VisualCue

# Indice

[1 Indice 2](#_Toc165643659)

[2 Introduzione 4](#_Toc165643660)

[2.1 Informazioni sul progetto 4](#_Toc165643661)

[2.2 Abstract 4](#_Toc165643662)

[2.3 Scopo 4](#_Toc165643663)

[3 Analisi 5](#_Toc165643664)

[3.1 Analisi del dominio 5](#_Toc165643665)

[3.2 Analisi e specifica dei requisiti 5](#_Toc165643666)

[3.3 Use case 11](#_Toc165643667)

[3.4 Pianificazione 12](#_Toc165643668)

[3.4.1 Analisi 12](#_Toc165643669)

[3.4.2 GUI 12](#_Toc165643670)

[3.4.3 Configurazioni ambiente di sviluppo 12](#_Toc165643671)

[3.4.4 Algoritmo 12](#_Toc165643672)

[3.5 Analisi dei mezzi 14](#_Toc165643673)

[3.5.1 Software 14](#_Toc165643674)

[3.5.2 Hardware 14](#_Toc165643675)

[3.5.3 Web server 14](#_Toc165643676)

[4 Progettazione 15](#_Toc165643677)

[4.1 Design procedurale 15](#_Toc165643678)

[4.2 Design dei dati e database 19](#_Toc165643679)

[4.3 Design delle interfacce 20](#_Toc165643680)

[4.3.1 Login 20](#_Toc165643681)

[4.3.2 Home 21](#_Toc165643682)

[4.3.3 Chat 22](#_Toc165643683)

[4.3.4 Crea 23](#_Toc165643684)

[4.3.5 Profilo 24](#_Toc165643685)

[4.3.6 Scopri 25](#_Toc165643686)

[5 Implementazione 26](#_Toc165643687)

[5.1 package.json 26](#_Toc165643688)

[5.2 index.js 27](#_Toc165643689)

[5.3 .env 28](#_Toc165643690)

[5.4 routes.js 28](#_Toc165643691)

[5.5 Models 29](#_Toc165643692)

[5.6 Middelwares 30](#_Toc165643693)

[5.7 Controllers 35](#_Toc165643694)

[5.7.1 Auth.js 35](#_Toc165643695)

[5.7.2 Home.js 36](#_Toc165643696)

[5.7.3 Catalog.js 38](#_Toc165643697)

[5.7.4 Create.js 38](#_Toc165643698)

[5.7.5 Discover.js 39](#_Toc165643699)

[5.7.6 Profile.js 40](#_Toc165643700)

[5.7.7 Comment.js 43](#_Toc165643701)

[5.7.8 Signup.js 45](#_Toc165643702)

[5.7.9 Login.js 47](#_Toc165643703)

[5.8 Utils 47](#_Toc165643704)

[5.9 Javascript 48](#_Toc165643705)

[5.9.1 Home.js 48](#_Toc165643706)

[5.9.2 GenPost.js 48](#_Toc165643707)

[5.9.3 Create.js 49](#_Toc165643708)

[5.9.4 Discover.js 51](#_Toc165643709)

[6 Test 52](#_Toc165643710)

[6.1 Protocollo di test 52](#_Toc165643711)

[6.2 Mancanze/limitazioni conosciute 90](#_Toc165643712)

[7 Consuntivo 90](#_Toc165643713)

[8 Conclusioni 93](#_Toc165643714)

[8.1 Sviluppi futuri 93](#_Toc165643715)

[8.2 Considerazioni personali 93](#_Toc165643716)

[9 Bibliografia 95](#_Toc165643717)

[9.1 Sitografia 95](#_Toc165643718)

[10 Glossario 95](#_Toc165643719)

[11 Indice delle figure 96](#_Toc165643720)

[12 Allegati 96](#_Toc165643721)

# Introduzione

## Informazioni sul progetto

Allievo: Riccardo Deiuri

Docente: Geo Petrini

Luogo: Scuola Arti e Mestieri di Trevano, TI

Sezione: Informatica

Data di inizio: 04.02.2025

Data di consegna/fine: 04.04.2025

## Abstract

The creation of the VisualCue web application revolutionizes the way users memorize and learn new terms by combining text and images into customizable flashcards. Users can create flashcards with custom content by uploading their own or having them generated. Entire collections of cards can also be automatically generated from user-supplied text input, with images created by a built-in algorithm.

VisualCue offers several test modes, such as memory matching, multiple choice, and timed challenges, to accommodate different learning styles. Users can organize their papers into private or public collections and engage in real-time challenges, fostering a competitive and interactive learning environment.

This innovative application leverages visual association to enhance memory retention, making it an effective and engaging tool for students of all ages and subjects.

## Scopo

Il progetto ha due scopi principali, il primo scopo, quello operativo, è quello di creare un social network innovativo, questo social permette di postare immagini, audio e testo, anche tutti e tre gli elementi insieme, permettere anche di visualizzare quale tipo di post pubblichi maggiormente sul tuo profilo attraverso un tag che bisogna inserire nel post.

Il secondo scopo è quello didattico, ovvero imparare a gestire un progetto di gruppo, perché non avendo mai lavorato in gruppo su un progetto di questa lunghezza e durata.

Questo progetto ci farà crescere molto sotto l’aspetto formativo.

# Analisi

## Analisi del dominio

L’applicativo Web da realizzare è un social-network indirizzato ad utenti di tutte le età, esistono già applicativi di questo tipo ben costruiti, però rispetto a questi altri implementa molte funzioni interessanti, tra cui l’aggiunta di post (audio, immagini, testo), personalizzazione del profilo, chat con altri utenti e scoprire altri utenti in modo innovativo, tutto questo in un unico social.  
Essi potranno accedere tramite e-mail da qualsiasi browser, la connessione è garantita con efficienza solo in Europa (sede del server).

## Analisi e specifica dei requisiti

**Priorità 1**

|  |  |
| --- | --- |
| **Req-001** | |
| **Nome** | Creazione interfaccia |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di creare un’interfaccia grafica dell’applicativo web |
| **Sotto requisiti** | |
| **Req-001\_1** | Creare pagina Home |
| **Req-001\_2** | Creare pagina Chat |
| **Req-001\_3** | Creare pagina Scopri |
| **Req-001\_4** | Creare pagina Crea |
| **Req-001\_5** | Creare pagina Profilo |
| **Req-001 6** | Creare pagina Login/Sing-up |

|  |  |
| --- | --- |
| **Req-002** | |
| **Nome** | Creare e configurare webserver |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di realizzare il webserver con tutto il necessario (vedere sotto requisiti) |
| **Sotto requisiti** | |
| **Req-002\_1** | Creare utenti sul server |
| **Req-002\_2** | Installare e configurare MySQL |
| **Req-002\_3** | Configurare IP e SSH |
| **Req-002\_4** | Installare Node |

|  |  |
| --- | --- |
| **Req-003** | |
| **Nome** | Creazione database |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di creare il database |
| **Sotto requisiti** | |
| **Req-003\_1** | Creare utenti |
| **Req-003\_2** | Creare tabelle |
| **Req-003\_3** | Creazione script di anonimizzazione dati e db per i test/monitoraggio/statistica. |

|  |  |
| --- | --- |
| **Req-004** | |
| **Nome** | Algoritmo per “Home” |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di creare l’algoritmo per far funzionare la sezione Home, ovvero la visualizzazione dei post. |

|  |  |
| --- | --- |
| **Req-005** | |
| **Nome** | Algoritmo per “Crea” |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di creare l’algoritmo per far funzionare la sezione Crea, ovvero la creazione di un post. |
| **Sotto requisiti** | |
| **Req-008\_1** | Implementare le immagini |
| **Req-008\_2** | Implementare le audio |
| **Req-008\_3** | Implementare le testo |
| **Req-008\_3** | Implementare le video |
| **Req-008\_3** | Creazione post tramite Drag&Drop |

|  |  |
| --- | --- |
| **Req-006** | |
| **Nome** | Eliminazione dei post |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Gli utenti possono modificare i propri post. |

|  |  |
| --- | --- |
| **Req-007** | |
| **Nome** | Implementazione login e signup |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di implementare il login e signup |

|  |  |
| --- | --- |
| **Req-008** | |
| **Nome** | Criptazione dati sensibili |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di criptare i dati sensibili, come ad esempio la password dell’utente |

|  |  |
| --- | --- |
| **Req-009** | |
| **Nome** | Connessione database |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessita di connettere il database in modo da utilizzarlo per l’applicativo web. |

|  |  |
| --- | --- |
| **Req-010** | |
| **Nome** | Modifica del profilo |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Gli utenti possono modificare il proprio profilo. |

**Priorità 2**

|  |  |
| --- | --- |
| **Req-011** | |
| **Nome** | Algoritmo per “Scopri” |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Si necessita di creare l’algoritmo per far funzionare la sezione Scopri, ovvero la visualizzazione dei profili consigliati e la possibilità di scegliere se seguirli. |

|  |  |
| --- | --- |
| **Req-012** | |
| **Nome** | Configurazione sicurezza del server |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Si necessita di configurare il server in modo che sia il più sicuro possibile |
| **Sotto requisiti** | |
| **Req-004\_1** | Monitoraggio delle richieste e stato dei servizi del server. |

**Priorità 3**

|  |  |
| --- | --- |
| **Req-013** | |
| **Nome** | Gestione file (immagini, video, audio) con NAS |
| **Priorità** | 3 |
| **Versione** | 1.0 |
| **Note** | Si necessita di gestire i file con un NAS. |

|  |  |
| --- | --- |
| **Req-014** | |
| **Nome** | Algoritmo per “Chat” |
| **Priorità** | 3 |
| **Versione** | 1.0 |
| **Note** | Si necessita di creare l’algoritmo per far funzionare la sezione Chat, ovvero le chat scambiate con gli altri utenti. |
| **Sotto requisiti** | |
| **Req-007\_1** | Salvataggio delle chat nel DB |
| **Req-007\_2** | Implementare immagini e audio |

**Spiegazione elementi tabella dei requisiti:**

**ID**: identificativo univoco del requisito

**Nome**: breve descrizione del requisito

**Priorità**: indica l’importanza di un requisito nell’insieme del progetto, definita assieme al committente. Ad esempio, poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

**Versione**: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l’ultima versione, mentre le vecchie dovranno essere inserite nei diari.

**Note**: eventuali osservazioni importanti o riferimenti ad altri requisiti.

**Sotto requisiti**: elementi che compongono il requisito.

## Use case

I casi d’uso rappresentano l’interazione tra i vari attori e le funzionalità del prodotto.

