Escola Superior de Educação de Coimbra

Licenciatura em Comunicação e Design Multimédia

Desenvolvimento de Aplicações Multimédia

Trabalho Prático #2

2022/2023

Plataforma Colaborativa para Edição de Documentos de Texto e para Comunicação entre Utilizadores

Christopher Lemos

2018039433

Introdução

O *Rapport* é uma aplicação colaborativa de redes sociais inspirada no Reddit, que incorpora a sua estrutura de subreddit e sistema de comentários.

Os utilizadores têm a capacidade de criar subreddits, estabelecendo assim comunidades encapsuladas para discussões e colaboração em torno de um determinado tema. Ao criar um subreddit, o utilizador define um nome e uma descrição que refletem o propósito e os objetivos da comunidade. Por exemplo, um subreddit pode ser criado para discutir sobre música, enquanto outro pode ser voltado para a discussão de livros ou filmes.

O sistema de comentários permite interagir com o conteúdo publicado, partilhar feedback, fazer perguntas, iniciar discussões e trocar ideias com outros utilizadores dentro de cada documento.

Uma das características distintivas do Rapport é a sua capacidade de colaboração em tempo real num documento. Tal como no Google Docs, vários utilizadores podem trabalhar em conjunto no mesmo documento, possibilitando a edição simultânea e visualização das alterações feitas pelos outros colaboradores em tempo real.

Funcionalidades não implementadas

Sistema de Upvotes

O sistema de upvotes ainda não está ativo. Embora a interface mostre os botões de upvote e downvote, atualmente eles não têm efeito no sistema de classificação do conteúdo.

Controlo de Versão

Embora seja possível colaborar em tempo real em um documento, o sistema de colaboração ainda não possui um controle completo de versão.

Pesquisa

Embora haja uma funcionalidade de pesquisa visualmente presente, ainda não está ativa e não fornece resultados de pesquisa relevantes. Os utilizadores atualmente não podem pesquisar subreddits, documentos ou conteúdos específicos dentro do aplicativo.

Sistema de Permissões

O sistema de permissões, que controla o acesso e as restrições para os subreddits e documentos, ainda não está implementado. Atualmente, todos os utilizadores têm acesso completo a todos os subreddits e documentos existentes, sem a capacidade de definir permissões diferenciadas com base em papéis ou níveis de autorização.

Estrutura do Projeto

Controle

O projeto recorre ao framework ExpressJS para as funções do servidor:

o setup dos *routes* GET; POST e PUT;

a configuração e renderização do *templating engine;*

a configuração do middleware;

validação e *error-handling*

Middleware

Para facilitar a comunicação entre o Controle e o Modelo recorre-se ao uso de middleware como o body-parser, que permite aceder aos *requests* efetuados ao servidor, e o express-session, que permite armazenar e persistir os dados do utilizador durante a sua sessão.

Model

Estrutura de dados

A estrutura dos dados segue maioritariamente o modelo relacional, no entanto, devido a certas funcionalidades adicionadas posteriormente, e para simplificar a sua implementação, por vezes recorre-se ao modelo hierarchical.
A "base de dados" consiste num conjunto de ficheiros JSON representado as entidades, que são manipulados através das funções CRUD.
As entidades: `user, subreddit, post e comment` são definidas pelos construtores que definem os seus atributos.

```
function User(id, username, email, password, dateCreated, avatarImage) {
  this.id = id
  this.username = username;
```

```
    this.email = email;
    this.password = password;
    this.dateCreated = dateCreated;
    this.avatarImage = avatarImage
}
```

CRUD

Para manipular os dados há um conjunto de funções que lida com uma multiplicidade de requisitos, desde funções de CRUD básicas a operações mais complexas.

```
function addUser(username, email, password) {
    // operação pretendida
    const id = getNextID(users)
    const avatarImage = faker.image.avatar()
    const user = new User(id, username, email, password, new Date(), avatarImage)

    // guardar na variável local
    users[id] = user

    // guardar em localstorage
    localStorage.setItem('users', JSON.stringify(users))
    return user
}
```

As funções CRUD por norma seguem este formato, realizando primeiro o código para obter os dados pretendidos, seguido pelo seu armazenamento na variável local, que por fim é guardado definitivamente no *localstorage*.

Helper Functions

Para facilitar certas operações recorrentes existem ainda as *helper functions*, que atuam como funções intermediárias para as funções principais.

```
function getUser(id) { return users[id] }
```

View

Templating

O projeto utiliza o view engine EJS, que em conjunto com o express, permite o uso de templates para renderizar conteúdo dinâmico.

