Dipartimento di Ingegneria e Scienza dell'Informazione

CORSO DI INGEGNERIA DEL SOFTWARE

Ypsilon 2

your own stats

Documento di progetto: Sviluppo dell'applicazione

Gruppo: T105

INDICE

1. Scopo del documento

Questo documento contiene tutte le informazioni necessarie per sviluppare parte di un'applicazione Ypsilon 2. In particolare, mostra tutti gli artefatti necessari per implementare i servizi di gestione dei dipendenti e dei reparti dell'applicazione Ypsilon 2.

Partendo da una descrizione del flusso utente relativo al ruolo di Amministratore dell'Applicazione (di seguito RA), la documentazione prosegue introducendo le API necessarie (tramite il Modello API e il Modello Risorsa) per poter visualizzare, inserire e modificare i dipartimenti tenuti ad applicare Ypsilon 2 e il personale.

Per ogni API creata, questo documento fornisce la relativa documentazione e i test eseguiti, oltre a una descrizione delle funzionalità fornite. Infine, è presente una sezione dedicata alle informazioni sul repository Git e sul deployment dell'applicazione stessa.

2. User Flows

In questa sezione del documento di sviluppo riportiamo gli “user flows” per il ruolo dell’utente studente.

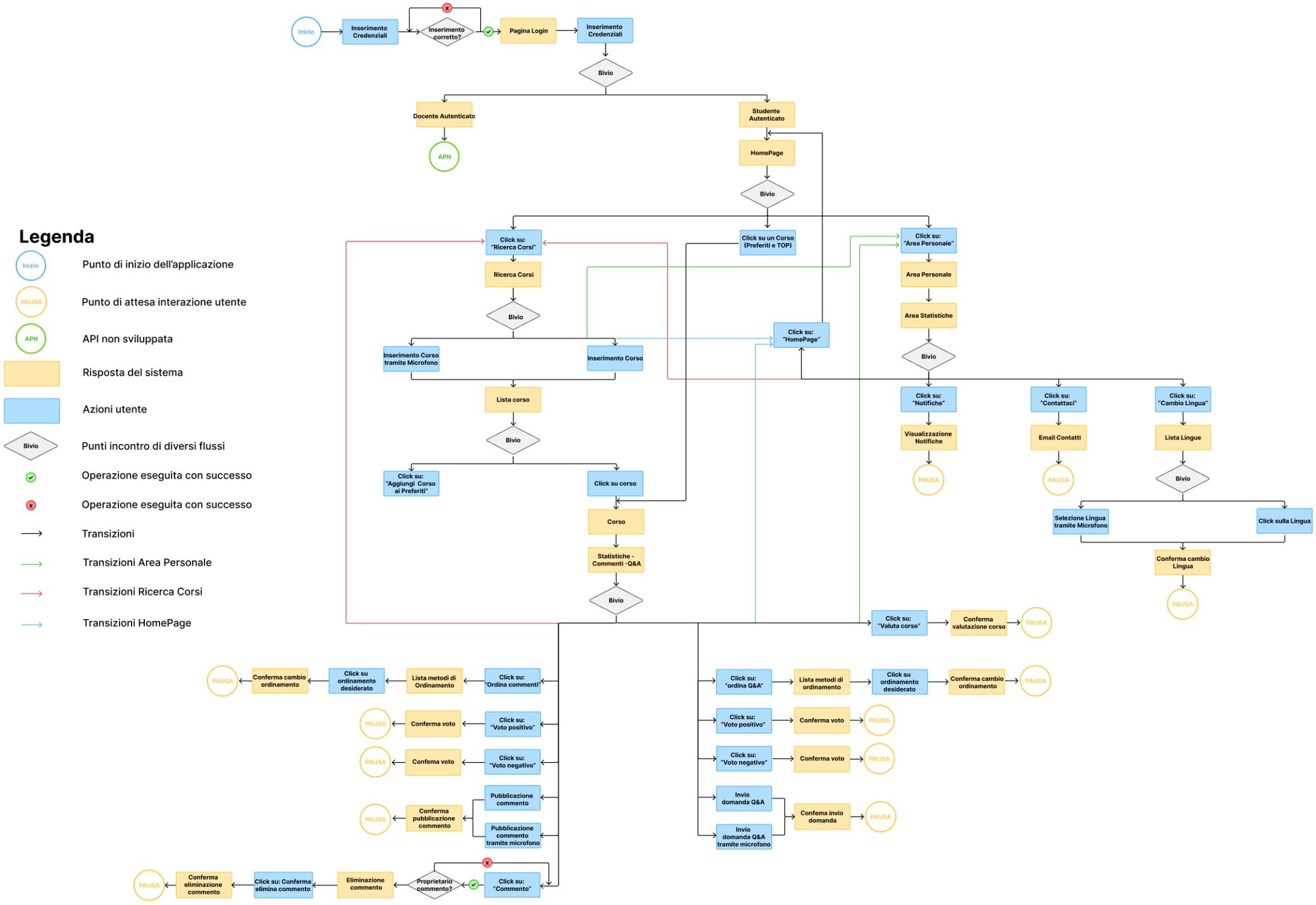
L’immagine che segue descrive lo user flow relativo alla gestione delle informazioni relative alle funzionalità che riguardano l’utente studente, nello specifico:

- le possibili funzionalità presenti all’interno della homepage
- le funzionalità riguardanti la ricerca dei corsi
- le possibili modifiche delle impostazioni del profilo personale e le funzionalità presenti nella pagina relativa
- la visualizzazione di un corso con le funzionalità presenti nella pagina ad esso dedicata, tra cui commenti e Q&A.

Attraverso questo diagramma vengono presentate inoltre le relazioni esistenti tra le varie azioni che l’utente studente può svolgere. Inoltre è presente una breve legenda che descrive i simboli utilizzati.

2.1 Diagramma User Flow

Ypsilon 2
your own stats



3. Application, Implementation and Documents

Nella sezione precedente vengono identificate le varie fetures che devono essere implementate per la nostra web app. Essa è stata sviluppata utilizzando NodeJS e VueJS per quanto riguarda il front-end, mentre per il back-end e le API è viene utilizzato JavaScript. Per la gestione dei dati viene utilizzato MongoDB.

3.1. Project Structure

Nella figura sottostante viene visualizzata la struttura del progetto, composto dai seguenti file e cartelle:

test: La cartella "Tests" è una cartella che contiene i file di test utilizzati per verificare il corretto funzionamento delle varie parti di un'applicazione o di un progetto.

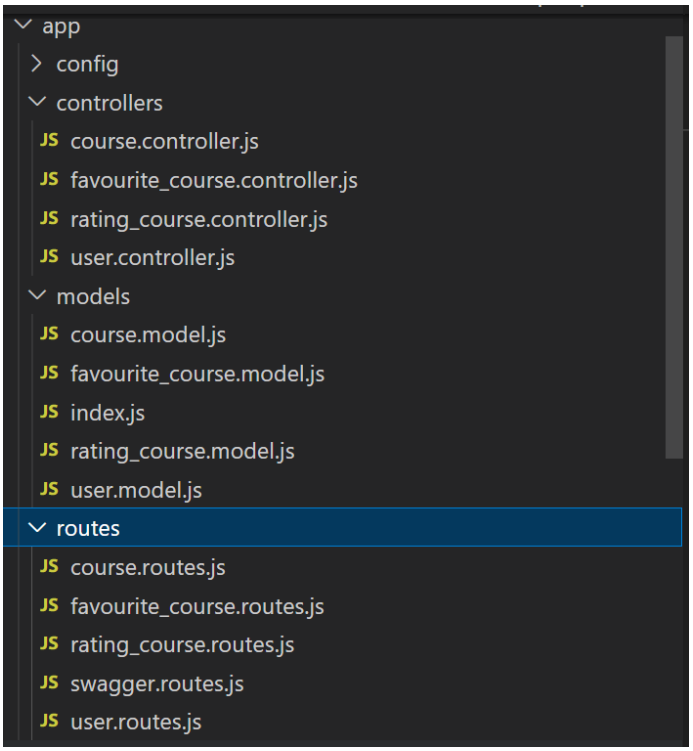
In particolare, sono presenti i file di test che sono stati utilizzati per verificare il corretto funzionamento delle API utilizzate dalla web app

Nella cartella ypsilon2 è presente:

- **Node_modules**: La cartella "Node_modules" è una cartella generata automaticamente quando viene eseguito il comando "npm init" nella directory del progetto. Questa cartella contiene tutte le dipendenze del progetto, ovvero tutti i moduli di software esterni utilizzati dal progetto.
- **Assets:** è la cartella in cui sono presenti le immagini e i file .vue secondari, che vengono successivamente utilizzati e implementati nell'app principale.
- **Components:** è la cartella in cui sono presenti i file .vue principali, in questo caso ogni file .vue rappresenta una diversa page della web-app
- **Router:** è la cartella che ha all'interno il file che gestisce le route dei file ".vue".

La parte Backend è divisa in:

- **Models**: è la cartella che contiene gli schemi di tutti i modelli che sono stati definiti per essere utilizzati con il database di MongoDB. Gli schemi dei modelli vengono creati utilizzando il framework Mongoose, che fornisce un modo per definire la struttura dei dati che verranno salvati nel database. Una volta che gli schemi dei modelli sono stati definiti, possono essere utilizzati per creare, leggere, aggiornare e eliminare i dati nel database di MongoDB.
- **Routes**: è la cartella che contiene i file che definiscono le rotte per le varie API del progetto.
- **Controllers:** è la cartella che contiene i file che definiscono le API e i metodi utilizzati per gestire le richieste HTTP e inviare le risposte appropriate.



3.2. Project Dependencies

Di seguito viene riportata la lista dei vari moduli utilizzati e aggiunti al Package.json

```
"dependencies": {
  "axios": "^1.2.1",
  "core-js": "^3.8.3",
  "crypto-browserify": "^3.12.0",
  "fs-extra": "^11.1.0",
  "net": "^1.0.2",
  "postman-request": "^2.88.1-postman.31",
  "tough-cookie": "^4.1.2",
  "url": "^0.11.0",
  "util": "^0.12.5",
  "view-router": "^0.2.3",
  "vue": "^3.2.45",
  "vue-router": "^4.1.6"
},
"devDependencies": {
  "@babel/core": "^7.12.16",
  "@babel/eslint-parser": "^7.12.16",
  "@vue/cli-plugin-babel": "~5.0.0",
  "@vue/cli-plugin-eslint": "~5.0.0",
  "@vue/cli-service": "~5.0.0",
  "eslint": "^7.32.0",
  "eslint-plugin-vue": "^8.0.3"
},
```

3.3. Project Data or DB

Per la gestione dei dati utili alla web app sono stati definite varie strutture dati che presentano diversi attributi. Di seguito vengono riportate le immagini di ciò che contengono i file JavaScript dedicati alle strutture dati.

3.3.1. User

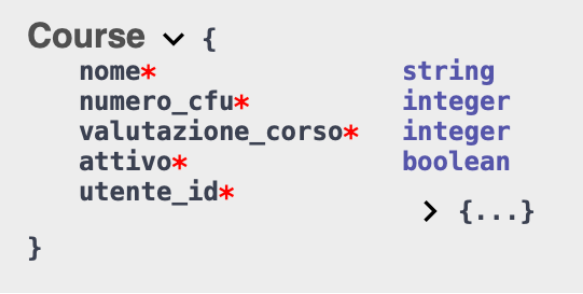
Rappresenta l'Utente Studente dopo aver effettuato l'accesso tramite web app. I dati che vengono salvati per ogni elemento utente sono: ***nome, cognome.***

```
User {
    nome*                   string
    cognome*                string
}
```

```
> models > JS user.model.js > ...
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

module.exports = mongoose => {
  var schema = mongoose.Schema(
    {
      nome: String,
      cognome: String
    },
    { timestamps: true }
  );

  schema.method("toJSON", function() {
    const { __v, _id, ...object } = this.toObject();
    object.id = _id;
    return object;
  });

  const User = mongoose.model("User", schema);
  return User;
};
```

3.3.2. Corso

Rappresenta il corso, che verrà automaticamente generato una volta che l'utente docente effettuerà il login utilizzando le credenziali universitarie. I dati che vengono salvati per ogni elemento sono: ***nome, numero_cfu, valutazione_corso, attivo, utente_id.***



```
var mongoosee = require('mongoose');
var Schemaa = mongoosee.Schema;

module.exports = mongoose => {
  var schema = mongoose.Schema(
    {
      nome: String,
      numero_cfu: Number,
      valutazione_corso: Number,
      attivo: Number,
      utente_id: {type: Schemaa.Types.ObjectId, ref: 'User'}
    },
    { timestamps: true }
  );

  schema.method("toJSON", function() {
    const { __v, _id, ...object } = this.toObject();
    object.id = _id;
    return object;
  });

  const Course = mongoose.model("Course", schema);
  return Course;
};
```