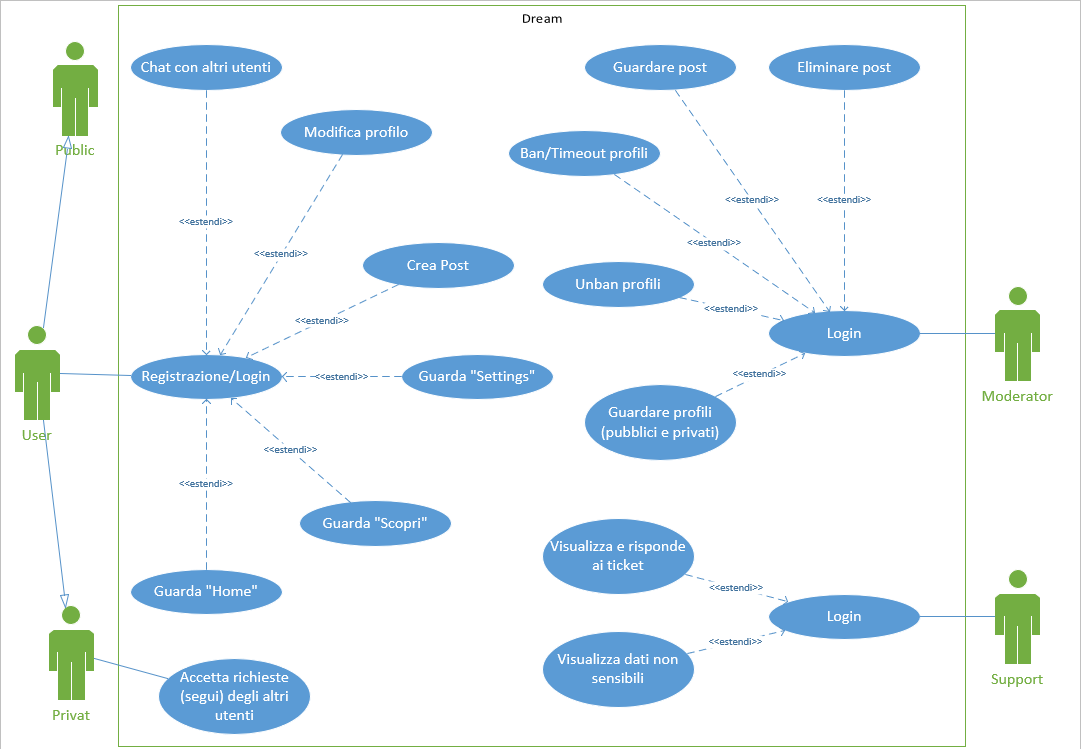


Figura 1 Use case

## Pianificazione

La pianificazione di questo progetto è stata fatta tramite il Gantt preventivo utilizzando il metodo di pianificazione agile, la pianificazione tramite Gantt viene usata come se fosse una linea guida essendo che le tempistiche vengono segnalate dalle milestone (sprint) definiti insieme al committente del progetto.

Il progetto suddiviso in quattro parti fondamentali che sono:

* Analisi
* GUI
* Configurazioni varie
* Algoritmo

### Analisi

La parte di analisi serve per capire le specifiche che deve avere il progetto discusse con il committente, creare l’Abstract e creare gli schemi del progetto che sono:

* Qdc
* Gantt
* Use case
* Diagramma ER
* Activity diagram

### GUI

La parte di GUI (interfaccia grafica) serve per creare le interfacce del nostro applicativo, inizialmente le pagine vengono solo disegnate (parte design) con mockflow per poi essere implementate nel linguaggio html.

### Configurazioni ambiente di sviluppo

La parte di configurazioni serve per installare il server e configuralo in modo da essere utilizzato, lo stesso discorso vale per nodeJS che verrà installato e configurato sul server.

### Algoritmo

La parte di algoritmo serve per definire la parte back-end del codice, configurare nodeJS (linguaggio utilizzato), implementare le varie funzionalità che avrà il nostro applicativo, far comunicare le pagine fra di loro e testare tutto ciò che è stato implementato.

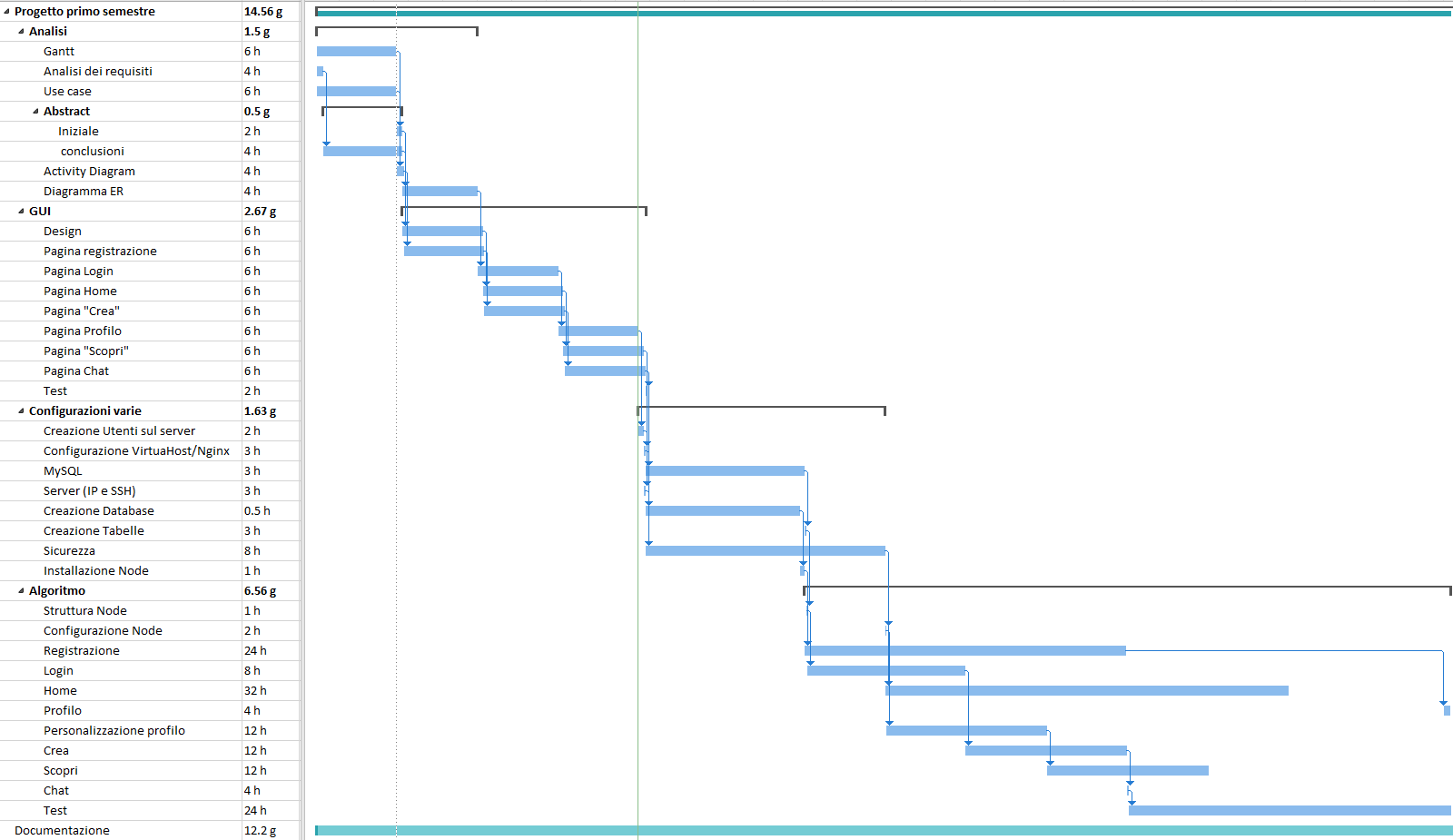


Figura 2 Gantt preventivo

## Analisi dei mezzi

### Software

Visual Studio Code: versione 1.87.2

Browser microsoft edge (indifferente il browser)

Node js: versione 20.11.0

Librerie Node Js produttivo:

* bcrypt: versione 5.1.1
* cookie-parser: versione 1.4.6
* dotenv: versione 16.4.1
* ejs: versione 3.1.9
* express: versione 4.18.2
* jsonwebtoken: versione 9.0.2
* multer: 1.4.5-lts.1
* mysql2: versione 3.9.1
* nodemon: versione 3.0.3
* sequelize: versione 6.37.1

Librerie Node Js development:

* jest: versione 29.7.0

### Hardware

#### Hardware Sviluppo

Gli Hardware di sviluppo sono dei computer forniti dalla scuola con queste specifiche:

* OS: Windows 10, 64 bit
* Processore: Intel i7-7700, 3.60GHz, 4 Cores
* Ram: 16 GB, 2400 MHz
* Scheda grafica: Intel HD Graphics 630, 8 GB
* Display: HP E24i G4, 1920x1200, 60Hz

#### Hardware Prodotto

Come Hardware prodotto serve una macchina che si possa collegare a internet e un browser per poter accedere al sito del nostro applicativo

### Web server

* Node js
* Memoria: 146 GiB
* Processore: Intel(R) Xeon(R) CPU E5640 @ 2.67GHz

# Progettazione

## Design procedurale

Activity Diagram che rappresentano i 4 principali usi del software (Discover, Home, Login e Register), per farli è stato usato il sito web Lucid app (nella sitografia il link).

***Discover***

Il primo Activity Diagram è quello del discover (o scopri), una volta nella pagina scopri, se tutti gli utenti sono già stati passati o seguiti uscirà la scritta “No users yet” e sarà possibile solo tornare indietro e vedere gli account già visti, se invece ci sono utenti ancora non scoperti sarà possibile seguirli e vedere i loro post oppure decidere di passare al prossimo utente.

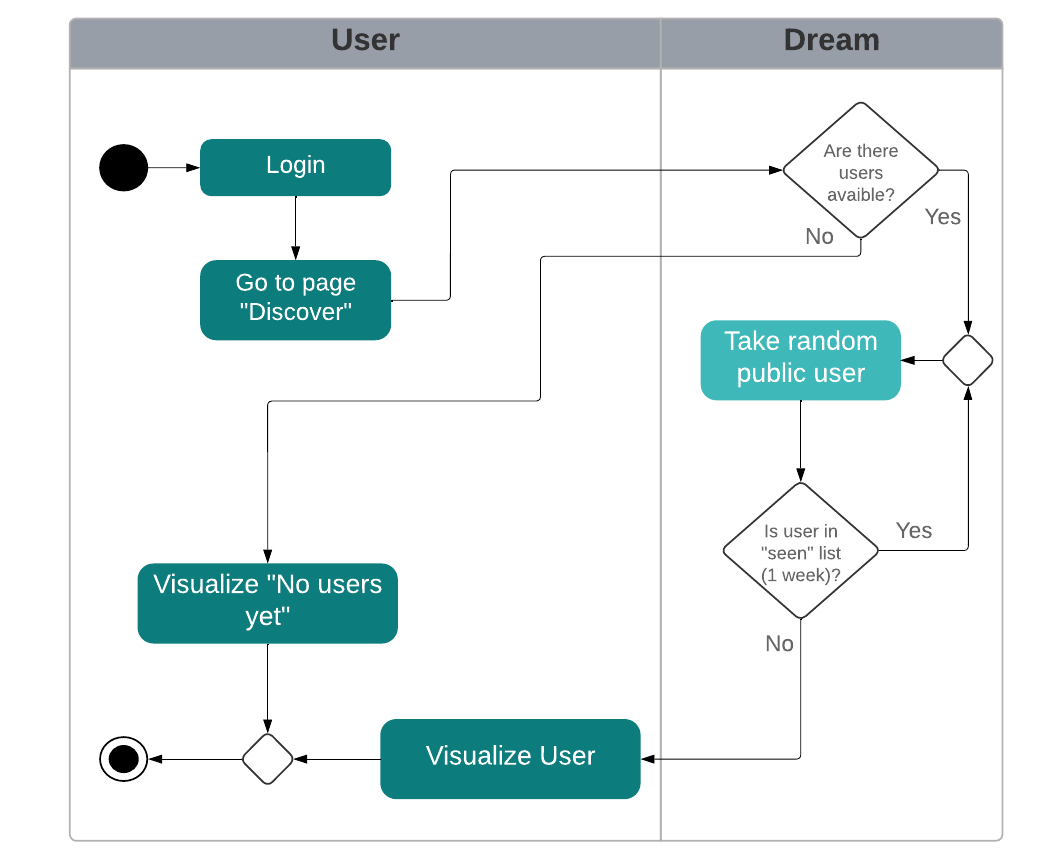


Figura 3 Discover Activity Diagram

***Home***

Il secondo Activity Diagram è quello della home, una volta nella pagina della home verranno caricati direttamente dal db i post che gli utenti che segui hanno pubblicato, se si arriva alla fine dei post perché non ce ne sono più verrà visualizzata la scritta “No post yet”.

