# UNIVERSE

Progetto ideato dagli studenti dell'università di Trento

Lorenzo Masè

Lorenzo Zanini

Pietro Bassa

Indice

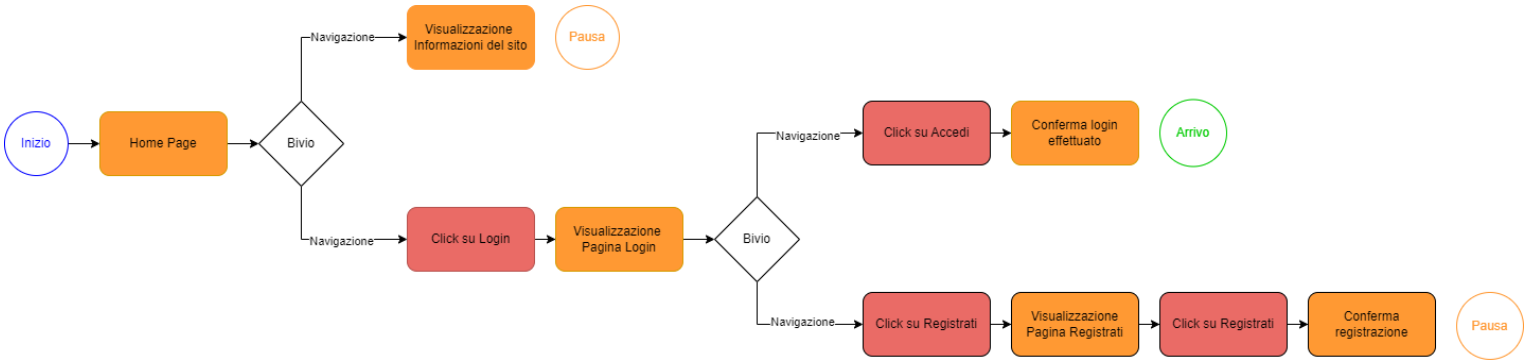
# Scopo del documento

Il seguente documento descrive una parte del funzionamento di UNIverse, partendo dagli user flows, dove descriviamo le funzioni e i percorsi che potrà seguire un utente anonimo, un utente autenticato, un tutor e un admin.
Successivamente viene descritta la struttura del sito e, andando più nel particolare, le API con il relativo testing e il FrontEnd con la sua implementazione.

# User Flows

## Utente Anonimo

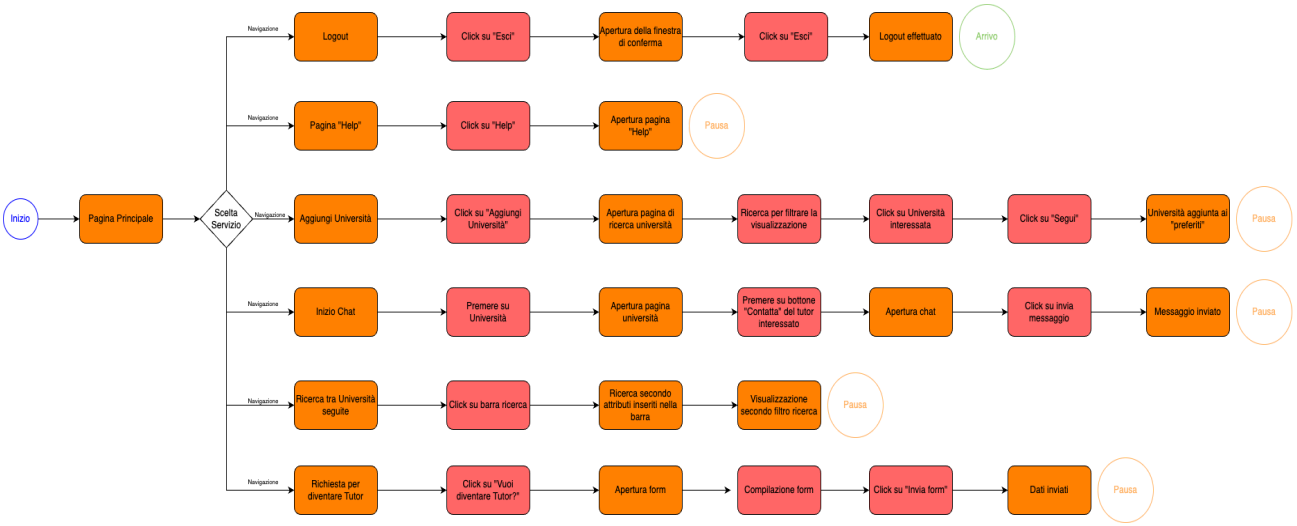
Una volta ricercato il sito, un utente anonimo si troverà di fronte alla home page.
All'interno di questa, l'utente non autenticato, potrà visualizzare le informazioni generali del sito oppure effettuare subito il Login.
Per la seconda scelta sarà necessario effettuare un click sul tasto Login, questo aprirà la pagina di Login.
All'interno dell'interfaccia l'utente potrà decidere se accedere, quindi premere sul bottone Accedi, oppure registrarsi, con annesso bottone Registrati.
Nel primo caso una volta inserite le credenziali esatte sarà effettuato l'accesso, nel secondo si verrà indirizzati ad un'altra pagina di registrazione, qui una volta inserite le credenziali si potrà effettuare la registrazione.



## Utente Autenticato

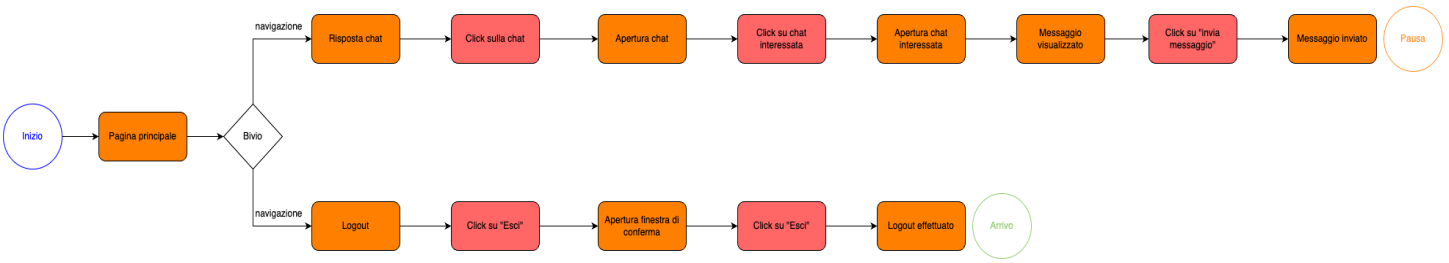
Inizialmente l'Utente Autenticato si troverà di fronte alla pagina principale, da questa avrà la possibilità di accedere a tutti i servizi:

- Potrà visualizzare le sue università "preferite" e ricercare tra queste per filtrare la visualizzazione e trovare più facilmente la ricercata.
- L'utente potrà aggiungere delle università premendo sul tasto "Aggiungi università" presente nel menù sulla sinistra dell'interfaccia, in questo modo ricercando tra le università presenti nel database potrà aggiungerne una nuova alla sua lista personale. Premendo su un'università avrà modo di visualizzare la sua pagina con relative info e possibili tutor.
- Se l'utente fosse interessato a contattare qualche tutor potrà allora iniziare una chat diretta con il suddetto premendo sul tasto "contatta".
- L'Utente potrà richiedere aiuto attraverso il tasto "Help", sempre presente nel menù sulla sinistra, questo tasto una volta premuto aprirà una pagina di aiuto, dove sarà presente un video che spiega all'utente come utilizzare il sito e la sezione delle FAQ.
- L'Utente potrà effettuare il logout da questa pagina premendo sul tasto "Esci" sempre nel menù a sinistra.
  In questo modo potrà uscire formalmente dal sito.
- L'Utente potrà effettuare una richiesta per poter diventare tutor, compilando un form dopo aver premuto sul bottone "Vuoi diventare tutor?".