3.3.3. Corso Preferito

Rappresenta il corso preferito, generato dopo aver inserito un corso nella lista dei corsi preferiti. I dati che vengono salvati per ogni elemento sono: ***utente_id, corso_id.***

```
Favourite_course ⌄ {
    utente_id*          string
    corso_id*           string
}
```

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
module.exports = mongoose => {
  var schema = mongoose.Schema(
    {
      utente_id: {type: Schema.Types.ObjectId, ref: 'user'},
      corso_id: {type: Schema.Types.ObjectId, ref: 'course'}
    },
    { timestamps: true }
  );

  schema.method("toJSON", function() {
    const { __v, _id, ...object } = this.toObject();
    object.id = _id;
    return object;
  });

  const User = mongoose.model("favourite_course", schema);
  return User;
};
```

3.3.4. Rating Corso

Rappresenta il rating che viene dato ad un corso, generato dopo che un utente da una valutazione ad un corso. I dati che vengono salvati per ogni elemento sono: ***utente_id, corso_id, valutazione.***

```
Rating_course {
    utente_id*          string
    corso_id*           string
    valutazione*        integer
}
```

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
module.exports = mongoose => {
  var schema = mongoose.Schema(
    {
      utente_id: {type: Schema.Types.ObjectId, ref: 'user'},
      corso_id: {type: Schema.Types.ObjectId, ref: 'course'},
      valutazione: Number,
    },
    { timestamps: true }
  );

  schema.method("toJSON", function() {
    const { __v, _id, ...object } = this.toObject();
    object.id = _id;
    return object;
  });

  const User = mongoose.model("rating_course", schema);
  return User;
};
```

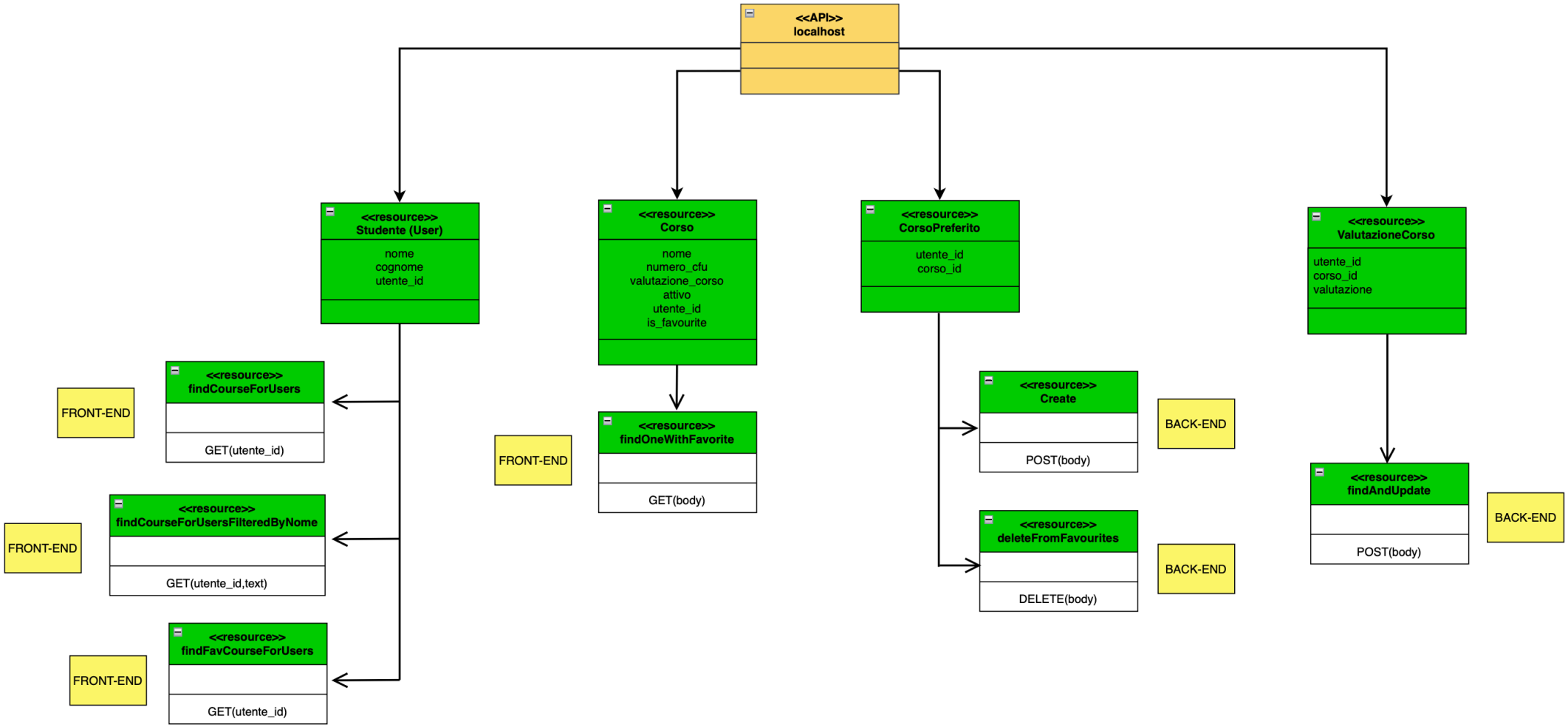
3.4. Project API

3.4.1. Resources extraction from the class diagram

Di seguito viene mostrato l'UML API Diagram, quindi una rappresentazione di tutte le risorse che vengono gestite a livello di API con le relative relazioni presenti nel progetto. Viene quindi definito come le componenti del sistema sono dipendenti e dipendono l'una dall'altra e le varie interazioni presenti tra esse.
Il seguente diagramma viene creato seguendo il diagramma delle classi (presente all'interno del documento D3), estraendo le risorse delle classi, dei metodi e i relativi parametri presi in input, la specifica del tipo di funzione (GET, POST, DELETE).
In seguito viene descritto se la risorsa è utilizzata per il front-end oppure per il back-end. Da sottolineare che le API che vengono mostrate nel diagramma sono quelle che sono state implementate ed utilizzate

In particolare:

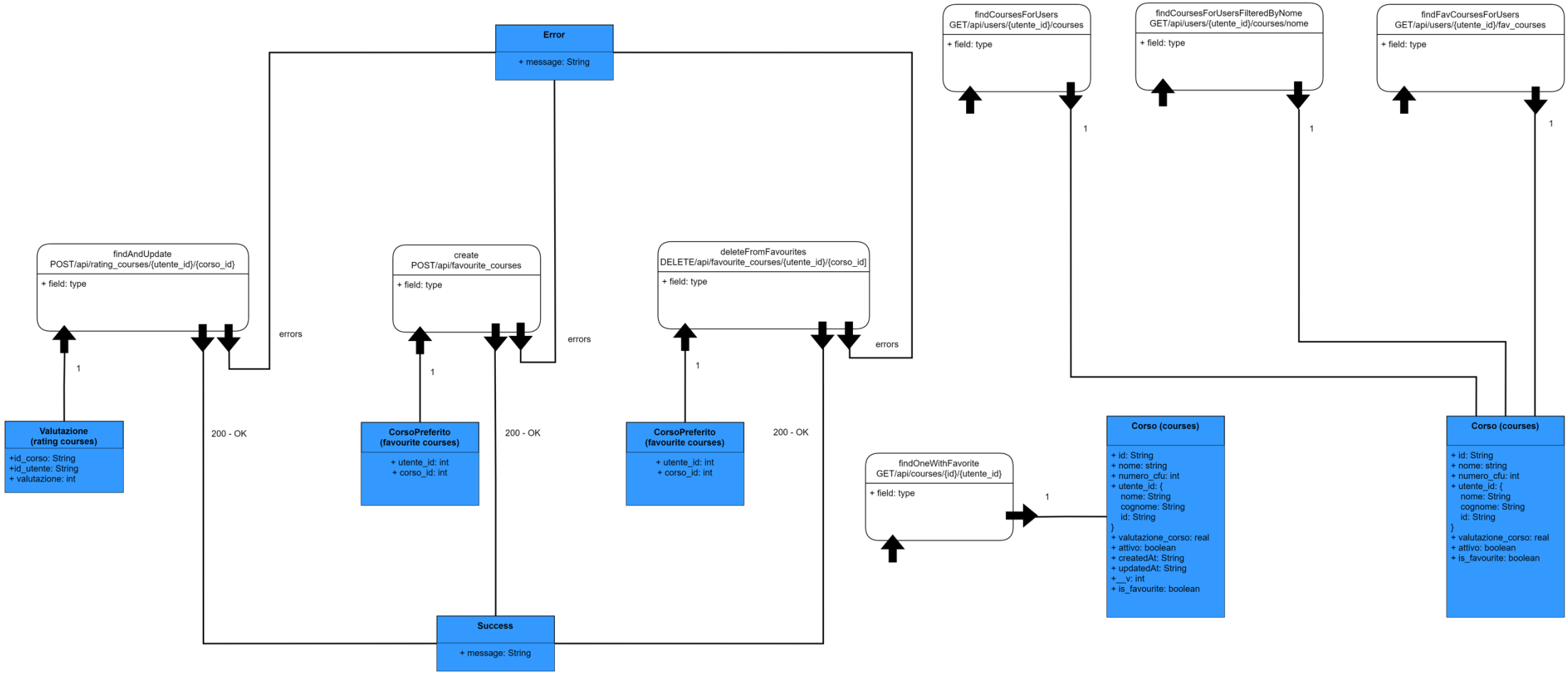
- Il root è L'URL del sito;
- Le risorse di primo livello vengono ricavate dalle classi con gli attributi presenti.
- Le risorse di secondo livello vengono individuate partendo dai metodi presenti all'interno delle classi, nelle quali vengono identificate le funzioni necessarie per lo sviluppo delle API.


Ypsilon 2
your own stats
<<API>>
localhost
<<resource>>
Studente (User)
nome
cognome
utente_id
<<resource>>
Corso
nome
numero_cfu
valutazione_corso
attivo
utente_id
is_favourite
<<resource>>
CorsoPreferito
utente_id
corso_id
<<resource>>
ValutazioneCorso
utente_id
corso_id
valutazione
<<resource>>
findCourseForUsers
GET(utente_id)
FRONT-END
<<resource>>
findCourseForUsersFilteredByNome
GET(utente_id,text)
FRONT-END
<<resource>>
findFavCourseForUsers
GET(utente_id)
FRONT-END
<<resource>>
findOneWithFavorite
GET(body)
FRONT-END
<<resource>>
Create
POST(body)
BACK-END
<<resource>>
deleteFromFavourites
DELETE(body)
BACK-END
<<resource>>
findAndUpdate
POST(body)
BACK-END

3.4.2. Resources Model

Di seguito viene mostrato il Resources Model (diagramma delle API), esso descrive graficamente le API che vengono utilizzate per il funzionamento della web app. In particolare per ogni risorsa vengono definite il nome, il metodo con cui vi è possibile accedersi e il path URL necessario. Ognuna di queste API elabora i dati che gli vengono passati, ed inviare una risposta alla richiesta effettuata.
Da sottolineare che le API che vengono mostrate nel diagramma sono quelle che sono state implementate ed utilizzate

Ypsilon 2
your own stats



4. Sviluppo API

Di seguito viene presentata la lista delle API che sono state sviluppate per questo progetto (anche quelle che non vengono utilizzate nell'effettivo) con annessa una breve descrizione. Nella sezione successiva (5. API Documentation) verrà mostrata una descrizione più precisa delle API che vengono utilizzate.

4.1 Users

- ***GET APIs:***

  findAll: permette di recuperare tutti gli utenti presenti nel database e li invia come risposta alla richiesta.

  findCoursesForUser: permette di recuperare tutti i corsi associati ad un determinato utente, identificato tramite il suo ID (utente_id), e li invia come risposta alla richiesta;

  findCousesForUserFilteredByNome: permette di recuperare tutti i corsi associati ad un determinato utente, identificato tramite il suo ID (utente_id), che soddisfano il filtro di ricerca (tramite il nome del corso), e li invia come risposta alla richiesta;

  findFavCoursesForUser: permette di recuperare tutti i corsi associati ad un determinato utente, identificato tramite il suo ID (utente_id), che vengono segnalati come preferiti e li invia come risposta alla richiesta.

- ***POST APIs:***

  create: permette la creazione di un nuovo oggetto 'user' (quindi la creazione di un nuovo utente) con i dati presenti nel corpo della richiesta e lo salva nel database;

4.2. Courses

- **GET APIs:**

  findAll: permette di recuperare tutti i corsi presenti nel database e inviarli come risposta alla richiesta

  findAllPublished: permette di recuperare tutti i corsi presenti nel database che siano ancora 'attivi' (con il campo 'attivo' settato a 'true'), e inviarli come risposta alla richiesta. In particolare serve per recuperare tutti i corsi che vengono ancora insegnati (per non lasciare nel sito i corsi che non vengono più svolti).

findOne: permette di recuperare un determinato corso, identificato tramite il sui ID e inviarlo come risposta alla richiesta.

findOneWithFavourite: permette di recuperare un determinato corso, identificato tramite il sui ID e verificare se è stato segnalato come preferito ad un determinato udente identificato anch'esso tramite il suo ID, e infine inviare il corso recuperato e le informazioni di segnalazione di preferenza come risposta alla richiesta.

