UNIVERSIDADE da MADEIRA

CTeSP

CURSOS TÉCNICOS SUPERIORES PROFISSIONAIS

TRABALHO

Relatório do Projeto Final

João António Sousa Abreu
Nº 2019718

João Pedro Maques
Nº 2035017

CTeSP em Tecnologias e Programação de Sistemas de Informação

UNIDADE CURRICULAR:
Desenvolvimento Web – Back-End

DOCENTE:
David Jardim

DATA:
11 de 06 de 2019

ESCOLA SUPERIOR DE TECNOLOGIAS E GESTÃO

Cofinanciado por:

MADEIRA 14-20

REGIÃO AUTÓNOMA DA MADEIRA

PORTUGAL 2020

UNIÃO EUROPEIA
Fundo Social Europeu

Índice

Índice das Figuras

Introdução

Este relatório é referente ao Projeto Final que visa juntar/interligar variados conteúdos ensinados de 3 disciplinas diferentes: Back-End, Front-End e Base de Dados.

O principal objetivo deste trabalho foi interligar as 3 matérias de maneira a criar uma "rede social", no nosso caso fizemos uma espécie de setup wars em que os utilizadores fazem "post" das suas imagens/setups dos seus PCs e depois tem uma secção de comentários referente a cada imagem.

O principal objetivo deste relatório será explicar o desenvolvimento deste Website o mais detalhadamente possível.

Base de Dados

```
CREATE DATABASE social;
USE social;

CREATE TABLE users(
    user_id INT NOT NULL AUTO_INCREMENT,
    fname VARCHAR(45),
    lname VARCHAR(45),
    username VARCHAR(30) NOT NULL,
    password VARCHAR(30) NOT NULL,
    age INT,
    email VARCHAR(50),
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    PRIMARY KEY(user_id)
);

CREATE TABLE posts(
    post_id INT NOT NULL AUTO_INCREMENT,
    user_id INT,
    title VARCHAR(45) NOT NULL,
    body TEXT NOT NULL,
    image LONGTEXT,
    likes INT,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    PRIMARY KEY (post_id),
    FOREIGN KEY(user_id) REFERENCES users(user_id)
);

CREATE TABLE comments(
    comment_id INT NOT NULL AUTO_INCREMENT,
    user_id INT NOT NULL,
    post_id INT NOT NULL,
    body TEXT NOT NULL,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    PRIMARY KEY (comment_id),
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    FOREIGN KEY (post_id) REFERENCES posts(post_id)
);
```

Figura 1 - Criação de tabelas para a base de dados

Criação da Database

Neste projeto começamos por criar a base de dados a utilizar ao longo do projeto.

Para executar o mesmo foi criado um schema como o da imagem anterior em que não só cria a database “social”, mas também as 3 tabelas que usamos.

A tabela “users” serve para registar o primeiro/ultimo nome do utilizador, o seu username, o email e password sendo a chave primaria user_id e a mesma é auto_incrememet, ou seja, nunca teremos 2 id’s iguais.

No que toca a tabela “posts” temos a chave primaria, post_id, a chave estrangeira user_id que serve para associar o post a quem o fez/efetuou, o titulo do post, a imagem, os likes e finalmente quando foi criado.

Finalmente para a tabela comments tem-se, como nas outras, a chave primaria comment_id, nesta tabela temos duas chaves estrangeiras para conectar quem fez o comentário e em que post o mesmo foi realizado, o body que neste caso e a mensagem e finalmente quando foi criado.