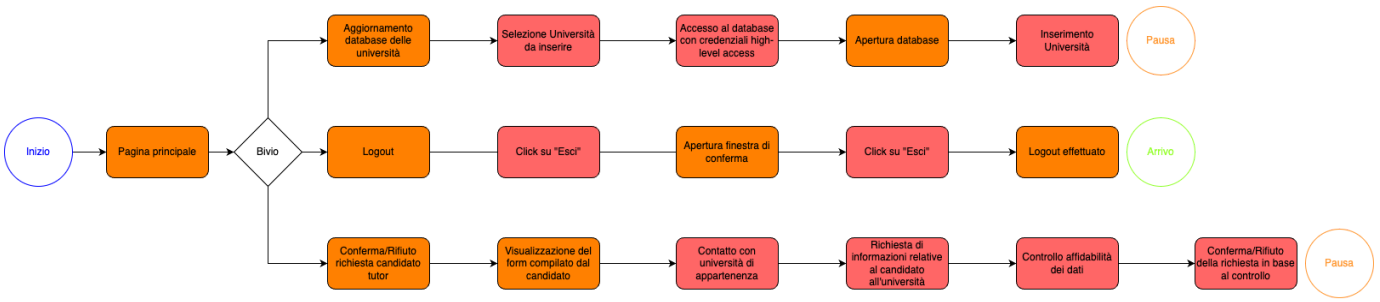

# Tutor

Il tutor per poter parlare con gli utenti avrà bisogno di utilizzare la chat.
Questa gli permetterà di gestire i messaggi in arrivo in modo tale da poter aprire le chat interessate senza perdere troppo tempo nella ricerca.
Il tutor potrà effettuare Logout allo stesso modo di un utente autenticato, quindi premendo sul bottone “Esci” e, dopo l’apertura della scheda di conferma, premendo di nuovo su bottone “Esci”.



## Admin

L’admin avrà il ruolo di accertarsi che l’identità del candidato tutor sia realmente quella che dichiara propria. Questo sarà assicurato tramite il contatto con l’università di appartenenza del candidato tutor.
L’admin avrà anche il ruolo di mantenere aggiornato il database del sito, inserendo nuove università.
L’accesso al database avverrà attraverso credenziali speciali ad alto livello di accesso.
Il logout potrà essere effettuato come ogni altro utente.

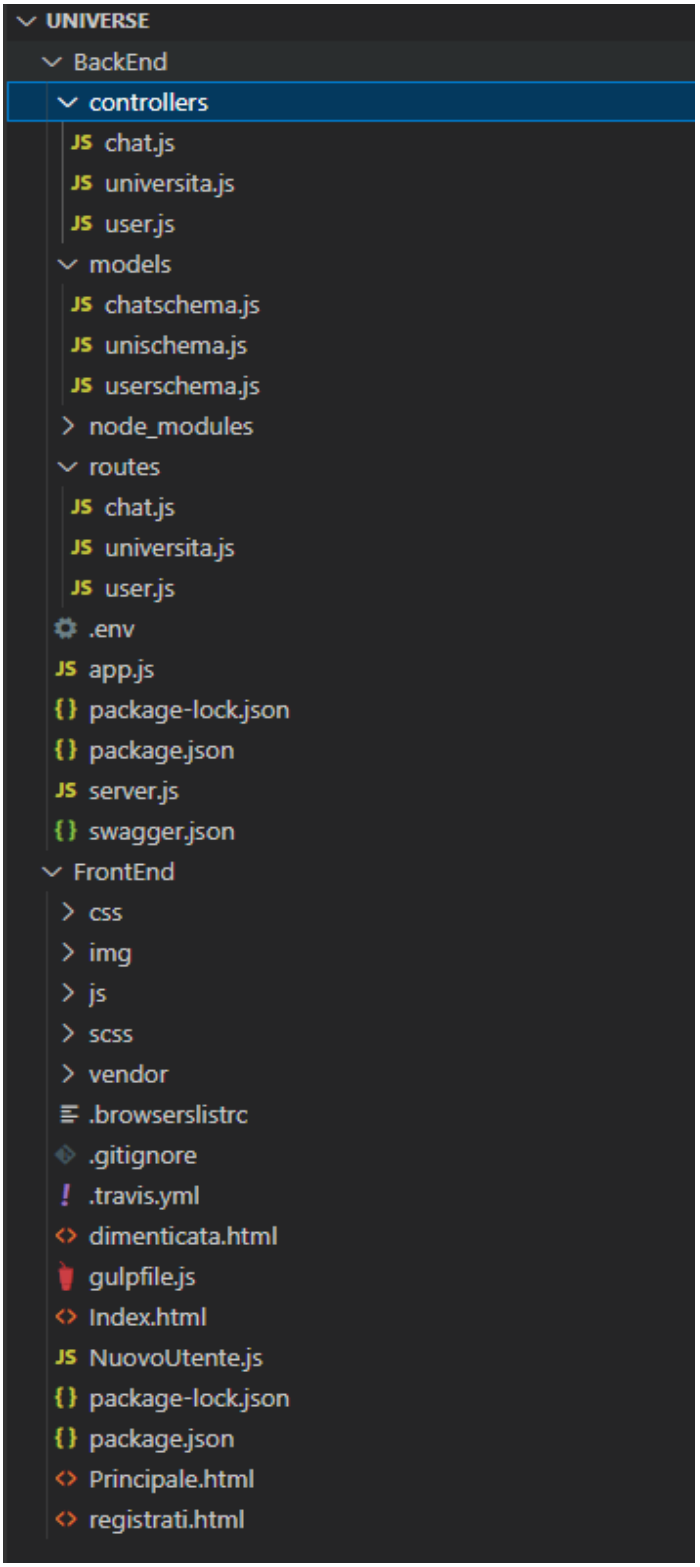


# Application Implementation and Documentation

Dopo aver definito i servizi e le funzioni che vogliamo presenti nel nostro sito, abbiamo pensato a come implementare il flusso.
Per lo sviluppo del sito useremo NodeJs, mentre per la gestione dei dati useremo MongoDB.

## Project Structure

Il sito è composto da 2 directory
-BackEnd
-FrontEnd
dove in BackEnd saranno presenti le API di comunicazione tra sito e database, in FrontEnd saranno presenti i file HTML, CSS e JS per la gestione visiva del sito.



## Project Dependencies

Il progetto dipenderà da le seguenti librerie

- dotenv (variabili d'ambiente)
- express (framework)
- mongoose (database)
- nodemailer (invio email)
- jest e superjet (testing)
- swagger-ui-express (documentazione)
- json (formato dati)
- routes (collegamenti)

## Project DataBase

Il Database del sito sarà strutturato con tre schemi, uno per le Università, uno per gli Utenti e uno per le chat. Graficamente in questo modo:

| Collection Name | Documents | Logical Data Size | Avg Document Size | Storage Size | Indexes | Index Size | Avg Index Size |
|---|---|---|---|---|---|---|---|
| chats | 13 | 1.69KB | 133B | 36KB | 1 | 36KB | 36KB |
| università | 4 | 597B | 150B | 36KB | 1 | 36KB | 36KB |
| users | 3 | 524B | 175B | 36KB | 1 | 36KB | 36KB |

## Users

Schema di come è implementato l'utente

```
_id: ObjectId('63ee04a2f374fd1e85e68f62')
name: "Lorenzo"
surname: "Mase"
email: "bsd@gmail.com"
password: "123321"
Is_tutor: false
› universita_seguite: Array
Is_Admin: true
__v: 0
```