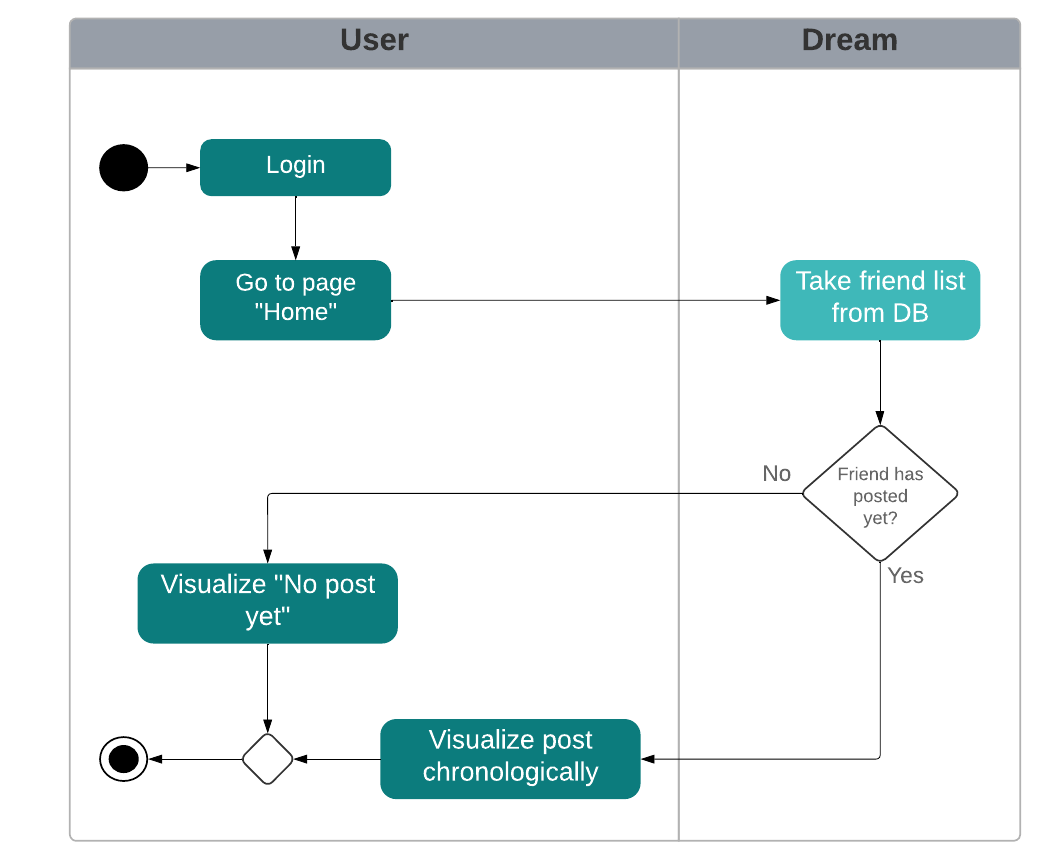


Figura 4 Home Activity Diagram

***Login***

Il terzo activity Diagram è quello del login, una volta nella schermata di login bisognerà immettere i dati e premere il bottone per effettuare il login, se i dati sono corretti, si accederà al proprio account, se i dati non sono corretti o inesistenti verrà mostrato un messaggio di errore e bisognerà riprovare.

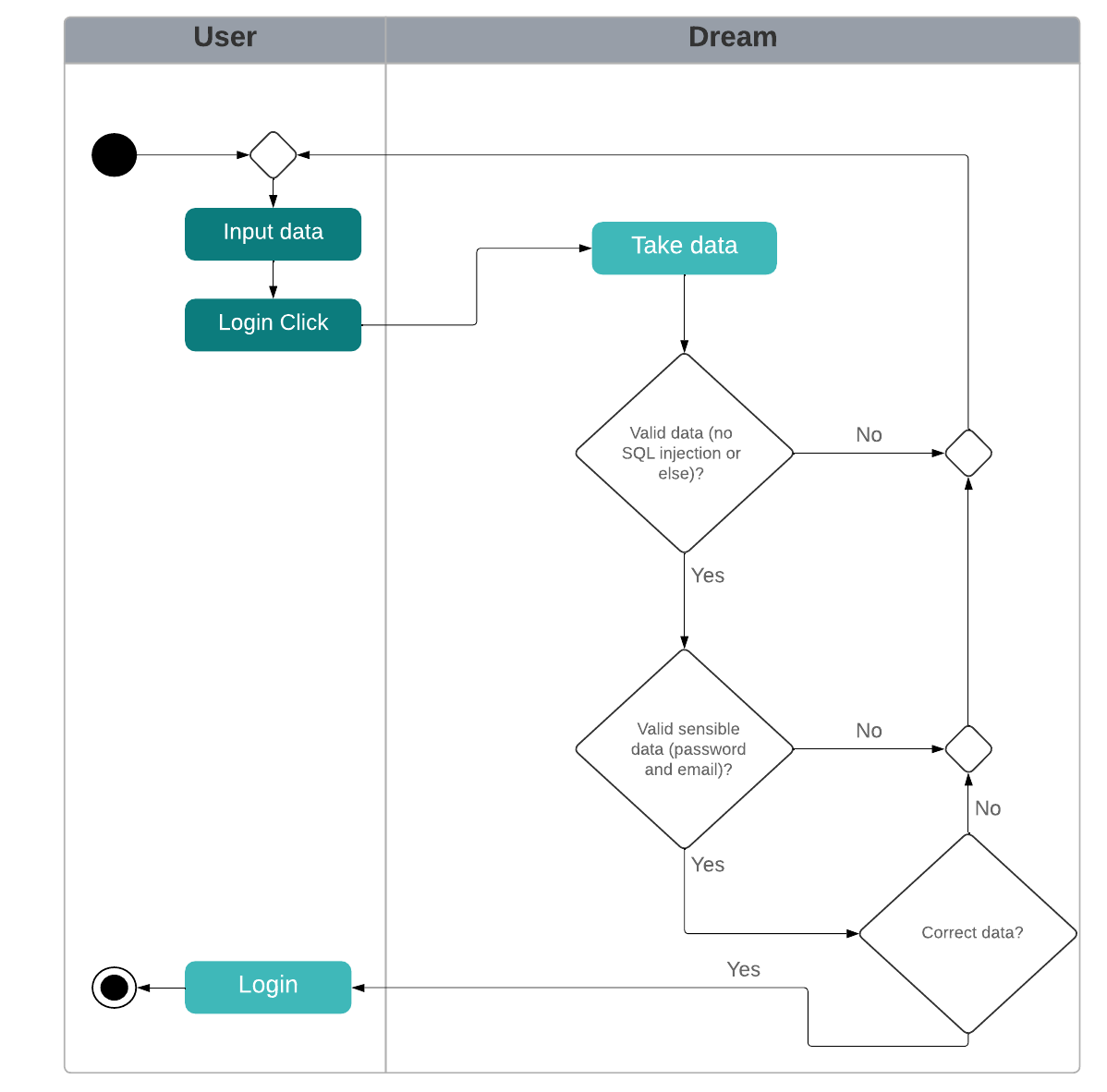


Figura 5 Login Activity Diagram

***Register***

Il quarto activity Diagram è quello del Register che è molto simile a quello del login, infatti una volta sulla pagina di registrazione bisognerà immettere i dati, se i dati sono corretti la registrazione verrà effettuata correttamente, se i dati inseriti non vanno bene (es. password troppo corta o mail senza dominio, …), verrà mostrato un messaggio di errore e si richiederà di riprovare, ovviamente l’utente non verrà creato.