Sendo o produto final o seguinte:

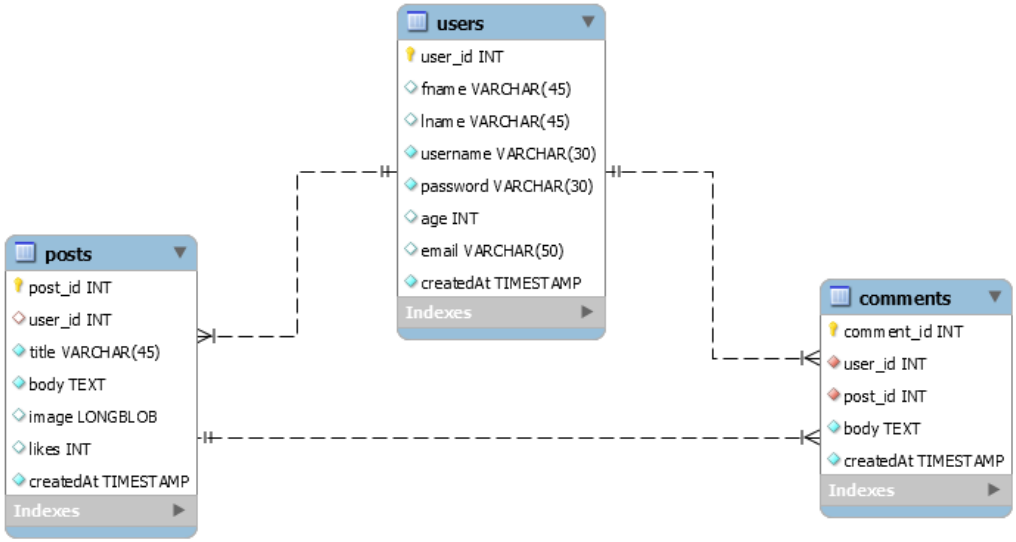

Figura 2 - Diagrama das tabelas no MySQL Workbench

Inserts

```
# Users
INSERT INTO users(username, password) VALUES("joao", "password");
INSERT INTO users(username, password) VALUES("user123", "user");


# Posts
INSERT INTO posts(user_id, title, body, image) VALUES(1, "Exemplo titulo",
"Exemplo body setup",
"https://external-preview.redd.it/BNKk6z8sgeioEyUfo8bf88WZAquZWcySHge8ShXdZKw.
jpg?width=640&crop=smart&auto=webp&s=f15aa4ec889927ae337d721a833cab53ad4534bb");
INSERT INTO posts(user_id, title, body, image) VALUES(2, "Exemplo titulo 2",
"Lorem ipsum do segundo comentario",
"https://preview.redd.it/oy36njp1bk231.jpg?
width=960&crop=smart&auto=webp&s=076ede78d385598cd63c2be91760b90cbe1e25d9");
INSERT INTO posts(user_id, title, body, image) VALUES(2, "Exemplo titulo 3",
"Lorem ipsum do terceiro comentario",
"https://external-preview.redd.it/1Gup7xK9Z7rtVM4KILMiqCrL12_hbE0wPetLVZerZ3o.jpg
width=640&crop=smart&auto=webp&s=10b4b128ccf938a577b6163ee8f339bc5f92af51");


# Comments
INSERT INTO comments(user_id, post_id, body) VALUES(1, 2, "Nice setup !");
```

Figura 3 - Inserir dados nas tabelas

(Código alterado de forma a que a imagem seja mais legível)

Primeiro fizemos o insert de utilizadores ou “users” em que indicamos o seu username e password de modo a que, caso tivéssemos incluído verificação, o mesmo poderia entrar e aí acabar que preencher o seu perfil.

De seguida inserimos alguns posts como exemplos indicando o seu titulo, metendo uma imagem relacionada com o mesmo e que utilizador o executou.

Finalmente foi metido alguns comentários sendo indicado o conteúdo do mesmo no body, quem o fez e a que post é relacionado.

Back-End

No que toca a Back-End primeiro foi instalado todos os node-modules a serem utilizados ao longo do projeto através do comando "npm install" de modo a que pudéssemos "chamá-los" no app.js tal como mostrado na imagem seguinte:

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var cors = require('cors');
var logger = require('morgan');
var bodyParser = require('body-parser');

var postRouter = require('./routes/posts');

var app = express();
```

Figura 4 - Bibliotecas do nodejs usadas

De seguida criamos um controlador para guardar todas a funções utilizadas ao longo do projeto denominado de PostController.