## Chats

Schema di come sono implementati i messaggi

```
_id: ObjectId('63ee4f2b6e70358c8ce8ecb1')
Sender: "bsd@gmail.com"
Receiver: "pietro.bassa@studenti.unitn.it"
Time: 2023-02-16T15:43:39.114+00:00
Text: "testo"
__v: 0
```

## Università

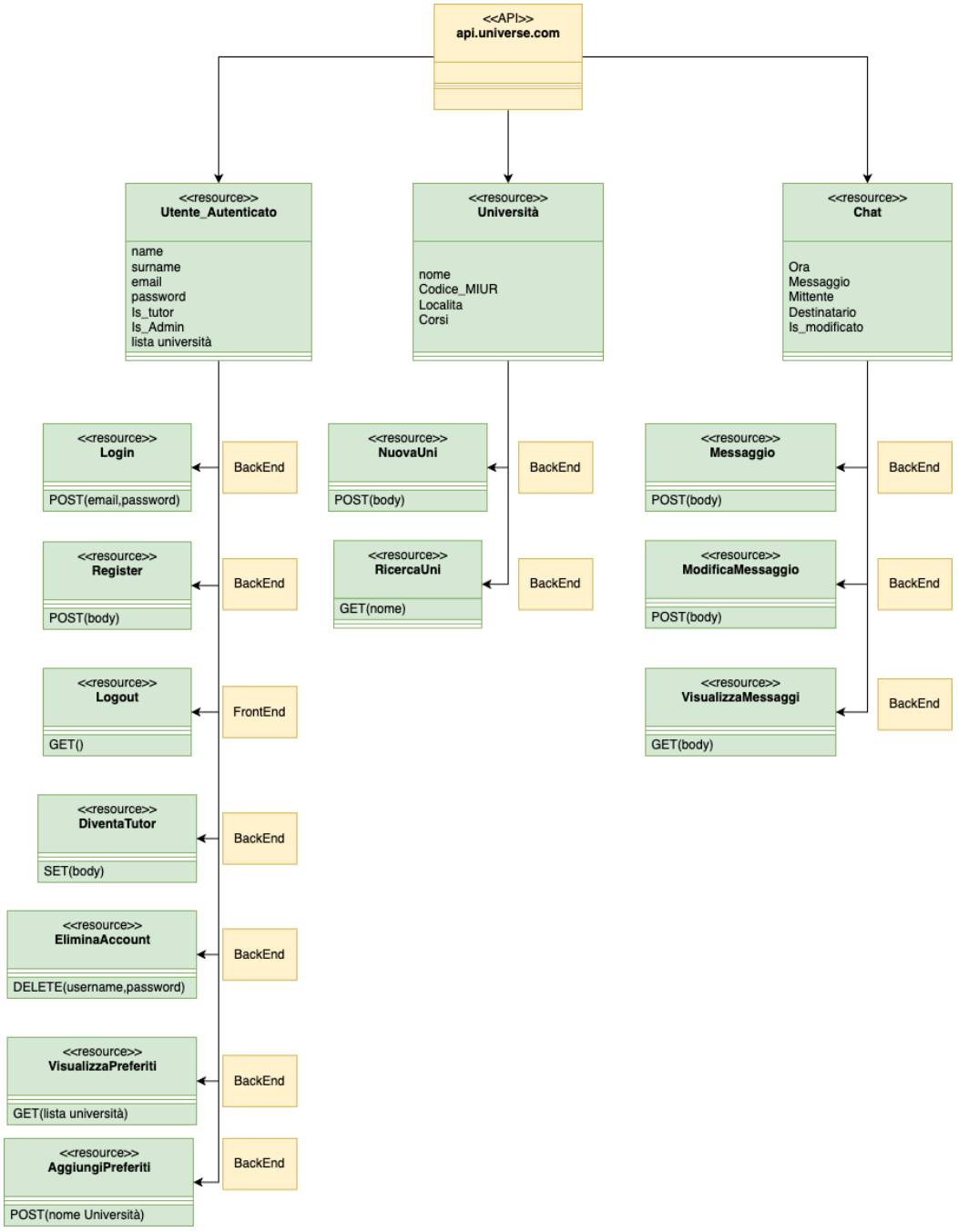
Schema di come sono implementate le università

```
_id: ObjectId('63ea5b3f10ec2361f35be98a')
UniName: "Università Degli Studi di Trento"
MIURcode: 123321
Location: "Trento"
› DgCourse: Array
› Tutors: Array
StudentsNumber: 50000
__v: 0
```

# Project APIs

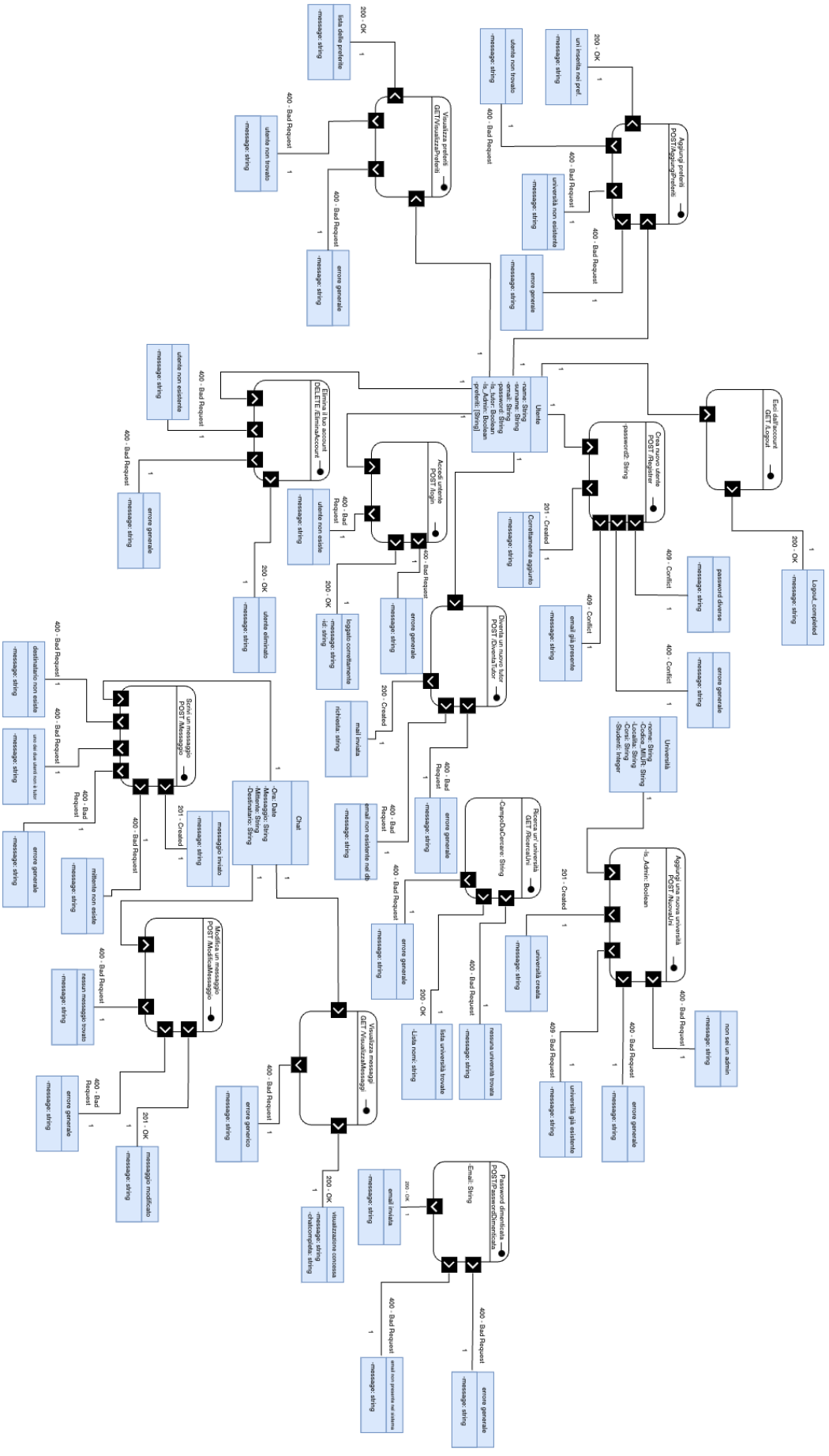
## Resources Extraction from the Class Diagram

Tramite il class diagram identifichiamo le risorse richieste per valutare quali API implementare nel sito.