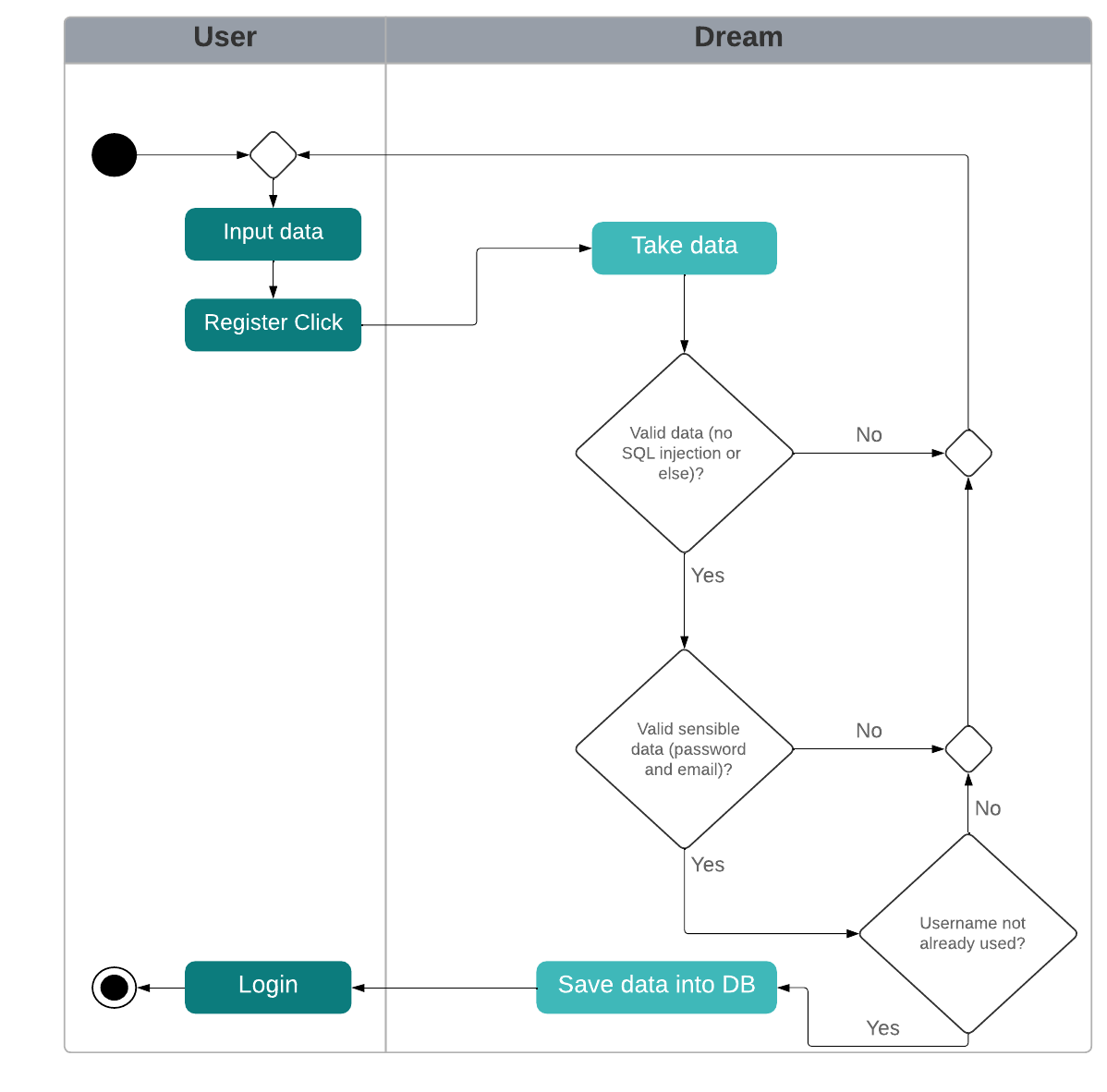


Figura 6 Register Activity Diagram

## Design dei dati e database

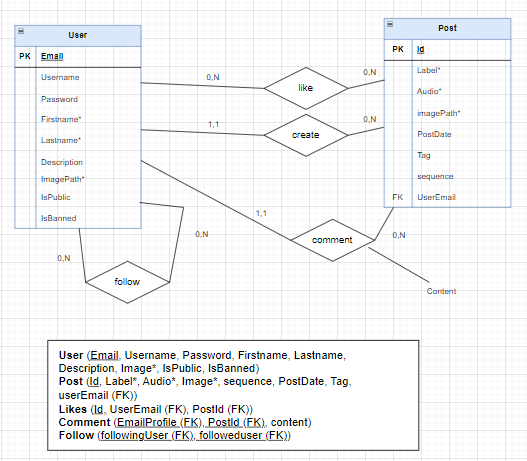


Figura 7 diagramma Er

## Design delle interfacce

Essendo un applicativo che funziona sia via PC che via mobile le interfacce presentano una leggera differenza a dipendenza del dispositivo utilizzato, infatti le interfacce sostanzialmente sono tutte uguali nella forma, infatti troviamo il nome dell’applicativo in alto a sinistra, uno spazio centrale dove verranno effettuate varie azioni a dipendenza della modalità che viene scelta.

La differenza tra i due dispositivi sta nella disposizione dei pulsanti con cui scegliere la modalità, infatti su PC li troviamo a sinistra dello spazio centrale, mentre su mobile li troviamo in basso alla schermata.

I pulsanti sono: “Home” , “Chat”  , “Crea” , “Scopri”  e “Profilo” 

### Login

Figura 8 Design pagina Login

Questa sarà la pagina di login, troviamo due campi di testo dove inserire username e password e un pulsante Login per effettuare appunto il login, sotto il pulsante troviamo una scritta “create an account” in rosso che una volta cliccata porterà alla pagina di Registrazione.

### Home

Figura 9 Design pagina Home

Questa sarà la pagina della modalità “Home” , in questa pagina possiamo uno spazio in cui verranno inseriti i post, i tasti per mettere mi piace, commentare e per condividere il post.

### Chat

Figura 10 Design pagina Chat

Questa sarà la pagina per la modalità “Chat” , troviamo nello spazio in centro in alto il nome e la foto profilo dell’ utente con cui ci stiamo scambiando i messaggi, in mezzo i messaggi che ci siamo scambiati e in basso un campo per scrivere il messaggio da inviare.

### Crea

Questa sarà la pagina per la modalità “Crea” , troviamo nello spazio centrale la possibilità di creare il tuo Dream con sotto un campo di testo per inserire dei tag e in basso a destra un pulsante per condividere il Dream (post).

#### PC

Figura 11 Design pagina PC Crea

#### Mobile

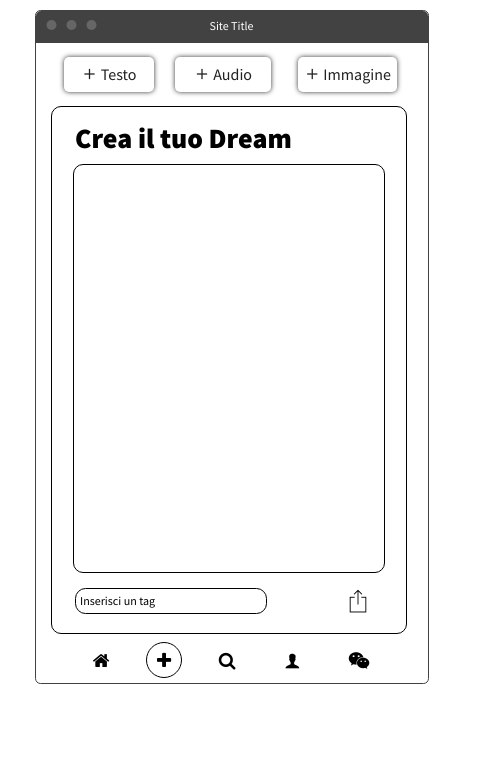


Figura 12 Design pagina Mobile Crea

### Profilo

Figura 13 Design pagina Profilo

Questa sarà la pagina per la modalità “Profilo” , troviamo nello spazio centrale la propria foto profilo con il proprio nome, la descrizione del profilo, la mail, e il nome utente e un pulsante per visualizzare i post, di fianco ad ogni campo troviamo una matita che è in realtà un pulsante per modificare il campo.

### Scopri

Figura 15 Design pagina Scopri

Figura 14 Design pagina Scopri

Questa sarà la pagina della modalità “Scopri” , troviamo nello spazio centrale un account con la foto profilo, il suo nome, la sua descrizione e un post, a destra e a sinistra dell’interfaccia delle frecce che permettono di andare al prossimo account o tornare a quello precedente.

# Implementazione

## package.json

Questo file contiene le informazioni sul progetto, tra le più importanti:

* dependecies: elenco delle dipendenze necessarie per il progetto. Ogni dipendenza è elencata con il suo nome e la versione minima richiesta.
* devDependecies: dipendenze necessarie solo per lo sviluppo, come ad esempio Jest per i test.
* scripts: definisce gli script di avvio e di test per il progetto. Il comando start viene eseguito per avviare il server usando Nodemon, che ricarica automaticamente il server quando vengono apportate modifiche. Il comando test esegue i test usando Jest

{

  …

  "dependencies": {

    "bcrypt": "^5.1.1",

    "cookie-parser": "^1.4.6",

    "dotenv": "^16.4.1",

    "ejs": "^3.1.9",

    "express": "^4.18.2",

    "jsonwebtoken": "^9.0.2",

    "multer": "^1.4.5-lts.1",

    "mysql2": "^3.9.1",

    "nodemon": "^3.0.3",

    "sequelize": "^6.37.1"

  },

  "scripts": {

    "start": "nodemon index.js",

    "test": "jest"

  },

  "devDependencies": {

    "jest": "^29.7.0"

  }

}

## index.js

Questo file configura il server web utilizzando Express in Node.js, definisce i middleware necessari per gestire le richieste HTTP e il routing delle richieste alle varie parti dell'applicazione, in questo caso il file “controllers/auth” per le richieste “/api” e “routes/routes” per le altre richieste (per esempio i link alle pagine).

Inoltre vengono pubblicate le cartelle pubbliche come: “js”, “css” e “attaches”.

Infine, avvia il server in ascolto sulla porta specificata, in questo caso la porta presente nel file “.env” oppure la 5000.

// Const

const express = require("express");

const cookie = require("cookie-parser");

const dotenv = require("dotenv").config();

const PORT = process.env.PORT || 5000;

const app = express();

app.use("/js", express.static(\_\_dirname + "/public/js"));

app.use("/css", express.static(\_\_dirname + "/public/css"));

app.use("/attaches", express.static(\_\_dirname + "/public/attaches"));

app.set("view engine", "ejs");

app.set("views", "./views");

app.use(cookie());

app.use(express.json());

app.use("/api", require("./controllers/auth"));

app.use("/", require("./routes/routes"));

app.listen(PORT);

## .env

Questo è il file di configurazione, viene utilizzato da noi per memorizzare le variabili d'ambiente del progetto che sono utilizzate nel codice per rendere più semplice configurare l'applicazione, consentendo di modificare facilmente le impostazioni senza dover modificare il codice sorgente direttamente.

# DATABASE

DB\_HOST = localhost

DB\_USER = root

DB\_PASSWORD = root

DB\_NAME = Dream\_test

DB\_DIALECT = mysql

# SERVER

PORT = 3000

# BCRYPT

SALT = "$2b$10$3DNi4nr/nLMDP.y2.8x84u"

# JWT

JWT\_SECRET = "{wZ9B+a$)-VG//\_#K-iApt5SNF{TdhbhtWbSG:\_\*i!F76d]HU46qvY&\*d[],yp\*S?btv!$aHLxjHDWeCcZK+7NuM77"

JWT\_EXPIRES = 8h

# COOKIE

COOKIE\_EXPIRES = 8

# PATH LINUX

# FILE\_PATH = "/dati/"

# PATH WINDOWS

FILE\_PATH = "..\..\6\_Database\dati\"

## routes.js

Questo codice definisce le route dell'applicativo utilizzando Express Router.

Questo file viene esportato per essere utilizzato nell'applicazione principale come modulo router.

Esempio di route:

router.get("/login", logout, (req, res) => {

  res.sendFile("login.html", { root: "./views/" });

});

questo codice specifico gestisce le richieste GET per la pagina di login.

## Models

Questa cartella permette di tenere il codice organizzato e pulito, infatti contiene al suo interno i modelli di dati, quest’ultimi definiscono la struttura dei dati e possono includere attributi, relazioni con altri modelli e metodi per interagire con i dati.

Tra le altre funzioni che hanno i file inseriti in questa cartella ci sono le iterazioni con il database, la validazione dei dati e la relazione tra i dati.

I models forniscono un'astrazione del database, facilitano l'organizzazione del codice e garantiscono l'integrità e la coerenza dei dati all'interno dell'applicazione.

Esempio di Model:

***likes.js:***

// Table likes

const Likes = db.define(

  "likes",

  {

    postId: {

      type: DataTypes.INTEGER,

      allowNull: false,

      primaryKey: true,

      references: {

        model: "post",

        key: "id",

      },

    },

…

***database.js***

const { Sequelize } = require("sequelize");

const dotenv = require("dotenv").config();

const sequelize = new Sequelize({

  dialect: process.env.DB\_DIALECT,

  host: process.env.DB\_HOST,

  username: process.env.DB\_USER,

  password: process.env.DB\_PASSWORD,

  database: process.env.DB\_NAME,

  logging: true,

});

module.exports = sequelize;

## Middelwares

***checkBannedUser.js***

Questo codice verifica se l'utente che ha effettuato una richiesta è stato bannato. Utilizza la funzione bannedUser per controllare se l'email dell'utente è presente nell'elenco dei ban.