```
var express = require('express');
var bodyParser = require('body-parser');
var mysql = require('mysql');
var app = express();

// Database connection
var connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'social'
});

connection.connect(function (err) {
  if (err) {
    console.error('error connecting: ' + err.stack);
    return;
  }

  console.log('connected to database');
});
```

Figura 5 - Ligação à base de dados

Após estabelecer todas as variáveis a serem utilizadas, o primeiro passo foi conectar o back-end a base de dados.

Para o mesmo foi criada a função "connection.connect" para inserir os dados da nossa database sendo que, caso a conexão seja estabelecida com sucesso, a mensagem "connected to database" irá aparecer no terminal.

Funções e Endpoints - 1

A seguinte função serve para obter todos os posts que foram inseridos na nossa base de dados.

```
// Gets posts
exports.posts = function (request, response) {
  query = "SELECT * FROM posts";

  connection.query(query, function (error, results, fields) {
    if (!error)
      response.send(results);
    else
      console.log('query failed');
  })
}
```

Figura 6 - Selecionar todos os posts

```
router.get("/", PostController.posts);
```

Figura 7 - Endpoint no ficheiro posts.js

A função é chamada no endpoint "/" sendo a sua finalidade mostrar todos os posts ou setups na pagina principal da nossa aplicação

A seguinte função serve para ver todas as setup feitas por um "id" em especifico sendo a mesma executada no endpoint "/:id" em que a mesma pede o "id" inserido através do "request.params.id"

```
// Gets post by id
exports.show = function (request, response) {
  id = request.params.id;
  query = "SELECT * FROM posts WHERE post_id = ?";

  connection.query(query, id, function (error, results, fields) {
    if (!error)
      response.send(results);
    else
      console.log('query failed');
  })
}
```

Figura 8 - Selecionar posts por id

```
router.get("/:id", PostController.show);
```

Figure 9 - Endpoint para selecionar os posts por id

Funções e endpoints - 2

```
// Add comments
exports.comment_add = function (request, response) {
  id = request.params.id;
  body = request.body.body;

  query = "INSERT INTO comments(user_id, post_id, body) VALUES(1, ?, ?)";

  connection.query(query, [id, body], function (error, results, fields) {
    if (!error)
      response.send(results);
    else
      console.log('query failed');
  })
}
```

Figura 10 - Adicionar comentario

```
router.post("/comment/reply/:id", PostController.comment_add);
```

Figura 11 - Endpoint para o comentário

Esta função é responsável por receber e guardar os comentários realizados pelos utilizadores guardando os mesmos na base da dados. O endpoint para o mesmo é "/comment/reply/:id". Como infelizmente não conseguimos implementar um sistema de login/autentificação, o id do utilizador foi pré-definido para 1 sendo o comentário o que for inserido na aplicação.

```
// Signup
exports.adduser = function (request, response) {
  username = request.body.username;
  password = request.body.password;

  query = "INSERT INTO users(username, password) VALUES(?, ?)";

  connection.query(query, [username, password], function (error, results, fields) {
    if (!error)
      response.send(results);
    else
      console.log('query failed');
  })
}
```

Figura 12 - Criação do utilizador

```
router.post("/adduser", PostController.adduser);
```

Figura 13 - Endpoint da criação do utilizador

A seguinte função, com o endpoint "/adduser", tal como indica, é para criar utilizadores.

Este processo só requere que seja inserido o que username o mesmo deseja utilizador e a sua password sendo o método utilizado para o mesmo o "Post".

Funções e endpoints - 3

```
// Create post
exports.create = function (request, response) {
  title = request.body.title;
  body = request.body.body;

  query = "INSERT INTO posts(title, body) VALUES(?, ?)";

  connection.query(query, [title, body], function (error, results, fields) {
    if (!error)
      response.send(results);
    else
      console.log('query failed');
  })
}
```

Figura 14 - Criação do post

```
router.post("/create", PostController.create);
```

Figura 15 - Endpoint da criação do post

Esta funções se responsabiliza por guardar os posts/setups inseridos sendo que a mesma requere o titulo e a descrição ou seja o body. O endpoint que foi atribuído a esta função foi o "/create" sendo o seu método "post".