- **POST APIs:**

create: permette la creazione di un nuovo oggetto 'course' (quindi la creazione di un nuovo corso) con i dati presenti nel corpo della richiesta e lo salva nel database.

- **DELETE APIs:**

delete: permette di eliminare un determinato corso, identificato tramite il suo ID, dal database.

deleteAll: permette di eliminare tutti i corsi, presenti nel database, dal database

- **PUT APIs:**

update: permette di aggiornare un determinato corso, identificato tramite il suo ID, con i dati presenti del corpo della richiesta.

4.3. Favourite_Courses

- **POST APIs:**

create: permette la creazione di un nuovo oggetto 'Favourite_corse' con l'ID relativo all'utente e l'ID relativo al corso specificati nel corpo della richiesta e lo salva nel database, a meno che non esista già un'associazione preferita per questo utente e corso. Più semplicemente salva nel database un determinato corso che viene aggiunto ai corsi preferiti da un determinato utente.

- **DELETE APIs:**

deleteFromFavourites: permette di eliminare l'associazione tra un corso con un determinato ID ed un utente con un determinato ID dal database. Più semplicemente viene usata per togliere un determinato corso dai corsi preferiti di un determinato utente.

4.4. Rating_Courses

- **POST APIs:**

  create: permette la creazione un nuovo oggetto ‘Rating_course’ con l’ID relativo ad un utente e l’ID relativo ad un corso specificati nel corpo della richiesta. Più semplicemente gestisce la creazione di una valutazione per un corso. Prima di creare la nuova valutazione, verifica che non esista già una valutazione data da quel determinato utente a quel determinato corso. In caso contrario salva la nuova valutazione nel database ed invia una risposta di successo al client.

  findAndUpdate: permette di gestire l’update di una valutazione data da un determinato utente, identificato tramite il suo ID, ad un determinato corso, identificato tramite il sui ID. In particolare trova la valutazione nel database e lo aggiorna con il nuovo voto, se il voto non viene trovato ne crea uno nuovo e lo salva nel database.

- **DELETE APIs:**

  delete: permette di eliminare la valutazione di un determinato corso, identificato tramite il suo ID, dato da un determinato utente, identificato tramite il suo ID.

5. API Documentation

Le API locali fornite dalla web app Ypsilon2 descritte nella sezione precedente sono state documentate utilizzando il modulo NodeJs chiamato Swagger UI Express. In questo modo la documentazione relativa alle API sviluppate è direttamente disponibile a chiunque possieda il codice sorgente.
Per poter generare l'endpoint dedicato alla presentazione delle API viene utilizzato Swagger UI, attraverso il quale è possibile generare una pagina web dalle definizioni delle specifiche OpenAPI.

Di seguito viene mostrata la pagina web dedicata alla documentazione delle API sviluppate ed utilizzate per la realizzazione. Nello sviluppo delle API vengono usate le funzioni GET, per recuperare dati dal database, POST, per creare una nuova risorsa sul database, DELETE, per eliminare una risorsa dal database.

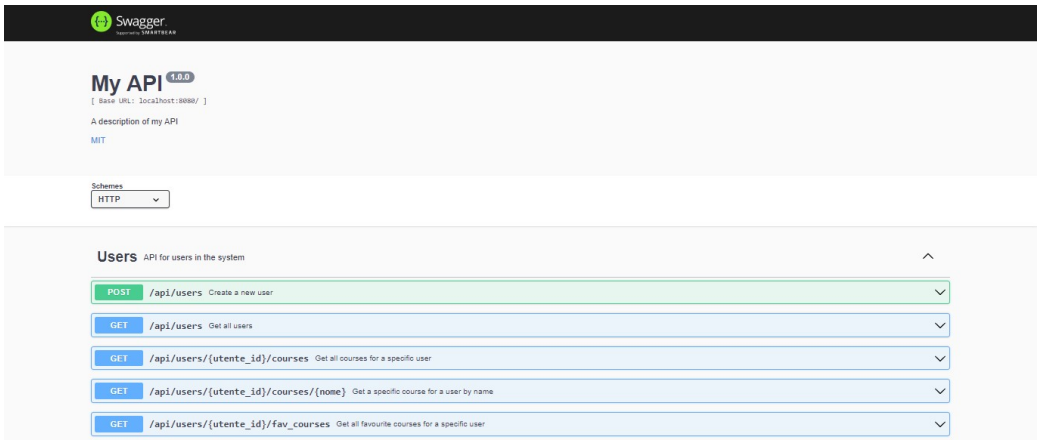
L'endpoint da invocare per raggiungere la documentazione è:

Localhost/8080/api-docs/

Da sottolineare che non tutte le API che sono state sviluppate vengono utilizzate per il funzionamento della web app, in quanto quello che è stato sviluppato e implementato è solamente una parte del progetto.

Di seguito viene riportata la schermata relativa alla documentazione delle API e successivamente una descrizione più approfondita delle API utilizzate per la realizzazione della web app, citate nel paragrafo precedente.

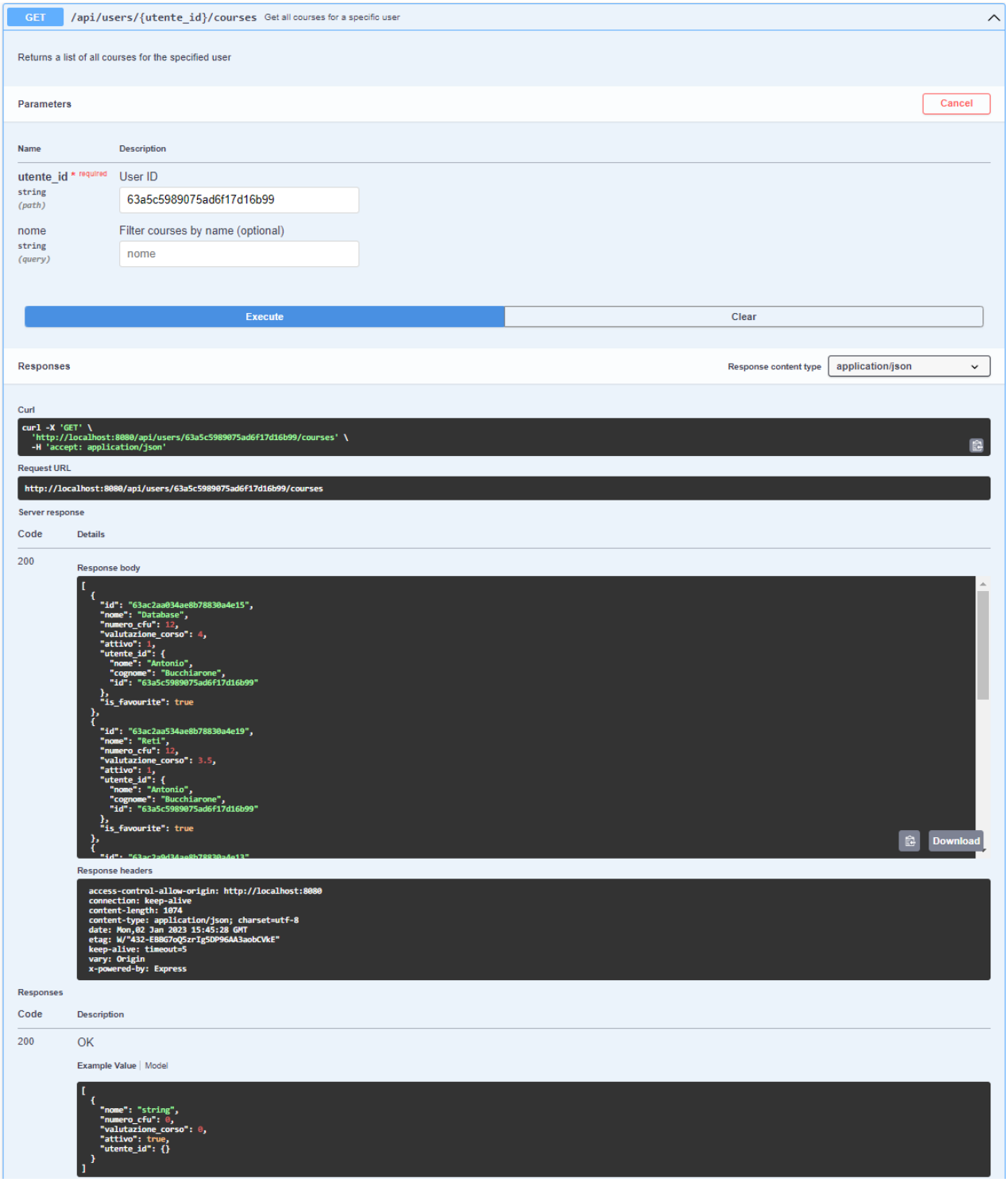
5.1 Users



Descrizione delle API utilizzate per gli utenti:

5.1.1 GET/api/users/{utente_id}/courses

Questa API di tipo GET permette di trovare tutti i corsi per un utente specifico restituirli come riposta. Quindi serve per restituire tutti i corsi che servono a riempire la lista dei corsi più votati.



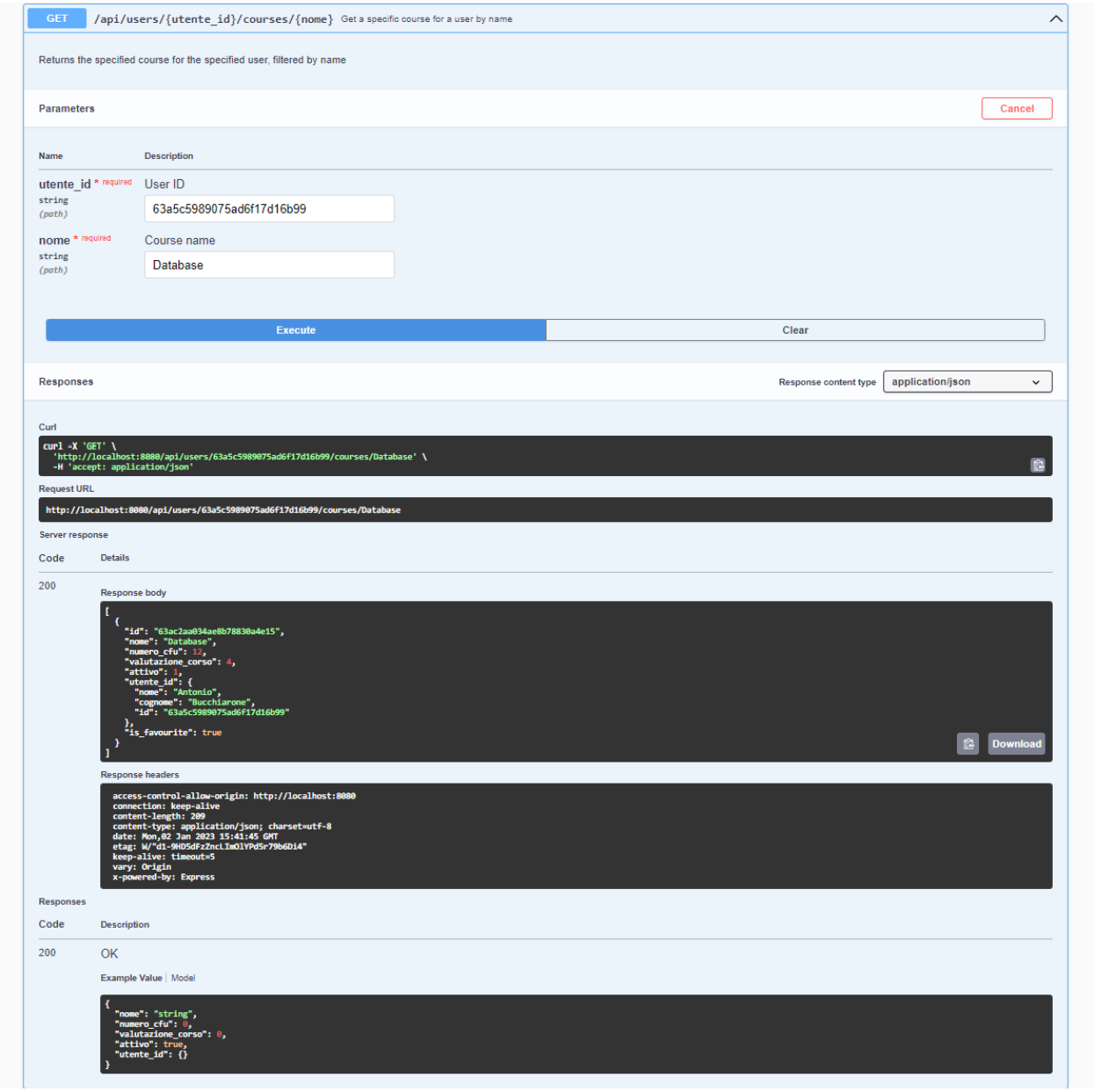
Di seguito il codice della funzione utilizzata:

```
// Retrieve all courses for a specific user
exports.findCoursesForUser = (req, res) => {
  // Validate request
  if (!req.params.utente_id) {
    res.status(400).send({ message: "User ID can not be empty!" });
    return;
  }

  // Find all course-user associations for the specified user
  Favourite_course.find({ utente_id: req.params.utente_id })
    .then((associations) => {
      // Extract the course IDs from the associations
      const courseIds = associations.map((association) => association.corso_id.toString());

      const nome = req.query.nome;
      var condition = nome ? { nome: { $regex: new RegExp(nome), $options: "i" } } : {};

      Course.find(condition)
        .populate("utente_id", "nome cognome")
        .sort({ valutazione_corso: -1 })
        .then(data => {
          const coursesWithIsFavourite = data.map((course) => {
            return {
              id: course.id,
              nome: course.nome,
              numero_cfu: course.numero_cfu,
              valutazione_corso: course.valutazione_corso,
              attivo: course.attivo,
              utente_id: course.utente_id,
              // Aggiungi qui altre proprietà se necessario
              is_favourite: courseIds.includes(course.id)
            }
          });
          res.send(coursesWithIsFavourite);
        })
        .catch(err => {
          res.status(500).send({
            message:
              err.message || "Some error occurred while retrieving courses."
          });
        });
    })
    .catch((err) => {
      res.status(500).send({
        message: "Error retrieving course-user associations for user.",
      });
    });
};
```