Se l'utente è stato bannato, viene restituito un file HTML "banned.html", in caso contrario, la richiesta viene passata alla funzione successiva. Se si verifica un errore durante il controllo, viene restituito un file HTML di errore "400.html" con uno status code 400.

// Require functions or packages

const bannedUser = require("../controllers/utils/bannedUser");

const checkBannedUser = async (req, res, next) => {

  try {

    if(await bannedUser(req.user.email)){

      return res.sendFile("banned.html", { root: "./views/errors" });

    }

    next();

  } catch (error) {

    return res.status(400).sendFile("400.html", { root: "./views/errors" });

  }

};

module.exports = checkBannedUser;

***checkLogin.js***

Questo codice controlla se l'utente è autenticato e ha effettuato l'accesso.

Utilizza un token JWT presente nei cookie della richiesta per verificare l'autenticità dell'utente. Se il token è assente o non valido, restituisce una risposta di errore (401), se il token è valido, verifica se l'utente esiste nel database, se l'utente esiste, aggiunge i dati dell'utente all'oggetto req e passa alla funzione successiva. Se l'utente non esiste nel database, restituisce una risposta di errore 401.

jwt.verify(token, process.env.JWT\_SECRET, (err, decoded) => {

    if (err) {

      return res.status(401).sendFile("login.html", { root: "./views/" });

    }

    const userExist = User.findOne({

      where: {

        email: decoded.email,

      },

    }).then((data) => {

      if (data) {

        req.user = data;

        next();

      } else {

        return res

          .status(401)

          .sendFile("login.html", { root: "./views/" });

      }

    });

  });

***countFollow.js***

Questo codice conta il numero di follower e utenti seguiti dall'utente utilizzato. Utilizza i models ‘Follow’ e ‘User’ per fare ciò. I conteggi vengono ottenuti attraverso le query al database. Una volta ottenuti i conteggi, vengono allegati all'oggetto req.

Se si verifica un errore durante il conteggio, viene gestito restituendo una risposta di errore 401 con una pagina di errore personalizzata.

const countFollowers = await Follow.count({

      where: {

        followedUser: req.userToFind.email,

      },

    });

    const countFollowing = await Follow.count({

      where: {

        followingUser: req.userToFind.email,

      }, });

    req.count = { followers: countFollowers, following: countFollowing };

***getAvatar.js***

Questo codice prende l’utente dal database e lo converte in formato base64 per essere utilizzato direttamente nell'applicazione web. Se l'utente ha un avatar personalizzato, il codice legge l'immagine dal file system, la converte in base64 e la memorizza nell'oggetto req per l'utilizzo successivo nell'applicazione. Una volta ottenuto l’utente o la sua rappresentazione in base64, il codice passa alla funzione successiva.

const getAvatar = async (req, res, next) => {

  const userAvatar = await User.findOne({

    where: {

      email: req.userToFind.email,

    },

    attributes: ["imagePath"],

  });

  if (userAvatar) {

    if (userAvatar.imagePath !== null) {

      let imagePath = await path.join(

        process.env.FILE\_PATH,

        userAvatar.imagePath

      );

      if (await fs.existsSync(imagePath)) {

        img =

          "data:image/" +

          imagePath.split(".").pop() +

          ";base64," +

          (await fs.promises.readFile(imagePath)).toString("base64");

      }

    }

  }

};

***getCredential.js***

Questo codice determina l'utente da cercare, dalla richiesta ricevuta. Se la richiesta include il parametro username, il codice cerca un utente corrispondente a quel nome utente nell'URL. Se l'utente esiste, le sue credenziali vengono memorizzate nell'oggetto req. Se invece il parametro username non è presente nella richiesta, il codice utilizza l'email dalla richiesta. Le credenziali dell'utente trovato vengono memorizzate nell'oggetto req. Se l'utente non viene trovato in entrambi i casi, il codice restituisce una risposta di errore 404 con una pagina di errore personalizzata.

const getCredentials = async (req, res, next) => {

  try {

    if (req.params.username) {

      const otherUser = await User.findOne({

        where: {

          username: req.params.username,

        },

      });

      if (otherUser) {

        req.userToFind = otherUser;

        return next();

      }

    } else {

      const otherUser = await User.findOne({

        where: {

          email: req.user.email,

        },

      });

      if (otherUser) {

        req.userToFind = otherUser;

        return next();

      }

    }    return res.status(404).sendFile("404.html", { root: "./views/errors" });

  }

***getPopoularTags.js***

Questo codice recupera i tag più usati dell'utente corrente. Utilizzando una query al database, cerca i post dell'utente e conta quanti post sono associati a ciascun tag. Quindi, ordina i tag in base al numero di occorrenze e seleziona i tre tag più popolari. Una volta ottenuti i tag più popolari, vengono memorizzati nell'oggetto req per l'utilizzo successivo nell'applicazione.

const getPopularTags = async (req, res, next) => {

    const tags = await Post.findAll({

        where: {

            userEmail: req.userToFind.email

        },

        attributes: [

            'tag',

            [Sequelize.fn('COUNT', Sequelize.fn('DISTINCT', Sequelize.col('id'))), 'tagCount']

        ],

        group: ['tag'],

        order: [[Sequelize.literal('tagCount DESC')]],

        limit: 3

    });

***logout.js***

Questo codice gestisce il logout dell'utente dalla piattaforma. Elimina il cookie "userLogin" dal browser del client che effettua il logout dell'utente. Dopo aver cancellato il cookie, il codice passa alla funzione middleware successiva.

const logout = async (req, res, next) => {

    await res.clearCookie("userLogin");

    next();

}

module.exports = logout;

## Controllers

### Auth.js

Questo file contiene i metodi a cui mandare le varie richieste, con i rispettivi middleware se necessari, inoltre vengono importati i metodi.

Le richieste inviate in questo file creano operazioni e non inviano pagine come nel “routes” (tranne pagine di errori).

const express = require("express");

const multer = require('multer');

const {signup, verifyUsername, verifyEmail, verifyPassword} = require("./signup");

const login = require("./login");

const {profile, changeImage, showFollowers, showFollowed } = require("./profile");

…

router.post("/verifyUsername", verifyUsername);

router.post("/verifyEmail", verifyEmail);

router.post("/verifyPassword", verifyPassword);

router.post("/signup", signup);

router.post("/login", login);

…

### Home.js

#### sendPost

Questa, probabilmente, è la funzione più grande e importante del progetto. Si occupa di prendere i post più recenti degli utenti che si seguono. Dopodiché, convertiamo l’immagine e l’audio del post in base64 affinché l’utente possa vederla. Stessa cosa per l’immagine dell’utente.

if (newPost.image !== null) {

  let imagePath = await path.join(process.env.FILE\_PATH, newPost.image);

  if (await fs.existsSync(imagePath)) {

    imageType = imagePath.split(".").pop();

    base64Image = (await fs.promises.readFile(imagePath)).toString(

      "base64"

    );

  }

}

Successivamente conto i like e i commenti del post.

const postLikes = await Likes.count({

  where: {

    postId: newPost.id,

  },

});

const postComments = await Comment.count({

  where: {

    postId: newPost.id,

  },

});

Infine, ritorno il json con tutte le informazioni ricavate.

return res.json({

  text: newPost.label,

  image: base64Image,

  imageType: imageType,

  audio: base64Audio,

  audioType: audioType,

  sequence: newPost.sequence,

  tag: newPost.tag,

  username: newUsername.username,

  postDate: newPost.postDate,

  id: newPost.id,

  likes: postLikes,

  comments: postComments,

  userImage: base64UserImage,

  userImageType: userImageType,

  hasLiked: hasLiked,

});

#### likePost

Questo metodo permette di mettere like ai post. Prima verifico che l’utente non ha già messo like, poi nel caso aggiungo il like.

const likedPost = await Likes.findOne({

  where: {

    postId: idPost,

    userEmail: req.user.email,

  },

});

if (!likedPost) {

  // If hadn't liked the post, create a like

  const newLike = await Likes.create({

    postId: idPost,

    userEmail: req.user.email,

  });

  return res.status(200).json({ created: true });

} else {

  // If had liked the post, delete it

  await likedPost.destroy();

  return res.status(200).json({ created: false });

}

### Catalog.js

#### catalogPost

Questo metodo è molto simile al sendPost nella Home.js (vedi [5.7.1.1](#_sendPost)), con l’unica differenza che prende i post di un solo determinato utente.

### Create.js

Nel controller create.js, all'interno della funzione createPost, prima viene effettuata una verifica per assicurarsi che i dati passati come parametri siano corretti e validi. Successivamente, tali dati vengono salvati nel database come un nuovo Post attraverso una query di inserimento. Infine, viene restituito il nuovo post appena creato. In Node.js, utilizzando Sequelize come ORM, questo processo viene eseguito nel modo seguente:

const newPost = await Post.create({

  userEmail: userEmail,

  tag: tag,

  label: label,

  image: pathImg,

  audio: pathAudio,

  sequence: sequence

});

return newPost;

La variabile “pathImg” è il percorso in cui le immagini vengono salvate nel server. “pathAudio” è il rispetto percorso per gli audio. In questo controller viene gestita la cartella per differenziare i due tipi di file, come segue:

if(file != undefined){

    pathImg = "Images/" + file.filename;

}

if(audio != undefined){

    pathAudio = "Audio/" + audio.filename;

}

### Discover.js

#### Search

In questa funzione si gestisce la ricerca degli utenti all’interno della pagina scopri. Si può cercare tramite username o nome o cognome. Verifico inoltre, che l’utente cercato non sia sé stesso o un utente bannato.

const results = await User.findAll({

where: {

  [Op.or]: [

    { username: { [Op.like]: `${value}%`, [Op.ne]: req.user.username} },

    { firstname: { [Op.like]: `${value}%`, [Op.ne]: req.user.firstname } },

    { lastname: { [Op.like]: `${value}%`, [Op.ne]: req.user.lastname } }

  ],

  isBanned: false

},

attributes: ['username']

});

#### getUserNotFollowed

Questo metodo permette di ricavare gli utenti non seguiti dall’utente loggato, questo ci è utile per poi proporre gli utenti nello scopri. Dopodiché prendo un utente a caso tra di essi e lo setto come params.username. In realtà getUserNotFollowed si può considerare un middleware.

const usersNotFollowed = await setUsersNotFollowed(req.user.email);

if(usersNotFollowed == null){

  return;

}

const num = parseInt(Math.random()\*usersNotFollowed.length);

req.params.username = usersNotFollowed[num].username;

req.isFollowed = false;

return next();

#### follow e unfollow

Questi due metodi permettono di seguire e smettere di seguire un utente. Nel codice viene fatta una semplice query di inserimento o di eliminazione a dipendenza del metodo.

const newFollow = await Follow.create({

  followedUser: req.userToFind.email,

  followingUser: req.user.email

});

const followToDelete = await Follow.findOne({

  where: {

    followedUser: req.userToFind.email,

    followingUser: req.user.email

  }

});

await followToDelete.destroy();

### Profile.js

#### profile

Questa è la funzione principale di questo controller, essa gestisce la modifica e il salvataggio dei dati del profilo, come ad esempio il nome utente o l’email.

if (type == 'name') {

    if (await filter.filterInputName(value) === true) {

        const firstName = value.substring(0, value.indexOf(' '));

        const lastname = value.substring(value.indexOf(' ') + 1);

        const userObject = await User.findOne({

            where: {

                email: req.user.email,

            }

        });

        if (userObject instanceof User) {

            userObject.firstname = firstName;

            userObject.lastname = lastname;

            await userObject.save();

            return res.json({

                content: "Name modified"

            });

        }

    }

}

#### changeImage

Questo metodo permette di cambiare o di aggiungere l’immagine del profilo utente.

try {

    var pathImg = null;

    if(file != undefined){

        pathImg = "Images/" + file.filename;

    }

    const user = await User.findOne({

        where: {

            username: username

        }

    });

    user.imagePath = pathImg;

    return await user.save();

} catch (error) {

    return "Errore: nel cambiamento del immagine del profilo.";

}

#### ShowFollowers e showFollowed

Sono due metodi che permettono di ricevere la lista degli utenti che si seguono o da cui siamo seguiti.