A sua implementação é efetuada de um modo padrão, com um *views directory* que contém as páginas a ser enviadas pelo router, os *components* que correspondem aos parciais que constroem a página e a estrutura básica da página HTML, e por fim o *public* que contém os stylesheets e scripts externos.

Os dados usados para gerar o conteúdo é enviado pelo ExpressJS, podem ser gerados quando o respetivo GET/POST request é efetuado ou ainda no caso do objeto criado pelo express-session que é enviado para todos os routes via a configuração inicial do express, como no exemplo seguinte:

```
// enviar o objeto session para todos os routes
app.use((req, res, next) => {
        res.locals.session = req.session
        next();
    });
```

```
<%- include('top') %>

<%- include('header', {session}) %>

<div class="main-content">
    <div class="container">

        <%- include('filter-home', {filter}) %>

        <div class="posts">
```

```
        <% posts.forEach( post => { %>
            <%- include('post-main', {post: post}) %>
        <% })%>
      </div>
    </div>
</div>

<%- include('bottom') %>
```

na página *homepage.ejs*, verifica-se o uso de diversos parciais, destacando-se o *top/bottom* que representam a estrutura html, o *header* que representa a barra de navegação e o *post-main* que representa o formato principal do *post*.
Para gerar os *posts,* percorre-se o array e insere-se a parcial *post-main* para cada um, passando-lhe a variável de cada iteração para subsequentemente gerar o conteúdo dinâmico presente nessa parcial.
Para determinar qual barra de navegação usar (logged in/signed out), passa-se a variable *session* para a parcial *header:*

```
<% if (session.loggedIn) { %>
    <%- include('header-logged-in') %>
    <% } else { %>
    <%- include('header-signed-out') %>
    <%- include('sign-up-dialog') %>
    <%- include('login-dialog') %>
    <% } %>
```

Consoante o valor de session.loggedIn, insere-se a parcial correspondente.

Código cliente

Para fornecer uma experiência interativa, há um uso extensivo de código do lado do cliente. Cada página EJS tem uma script tag na qual realiza-se o código específico à página ou no caso de ser preciso usar módulos do node no browser, este é acedido via o ficheiro bundle.js criado através do webpack.

Manual de Utilização

Setup Inicial

O projeto vem sem a pasta node_modules devido ao seu tamanho, por isso para iniciar o projeto utiliza-se primeiro o comando

```
npm install
```

Para correr o servidor

```
node colaborativa.js
```

Por fim, abre-se o website no port 3000

Funcionalidades

Fóruns de discussão de acesso livre ou condicionado a todos os utilizadores

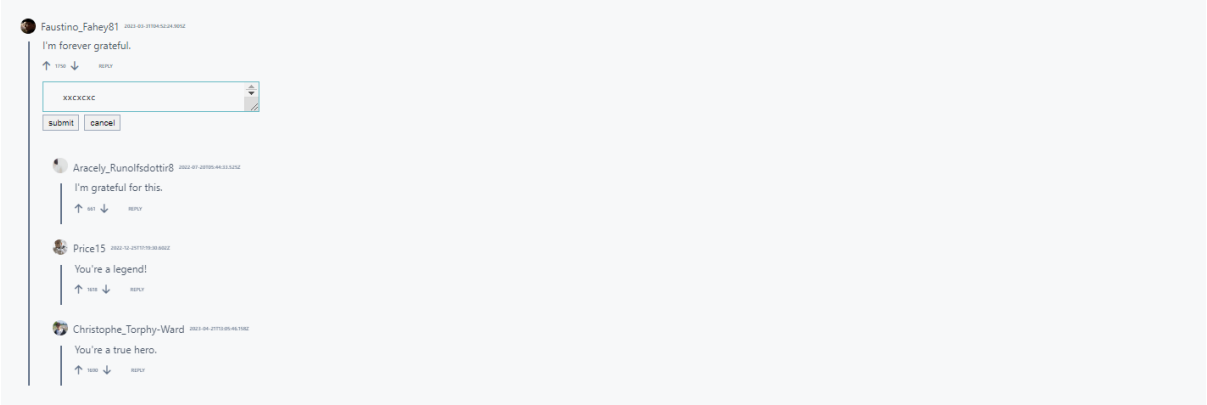
A página principal disponibiliza todos os posts do Rapport, sendo organizado por padrão pelo modo *hot* que utiliza uma fórmula tendo em conta a sua recência e o número de upvotes. É possível ainda organizar por *new*, retornando os posts organizados do mais recente ao mais antigo, e por *top*, que organiza os posts pelo seus upvotes.

```
function joinedHotPosts() {
  const now = Date.now();
  const oneHour = 3600 * 1000;
  const hotPosts = posts.map(post => {
    const subreddit = subreddits.find(s => s.id == post.subredditID);
    const author = users.find(u => u.id == post.authorID);
    const ageHours = (now - new Date(post.dateCreated).getTime()) / oneHour;
    const score = Math.log10(post.upvotes - post.downvotes) + ageHours / 24;

    return {
      id: post.id,
      title: post.title,
      body: post.body,
      upvotes: post.upvotes,
      downvotes: post.downvotes,
      type: post.type,
      image: post.image,
      date: post.dateCreated,
      subreddit: subreddit,
      author: author,
      hotScore: score.toFixed(5)
    };
  }).filter(post => post.hotScore >= 0).sort((a, b) => b.hotScore - a.hotScore);
```

```
  return hotPosts;
}
```