Prima di tutto viene validata la richiesta per assicurarsi che l'ID dell'utente sia presente per poi trovare le associazioni tra il corso e l'utente ed estrarre l'ID dei corsi dalle associazioni. Viene poi effettuata una query per trovare tutti i corsi che soddisfano una determinata condizione, popolandoli con le informazioni dell'utente e ordinandoli per valutazione del corso in ordine decrescente.
Infine, per ogni corso trovato viene creato un oggetto contenente tutte le informazioni del corso, compreso se il corso è uno dei preferiti dell'utente, determinandolo confrontando l'ID del corso con gli ID estratti dalle associazioni tra il corso e l'utente precedentemente fatte.
Ognuno di questi oggetti viene restituito come risposta.

5.1.2. GET/api/users/{utente_id}/courses/{nome}

Questa API di tipo GET permette di recuperare tutti i corsi associati ad un determinato utente, identificato tramite il suo ID (utente_id), che soddisfano il filtro di ricerca (tramite il nome del corso), e li invia come risposta alla richiesta. Quindi viene utilizzata per restituire i corsi che, in base al nome, che vengono cercati da un determinato utente, che soddisfino una determinata ricerca.



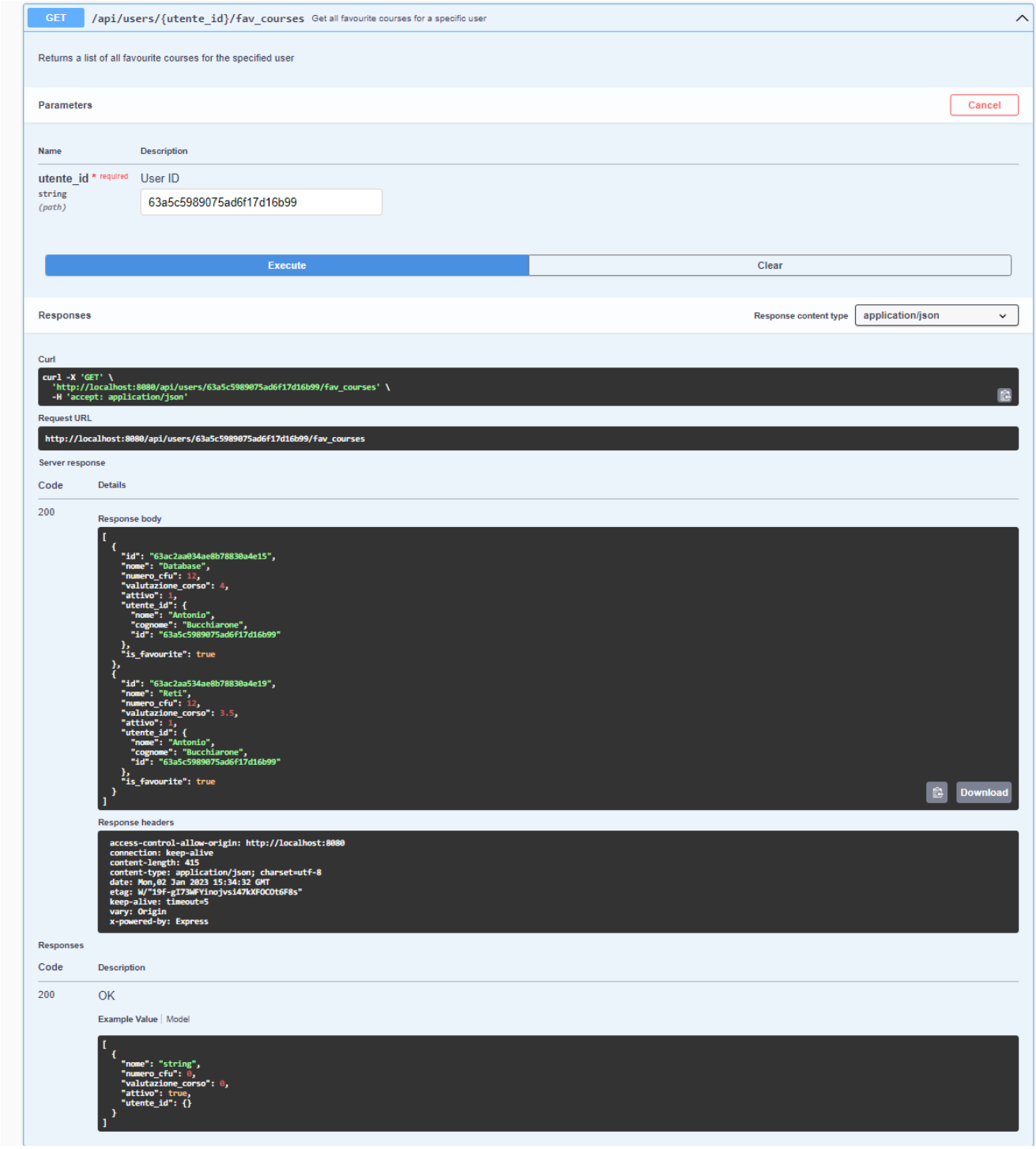
Di seguito il codice della funzione utilizzata:

```
exports.findCoursesForUserFilteredByNome = (req, res) => {
  // Validate request
  if (!req.params.utente_id) {
    res.status(400).send({ message: "User ID can not be empty!" });
    return;
  }

  // Find all course-user associations for the specified user
  Favourite_course.find({ utente_id: req.params.utente_id })
    .then((associations) => {
      // Extract the course IDs from the associations
      const courseIds = associations.map((association) => association.corso_id.toString());

      const prefix = req.params.nome;

      Course.find({ nome: { $regex: "^" + prefix, $options: "i" } })
        .populate("utente_id", "nome cognome")
        .sort({ valutazione_corso: -1 })
        .then(data => {
          const coursesWithIsFavourite = data.map((course) => {
            return {
              id: course.id,
              nome: course.nome,
              numero_cfu: course.numero_cfu,
              valutazione_corso: course.valutazione_corso,
              attivo: course.attivo,
              utente_id: course.utente_id,
              // Aggiungi qui altre proprietà se necessario
              is_favourite: courseIds.includes(course.id)
            }
          });
          res.send(coursesWithIsFavourite);
        })
        .catch(err => {
          res.status(500).send({
            message:
              err.message || "Some error occurred while retrieving courses."
          });
        });
    })
    .catch((err) => {
      res.status(500).send({
        message: "Error retrieving course-user associations for user.",
      });
    });
};
```

Prima di tutto viene validata la richiesta per assicurarsi che l'ID dell'utente sia presente per poi trovare l'associazione tra il corso e l'utente ed estrarre gli ID dei corsi delle associazioni. Successivamente viene effettuata una query per trovare tutti i corsi che hanno il nome che inizia con il prefisso specificato nella richiesta e popolandoli con le informazioni dell'utente, dopodiché i corsi vengono ordinati per valutazione del corso in ordine decrescente. Infine per ogni corso viene creato un oggetto contenente tutte le informazioni del corso, compreso se il corso è tra i corsi preferenti dell'utente specificato, determinandolo confrontando l'ID del corso con gli ID dei corsi estratti dalle associazioni tra i corsi e gli utenti.
Tutti questi oggetti vengono quindi restituiti come risposta.

3.1.2. GET/api/users/{utente_id}/fav_courses

Questa API di tipo GET permette di recuperare tutti i corsi associati ad un determinato utente, identificato tramite il suo ID (utente_id), che vengono segnalati come preferiti e li invia come risposta alla richiesta. Viene utilizzata quindi per riempire la lista dei corsi preferiti di un determinato utente.



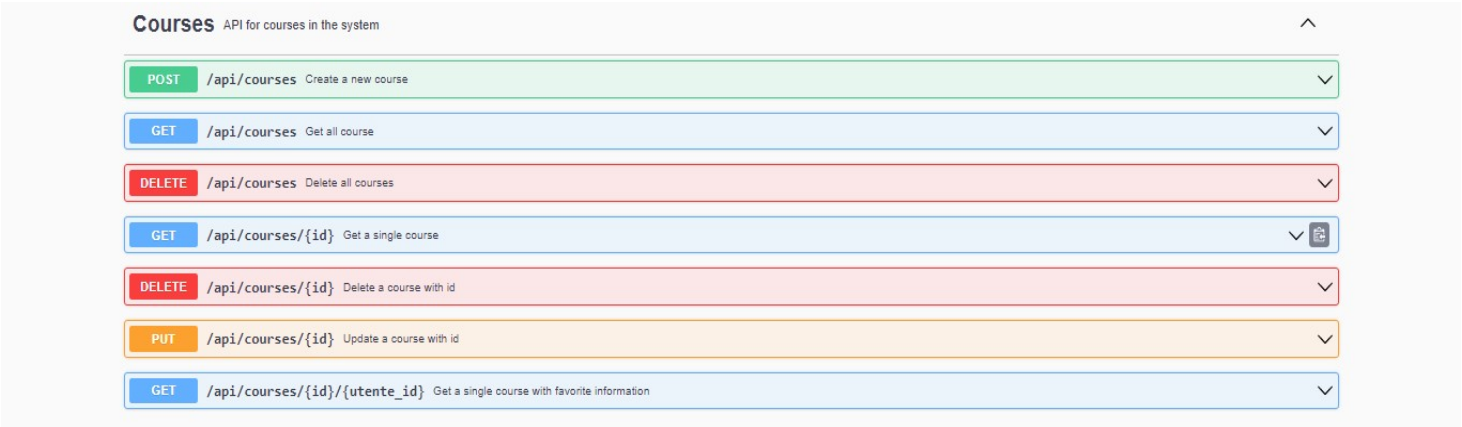
Di seguito il codice della funzione utilizzata:

```
exports.findFavCoursesForUser = (req, res) => {
  // Validate request
  if (!req.params.utente_id) {
    res.status(400).send({ message: "User ID can not be empty!" });
    return;
  }

  // Find all course-user associations for the specified user
  Favourite_course.find({ utente_id: req.params.utente_id })
    .then((associations) => {
      // Extract the course IDs from the associations
      const courseIds = associations.map((association) => association.corso_id);

      // Find all courses with the extracted IDs
      Course.find({ _id: { $in: courseIds } })
        .populate("utente_id", "nome cognome")
        .then((data) => {
          const coursesWithIsFavourite = data.map((course) => {
            return {
              id: course.id,
              nome: course.nome,
              numero_cfu: course.numero_cfu,
              valutazione_corso: course.valutazione_corso,
              attivo: course.attivo,
              utente_id: course.utente_id,
              // Aggiungi qui altre proprietà se necessario
              is_favourite: true
            }
          });
          res.send(coursesWithIsFavourite);
        })
        .catch((err) => {
          res.status(500).send({
            message: "Error retrieving courses for user.",
          });
        });
    })
    .catch((err) => {
      res.status(500).send({
        message: "Error retrieving course-user associations for user.",
      });
    });
};
```

Prima di tutto viene validata la richiesta per assicurarsi che l'ID dell'utente sia presente, per poi trovare le associazioni tra il corso e l'utente per l'utente specificato ed estrarre tutti gli ID delle associazioni. Successivamente viene effettuata una query per trovare tutti i corsi con gli ID estratti e popolarli con le informazioni dell'utente. Infine per ogni corso viene creato un oggetto con all'interno tutte le informazioni del corso e la proprietà 'is_favourite' viene impostata su 'true', in quanto tutti questi corsi sono stati selezionati come preferiti dall'utente.