## Resource Models

Raffiguriamo le API tramite il resource model



# Sviluppo API

Sviluppiamo le API descritte nel resource model:

## Login

API che permette di effettuare il login a un utente che possiede già un account del sito.
Per poter effettuare l'accesso si controlla innanzitutto se è presente l'account nel database, se questo viene trovato si conferma l'accesso, altrimenti si ritorna un errore.

```
const  login = (req,res) => {
    try {
        var dati = req.body;
        //Controllo se l'utente esiste nel db
        User.find({email: dati.email,password: dati.password}, function(err,obj){
          //Utente trovato
          if(obj.length){
            res.status(200).json(dati.email);
          }
          //Utente non trovato
          else{
            res.status(400).json("Utente non esistente");
          }
        })
        //Errore generico dento la funzione
      } catch (err) {
        res.status(400).json(err);
      }
}
```

## Registrazione

API che permette la registrazione di un nuovo account nel database del sito, l'utente invia i dati con cui vuole registrarsi, vengono controllate sia nome utente che mail in caso siano già state utilizzate da altri utenti.
Successivamente se queste sono uniche nel database si procede ad aggiungerle, altrimenti si ritorna un errore.

```
const register = async(req,res) => {
    try {
        // Creazione di un nuovo utente con i dati inviati dalla form
        var dati = req.body;
        //controllo che le password siano uguali
        if(dati.password==dati.password2){
          //controllo email esistente
          User.find({email: dati.email}, function(err,obj) {
            //se non presente aggiungo nuovo utente
            if(!obj.length){
              const user = new User({
                name: req.body.name,
                surname: req.body.surname,
                email: req.body.email,
                password: req.body.password,
                //Is_tutor: req.body.Is_tutor,
                //universita_seguite: req.body.universita_seguite
              });
              // Salvataggio dell'utente nel database
              user.save((err, data) => {
                if (err) return res.json({ Error: err });
                return res.status(201).json(data.email);
                })
            }
            //Errore utente già esistente
            else{
              res.status(409).json("Email già presente");
            }
          });
        }
        //Errore password diverse
        else{
          res.status(409).json("Password Diverse");
        }
        //Errore generico dento la funzione
      } catch (err) {
        res.status(400).json(err);
      }
}
```

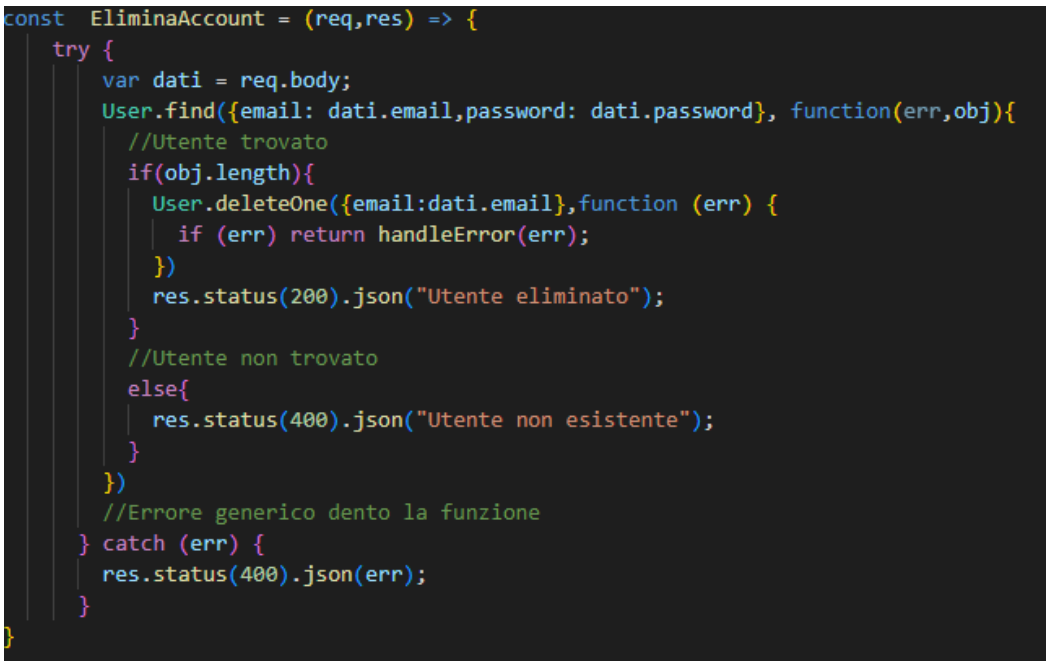
## Diventa Tutor

API che gestisce le richieste per diventare tutor da parte di un utente.
L'utente invia una Email al sito con una richiesta e l'admin, controllando le informazioni contenute decide se confermare o rifiutare la richiesta.

```
const  DiventaTutor = (req,res) => {
    try{
        var dati = req.body;
        var rich="Richiesta Tutor "+dati.email;
        User.find({email: dati.email}, function(err,obj){
          if(obj.length){
            recEmail(dati.email, rich, JSON.stringify(obj[0]));
            res.status(200).json("Inviata");
          }else{
            res.status(400).json("Email non esiste nel database");
          }
        });
      //Errore generico dento la funzione
    } catch (err) {
      res.status(400).json(err);
    }
}
```

## Elimina Account

API che gestisce l'eliminazione di un account dal database.
Si controlla che l'utente che richiede l'eliminazione sia presente nel database, poi si esegue l'eliminazione dei dati.
In caso l'utente non venisse trovato, si ritorna un errore.



```
const  EliminaAccount = (req,res) => {
    try {
        var dati = req.body;
        User.find({email: dati.email,password: dati.password}, function(err,obj){
          //Utente trovato
          if(obj.length){
            User.deleteOne({email:dati.email},function (err) {
              if (err) return handleError(err);
            })
            res.status(200).json("Utente eliminato");
          }
          //Utente non trovato
          else{
            res.status(400).json("Utente non esistente");
          }
        })
        //Errore generico dento la funzione
      } catch (err) {
        res.status(400).json(err);
      }
}
```

## Password Dimenticata

API che gestisce la richiesta di rinnovo password da parte di un utente, la richiesta viene accettata e si invia una nuova email all'indirizzo collegato all'account con una nuova password, collegata al database.

```
const passwordDimenticata = (req,res) => {
  try{
    var dati = req.body;
      var rich="Richiesta Passowrd Dimenticata Universe "+dati.email;
      User.findOne({email: dati.email}, function(err,obj){
        try{
          if(!obj.length){

            sendEmail(dati.email,rich,"la tua password è:" + obj.password);
            res.status(200).json("Inviata");
          }
        }catch(err){
          res.status(400).json("Email non esiste nel database");
        }
      });

  }catch (err) {
    res.status(400).json(err);
  }
}
```