Ecco una parte di un dei metodi, dove cerca tutti i followers e li inserisce in un array.

Follow.findAll({

  where: {

        followedUser: email

    },

    attributes: ['followingUser']

}).then(followers => {

    const followingUsers = followers.map(user => user.followingUser);

    User.findAll({

        where: {

            email: followingUsers

        },

        attributes: ['username']

    }).then(users => {

        const usernames = users.map(user => user.username);

        return res.json({

status: 'success', content: "followers trovati",

users : usernames

});

   }).catch(error => {

        return res.status(500).json({ status: 'error', message: "Errore durante la ricerca degli utenti" });

    });

})

### Comment.js

#### createComment

Questa funzione crea un nuovo post nel database.

Comment.create({

  userEmail: req.user.email,

  postId: idPost,

  content: text,

}).then((newComment) => {

    return res.json({

      status: "success",

      content: `Commento creato con successo.`,

    });

  })

  .catch((error) => {

    return res.json({

      status: "error",

      content: `Errore nell'invio del commento.`,

    });

  });

#### showComments

Questa funzione permette di ricavare i commenti di un determinato post. Innanzitutto, verifico che il post esiste, dopodiché prendo tutti i commenti del post.

if (postExists instanceof Post) {

  var comments = Comment.findAll({

    where: {

      postId: idPost,

    },

  }).then((comments) => {

      const userEmails = comments.map((comment) => comment.userEmail);

      return User.findAll({

        attributes: ["email", "username"],

        where: {

          email: userEmails,

          isBanned: false

        },

      }).then((users) => {

          const userMap = {};

          users.forEach((user) => {

            userMap[user.email] = user.username;

          });

          const commentsArray = comments.map((comment) => {

            return {

              ...comment.toJSON(),

              username: userMap[comment.userEmail],

          };

        });

        return res.json({

          status: "success",

          content: `Commento creato con successo.`,

          comments: commentsArray,

          userLogged: req.user.username

        });

      })

      .catch((error) => {

        return res.json({

          status: "error",

          content: `Errore nell'invio del commento.`,

        });

      });

  })

  .catch((error) => {

    return res.json({

      status: "error",

      content: `Errore nell'invio del commento.`,

    });

#### deleteComment

Questa funzione permette di eliminare i commenti.

const id = req.body.id;

const commentToDelete = await Comment.findOne({

  where: {

    id: id

  }

});

await commentToDelete.destroy();

### Signup.js

In questo controller ci sono i metodi per ogni dato, ovvero: username, email e password. I metodi doVerify verificano che esiste il dato. Di seguito un esempio per il nome utente:

async function doVerifyUsername(username){

    const userExist = await User.findOne({

        attributes: [

            'username'

        ],

        where: {

            username: username

        }

    });

    if(userExist === null){

        const usernameRegex = /^[^\s@]{1,32}$/;

        if(usernameRegex.test(username)){

            return true;

        }else{

            return 2;

        }

    }else{

        return false;

    }

}

Così facendo il controllo viene effettuato quando viene inserito il dato e un’altra volta alla fine, prima della creazione dell’utente.

if(await doVerifyUsername(username) && await doVerifyEmail(email) && await

doVerifyPassword(password)){

        if(password === confirmPassword){

            try{

                var hashedPassword = await bcrypt.hash(password, process.env.SALT);

                const newUser = User.create({

                    username: username,

                    email: email,

                    password: hashedPassword

                }).then(newUser => {

                    return res.json({status: "success", content: `Utente

${newUser.username} creato.`});

                }).catch(error => {

                    return res.json({status: "error", content: "Errore nella

creazione utente."});

                });

            }catch{

                return res.json({status: "error", content: "Password non

valida."});

            }

        }else{

            return res.json({status: "error", content: "Le password inserite non

corrispondono."});

        }

    }else{

        return res.json({status: "error", content: "Username invalido."});

}

### Login.js

Per quanto riguarda il login innanzitutto criptiamo la password utilizzando il pacchetto bcrypt.

password = await bcrypt.hash(password, process.env.SALT);

Dopodiché, a dipendenza se è un email o uno username fa accedere l’utente.

if (username.includes("@")) {

  userExist = await User.findOne({

    attributes: ["email", "password"],

    where: {

      email: username,

    },

  });

} else {

  userExist = await User.findOne({

    attributes: ["email", "password"],

    where: {

      username: username,

    },

  });

}

## Utils

Questa cartella contiene metodi che eseguono semplici operazioni e che possono essere anche importate ed esportati per altri progetti, poiché universali.

File presenti nella cartella:

* bannedUser.js
* inputFilter.js
* setCookies.js
* upload.js

## Javascript

### Home.js

#### loadPost

Questa funzione serve a gestire lo scroll nella home, ovvero quando scrollo nel div carica un altro post.

function loadPost() {

  if (postContainer.clientHeight + postContainer.scrollTop ==

postContainer.scrollHeight) {

    getPost();

  }

}

### GenPost.js

Questo javascript permette di creare il codice html necessario per generare un post.

Di seguito c’è la funzione per l’html dell’immagine.

function getHtmlImage(image) {

  return `

      <div class="row post-image-container" align="center">

          <img src="${image}" class="postImage">

      </div>`;

}

### Create.js

#### Request

Tramite al metodo request facciamo una richiesta fetch per passare gli elementi come testo, tag, immagini, audio e commenti. Dopo vari tentativi per passare un Blob, abbiamo optato ad utilizzare un FormData.

const formData = new FormData();

if(img != null){

    formData.append('file', img);

}

if(audio != null){

    formData.append('audio', audio);

}

if(comment){

    formData.append('text', comment.value);

}

formData.append('tag', tag.value);

formData.append('sequence', sequence);

fetch("/api/create", {

    method: "POST",

    body: formData

})

Dopodiché aspettiamo la risposta e a dipendenza dello status di essa mandiamo i messaggi d’errore o di successo.

if (data.status == "error") {

if(data.message == "Errore: inserire almeno un testo, un'immagine o un audio."){

    const classes = struments.getAttribute('class');

    struments.setAttribute('class', classes + ' border-danger');

}else if …

#### getSequence

Tramite la funzione getSequence gestiamo l’ordine del testo, delle immagini e degli audio così da permettere agli utenti più libertà e originarietà nella creazione dei post.

function getSequence(children, comment, imgInput, audioInput) {

    let sequence = '';

    children.forEach(child => {

        if (child.firstChild === comment) {

            sequence += "1";

        } else if (child.firstChild === imgInput) {

            sequence += "2";

        } else if (child.firstChild === audioInput) {

            sequence += "3";

        }

    });

    while (sequence.length < 3) {

        if (!sequence.includes("1")) {

            sequence += "1";

        } else if (!sequence.includes("2")) {

            sequence += "2";

        } else if (!sequence.includes("3")) {

            sequence += "3";

        }

    }

    return sequence;

}

### Discover.js

#### LoadProfile

LoadProfile è il metodo principale di questo script, esso esegue una richiesta fetch per far caricare il profilo di un utente. Questo utente può essere definito dal parametro nel caso di una ricerca, oppure se il parametro è null verrà preso un utente casuale tra quelli che non segue l’utente loggato (vedi il controller discover).

fetch(`/api/discover/${username}`, {

    method: "POST"

})

Quindi, al caricamento della pagina chiama una volta il metodo con null come parametro e dopodiché viene chiamato dal click dei bottoni follow e unfollow.

#### viewFollowers

Questo metodo permette di visualizzare gli utenti seguiti e che si segue. Questo avviene solo se si segue l’utente in questione.

const username = {

    username: this.username.innerHTML

}

fetch(`/api/${type}`, {

    method: "POST",

    body: JSON.stringify(username),

    headers: {

        "Content-Type": "application/json"

    }

}).then(res => res.json())

    .then(data => {

        if (data.status == "error"){

            document.getElementById('modalContent').innerHTML = data.content;

        }else{

            document.getElementById('modalContent').innerHTML = '';

            document.getElementById('exampleModalLabel').innerText = type.substring()

        data.users.forEach(user => {

            showHtmlUsers(user);

        });

    }

})