Tutti questi oggetti vengono quindi restituiti come risposta.

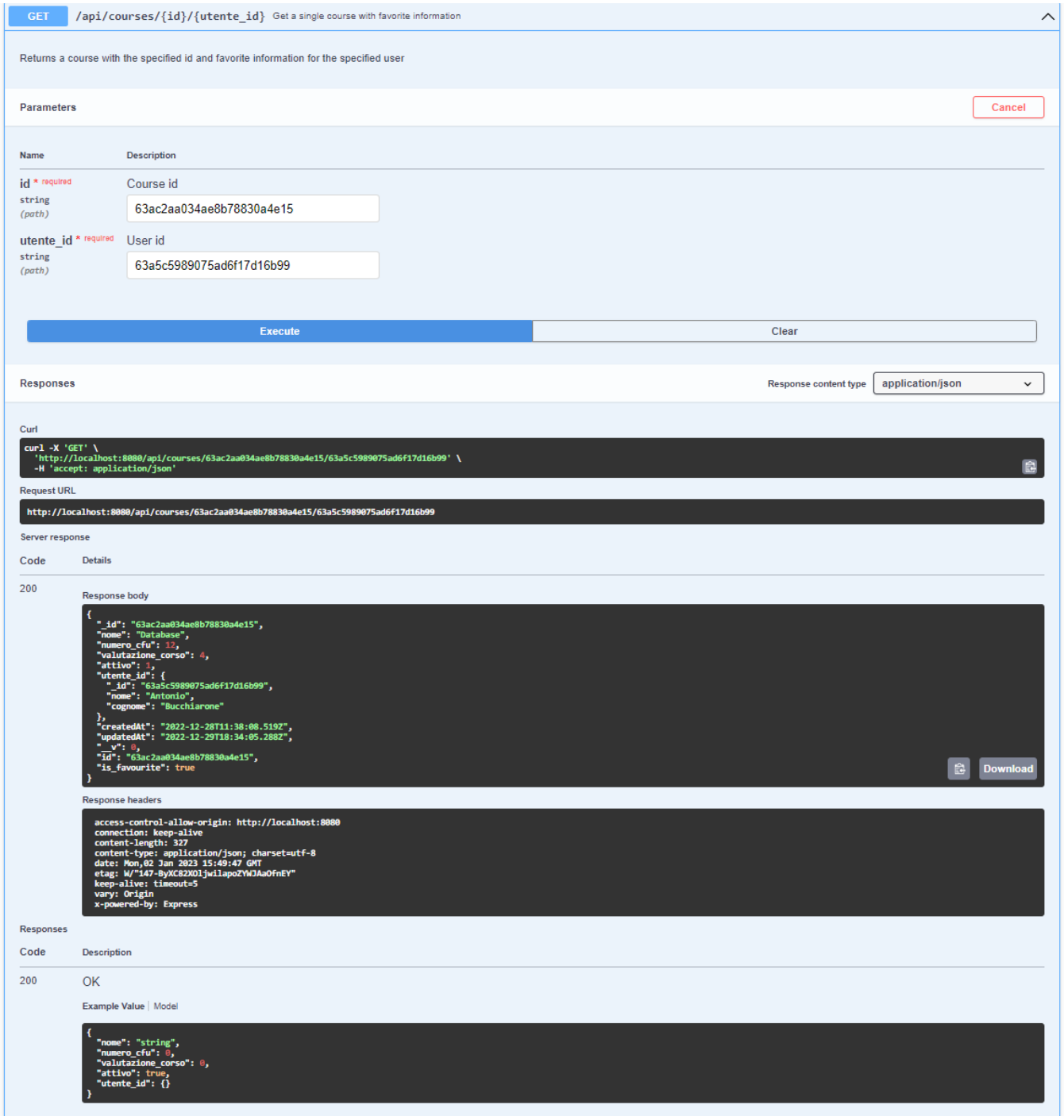
5.2 Courses



Descrizione delle API utilizzate per courses:

5.2 GET/api/courses/{id}/{utente_id}

Questa API di tipo GET permette di recuperare un determinato corso, identificato tramite il sui ID e verificare se è stato segnalato come preferito da un determinato udente identificato anch'esso tramite il suo ID, e infine inviare il corso recuperato e le informazioni di segnalazione di preferenza come risposta alla richiesta. Viene utilizzata quindi per inserire tra i preferiti un determinato corso da un determinato utente.



GET /api/courses/{id}/{utente_id} Get a single course with favorite information

Returns a course with the specified id and favorite information for the specified user

Parameters Cancel

Name	Description
id * required string (path)	Course id 63ac2aa034ae8b78830a4e15
utente_id * required string (path)	User id 63a5c5989075ad6f17d16b99

Execute Clear

Responses Response content type application/json

Curl

```
curl -X 'GET' \
  'http://localhost:8080/api/courses/63ac2aa034ae8b78830a4e15/63a5c5989075ad6f17d16b99' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:8080/api/courses/63ac2aa034ae8b78830a4e15/63a5c5989075ad6f17d16b99
```

Server response

Code Details

200

Response body

```
{
  "_id": "63ac2aa034ae8b78830a4e15",
  "nome": "Database",
  "numero_cfu": 12,
  "valutazione_corso": 4,
  "attivo": 1,
  "utente_id": {
    "_id": "63a5c5989075ad6f17d16b99",
    "nome": "Antonio",
    "cognome": "Bucchiarone"
  },
  "createdAt": "2022-12-28T11:38:08.519Z",
  "updatedAt": "2022-12-29T18:34:05.288Z",
  "__v": 0,
  "id": "63ac2aa034ae8b78830a4e15",
  "is_favourite": true
}
```

Download

Response headers

```
access-control-allow-origin: http://localhost:8080
connection: keep-alive
content-length: 327
content-type: application/json; charset=utf-8
date: Mon,02 Jan 2023 15:49:47 GMT
etag: W/"147-ByXC82XOljwilapoZYWJAaOfnEY"
keep-alive: timeout=5
vary: Origin
x-powered-by: Express
```

Responses

Code Description

200 OK

Example Value | Model

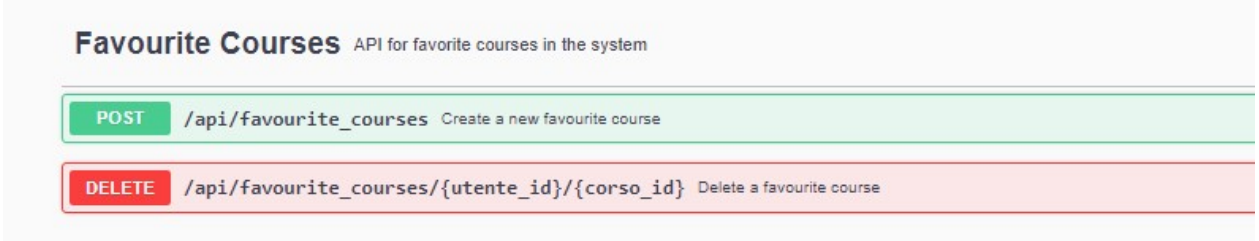
```
{
  "nome": "string",
  "numero_cfu": 0,
  "valutazione_corso": 0,
  "attivo": true,
  "utente_id": {}
}
```

Di seguito il codice della funzione utilizzata:

```
exports.findOneWithFavorite = (req, res) => {
  const id = req.params.id;
  const utente_id = req.params.utente_id;

  Course.findById(id)
    .populate("utente_id", "nome cognome")
    .then(async data => {
      if (!data) {
        res.status(404).send({ message: "Not found Course with id " + id });
      } else {
        const favouriteCourse = await Favourite_course.findOne({ utente_id: utente_id, corso_id: id });
        data = data.toObject()
        data.id = data._id
        if (favouriteCourse) {
          data.is_favourite = true;
        } else {
          data.is_favourite = false;
        }
        res.send(data)
      }
    })
    .catch(err => {
      res
        .status(500)
        .send({ message: "Error retrieving Course with id=" + id });
    });
};
```

Prima di tutto viene ottenuto l'ID del corso e l'ID dell'utente dai parametri della richiesta, per poi effettuare una query al database per trovare il corso con l'Id specificato e popolarlo con le informazioni dell'utente. Viene poi effettuata un'altra query alla collezione delle associazioni tra corsi e utenti per trovare una associazione specifica che corrisponda all'ID dell'utente e l'ID del corso. Se l'associazione viene trovata, il corso viene contrassegnato come preferito (viene quindi impostata la proprierà 'is_favorite' su 'true'), altrimenti viene contrassegnato come non preferito (viene quindi impostata la proprietà 'is_favorite su 'false').
Infine, il corso viene restituito come riposta.

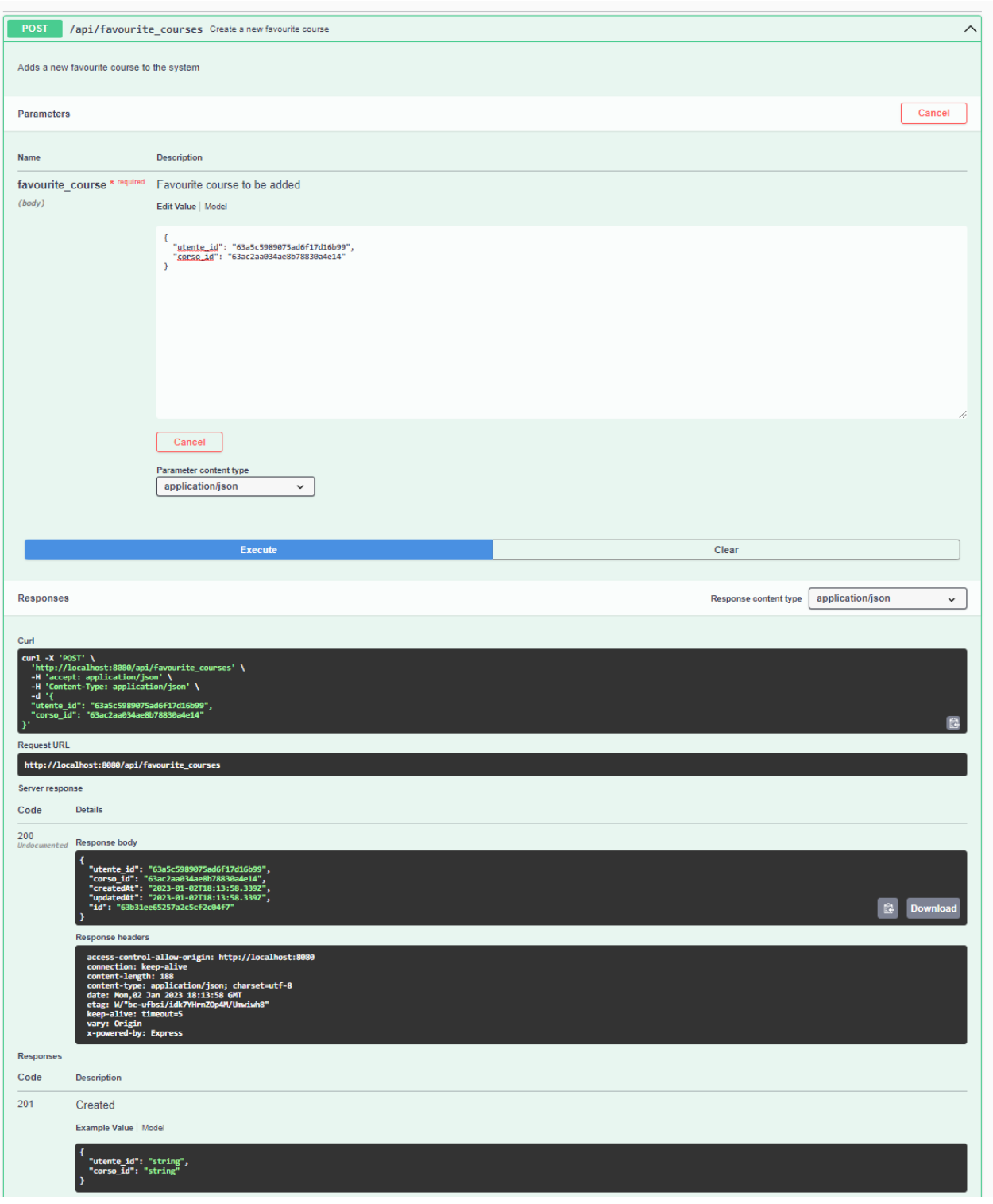
5.3 Favorite Courses



Descrizione delle API utilizzate per Favourite Course:

5.3.1 POST/api/favourite_courses

Questa API di tipo POST permette la creazione di un nuovo oggetto ‘Favourite_corse’ con l’ID relativo all’utente e l’ID relativo al corso specificati nel corpo della richiesta e lo salva nel database, a meno che non esista già un’associazione preferita per questo utente e corso. Più semplicemente salva nel database un determinato corso che viene aggiunto ai corsi preferiti da un determinato utente.