## Ricerca Università

API che gestisce la ricerca tra le università, questa ricerca si basa su un oggetto di ricerca, si analizzano i dati delle varie università disponibili e si ritorna una lista con tutte le università inerenti all'oggetto.

```
const RicercaUni = (req,res) => {
    var dati = req.body;
    uni.find({UniName: { $regex: new RegExp(dati.UniName, "i") }}, function(err,obj){
    if(obj.length){
      let lista = "";
      obj.forEach(element => {
        let aa = element.UniName + "\n";
        lista = lista + aa ;
      });
      res.status(200).json(lista);
    }else{
      //Non trovato
      res.status(400).json("Università non trovata");
    }
  });
}
```

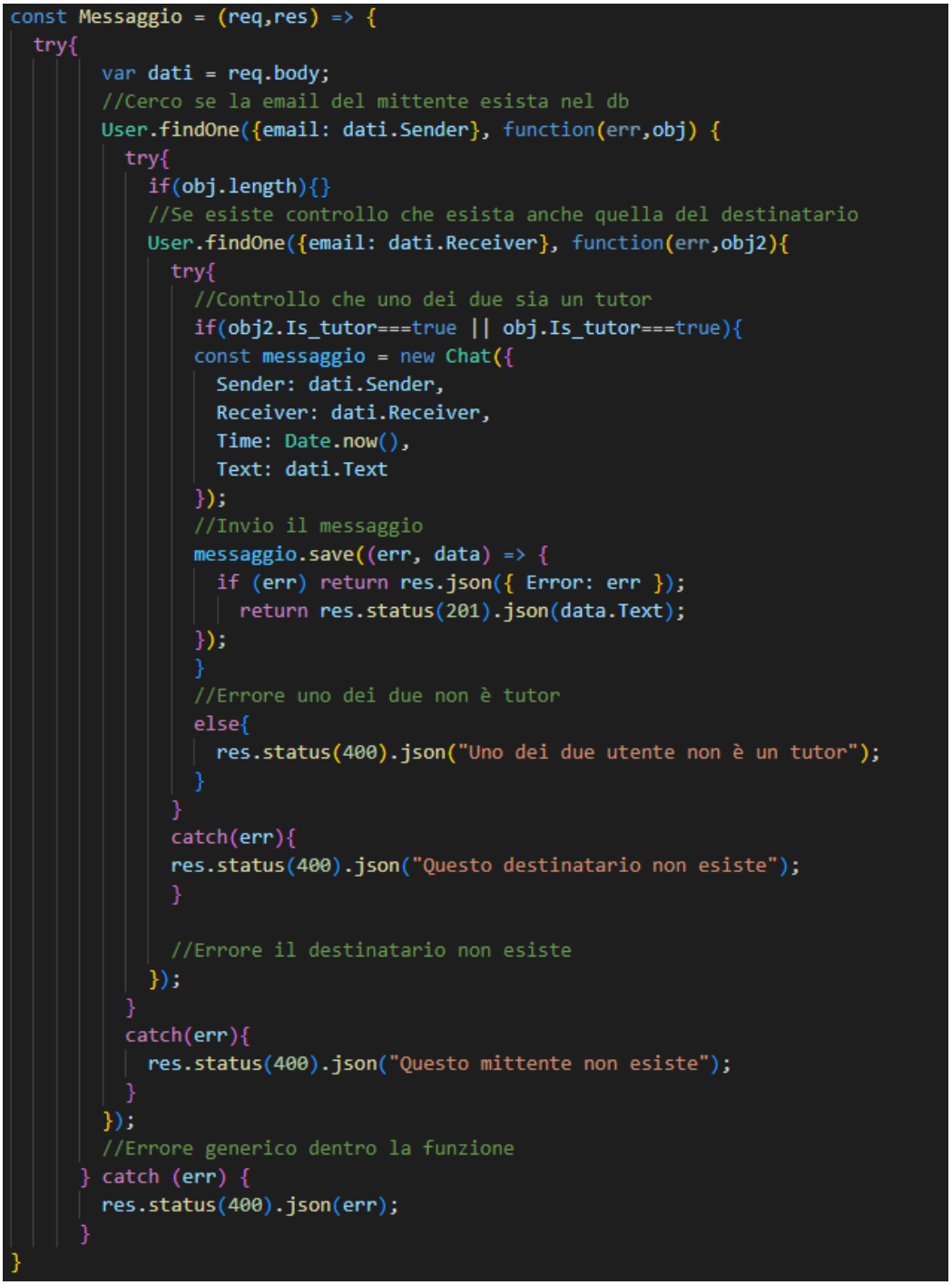
## Aggiungi Università

API che gestisce l'aggiunta di una nuova università al database, soltanto un admin può inoltrare questa richiesta.
Si controlla se i dati inseriti nella form dall'admin inerenti l'università sono già presenti nel database, in caso questi fossero unici si procede all'inserimento altrimenti si ritorna un errore.

```
const  NewUni = (req,res) => {
    try {
        // Creazione di una nuova università con i dati inviati dalla form
        var dati = req.body;
        User.findOne({_id: dati.id}, function (err, utt) {
          if(utt.Is_Admin==true){
            uni.find({UniName: dati.UniName}, function(err,obj) {
              //se non presente aggiungo l'università nuova
              if(!obj.length){
                const universitàS = new uni({
                  UniName : dati.UniName,
                  MIURcode: dati.MIURcode,
                  Location: dati.Location,
                  DgCourse: dati.DgCourse,
                  Tutors: dati.Tutors,
                  StudentsNumber: dati.StudentsNumber
                });
                // Salvataggio dell'università nel database
                universitàS.save((err, data) => {
                  if (err) return res.json({ Error: err });
                  return res.status(201).json(data.UniName);
                })
              }
              //Errore Università già esistente nel database
              else{
                res.status(409).json("Università già presente");
              }
            });
          }else{
            res.status(400).json("Non sei un admin");
          }
        });
        //Errore generico dento la funzione
      } catch (err) {
        res.status(400).json(err);
      }
}
```

## Invia un messaggio

API che gestisce l'invio dei messaggi all'interno di una chat, questa controlla innanzitutto se la email del mittente è presente nel database, se ritorna true, proseguiamo controllando quella del destinatario è presente nel database, se è confermato si controlla che uno dei due sia un tutor.
Se tutte queste richieste sono confermate, il messaggio è libero di transitare, altrimenti si ritornano degli errori appositi.



```
const Messaggio = (req,res) => {
  try{
        var dati = req.body;
        //Cerco se la email del mittente esista nel db
        User.findOne({email: dati.Sender}, function(err,obj) {
          try{
            if(obj.length){}
            //Se esiste controllo che esista anche quella del destinatario
            User.findOne({email: dati.Receiver}, function(err,obj2){
              try{
                //Controllo che uno dei due sia un tutor
                if(obj2.Is_tutor===true || obj.Is_tutor===true){
                const messaggio = new Chat({
                  Sender: dati.Sender,
                  Receiver: dati.Receiver,
                  Time: Date.now(),
                  Text: dati.Text
                });
                //Invio il messaggio
                messaggio.save((err, data) => {
                  if (err) return res.json({ Error: err });
                    return res.status(201).json(data.Text);
                });
                }
                //Errore uno dei due non è tutor
                else{
                  res.status(400).json("Uno dei due utente non è un tutor");
                }
              }
              catch(err){
              res.status(400).json("Questo destinatario non esiste");
              }

              //Errore il destinatario non esiste
            });
          }
          catch(err){
            res.status(400).json("Questo mittente non esiste");
          }
        });
        //Errore generico dentro la funzione
      } catch (err) {
        res.status(400).json(err);
      }
}
```