Em todos os filtros o processo é o mesmo, é utilizado o método .map() para poder manipular cada post individual, isto é usado para dois fins: um join entre os posts, subreddits e autores para criar um objeto final que facilite a implementação dos dados no lado do cliente, e para criar os dados relevantes a cada filtro. No caso do *hot* isto refere-se à variável *hotScore* que consiste numa normalização dos valores obtidos na fórmula aplicada na variável *score.*

Através dos posts na página inicial, o utilizador pode direcionar-se ao post individual ou ainda ao subreddit correspondente através do título do post e do avatar do subreddit. Ao aceder ao post o utilizador, caso tenha uma sessão iniciada poderá comentar diretamente ao post e ainda face a qualquer comentário, criando deste modo um sistema de comentários encadeados.



Para comentar no post em si, basta escrever uma mensagem no texto abaixo do post e submeter.

```
let postReplyForm = document.getElementById('post-reply-form')
 postReplyForm.addEventListener('submit', (event) => {
```

```
    event.preventDefault()

    let formData = new FormData(postReplyForm)
    let body = formData.get('post-reply')

    let commentData = {
            postID: `<%= post.id %>`,
            parentCommentID: null,
            authorID: `<%= session.userID %>`,
            reply: body
        }

    fetch('/r/:subreddit/:postID', {
            method: 'POST',
            body: JSON.stringify(commentData),
            headers: {
            'Content-Type': 'application/json'
        }
        })

    location.reload()
})
```

Ao submeter é criado um *FormData* com os conteúdos do texto e realiza-se um POST request via o fetch(), enviando os dados, após serem enviados a página realiza um refresh.

Para responder ao comentário de outro utilizador, o processo básico é semelhante, tendo apenas que ter em conta a estrutura recursiva dos comentários.

Cada comentário tem a seguinte estrutura:

```
{
   "id": "2a8bb6f4-d43c-4371-99e9-30d8c6fb3fc9",
   "postID": 24,
   "parentCommentID": null,
   "upvotes": 814,
   "body": "You're a true virtuoso of comments.",
   "authorID": 9,
   "dateCreated": "2023-05-16T19:02:50.034Z",
   "replies": []
 },
```

Quando um novo comentário é adicionado, é verificado se ele possui um *comentário-pai*. Se não tiver, significa que é um comentário de nível superior. Isto é estabelecido pelo elemento a qual se dirige o comentário, se for ao post em si, o *comentário-pai* tem um valor de null, se for dirigido a outro comentário, será assinalado o id desse comentário.

```
if (parentCommentID === null ) {
        comments.push(comment)
    } else {
        findParentCommentById(comments, parentCommentID, comment)
    }
```

Se o novo comentário tiver um *comentário-pai*, ele é anexado como uma resposta ao *comentário-pai* correspondente.

```
function findParentCommentById(commentsArray, parentCommentID, comment) {
    commentsArray.forEach( c => {
```

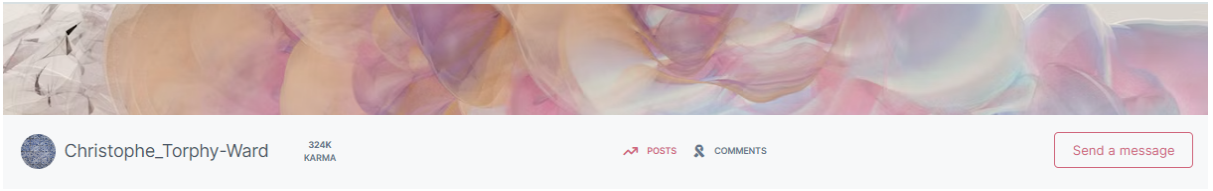
```
        if (c.id == parentCommentID) {
            c['replies'].push(comment)
        }
        if (c.replies.length > 0) {
            let foundComment = findParentCommentById(c.replies, parentCommentID,
comment)
            if (foundComment) {
                c['replies'].push(comment)
            }
        }
    })
}
```

Caso o *comentário-pai* tenha respostas adicionais, a função é chamada novamente, procurando o *comentário-pai* dentro das respostas existentes. Este processo continua recursivamente até que o comentário pai correspondente seja encontrado ou até que a hierarquia de comentários seja totalmente percorrida. Quando o comentário pai correspondente é encontrado, o novo comentário é adicionado como uma resposta a esse comentário pai.