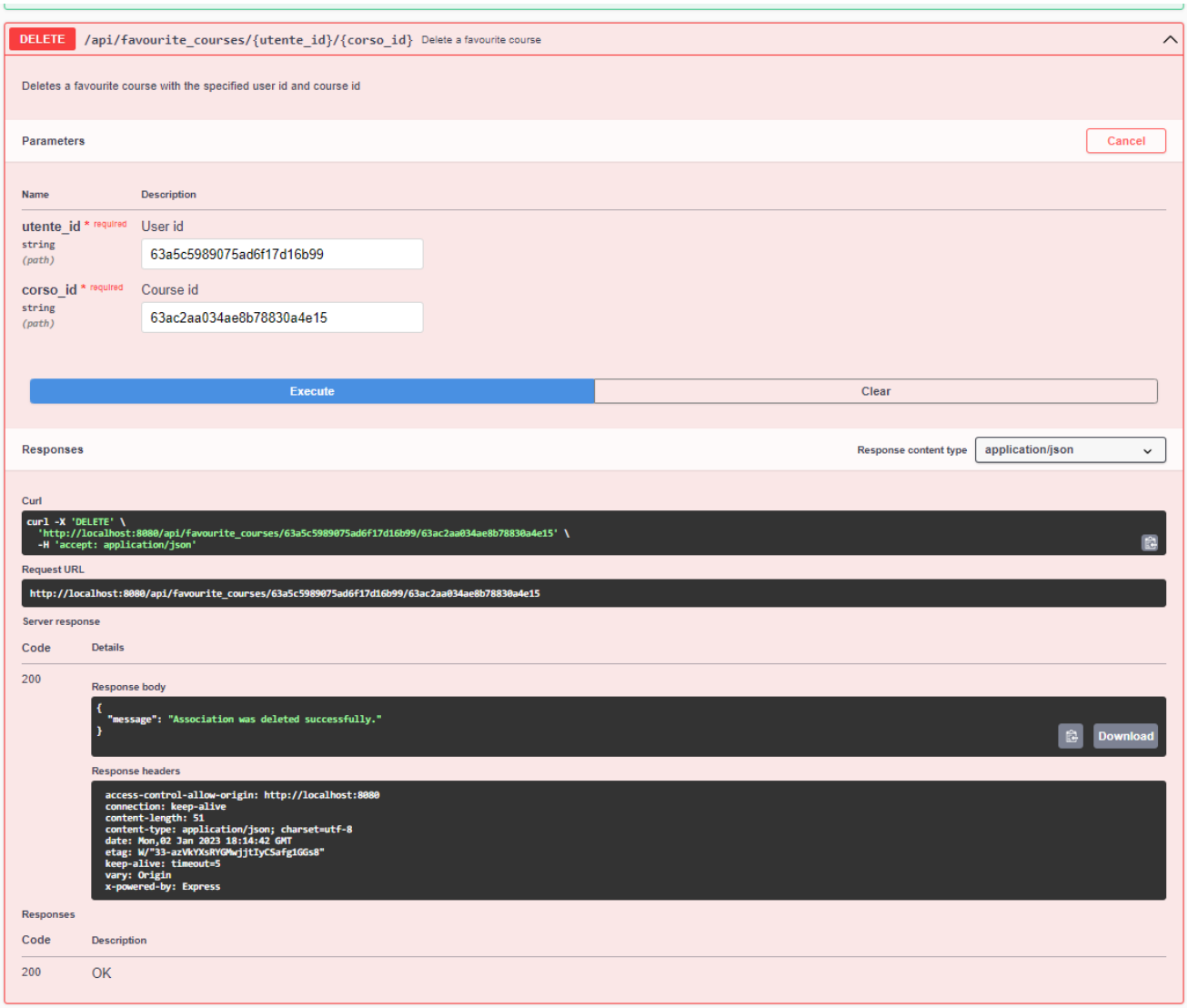
Di seguito il codice della funzione utilizzata:

```
exports.create = (req, res) => {
  // Validate request
  if (!req.body.utente_id || !req.body.corso_id) {
    res.status(400).send({ message: "Content can not be empty!" });
    return;
  }

  const favourite_course = new Favourite_course({
    utente_id: req.body.utente_id,
    corso_id: req.body.corso_id,
  });

  Favourite_course.find({ utente_id: req.body.utente_id, corso_id: req.body.corso_id })
  .then((associations) => {
    if (associations.length > 0){
      res.status(500).send({
        message: "There is already a favourite association for this user."
      });
    } else {
      favourite_course.save(favourite_course)
      .then(data => {
          res.send(data);
      })
      .catch(err => {
        res.status(500).send({
        message:
            err.message || "Some error occurred while creating the fav_course."
        })
      });
    }
  })
  .catch((err) => {
    res.status(500).send({
      message: err.message || "Error occurred while checking for existing favourite association."
    });
  });
};
```

Prima di tutto viene validata la richiesta per assicurarsi che siano presenti nel database sia l'ID dell'utente che l'ID del corso, al fine di creare un nuovo oggetto di associazione tra corso e utente utilizzando l'ID dell'utente e l'ID del corso forniti nella richiesta. Successivamente viene effettuata una query alle collezioni delle associazioni tra corsi e utenti per trovare un'associazione specifica che corrisponde all'ID dell'utente e l'ID del corso. Se l'associazione viene trovata, viene inviata una risposta di errore indicando che l'associazione esiste già, altrimenti viene salvata la nuova associazione tra corso e utente nel database e viene restituita come risposta

5.3.2 DELETE/api/favourite_courses/{utente_id}/{corso_id}

Questa API di tipo DELETE permette di eliminare l'associazione tra un corso con un determinato ID ed un utente con un determinato ID dal database. Più semplicemente viene usata per togliere un determinato corso dai corsi preferiti di un determinato utente.

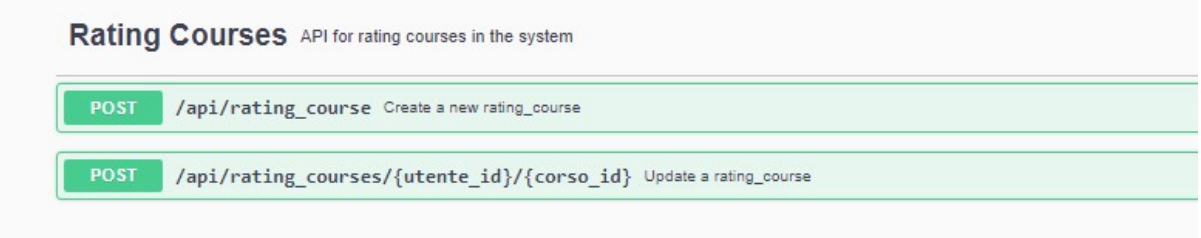


Di seguito il codice della funzione utilizzata:

```
exports.deleteFromFavourites = (req, res) => {
  // Verifica che l'ID utente e l'ID corso siano presenti nella richiesta
  if (!req.params.utente_id || !req.params.corso_id) {
    res.status(400).send({ message: "Utente ID and course ID can not be empty." });
    return;
  }

  // Cerca l'associazione corso-utente da eliminare
  Favourite_course.findOneAndDelete({ utente_id: req.params.utente_id, corso_id: req.params.corso_id })
    .then((association) => {
      if (!association) {
        res.status(404).send({ message: "Association not found." });
        return;
      }
      res.send({ message: "Association was deleted successfully." });
    })
    .catch((err) => {
      if (err.kind === "ObjectId" || err.name === "NotFound") {
        res.status(404).send({ message: "Association not found." });
      } else {
        res.status(500).send({ message: "Could not delete association." });
      }
    });
};
```

Prima di tutto verificato che sia presente sia l'ID dell'utente che l'ID del corso nel database nella richiesta, per poi effettuare una query alla collezione delle associazioni tra corsi e utenti per trovare una associazione specifica che corrisponde all'ID dell'utente e all'ID del corso forniti nella richiesta. Se l'associazione viene trovata, essa viene eliminata utilizzando il metodo 'findOneAndDelete'. Nel caso in cui l'associazione non viene trovata, viene inviata una risposta di errore indicando che l'associazione tra il corso e l'utente non è stata trovata.
Infine, se l'associazione è stata eliminata correttamente viene inviata una risposta di successo.

5.4 Rating Courses



Descrizione delle API utilizzate per Rating Courses:

5.4.1 POST/api/rating_courses/{utenti_id}/{corso_id}

Questa API di tipo POST permette di gestire l'update di una valutazione data da un determinato utente, identificato tramite il suo ID, ad un determinato corso, identificato tramite il sui ID. In particolare trova la valutazione nel database e lo aggiorna con il nuovo voto, se il voto non viene trovato ne crea uno nuovo e lo salva nel database. Si occupa anche di fare la media tra tutte le valutazioni di un determinato corso e aggiornare la valutazione di quel corso.

POST /api/rating_courses/{utente_id}/{corso_id} Update a rating_course

Updates the rating of a rating_course with the specified user and course IDs

Parameters Cancel

Name	Description
utente_id * required string (path)	ID of the user 63a5c5989075ad6f17d16b99
corso_id * required string (path)	ID of the course 63ac2aa034ae8b78830a4e15
valutazione * required (body)	New rating for the rating_course Edit Value | Model

```
{
  "utente_id": "63a5c5989075ad6f17d16b99",
  "corso_id": "63ac2aa034ae8b78830a4e15",
  "valutazione": 3
}
```

Cancel

Parameter content type: application/json

Execute Clear

Responses Response content type: application/json

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/rating_courses/63a5c5989075ad6f17d16b99/63ac2aa034ae8b78830a4e15' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "utente_id": "63a5c5989075ad6f17d16b99",
  "corso_id": "63ac2aa034ae8b78830a4e15",
  "valutazione": 3
}'
```

Request URL

```
http://localhost:8080/api/rating_courses/63a5c5989075ad6f17d16b99/63ac2aa034ae8b78830a4e15
```

Server response

Code	Details
200	Response body

```
{
  "corso_id": "63ac2aa034ae8b78830a4e15",
  "utente_id": "63a5c5989075ad6f17d16b99",
  "createdAt": "2022-12-29T17:16:58.239Z",
  "updatedAt": "2023-01-02T18:17:02.754Z",
  "valutazione": 3,
  "id": "63adcb8abb34c624a92a04bd"
}
```

Download

Response headers

```
access-control-allow-origin: http://localhost:8080
connection: keep-alive
content-length: 204
content-type: application/json; charset=utf-8
date: Mon,02 Jan 2023 18:17:02 GMT
etag: W/"cc-hPIBEUq2wSPjRwOP65ECTMQ/Sis"
keep-alive: timeout=5
vary: Origin
x-powered-by: Express
```

Responses

Code	Description
200	Successfully updated the rating_course Example Value | Model

```
{
  "utente_id": "string",
  "corso_id": "string",
  "valutazione": 0
}
```

500	Error updating rating_course

Di seguito il codice della funzione utilizzata con i rispettivi metodi chiamati:

```
exports.findAndUpdate = (req, res) => {
  // Trova il documento con l'utente e il corso specificati
  Rating_course.findOneAndUpdate(
    { utente_id: req.params.utente_id, corso_id: req.params.corso_id },
    { $set: { valutazione: req.body.valutazione } },
    { upsert: true, new: true }
  // con upsert se non lo trova lo crea nuovo!
  )
    .then((rating) => {
      updateCourseRating(req.params.corso_id)
      res.send(rating);
    })
    .catch((err) => {
      res.status(500).send({
        message: "Error updating rating.",
      });
    });
};
```

```
const updateCourseRating = (corso_id) => {
  // Trova tutti i documenti di Rating_course con il corso_id specificato
  Rating_course.find({ corso_id: corso_id })
    .then((ratings) => {
      // Calcola la media delle valutazioni
      const ratingSum = ratings.reduce((acc, rating) => acc + rating.valutazione, 0);
      const ratingCount = ratings.length;
      const ratingAvg = ratingSum / ratingCount;