# Test

## Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l’adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-001  REQ-001 | **Nome** | Creazione interfaccia - Login |
| **Descrizione** | Si necessita di creare un’interfaccia grafica dell’applicativo web. | | |
| **Prerequisiti** |  | | |
| **Risultato** | Visualizzazione Login - PC, Telefono: | | |
| **Risultati attesi** | L’interfaccia è visibile, corretta e responsive. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-002  REQ-001 | **Nome** | Creazione interfaccia - Signup |
| **Descrizione** | Si necessita di creare un’interfaccia grafica dell’applicativo web. | | |
| **Prerequisiti** |  | | |
| **Risultato** | Visualizzazione Signup - PC, Telefono: | | |
| **Risultati attesi** | L’interfaccia è visibile, corretta e responsive. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-003  REQ-001 | **Nome** | Creazione interfaccia - Home |
| **Descrizione** | Si necessita di creare un’interfaccia grafica dell’applicativo web. | | |
| **Prerequisiti** |  | | |
| **Risultato** | Visualizzazione Home - PC:  Visualizzazione Home - Telefono | | |
| **Risultati attesi** | L’interfaccia è visibile, corretta e responsive. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-004  REQ-001 | **Nome** | Creazione interfaccia - Crea |
| **Descrizione** | Si necessita di creare un’interfaccia grafica dell’applicativo web. | | |
| **Prerequisiti** |  | | |
| **Risultato** | Visualizzazione Crea - PC:  Visualizzazione Crea - Telefono | | |
| **Risultati attesi** | L’interfaccia è visibile, corretta e responsive. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-005  REQ-001 | **Nome** | Creazione interfaccia - Profilo |
| **Descrizione** | Si necessita di creare un’interfaccia grafica dell’applicativo web. | | |
| **Prerequisiti** |  | | |
| **Risultato** | Visualizzazione Profilo - PC:  Visualizzazione Profilo - Telefono | | |
| **Risultati attesi** | L’interfaccia è visibile, corretta e responsive. | | |
| **Esito** | Parzialmente passato, l’immagine profilo non è responsive su telefono. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-006  REQ-001 | **Nome** | Creazione interfaccia - Scopri |
| **Descrizione** | Si necessita di creare un’interfaccia grafica dell’applicativo web. | | |
| **Prerequisiti** |  | | |
| **Risultato** | Visualizzazione Scopri - PC:  Visualizzazione Scopri - Telefono | | |
| **Risultati attesi** | L’interfaccia è visibile, corretta e responsive. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-007  REQ-001 | **Nome** | Creazione interfaccia - Catalogo |
| **Descrizione** | Si necessita di creare un’interfaccia grafica dell’applicativo web. | | |
| **Prerequisiti** |  | | |
| **Risultato** | Visualizzazione Catalogo - PC:  Visualizzazione Catalogo - Telefono | | |
| **Risultati attesi** | L’interfaccia è visibile, corretta e responsive. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-008  REQ-001 | **Nome** | Creazione interfaccia - Chat |
| **Descrizione** | Si necessita di creare un’interfaccia grafica dell’applicativo web. | | |
| **Prerequisiti** |  | | |
| **Risultato** | Visualizzazione Chat – PC, Telefono:  INTERFACCIA NON CREATA | | |
| **Risultati attesi** | L’interfaccia è visibile, corretta e responsive. | | |
| **Esito** | Non passato, l’interfaccia non esiste. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-009  REQ-002 | **Nome** | Creare e configurare webserver |
| **Descrizione** | Si necessita di realizzare il webserver con tutto il necessario. | | |
| **Prerequisiti** |  | | |
| **Risultato** | Utente (principale):  IP e SSH:      Installazione Node:    Installazione MySQL: | | |
| **Risultati attesi** | L’utente esiste, si può accedere tramite SSH, Node e MySQL sono installati. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-010  REQ-003 | **Nome** | Creazione Database |
| **Descrizione** | Si necessita di creare il database. | | |
| **Prerequisiti** | REQ-002 | | |
| **Risultato** | Creazione Tabelle e Viste:  Creazione Utenti:  Script di anonimizzazione (password): | | |
| **Risultati attesi** | Gli utenti, le tabelle e lo script di anonimizzazione dei dati funzionano. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |
| **Test Case**  **Riferimento** | TC-011  REQ-004 | **Nome** | Algoritmo Home |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** |  | | |
| **Risultati attesi** | Si riesce a visualizzare i post degli utenti che si seguono. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-012  REQ-005 | **Nome** | Algoritmo Crea – Gestione errori |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Nessun tag inserito:  Nessun campo inserito o vuoti: | | |
| **Risultati attesi** | Non funziona se alcuni campi sono mancanti (segnalano gli errori). | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |
| **Test Case**  **Riferimento** | TC-013  REQ-005 | **Nome** | Algoritmo Crea – Testo |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Inserimento con il testo: | | |
| **Risultati attesi** | Si può creare un post con il testo. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-014  REQ-005 | **Nome** | Algoritmo Crea – Audio |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Inserimento con l’audio: | | |
| **Risultati attesi** | Si può creare un post con l’audio. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-015  REQ-005 | **Nome** | Algoritmo Crea – Immagine |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Inserimento con l’immagine: | | |
| **Risultati attesi** | Si può creare un post con l’immagine. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-016  REQ-005 | **Nome** | Algoritmo Crea – Video |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Inserimento con il video: NON È POSSIBILE AGGIUNGERE UN VIDEO. | | |
| **Risultati attesi** | Si può creare un post con un video. | | |
| **Esito** | Non passato, non è gestito e non esiste l’input. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-017  REQ-005 | **Nome** | Algoritmo Crea – Completo |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Inserimento con tutti i campi: | | |
| **Risultati attesi** | Si può creare un post con i campi che si vogliono (questo caso tutti e 3). | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-018  REQ-005 | **Nome** | Algoritmo Crea – Drag&Drop |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Inserimento di un campo tramite Drag&Drop: | | |
| **Risultati attesi** | Si può trascinare con il Drag&Drop per creare il post. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-019  REQ-006 | **Nome** | Eliminazione dei post |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Eliminazione di un post: | | |
| **Risultati attesi** | Si può eliminare un post. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-020  REQ-007 | **Nome** | Algoritmo login – Gestione errori |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Login con campi sbagliati: | | |
| **Risultati attesi** | Vengono stampati degli errori (campi sbagliati). | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-021  REQ-007 | **Nome** | Algoritmo login |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Login: | | |
| **Risultati attesi** | Viene eseguito il login correttamente (si apre la schermata home). | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-022  REQ-007 | **Nome** | Algoritmo signup – Gestione errori |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Signup con campi sbagliati:   (Non si può andare avanti)   (Non si può andare avanti)       (Nel caso l’utente inserisce campi non validi o cambia i dati tramite codice). | | |
| **Risultati attesi** | Vengono stampati degli errori (campi sbagliati). | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-023  REQ-007 | **Nome** | Algoritmo signup |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Signup:  … | | |
| **Risultati attesi** | L’utente viene registrato (si può eseguire il login). | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-024  REQ-008 | **Nome** | Criptazione dati sensibili |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Password inserita nel Database: | | |
| **Risultati attesi** | La password viene hashata tramite bcrypt nel Database. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-025  REQ-009 | **Nome** | Connessione Database |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Connessione Database: | | |
| **Risultati attesi** | Viene stampato a terminale la connessione eseguita al Database. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-026  REQ-010 | **Nome** | Modifica del Profilo |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Profilo di base:   Cambio dati (email, nome, cognome, descrizione, username):    Cambio immagine profilo: | | |
| **Risultati attesi** | Vengono modificati i dati. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-027  REQ-010 | **Nome** | Modifica del Profilo – Gestione errori |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Errore in un campo: | | |
| **Risultati attesi** | Viene visualizzato un messaggio di errore. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-028  REQ-011 | **Nome** | Algoritmo Scopri |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Seguo l’utente (utente di test è TestScopri):    Tolgo il segui:  L’utente segue tutti gli utenti: | | |
| **Risultati attesi** | Si possono seguire gli utenti e anche togliere il segui, inoltre quando gli utenti seguiti sono tutti, essi non verranno più mostrati. | | |
| **Esito** | Passato. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-029  REQ-012 | **Nome** | Configurazione sicurezza sul server. |
| **Descrizione** | Dev’essere presente il servizio, con i controlli e il monitoraggio. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Servizio inesistente e nessun monitoraggio. | | |
| **Risultati attesi** | Viene monitorato il servizio del sito web (online). | | |
| **Esito** | Non passato, non esiste il servizio e neanche gli script di monitoraggio. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-030  REQ-013 | **Nome** | Gestione file con NAS. |
| **Descrizione** | Dev’essere presente il servizio, con i controlli e il monitoraggio. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | NAS inesistente. | | |
| **Risultati attesi** | Il NAS esiste e i file vengono gestiti tramite esso. | | |
| **Esito** | Non passato, non esiste il NAS. | | |
| **Data** | 03.05.2024 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case**  **Riferimento** | TC-031  REQ-014 | **Nome** | Algoritmo Chat. |
| **Descrizione** | Si necessita di controllare il funzionamento dell’algoritmo. | | |
| **Prerequisiti** | REQ-001, REQ-002, REQ-003 | | |
| **Risultato** | Pagina delle Chat inesistente. | | |
| **Risultati attesi** | Si può fare una chat con un utente e inviare messaggi e files. | | |
| **Esito** | Non passato, non la pagina della Chat. | | |
| **Data** | 03.05.2024 | | |

## Mancanze/limitazioni conosciute

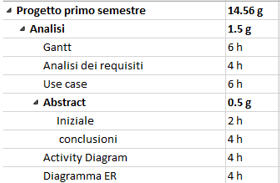
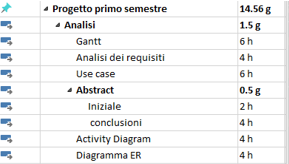
Le mancanze più importanti che ha questo progetto sono la sezione chat, infatti non è ne stata creata la pagina ne implementato il codice e l’impossibilità di cambiare la password (in caso di dimenticanza o per qualsiasi altro motivo) tramite l’invio di una mail o in qualsiasi altro modo, entrambe le mancanze sono dovute ad una questione di tempo, infatti il tempo è stato una variabile importante in questo progetto che non ci ha permesso di implementare tutto come avevamo pianificato.

# Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap. 3.4)

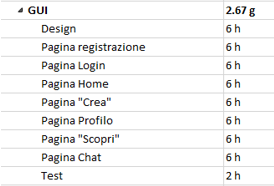
***Analisi:***

preventivo CONSUNTIVO



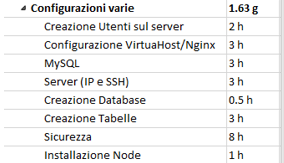
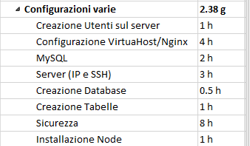
La parte di analisi dei dati è andata esattamente come avevamo pianificato all’inizio

***GUI:***



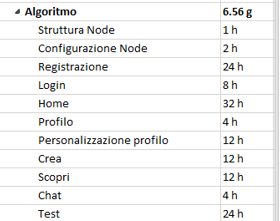
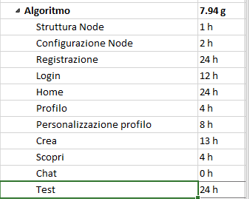
La parte di GUI invece non è andata come speravamo, infatti per colpa del fatto che non avevamo mai usato bootsrap c’è voluto molto tempo per impararlo e questo ha condizionato molto l’andamento dello sviluppo

***Configurazioni varie:***



La parte di analisi dei dati è andata esattamente come avevamo pianificato all’inizio

***Algoritmo:***



La Parte di GUI non è andata come avevamo progettato perché ci sono state in varie occasioni problemi che non avevamo messo in conto durante la fase di progettazione

che hanno rallentato lo sviluppo.