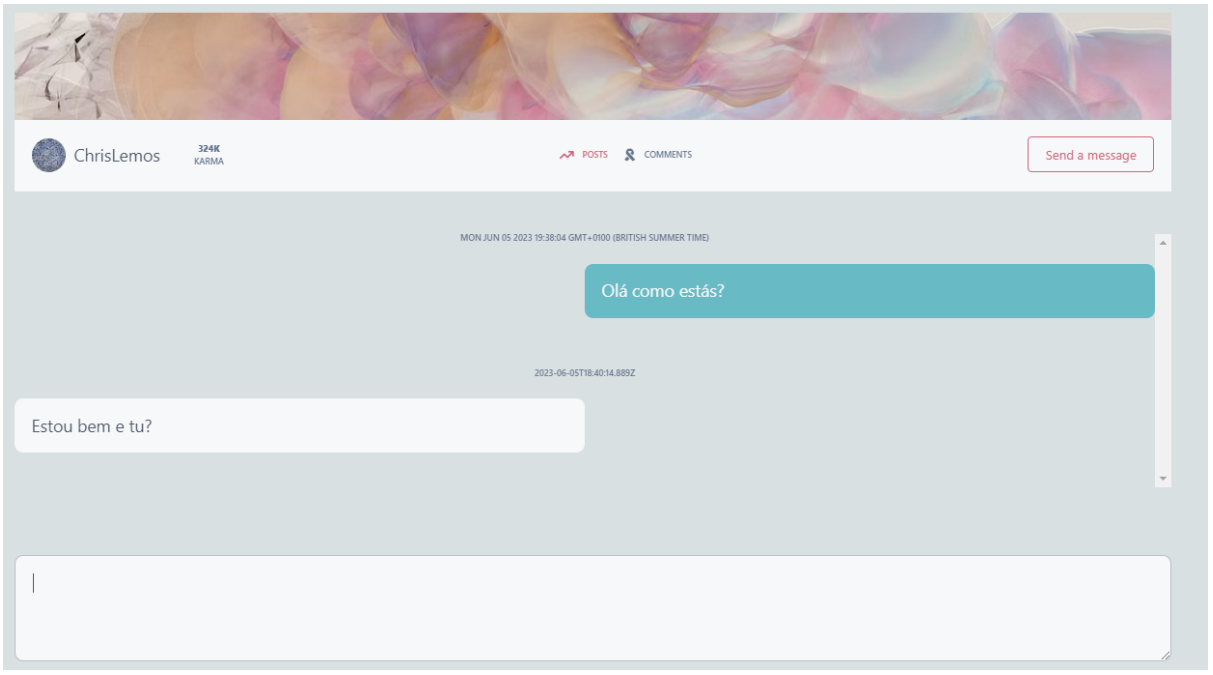
Isto cria uma estrutura em árvore, onde os comentários são organizados hierarquicamente, permitindo que os utilizadores respondam a comentários específicos e visualizem as discussões de forma estruturada.

Comunicação Individualizada entre dois utilizadores

Para enviar um mensagem privada a um utilizador basta aceder à página de perfil do utilizador via o seu avatar/nome. No seu banner clique na botão “Send a message”



Será direcionado a um chat exclusivo entre os dois utilizadores, carregando mensagens anteriores caso existam. Para mandar uma mensagem basta escrever na textarea no fim da página.