## Modifica messaggio

API che gestisce la modifica di un messaggio, questo permette all'utente e al tutor di modificare i messaggi per prepararli all'invio.
In caso il messaggio non fosse ritrovato si ritorna un errore.

```
const ModificaMessaggio = (req,res) => {
    try{
        var countpre;
        Chat.countDocuments({}, function(err, count) {
          var dati = req.body;
          //Time: dati.Time
          Chat.deleteOne({Sender: dati.Sender, Receiver : dati.Receiver, Text: dati.Text },async (err) => {});
          Chat.countDocuments({}, function(err, count2) {
            if(count!=count2){
              const chat = new Chat({
                Sender: dati.Sender,
                Receiver: dati.Receiver,
                Time: dati.Time,
                Text: dati.Text2
              });
              // Salvataggio dell'utente nel database
              chat.save((err, data) => {
                if (err) return res.json({ Error: err });
                return res.status(201).json(data.Text);
                })
              //Nessun messaggio trovato
            }else{
              res.status(400).json("Nessun messaggio trovato");
            }
          });
        });
      } catch (err) {
        res.status(400).json(err);
      }
}
```

## Visualizza messaggi

API che gestisce la visualizzazione dei messaggi da parte di un utente o di un tutor.
Troviamo sempre presente un errore generico in caso di bug.

```
const VisualizzaMessaggi = (req,res) => {
    try{
        var dati = req.body;
        Chat.find({Sender : dati.ut2, Receiver: dati.ut1}, function(err,obj){
          Chat.find({Sender: dati.ut1, Receiver: dati.ut2}, function (err,obj2) {
            obj3 = obj.concat(obj2);
            obj3.sort(function(a, b) {
              return a.Time - b.Time;
            });
            let lista = "";
            let promises = [];

            for (let i = 0; i < obj3.length; i++) {
              const oggetto = obj3[i];
              promises.push(
                User.findOne({email: oggetto.Sender}).exec()
                .then(trovanome => {
                  let aa = trovanome.name.concat(": ",oggetto.Text,"\n");
                  lista = lista + aa;
                })
              );
            }

            Promise.all(promises)
            .then(() => {
              res.status(200).json(lista);
            })

          })
        });
      }
      //Errore generico dentro la funzione
      catch (err) {
        res.status(400).json(err);
      }
}
```

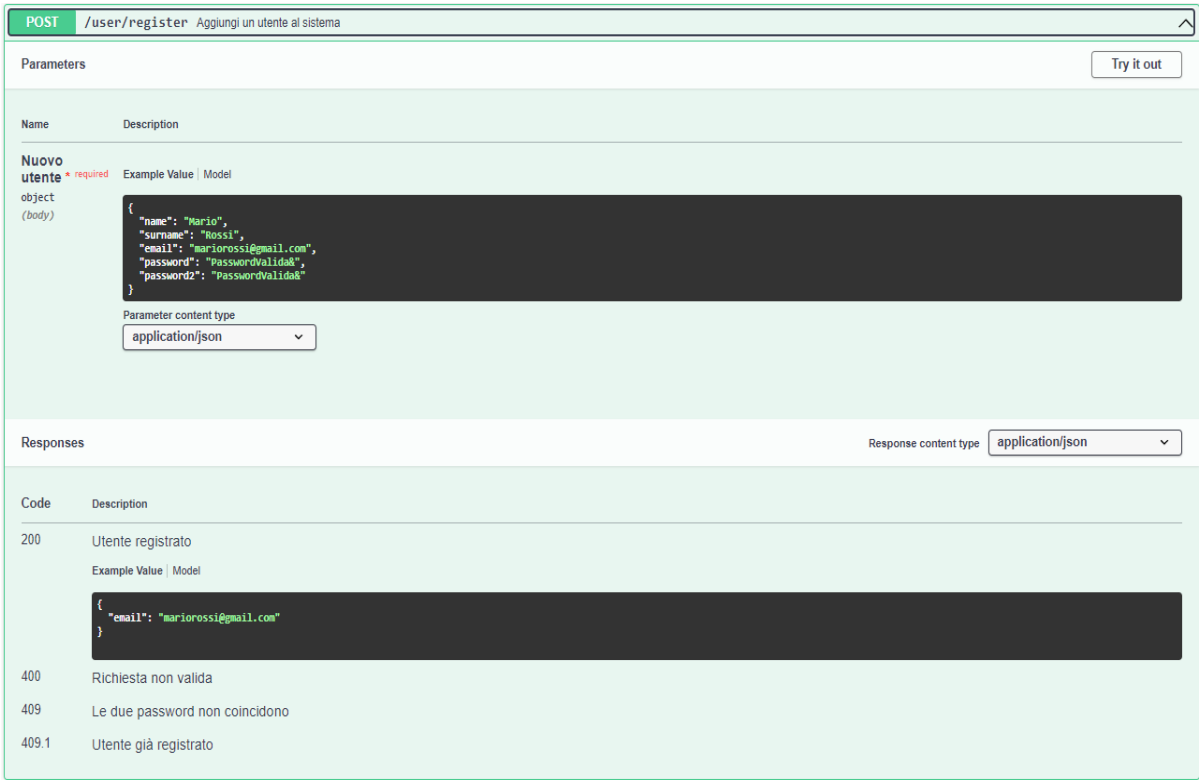
# API documentation

Le API elencate nella sezione precedente sono state documentate utilizzando il modulo swagger-ui-express, seguendo lo standard OpenAPI.
Sono stati descritti i vari endpoint e i modelli dati che abbiamo considerato finora: User, Università e Chat.

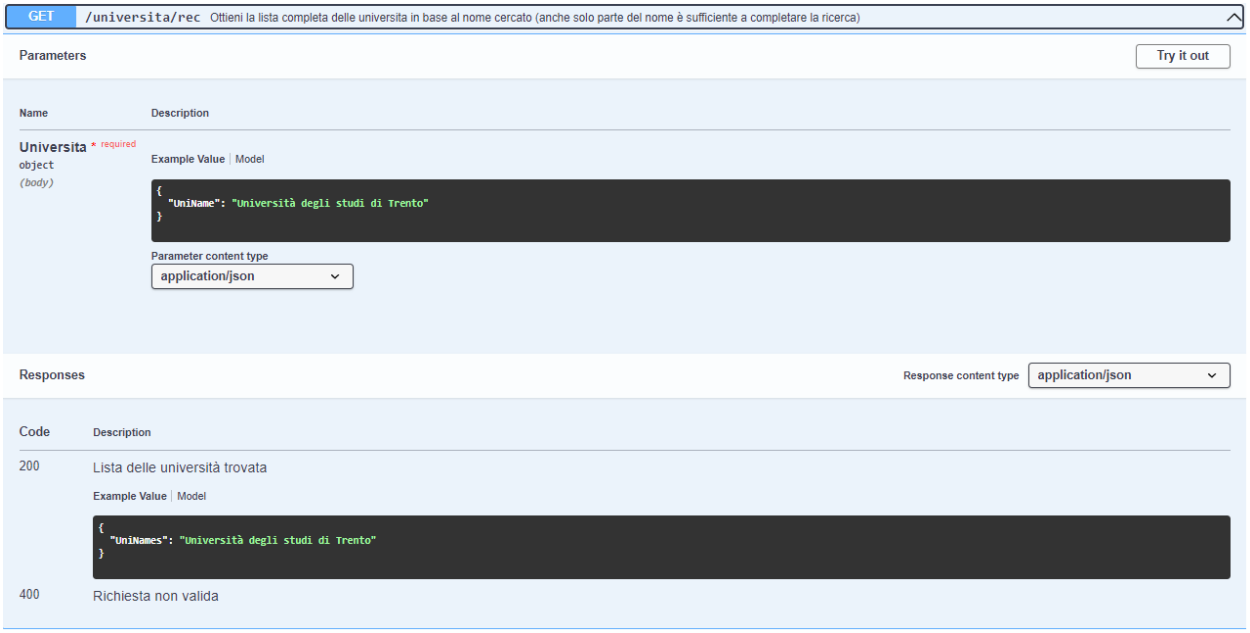
Per facilitare l'ingresso alla documentazione è possibile visitare una pagina dove è presente l'interfaccia Swagger UI, un modo semplice e chiaro per visualizzare ciò che riguarda l'API: dall'URI, dal metodo HTTP, alla struttura della richiesta e alle possibili risposte.
Questa modalità di visualizzazione di SWAGGER UI permette oltretutto il testing delle API direttamente dalla pagina web.
L'endpoint che rimanda a questa documentazione è il seguente: /api-docs.