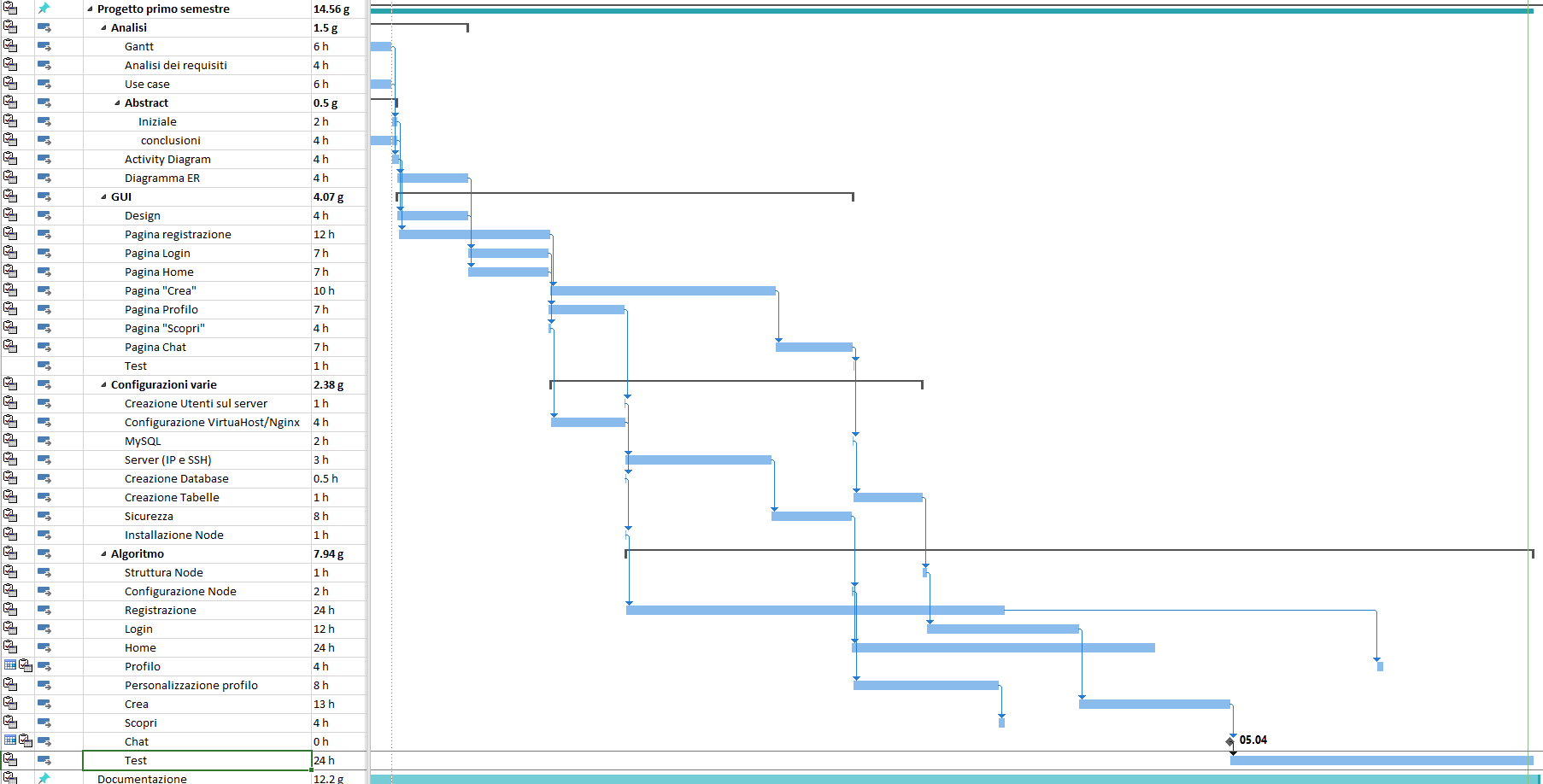


Figura 15 Gantt consuntivo

# Conclusioni

## Sviluppi futuri

Come sviluppi futuri ci saranno sicuramente l’aggiunta della sezione di chat e la possibilità di cambiare la password tramite la verifica mail, il codice rimane comunque molto flessibile e adatto a cambiamenti futuri senza stravolgere l’operato svolto da noi.

## Considerazioni personali

***Edoardo Antonini:***

Da questo progetto ho imparato l’utilizzo del framework bootsrap, framework che non avevo mai utilizzato e che una volta imparato ho trovato molto utile perché permette di creare uno stile molto bello e di rendere la pagina responsive in un tempo breve.

Ho affinato le mie competenze nell’ambito dei linguaggi HTML e CSS, prima di questo progetto le mie capacità in questi linguaggi non erano tanto elevate anche perché ho sempre cercato di concentrarmi maggiormente su altri aspetti, ma alla fine di questo progetto posso dire che sono migliorato e che sicuramente metterò mano ancora a pagine web perché mi sono in un certo senso appassionato a quest’ambito.

Ho imparato un nuovo metodo di lavorare (agile) con il quale mi sono trovato molto bene e penso che riuserò in futuro, mi sono trovato bene perché vedendo sempre ad ogni sprint che il prodotto prendeva forma mi stimolava molto a continuare a lavorare, oltre al fatto che trovo molto utile riunirsi con i compagni di progetto e con il mittente per capire cosa sta andando bene e cosa invece è meglio evitare di fare.

Ho imparato cosa vuol dire fare un progetto grande in gruppo (sicuramente più bello e nel complesso più semplice, dico nel complesso perché ci sono state situazioni di incomprensioni che hanno rallentato lo svolgimento).

In conclusione mi sono divertito molto nel realizzare questo progetto e sono soddisfatto del lavoro svolto anche se non siamo riusciti a completare tutte i requisiti che avevamo prefissato all’inizio.

***Nicolà Bertolazzi:***

In conclusione posso dire che sono contento del risultato del progetto, di tutto quello che imparato e del lavoro di squadra compiuto con i miei colleghi, come la gestione di un progetto da multiple persone grazie ad un repository su Github, però sono consapevole che gestendo meglio il tempo e le risorse avremmo potuto completare tutto il progetto concludendo i requisiti di livello 3 (ad esempio la chat) e magari implementato altre funzionalità e migliorando considerevolmente il codice.  
D’altro canto sono sicuro di essere migliorato molto rispetto ai progetti passati e lavorando con la metodologia agile mi sono trovato molto meglio rispetto a quella a cascata (Waterfall).  
Grazie a questo progetto ho imparato ad utilizzare molto meglio il framework Bootstrap e anche il linguaggio NodeJS, sono riuscito a comprendere i vantaggi del creare un sito web con questo linguaggio e anche le modifiche da apportare nella realizzazione di un sito in futuro, per esempio l’utilizzo di un Database MongoDB invece di MySQL, poiché MongoDB sarebbe stato molto più utile per l’uso dei json (nel progetto vengono utilizzati di continuo).  
Inoltre credo di aver capito al meglio come usare un ORM a 360 gradi (come Sequelize), visto che ho utilizzato molto del mio tempo per fare pratica e la ricerca dell’ORM.  
L’unico aspetto di cui sono molto deluso è il non aver concluso la chat, poiché mi sarebbe interessato molto imparare ad usare i Socket pure con NodeJS e gestire le chat.  
In futuro spero di continuare a lavorare con questa metodologia e anche con il linguaggio NodeJS, poiché molto flessibile, documentato e all’avanguardia.

***Riccardo Deiuri:***

In conclusione posso dire che questo progetto mi ha permesso di migliorare nella programmazione, scoprendo e applicando aspetti nuovi che non avevo mai visto nella mia formazione. Ho utilizzato per la prima volta NodeJs che è un linguaggio molto utile da conoscere se voglio continuare nel ramo di programmazione web. All’inizio è stato impegnativo lavorarci, tuttavia essendo un linguaggio basato su Javascript, ovvero un linguaggio di mia conoscenza, sono riuscito ad adattarmi velocemente.

Inoltre, mi ha permesso di ampliare la mia esperienza nella gestione di un progetto di queste dimensioni, essendo il mio secondo progetto effettivo. È stato, invece, il mio primo progetto a gruppi e questo fattore non è da sottovalutare. Lavorare in gruppo non è evidente, per le divergenze e i disaccordi tra i membri, per il salvataggio del progetto nel git, ovvero a volte si perde del lavoro per un errore di un compagno.

Per quanto riguarda le tempistiche di questo progetto non sono molto soddisfatto dal momento che siamo arrivati tiratissimi alla consegna. Questo è anche comprensibile siccome non conoscevamo Node e quindi all’inizio si lavorava più lentamente, tuttavia, secondo me, avremmo dovuto accelerare il passo prima. Inoltre, è stato il primo progetto con metodologia Agile, quest’ultima mi è piaciuta molto rispetto al Waterfall.

Per concludere, sono comunque contento del risultato e del progetto.

# Bibliografia

## Sitografia

* <https://mockflow.com/>.
* <https://getbootstrap.com/>
* <https://www.w3schools.com/>
* <https://stackoverflow.com/>
* <https://www.npmjs.com/>
* <https://chat.openai.com/>
* <https://www.youtube.com/?hl=it&gl=IT>
* <https://it.wikipedia.org/wiki/>
* <https://lucid.app/>

# Glossario

|  |  |
| --- | --- |
| **Termine** | **Significato** |
| Tag | Parola utilizzata per descrivere il tipo di post che viene pubblicato |
| Post | Foto, video o testo (possibilità di averli anche insieme) che viene pubblicato online sul sito |
| VirtualHost | Programma utilizzato per la creazione e l’uso di macchine virtuali |
| Nginx | Web server leggero e ad alte prestazioni, che può essere usato anche come reverse proxy, load balancer, cache HTTP e proxy di posta elettronica |
| Nodemon | Strumento di monitoraggio per node. js e qualsiasi applicazione basata su di esso |
| token JWT | definisce un modo compatto e autonomo per trasmettere in modo sicuro le informazioni tra le parti come oggetto JSON |
| NAS | dispositivo di archiviazione dedicato ai file che rende i dati sempre disponibili ai dipendenti per collaborare in maniera efficace in una rete |
| Javascript | è un linguaggio di programmazione che gli sviluppatori utilizzano per realizzare pagine Web interattive |
| NodeJs | È un runtime system open source multipiattaforma orientato agli eventi per l'esecuzione di codice JavaScript |

# Indice delle figure

[Figura 1 Use case 11](#_Toc165643755)

[Figura 2 Gantt preventivo 13](#_Toc165643756)

[Figura 3 Discover Activity Diagram 15](#_Toc165643757)

[Figura 4 Home Activity Diagram 16](#_Toc165643758)

[Figura 5 Login Activity Diagram 17](#_Toc165643759)

[Figura 6 Register Activity Diagram 18](#_Toc165643760)

[Figura 7 diagramma Er 19](#_Toc165643761)

[Figura 8 Design pagina Login 20](file:///D:\Dream\3_Documentazione\Documentazione_Dream.docx#_Toc165643762)

[Figura 9 Design pagina Home 21](file:///D:\Dream\3_Documentazione\Documentazione_Dream.docx#_Toc165643763)

[Figura 10 Design pagina Chat 22](file:///D:\Dream\3_Documentazione\Documentazione_Dream.docx#_Toc165643764)

[Figura 11 Design pagina PC Crea 23](file:///D:\Dream\3_Documentazione\Documentazione_Dream.docx#_Toc165643765)

[Figura 12 Design pagina Mobile Crea 23](file:///D:\Dream\3_Documentazione\Documentazione_Dream.docx#_Toc165643766)

[Figura 13 Design pagina Profilo 24](file:///D:\Dream\3_Documentazione\Documentazione_Dream.docx#_Toc165643767)

[Figura 14 Design pagina Scopri 25](file:///D:\Dream\3_Documentazione\Documentazione_Dream.docx#_Toc165643768)

[Figura 15 Gantt consuntivo 92](#_Toc165643769)

# Allegati

* Diari di lavoro
* Codici sorgenti
* Macchine virtuali
* Qdc
* Abstract
* Database
* Activity Diagram
* Gantt
* Gantt consuntivo
* Use Case
* Diagramma E-R
* Foto logo
* Foto design iterfaccie
* Prodotto