      // Aggiorna il campo "valutazione_corso" del corso con il nuovo rating calcolato
      Course.findOneAndUpdate(
        { _id: corso_id },
        { $set: { valutazione_corso: ratingAvg } },
        { new: true }
      )
        .then((course) => {
          console.log(`Updated rating for course with id ${corso_id}: ${ratingAvg}`);
        })
        .catch((err) => {
          console.log(`Error updating rating for course with id ${corso_id}: ${err}`);
        });
    })
    .catch((err) => {
      console.log(`Error finding ratings for course with id ${corso_id}: ${err}`);
    });
}
```

Prima di tutto viene effettuata una query alla collezione delle valutazioni corso-utente per trovare una valutazione specifica che corrisponde all'ID dell'utente e all'ID del corso forniti alla richiesta. Se la valutazione viene trovata, viene aggiornata utilizzando il metodo 'findOneAndUpdate', mentre se la valutazione non viene trovata, viene creata una nuova valutazione utilizzando lo stesso metodo ma con il flag 'upsert' impostato a 'true'. Al termine di questa operazione, viene inviata una risposta con la valutazione aggiornata o creata.

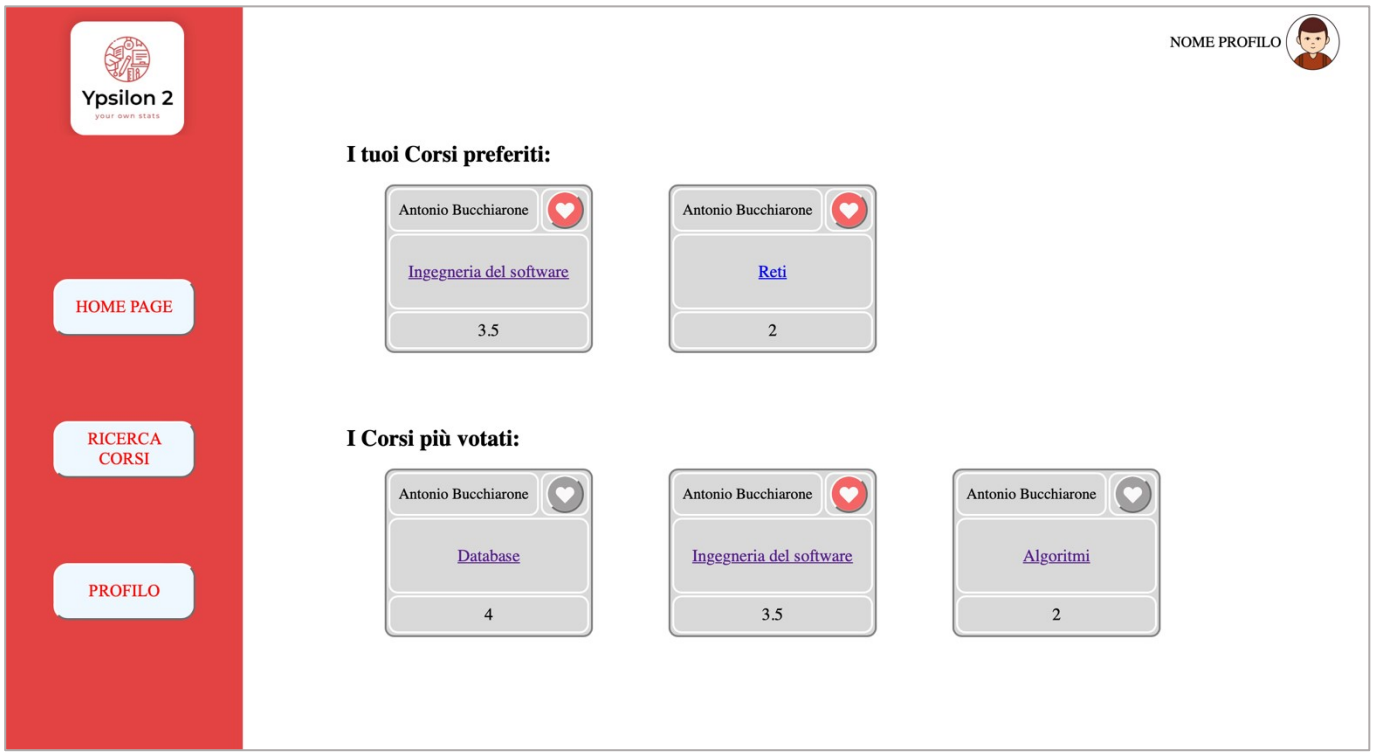
Viene poi eseguita una funzione chiamata ‘updateCourseRating’, nella quale viene passato l’ID del corso come parametro; questa funzione cerca tutte le valutazioni corsoutente per il corso specificato, calcola la media delle valutazioni e aggiorna il campo ‘valutazione_corso’ del corso con il nuovo rating calcolato.
Se si verica un errore durante questa operazione, viene stampato un messaggio di errore.

6. Front-end Implementation

Il front-end fornisce le funzionalità di visualizzazione, inserimento e cancellazione dei dati fornite dalla web app Ypsilon2 dell’utente studente attraverso l’utilizzo delle API descritte in precedenza. In particolare la web app è composta da una HomePage, una pagina per la ricerca dei corsi, una pagina per il profilo personale e una pagina dedicata ai corsi.

6.1 Homepage

Di seguito è presente la schermata della HomePage:



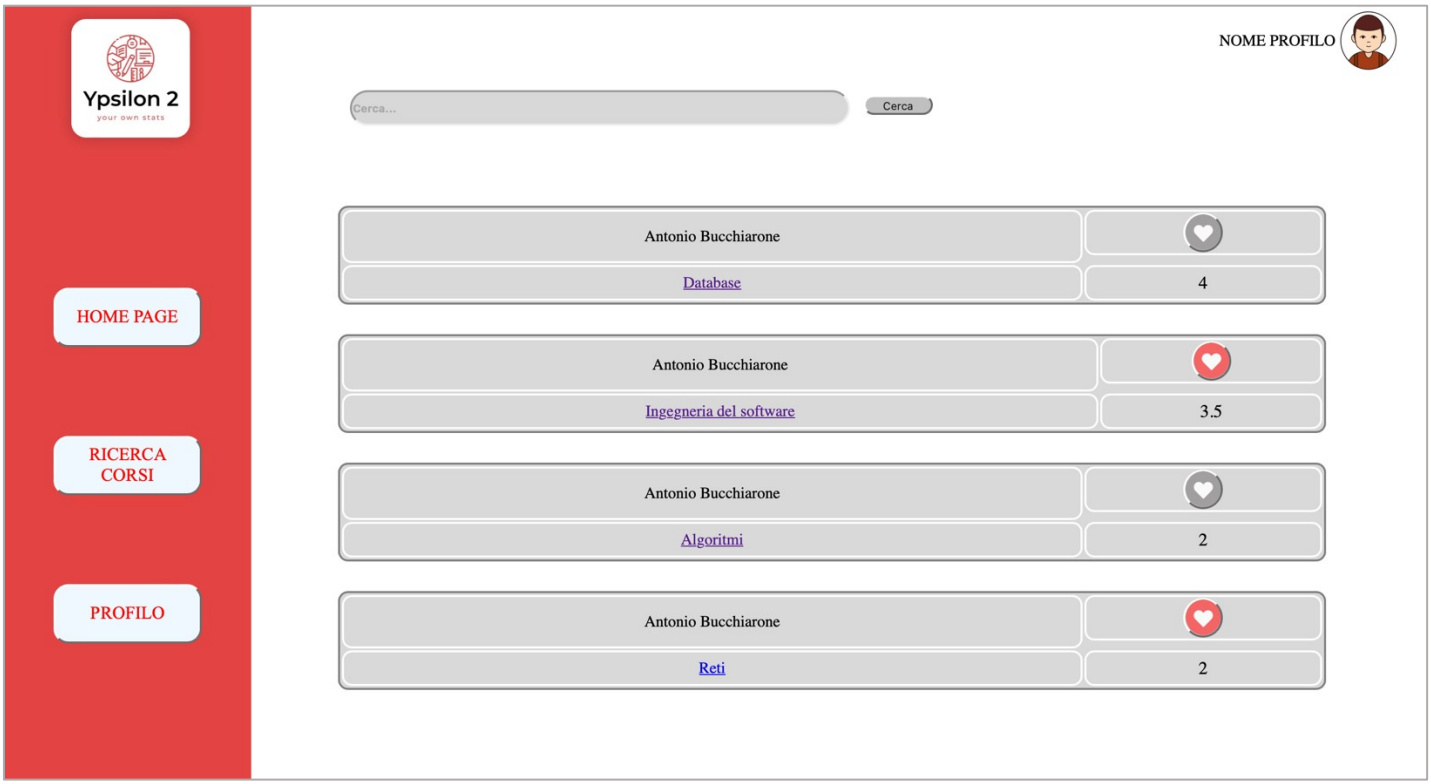
Entrando nel sito l’utente visualizza la HomePage, questa propone la lista dei corsi preferiti e dei corsi più votati; per ogni corso viene descritto il nome del docente relativo al corso, il nome del corso, la media delle valutazioni del corso e la relativa preferenza. Inoltre viene rappresentata una barra laterale con dei pulsanti con cui navigare in altre pagine del sito.
È presente anche il nome dell’utente loggato e la sua immagine profilo (Per questa risorsa è stata implementata solo la parte front-end non sono state implementate ancora le API).

In questa pagina è possibile:

- Premendo il pulsante relativo alla **ricerca corsi** essere reindirizzati alla pagina dedicata alla ricerca dei corsi.
- Premendo il pulsante relativo a **profilo** essere reindirizzati alla pagina dedicata al profilo personale dell'utente.
- Premendo il pulsante con il simbolo del **cuore** inserire un corso nei preferiti (il cuore diventerà rosso), rimuovere un corso dai preferiti (il cuore diventerà grigio).
- Premendo sul nome di un corso essere reindirizzati alla pagina relativa al corso.

6.2. Ricerca Corsi

Di seguito è presente la schermata relativa alla Ricerca Corso:



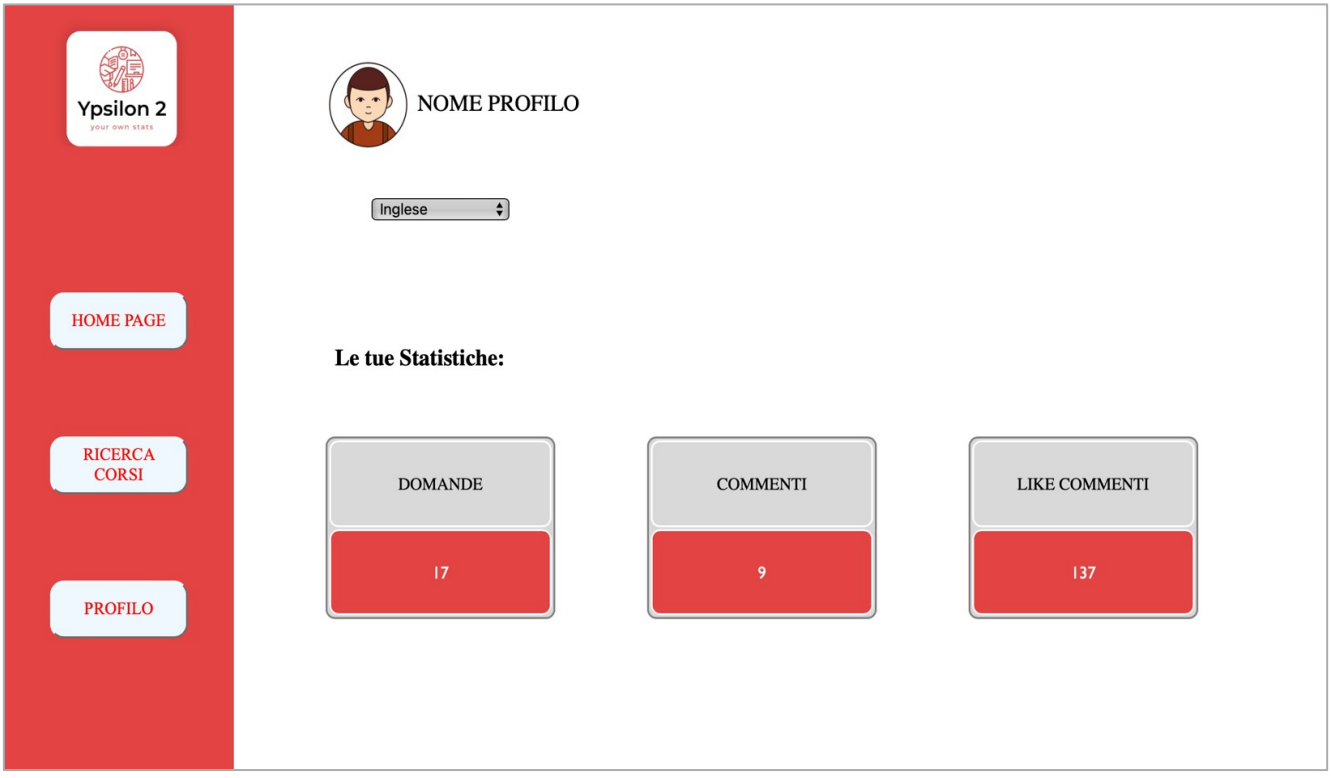
In questa pagina, vengono rappresentati la barra per il filtraggio dei corsi in base al loro nome e la lista dei corsi presenti all'interno del database; ogni corso ha la sua relativa descrizione esattamente come nella HomePage.
Inoltre, come in tutte le altre schermate è presente una barra laterale a sinistra con cui navigare in altre pagine del sito.
È presente anche il nome dell'utente loggato e la sua immagine profilo (Per questa risorsa è stata implementata solo la parte front-end non sono state implementate ancora le API)

In questa pagina è possibile:

- Premendo il pulsante relativo alla **HomePage** essere reindirizzati alla pagina principale.
- Premendo il pulsante relativo a **profilo** essere reindirizzati alla pagina dedicata al profilo personale dell'utente.
- Premendo il pulsante con il simbolo del **cuore** inserire un corso nei preferiti (il cuore diventerà rosso), rimuovere un corso dai preferiti (il cuore diventerà grigio).
- Scorrere la lista dei corsi utilizzando la barra di scorrimento.
- Premendo sul nome di un corso essere reindirizzati alla pagina relativa al corso.
- Inserendo il nome di un corso nella barra di ricerca eseguire un filtraggio tra i corsi presenti nel database e restituire il corso cercato.

