1. GET /students/all
   * Descrizione: recupera la lista di tutti gli studenti presenti nel sistema
   * Request parameters: nessuno
   * Request body: nessuno
   * Response body: lista di oggetti StudentDto contenenti le informazioni degli studenti
   * Possibili status code restituiti:
     + 200 (OK): la lista degli studenti è stata recuperata correttamente
     + 500 (Internal Server Error): errore interno del server durante la ricerca degli studenti
2. GET /students/{id}
   * Descrizione: recupera le informazioni di uno studente specifico in base al suo ID
   * Request parameters: id (intero) corrispondente all'ID dello studente da recuperare
   * Request body: nessuno
   * Response body: oggetto StudentDto contenente le informazioni dello studente cercato
   * Possibili status code restituiti:
     + 200 (OK): le informazioni dello studente sono state recuperate correttamente
     + 404 (Not Found): lo studente cercato non è stato trovato
     + 500 (Internal Server Error): errore interno del server durante la ricerca dello studente
3. POST /students/add
   * Descrizione: aggiunge un nuovo studente al sistema
   * Request parameters: nessuno
   * Request body: oggetto StudentDto contenente le informazioni dello studente da aggiungere
   * Response body: oggetto Student contenente le informazioni dello studente appena creato
   * Possibili status code restituiti:
     + 201 (Created): lo studente è stato creato correttamente
     + 401 (Unauthorized): non è stato possibile creare lo studente a causa di dati non validi
     + 500 (Internal Server Error): errore interno del server durante la creazione dello studente
4. PUT /students/update
   * Descrizione: aggiorna le informazioni di uno studente esistente nel sistema
   * Request parameters: nessuno
   * Request body: oggetto StudentDto contenente le informazioni aggiornate dello studente
   * Response body: oggetto Student contenente le informazioni dello studente aggiornato
   * Possibili status code restituiti:
     + 200 (OK): le informazioni dello studente sono state aggiornate correttamente
     + 404 (Not Found): lo studente cercato non è stato trovato
     + 409 (Conflict): non è stato possibile aggiornare le informazioni dello studente a causa di un conflitto con altri dati nel sistema
     + 500 (Internal Server Error): errore interno del server durante l'aggiornamento delle informazioni dello studente
5. DELETE /students/delete/{id}
   * Descrizione: elimina uno studente dal sistema in base al suo ID
   * Request parameters: id (intero) corrispondente all'ID dello studente da eliminare
   * Request body: nessuno
   * Response body: nessuno
   * Possibili status code restituiti:
     + 200 (OK): lo studente è stato eliminato correttamente
     + 404 (Not Found): lo studente cercato non è stato trovato
     + 500 (Internal Server Error): errore interno del server

# TeacherRestController

1. GET /teachers/all
   * Descrizione: recupera la lista di tutti i docenti registrati nel sistema.
   * Request parameters: nessuno.
   * Request body: nessuno.
   * Response body: lista di oggetti TeacherDto, ognuno contenente le informazioni di un docente.
   * Possibili status code restituiti:
     + 200 (OK): lista dei docenti recuperata correttamente.
     + 500 (Internal Server Error): errore interno del server.
2. GET /teachers/{id}
   * Descrizione: recupera le informazioni di un docente specificato dal suo ID.
   * Request parameters: ID del docente.
   * Request body: nessuno.
   * Response body: oggetto TeacherDto contenente le informazioni del docente.
   * Possibili status code restituiti:
     + 200 (OK): informazioni del docente recuperate correttamente.
     + 404 (Not Found): docente non trovato.
     + 500 (Internal Server Error): errore interno del server.
3. GET /teachers/{id}/courses
   * Descrizione: recupera la lista dei corsi insegnati da un docente specificato dal suo ID.
   * Request parameters: ID del docente.
   * Request body: nessuno.
   * Response body: lista di oggetti CourseDto, ognuno contenente le informazioni di un corso.
   * Possibili status code restituiti:
     + 200 (OK): lista dei corsi recuperata correttamente.
     + 404 (Not Found): docente non trovato.
     + 500 (Internal Server Error): errore interno del server.
4. GET /teachers/{id}/exams
   * Descrizione: recupera la lista degli esami corretti da un docente specificato dal suo ID.
   * Request parameters: ID del docente.
   * Request body: nessuno.
   * Response body: lista di oggetti ExamJoinCourseDto, ognuno contenente le informazioni di un esame e del corso a cui appartiene.
   * Possibili status code restituiti:
     + 200 (OK): lista degli esami recuperata correttamente.
     + 404 (Not Found): docente non trovato.
     + 500 (Internal Server Error): errore interno del server.
5. POST /teachers/add
   * Descrizione: aggiunge un nuovo docente al sistema.
   * Request parameters: nessuno.
   * Request body: oggetto TeacherDto contenente le informazioni del nuovo docente.
   * Response body: oggetto Teacher contenente le informazioni del docente appena creato.
   * Possibili status code restituiti:
     + 201 (Created): docente creato correttamente.
     + 400 (Bad Request): dati del docente non validi.
     + 500 (Internal Server Error): errore interno del server.
6. PUT /teachers/update
   * Descrizione: permette di aggiornare le informazioni di un insegnante nel sistema
   * Request parameters: nessuno
   * Request body: oggetto TeacherDto contenente le nuove informazioni dell'insegnante
   * Response body: oggetto Teacher contenente le informazioni aggiornate dell'insegnante
   * Possibili status code restituiti:
     + 200 (OK): insegnante aggiornato correttamente
     + 400 (Bad Request): richiesta non valida (ad esempio, mancano dei parametri obbligatori)
     + 404 (Not Found): insegnante non trovato
     + 409 (Conflict): l'aggiornamento dell'insegnante non è andato a buon fine a causa di un conflitto con un altro insegnante (ad esempio, l'email è già utilizzata da un altro insegnante)
     + 500 (Internal Server Error): errore interno del server
7. DELETE /teachers/delete/{id}
   * Descrizione: permette di eliminare un insegnante dal sistema
   * Request parameters: id (intero) - l'ID dell'insegnante da eliminare
   * Request body: nessuno
   * Response body: nessuno
   * Possibili status code restituiti:
     + 200 (OK): insegnante eliminato correttamente
     + 404 (Not Found): insegnante non trovato
     + 500 (Internal Server Error): errore interno del server