Vengono qui riportate, oltre allo schema SWAGGER risultante, tre tipi di metodi HTTP diversi, rispettivamente POST, GET e DELETE.
Nessun tipo PATCH è stato utilizzato, sebbene per l'API /chat/modificaMessaggio sarebbe stato opportuno utilizzarlo.

POST:




GET:



DELETE:


Questi tre metodi differiscono a livello HTML.
Di seguito, inoltre, viene riportata la definizione dei modelli utilizzati.

# My User Project CRUD 1.0.0

[ Base URL: localhost:3000/ ]

My User Project Application API

MIT

Schemes

HTTP

## Utenti API per gli utenti del sistema

- POST /user/register Aggiungi un utente al sistema
- POST /user/login Effettua l'accesso al sistema
- DELETE /user/elimina Rimuovi un account dal sistema
- POST /user/tutor Richiesta per diventare tutor, una mail viene inviata all'Admin per notificare la richiesta; sarà poi l'Admin ad accertarsi della validità di questa
- POST /user/passwordDimenticata Sistema di recupero password

## Universita API per le universita del sistema

- POST /universita/NewUni Aggiungi un utente al sistema
- GET /universita/rec Ottieni la lista completa delle universita in base al nome cercato (anche solo parte del nome è sufficiente a completare la ricerca)
- PATCH /universita/pref Aggiungi o rimuovi un universita dai preferiti di un utente

## Chat API per le chat del sistema

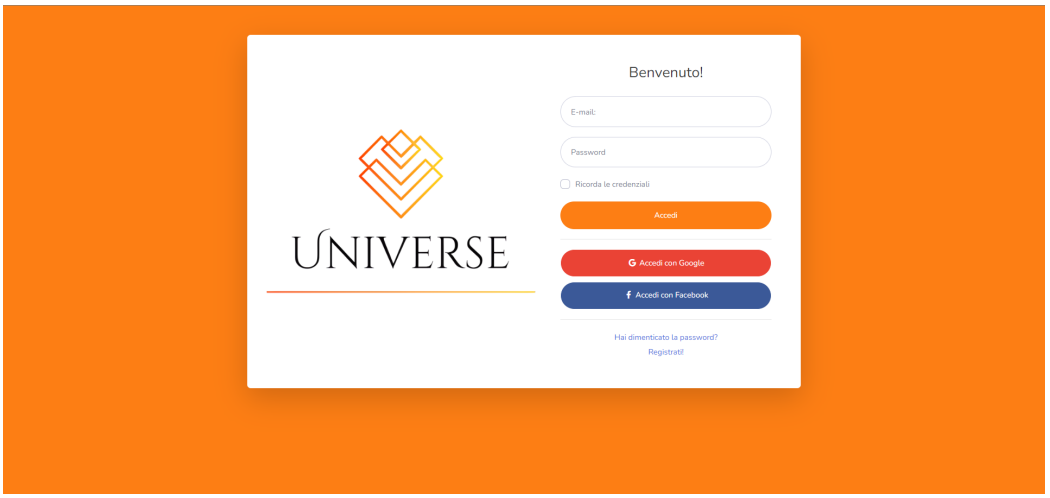
### Models

- Utente ›
- Universita ›
- Chat ›

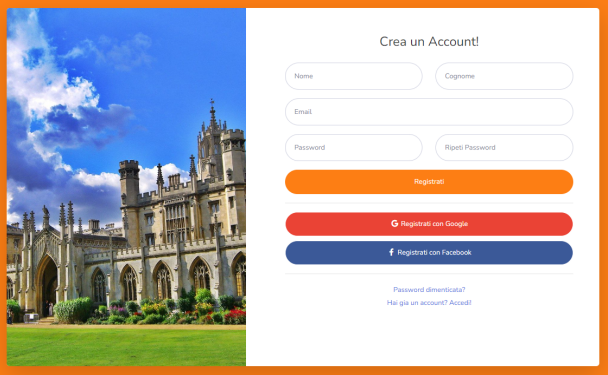
# FrontEnd Implementation

Il FrontEnd è composto dalle pagine di visualizzazione che danno modo all'utente di inserire i propri dati e usufruire dei servizi.
L'applicazione è formata dalla Pagina Home, Pagina di Login, Pagina di Registrazione, Pagina di recupero password, Pagina Principale, Pagina Help, Pagina delle Università e schermata di conferma logout.
Per quanto riguarda il FrontEnd le immagini con i loghi delle università mancano, e di conseguenza la loro visualizzazione nel sito, e verranno implementate in una versione futura del DataBase.

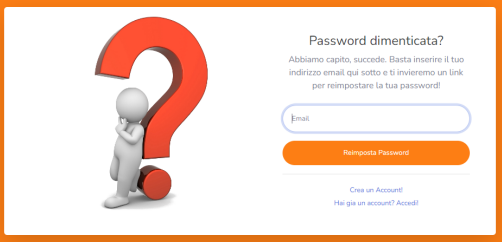
- **Pagina di Login**, dove l'utente avrà modo di inserire le proprie credenziali, oppure effettuare la registrazione.



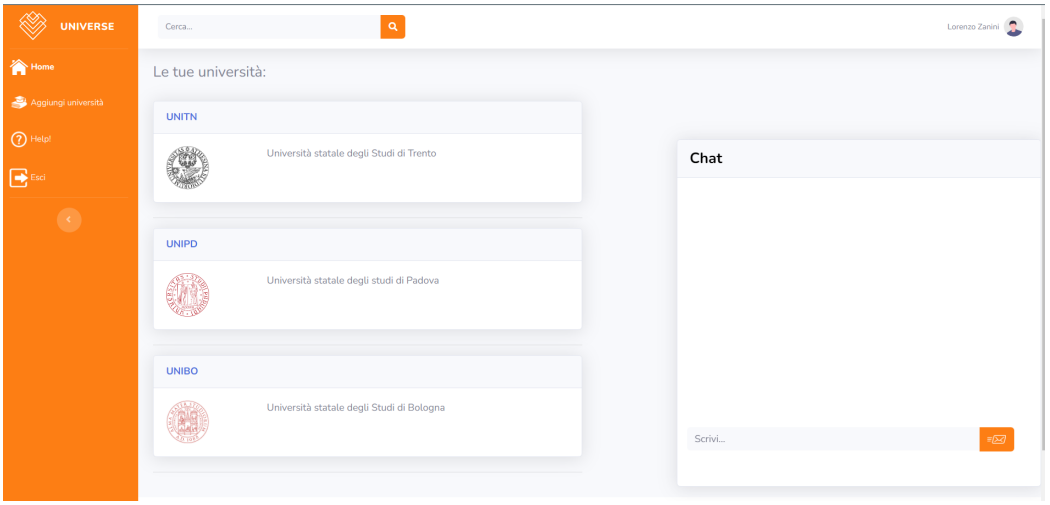
- **Pagina di Registrazione**, permette l'utente di creare un proprio spazio nel database con i propri attributi.