6.3. Profilo

Di seguito è presente la schermata relativa al profilo personale dell'utente:



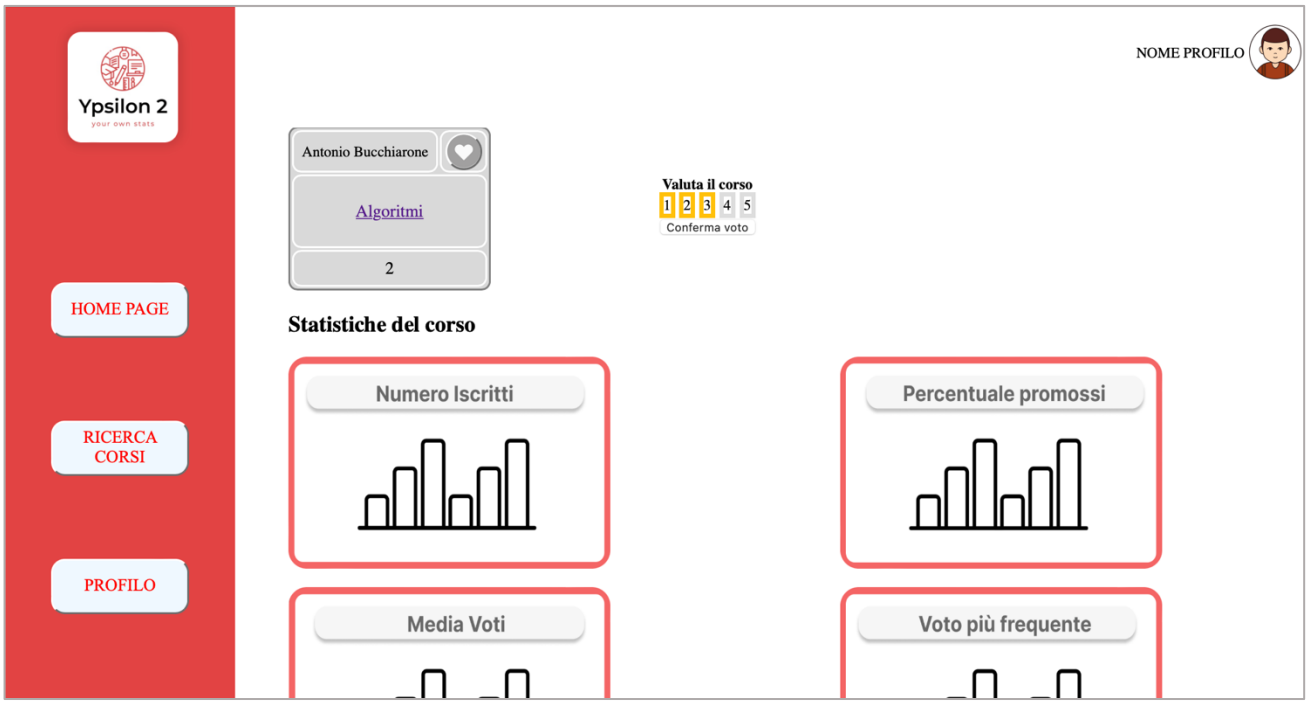
In questa pagina vengono rappresentate le informazioni dell'utente. In particolare vengono visualizzate il nome dell'utente e la sua immagine profilo, le statistiche personali e il cambio della lingua. (Per queste risorse è stata implementata solo la parte front-end, mentre non sono state implementate ancora le API).
Inoltre, come in tutte le altre schermate è presente una barra laterale a sinistra con cui navigare in altre pagine del sito.

In questa pagina è possibile:

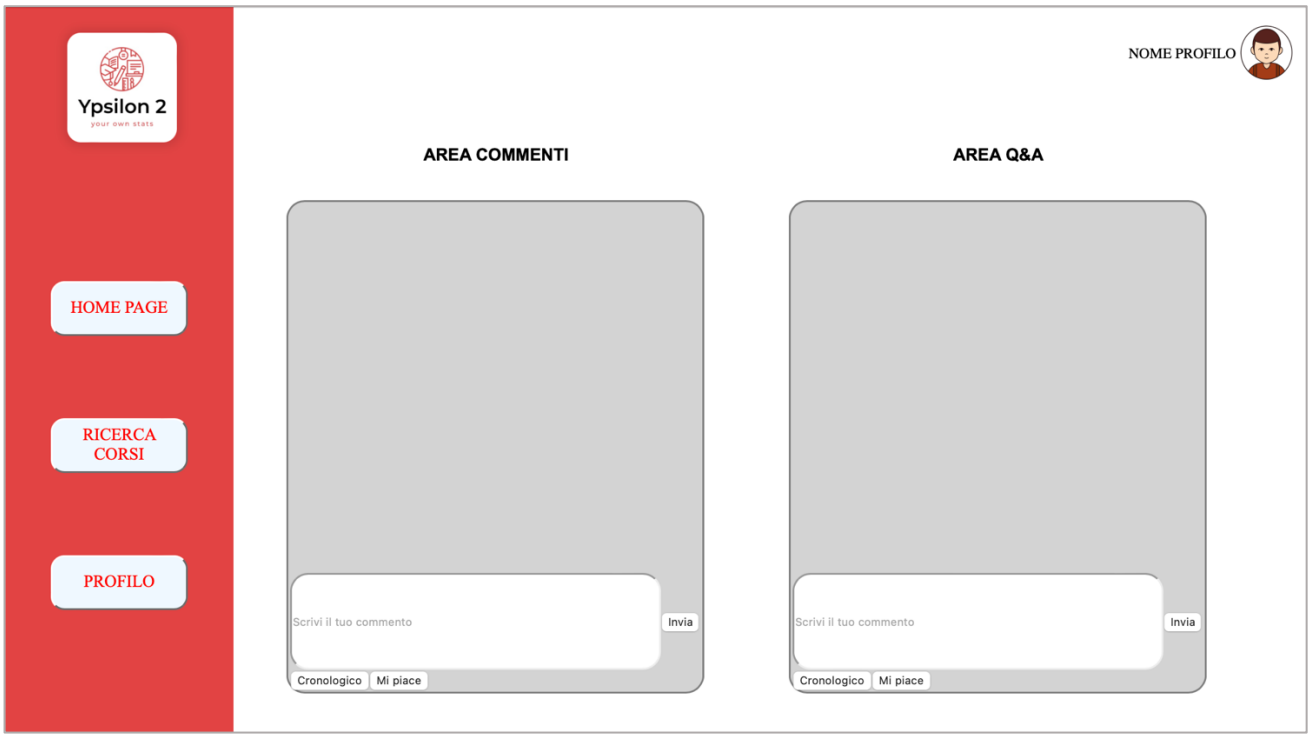
- Premendo il pulsante relativo alla **HomePage** essere reindirizzati alla pagina principale.
- Premendo il pulsante relativo alla **ricerca corsi** essere reindirizzati alla pagina dedicata alla ricerca dei corsi.

6.4. Corso

Di seguito è presente la schermata relativa ad un corso:



Scrollando nella pagina del corso troviamo questo:



Questa pagina viene dedicata ad un relativo corso; in questa pagina viene visualizzata la descrizione del corso relativo a quella pagina (nome corso, nome docente relativo, valutazione del corso) e il metodo di valutazione del corso.
Inoltre, come in tutte le altre schermate è presente una barra laterale a sinistra con cui navigare in altre pagine del sito.
È presente una sezione dedicata ai commenti e una sezione dedicata a Q&A (Queste risorse non sono state ancora implementate).
È presente la sezione dedicata alle statistiche, il nome dell'utente e l'immagine profilo (Per queste risorse è stata implementata solo la parte front-end, mentre non sono state implementate ancora le API).

In questa è pagina è possibile:

- Premendo il pulsante relativo alla **HomePage** essere reindirizzati alla pagina principale.
- Premendo il pulsante relativo alla **ricerca corsi** essere reindirizzati alla pagina dedicata alla ricerca dei corsi.
- Premendo il pulsante relativo a **profilo** essere reindirizzati alla pagina dedicata al profilo personale dell'utente.
- Dare una valutazione al corso (che farà media con le valutazioni date dagli altri utenti allo stesso corso) da 1 a 5, premendo sulla casella relativa al numero, e confermarla premendo sul pulsante **conferma voto.**
- Scorrere la pagina utilizzando la barra di scorrimento.

Inoltre è presente un esempio di funzionamento dell'area dei commenti, infatti è possibile scrivere un commento nella barra dedicata ed inviarlo premendo il tasto invia. Il commento inviato verrà visualizzato nella zona superiore; per ogni commento è presente il nome dell'utente e vi è la possibilità di mettere una valutazione positiva oppure una valutazione negativa (sempre modificabili). Premendo sul tasto 'cronologico' la lista dei commenti verrà ordinata in base all'ordine cronologico, mentre premendo sul tasto 'mi piace' la lista dei commenti verrà ordinata in base al commento con più valutazioni positive.

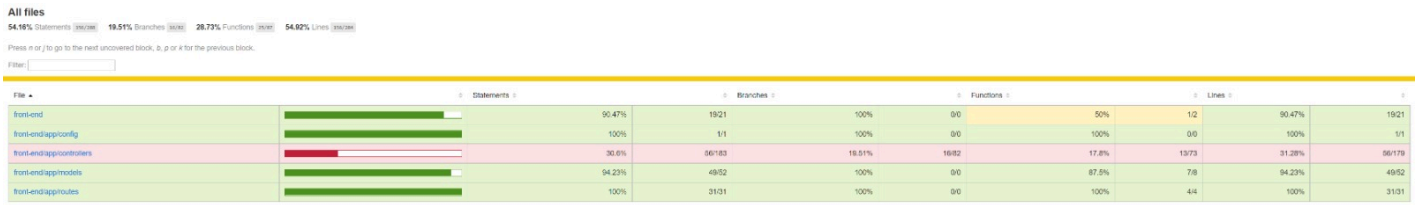
7. GitHub Repository and Deployment Info

Il progetto Ypsilon2 è disponibile al seguente link:

8. Testing

Per eseguire il testing è presente nel progetto una cartella di nome tests al cui interno è presente il file 'course.test.js' che viene usato per effettuare dei casi di test per le API utilizzate per la nostra web app.

Il comando per eseguire la suite di test è: "*npm test*".



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	54.16	19.51	28.73	54.92	
front-end	90.47	100	50	90.47	
app.js	90.47	100	50	90.47	19-20
front-end/app/config	100	100	100	100	
db.config.js	100	100	100	100	
front-end/app/controllers	30.6	19.51	17.8	31.28	
course.controller.js	31.03	12.5	19.04	31.03	9-10,29,37-46,55-65,72-93,101-118,131,141,149-156
favourite_course.controller.js	53.84	40	62.5	53.84	8-9,20,29-37,46-47,54-55,60-63
rating_course.controller.js	17.94	0	0	18.42	8-39,47-58,64-75,83-104
user.controller.js	28.33	22.72	14.81	29.82	11-12,27,42,52-92,101-140,149-185
front-end/app/models	94.23	100	87.5	94.23	
course.model.js	100	100	100	100	
favourite_course.model.js	100	100	100	100	
index.js	100	100	100	100	
rating_course.model.js	70	100	50	70	14-16
user.model.js	100	100	100	100	
front-end/app/routes	100	100	100	100	
course.routes.js	100	100	100	100	
favourite_course.routes.js	100	100	100	100	
rating_course.routes.js	100	100	100	100	
user.routes.js	100	100	100	100	

Test Suites: 1 passed, 1 total
Tests: 7 passed, 7 total
Snapshots: 0 total
Time: 17.821 s
Ran all test suites.