```
// Delete user
exports.delete_user = function (request, response) {
  id = request.params.id;

  query = "DELETE FROM users WHERE user_id = ?";

  connection.query(query, id, function (error, results, fields) {
    if (!error)
      response.send(results);
    else
      console.log('query failed');
  })

}
```

Figura 16 - Apagar o utilizador

Finalmente temos o delete user que, tal como o nome indica, serve para apaga/remover os utilizadores previamente criados.

Front-End

A seguinte imagem mostra a pagina principal da nossa aplicação:

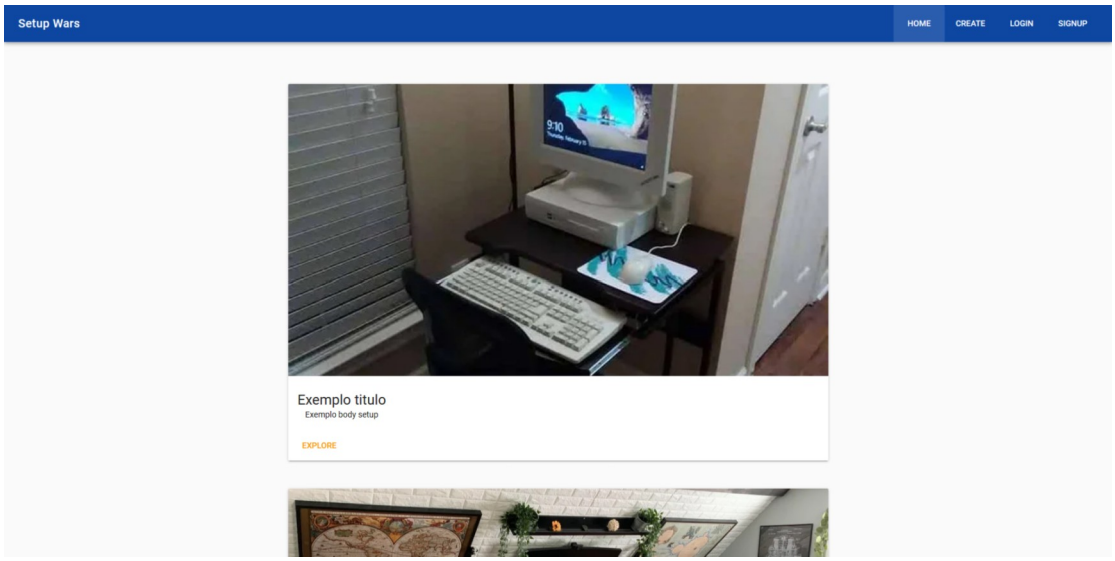

Figura 17 *- Página Inicial*

Como podemos observar temos uma barra de navegação com o botão “home” (pagina principal), o create para criar os posts, o login (função não implementada) e o signup para se inscreverem.