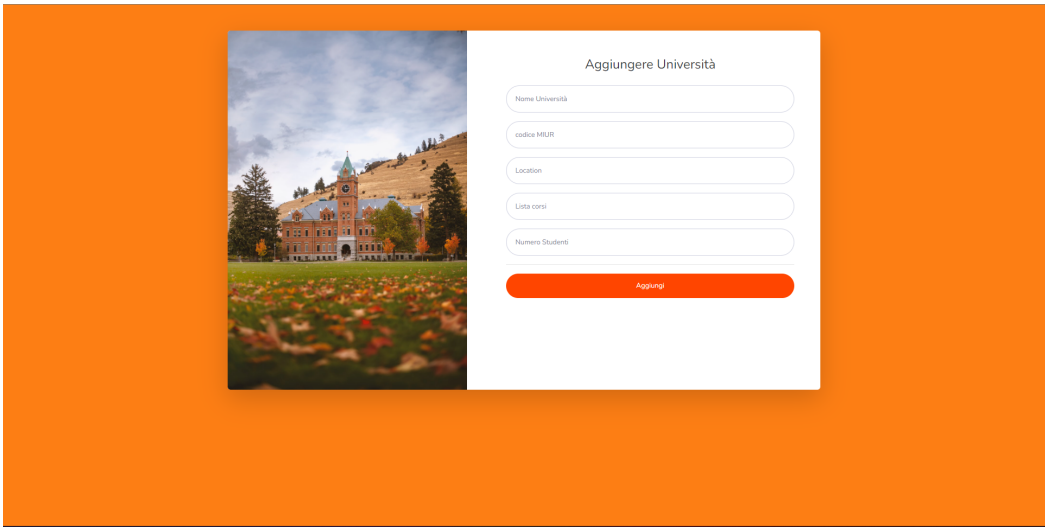

- **Pagina di recupero password,** per poter ottenere una nuova password in caso l'utente si dimenticasse della propria.



- **Pagina Principale,** dove l'utente autenticato vede le proprie università seguite, le proprie chat e il menù per usufruire dei servizi.




- **Pagina per aggiungere un università** al database, dove è possibile compilare il form e inviare le informazioni al database.



# GitHub Repository and Deployment Info

Nel seguente Link si ritrova la repository dove abbiamo salvato i file che strutturano l'applicazione.
In questo troviamo:

- Gli schemi, cioè i diagrammi costruiti con draw.io.
- Le schermate, screen del FrontEnd costruito.
- UNIverse, cioè l'applicazione.
- Le versioni che abbiamo scartato.
- I Documenti.
- Il README

https://github.com/LorenzoMase/Deliverables

La parte di BackEnd e FrontEnd sarà hostata dalla piattaforma Render.
BackEnd: https://universebackend.onrender.com
FrontEnd:
https://universefrontend.onrender.com/Index.html

# Testing

Per testare le nostre API abbiamo utilizzato il Framework jest. Per ogni API abbiamo creato il relativo file di test che servirà per fare i controlli.

## register.test

Controlliamo che l'API per la registrazione restituisca gli output previsti.

```
npm test register.test.js

> universe@1.0.0 test
> jest --coverage register.test.js

  console.log
    Your port is listening on port 3000

      at Server.log (server.js:19:13)

 PASS  testing/register.test.js
  Test per api /Register
    √ test password diverse (156 ms)
    √ test email già presente (605 ms)
    √ Registrazione corretta (95 ms)
```

## login.test

Controlliamo che l'API per il login restituisca gli output previsti.

```
npm test login.test.js

> universe@1.0.0 test
> jest --coverage login.test.js

  console.log
    Your port is listening on port 3000

      at Server.log (server.js:19:13)

 PASS  testing/login.test.js
  Test per api /login
    √ Questo utente non esiste (620 ms)
    √ Utente Loggato (66 ms)
```

## elimina.test

Controlliamo che l'API per l'eliminazione dell'account restituisca gli output previsti.

```
npm test elimina.test.js

> universe@1.0.0 test
> jest --coverage elimina.test.js

  console.log
    Your port is listening on port 3000

      at Server.log (server.js:19:13)

 PASS  testing/elimina.test.js
  Test per api /EliminaAccount
    √ Questo utente non esiste (643 ms)
    √ Utente Eliminato (65 ms)
```

## diventaTutor.test

Controlliamo che l'API per fare la richiesta di diventare un tutor restituisca gli output previsti.

```
npm test diventaTutor.test.js

> universe@1.0.0 test
> jest --coverage diventaTutor.test.js

  console.log
    Your port is listening on port 3000

      at Server.log (server.js:19:13)

 PASS  testing/diventaTutor.test.js
  Test per api /DiventaTutor
    √ Email non presente (579 ms)
    √ Email inviata (96 ms)
```

## passwordDimenticata.test

Controlliamo che l'API per fare la richiesta di ricevere la propria password per e-mail restituisca gli output previsti.

```
npm test passwordDimenticata.test.js
> universe@1.0.0 test> jest --coverage passwordDimenticata.test.js

  console.log
    Your port is listening on port 3000

      at Server.log (server.js:19:13)

 PASS  testing/passwordDimenticata.test.js
  Test per api /PasswordDimenticata
    √ Email non presente nel database (612 ms)
    √ Richiesta inviata (84 ms)
```

## nuovaUni.test

Controlliamo che l'API per inserire una nuova università nel database restituisca gli output previsti.

```
npm test nuovaUni.test.js

> universe@1.0.0 test
> jest --coverage nuovaUni.test.js

  console.log
    Your port is listening on port 3000

      at Server.log (server.js:19:13)

 PASS  testing/nuovaUni.test.js
  Test per api /NuovaUni
    √ Università già presente (646 ms)
    √ Non sei un admin (34 ms)
    √ Università inserita correttamente (118 ms)
```

## visualizzaMess.test

Controlliamo che l'API per visualizzare tutti i messaggi tra due utenti restituisca gli output previsti.

```
npm test visualizzaMess.test.js

> universe@1.0.0 test
> jest --coverage visualizzaMess.test.js

  console.log
    Your port is listening on port 3000

      at Server.log (server.js:19:13)

 PASS  testing/visualizzaMess.test.js
  Test per api /VisualizzaMessaggi
    √ Visualizza i messaggio (696 ms)
```

## scriviMessaggio.test

Controlliamo che l'API per scrivere un messaggio restituisca gli output previsti.

```
npm test scriviMessaggio.test.js

> universe@1.0.0 test
> jest --coverage scriviMessaggio.test.js

  console.log
    Your port is listening on port 3000

      at Server.log (server.js:19:13)

 PASS  testing/scriviMessaggio.test.js
  Test per api /Messaggio
    √ Uno dei due untenti non è un tutor (743 ms)
    √ Destinatario non esistente (77 ms)
    √ Mittente non esistente (36 ms)
    √ Messaggio inviato (121 ms)
```

## modificaMessaggio.test

Controlliamo che l'API per modificare un messaggio restituisca gli output previsti.

```
npm test modificaMessaggio.test.js

> universe@1.0.0 test
> jest --coverage modificaMessaggio.test.js

  console.log
    Your port is listening on port 3000

      at Server.log (server.js:19:13)

 PASS  testing/modificaMessaggio.test.js
  Test per api /ModificaMessaggio
    √ Nessun messaggio trovato (770 ms)
    √ Messaggio modificato (278 ms)
```

## ricercaUni.test

Controlliamo che l'API per ricerca un' università nel database restituisca gli output previsti.

```
npm test ricercaUni.test.js

> universe@1.0.0 test
> jest --coverage ricercaUni.test.js

  console.log
    Your port is listening on port 3000

      at Server.log (server.js:19:13)

 PASS  testing/ricercaUni.test.js
  Test per api /RicercaUni
    √ Lista università trovate (585 ms)
```