# TokenRestController

1. POST /token/verify-token
   * Descrizione: verifica la validità di un token di accesso
   * Request parameters: nessuno
   * Request body: stringa contenente il token di accesso da verificare
   * Response body: una stringa contenente l'ID utente associato al token di accesso
   * Possibili status code restituiti:
     + 200 (OK): token di accesso verificato correttamente, con ID utente restituito come corpo della risposta
     + 404 (Not Found): token di accesso non trovato
     + 401 (Unauthorized): token di accesso scaduto o non valido
     + 500 (Internal Server Error): errore interno del server
2. PUT /token/change-password
   * Descrizione: cambia la password dell'utente corrispondente al token di accesso fornito
   * Request parameters: nessuno
   * Request body: oggetto UserDto contenente l'ID dell'utente e la nuova password
   * Response body: nessuno
   * Possibili status code restituiti:
     + 200 (OK): password utente aggiornata correttamente
     + 500 (Internal Server Error): errore interno del server

# UserRestController

1. GET /users/update
   * Descrizione: questo endpoint permette di aggiornare le informazioni di un utente esistente.
   * Request body: un oggetto UserDto contenente le informazioni dell'utente da aggiornare.
   * Possibili status code restituiti:
     + HTTP status code 200 se l'aggiornamento è stato effettuato con successo.
     + HTTP status code 400 se i parametri passati sono invalidi.
2. POST /users/add
   * Descrizione: questo endpoint permette di aggiungere un nuovo utente.
   * Request body: un oggetto UserDto contenente le informazioni del nuovo utente.
   * Possibili status code restituiti:
     + HTTP status code 200 se l'utente è stato creato con successo.
3. POST /users/login
   * Descrizione: questo endpoint permette di effettuare il login di un utente e ottenere un token JWT.
   * Request body: un oggetto UserDto contenente le credenziali dell'utente.
   * Possibili status code restituiti:
     + HTTP status code 200 se il login è stato effettuato con successo, e il body della risposta contiene il token JWT.
     + HTTP status code 401 se le credenziali fornite sono errate.
     + HTTP status code 404 se l'utente non esiste.
     + HTTP status code 500 in caso di errori interni del server.
4. DELETE /users/delete/{id}
   * Descrizione: questo endpoint permette di eliminare un utente esistente.
   * Request parameters: l'ID dell'utente da eliminare.
   * Possibili status code restituiti:
     + HTTP status code 200 se l'utente è stato eliminato con successo.
     + HTTP status code 500 in caso di errori interni del server.
5. POST /users/recupera-password
   * Descrizione: questo endpoint permette di inviare una email contenente la password dell'utente.
   * Request body: una stringa contenente l'ID dell'utente.
   * Possibili status code restituiti:
     + HTTP status code 200 se l'email è stata inviata con successo.
     + HTTP status code 404 se l'utente non esiste.
     + HTTP status code 500 in caso di errori interni del server.
6. PUT/users/change-userid
   * Descrizione: questo endpoint permette di modificare l’ID dell’utente.
   * Request body: un oggetto ChangeUserIdDto contenente il vecchio ID dell’utente e quello nuovo.
   * Possibili status code restituiti:
     + HTTP status code 200 se l’ID è stato modificato con successo.
     + HTTP status code 404 se l'utente non esiste.
     + HTTP status code 401 se il nuovo ID scelto dall’utente è già presente nel database.
     + HTTP status code 500 in caso di errori interni del server.