Logo abaixo temos os setups e a sua respetiva “descrição” sendo que o botão amarelo declarado de “explore” nos reencaminha para a seção de comentários do setup desejado.

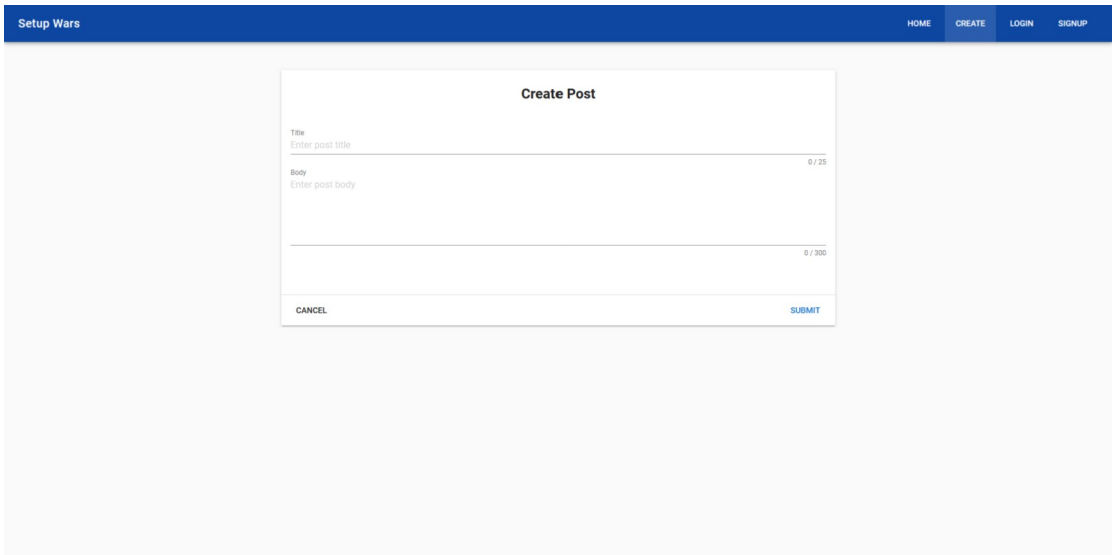

Figura 18 *- Criar post*

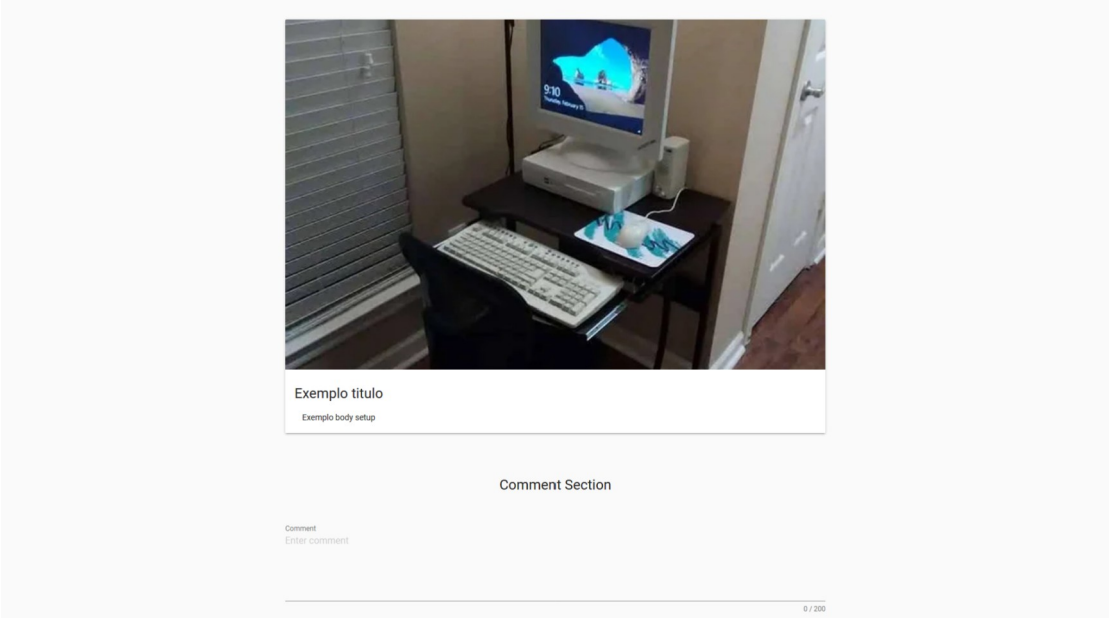

Figure 19 *- Página do post e secção de comentários*

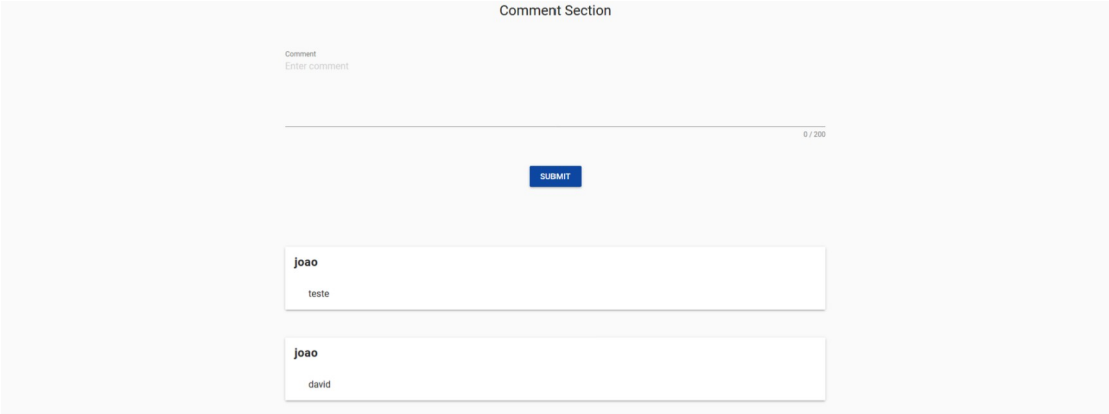

Figure 20 *- Comentários inseridos*

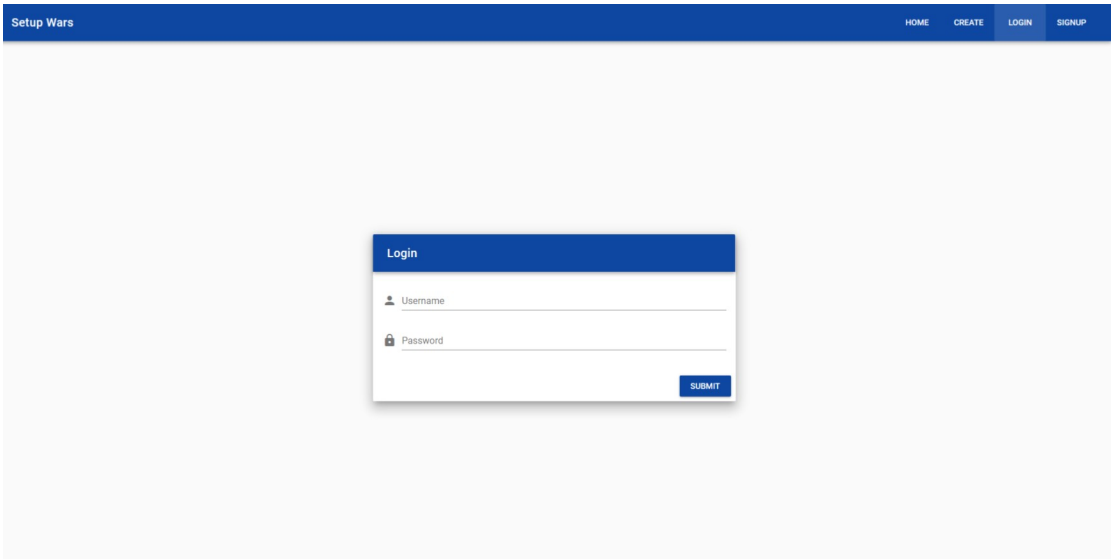

Figure 21 *- Página de login*

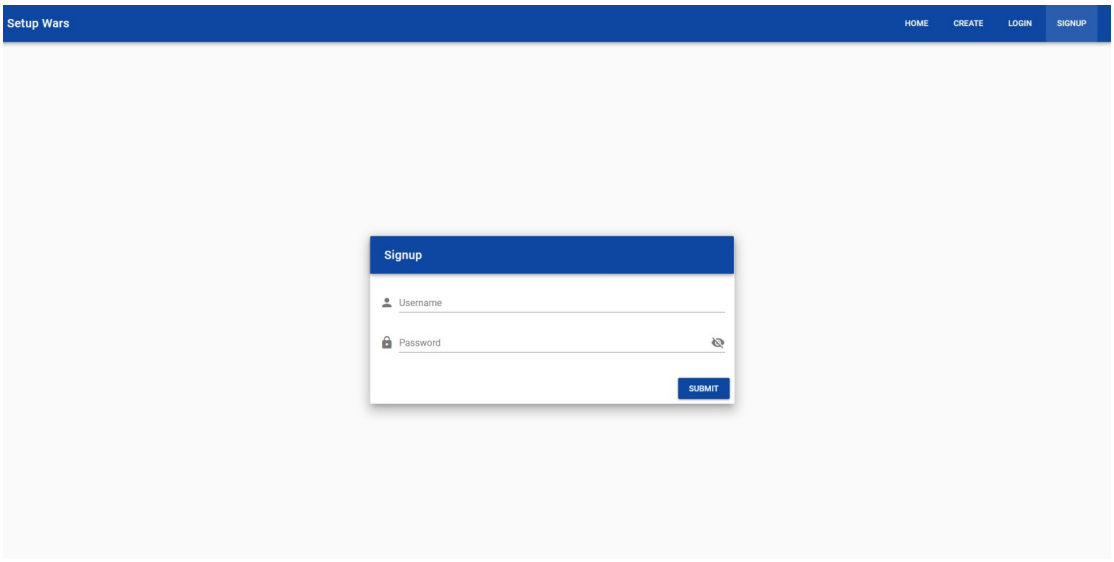

Figure 22 *- Página de registo*

Não implementado

Apesar de tudo o que fizemos neste projeto houve 2 partes que não conseguiram ser implementadas com sucesso, o Passport e o upload de imagens. No caso do passport, apesar de se conseguir aceder ao dados da base de dados com sucesso e de se fazer uma verificação mínima do utilizador, o maior problema que tive foi tanto na resposta que recebia mas também no reencaminhamento para outra pagina tipo "perfil" pois não sabíamos muito bem como, após a validação em back-end, mandar o vue alterar de pagina do website.

No caso do upload de imagens, conseguimos fazer uma espécie de "teste" utilizando o multer com sucesso, que só utilizava back-end sendo que o mesmo não só permitia o utilizador escolher qualquer imagem que se deseja, mas também tanto guardava-as numa pasta e mostrava a imagem submetida.

Ainda mais, conseguimos implementar uma verificação do ficheiro que não verificava a extensão do ficheiro, mas também o mime de modo a que não fosse permitido submeter ficheiros em que alteraram a extensão.

```
// Verificar o ficheiro
function checkFileType(file,cb){
    // extensões permitidas
    const filetypes = /jpeg|jpg|png|gif/;
    // verificar extensão
    const extname = filetypes.test(path.extname(
        file.originalname).toLowerCase());
    // verificação de mime
    const mimetype = filetypes.test(file.mimetype);

    if(mimetype && extname){
        return cb(null, true);
    }else {
        cb('Tipo de ficheiro invalido')
    }
}
```

Figura 23 - Verificação da extensão do ficheiro selecionado

Infelizmente o problema neste caso foi a adaptação para o vue pois não consegui encontrar nenhuma função que mandasse a imagem diretamente para o backend, só encontrei exemplos que mandavam o nome sendo que outro problema era como se iria armazenar os dados no back-end pois cada vez que pesquisava na net sobre o assunto a única opção que havia era o "blob" e não sabia o suficiente sobre o mesmo para o implementar no projeto com sucesso.

Conclusão

Este trabalho não foi deveras fácil de executar pois tanto Front-End como Back-End, em relação a matéria, só nos ensinaram como interagir com a base de dados e não um com o outro sendo isso a parte que deu mais “dores de cabeça” ao executar de maneira a ter os resultados desejados.

Apesar das inúmeras dificuldades que este trabalho nos apresentou e também o curto espaço de tempo que foi disponibilizado para a execução do mesmo, conseguimos executar a maior parte das tarefas que nos foi requerido, isto é, dar o upload de imagens, mostras todas as setups na pagina principal e ao clicar nelas, mostrar os comentários relacionados as mesmas