Quando a conversa é iniciada, é realizada a seguinte função, acedendo às conversas anteriores caso existem.

```
function getConversation(user1ID, user2ID) {
    return users[user1ID].conversations.find( u => u.userID == user2ID).messages
}
```

Ao enviar a mensagem executa-se a seguinte função

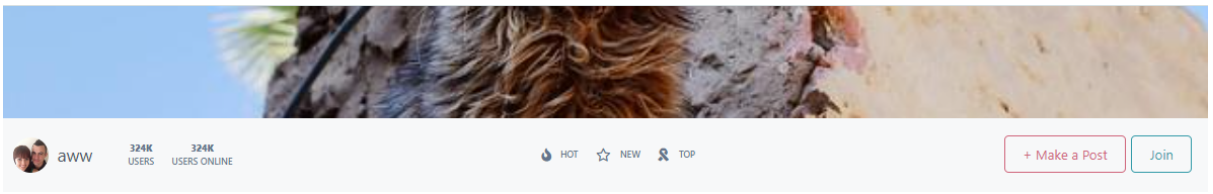
```
function sendMessage(senderID, receiverID, content ) {
  let sender = users.filter( u => u.id == senderID)[0]
  let receiver = users.filter( u => u.id == receiverID)[0]
  const conversationExists = sender.conversations.some(convo => convo.userID == receiverID)

 if (!conversationExists) {
  sender.conversations = [{'userID': receiverID, 'messages': []}]
  receiver.conversations = [{'userID': senderID, 'messages': []}]
 }

 let message = new Message(senderID, content)
 senderConvo = sender.conversations.find( convo => convo.userID == receiverID)
 receiverConvo = receiver.conversations.find( convo => convo.userID == senderID)

 senderConvo.messages.push(message)
 receiverConvo.messages.push(message)

 users[senderID].conversations = sender.conversations
 users[receiverID].conversations = receiver.conversations
```

```
  localStorage.setItem('users', JSON.stringify(users))
}
```

Verifica-se a existência de uma conversa entre os dois utilizadores, caso não exista cria-se, de seguida cria-se um objeto *message* através do seu construtor. Anexa-se a mensagem em ambos os utilizadores, tomando nota de quem o enviou.

Edição colaborativa de um documento

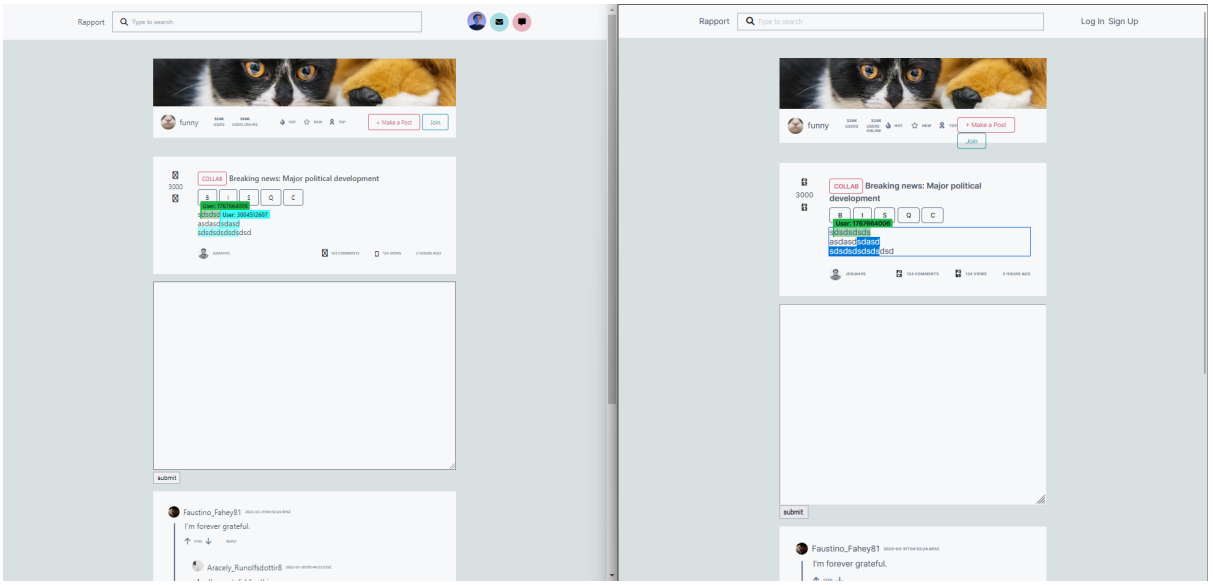
O utilizador pode criar um novo documento colaborativo via o botão “Make a post” no banner de cada subreddit



ou pode aceder a um post colaborativo, claramente identificado.

O editor de texto dos textos colaborativos são distintos dos textarea utilizados no resto do website, optando por um módulo externo chamado TipTap, que proporciona uma framework para criar editores de texto custom. Isto verifica-se pela adição de botões para mudar o estilo do texto que foram adicionados manualmente usando funções do TipTap. o

TipTap permite ainda, em conjunto com a sua solução de servidor Hocuspocus, a edição simultânea do documento.



Quando o post é acedido, é chamado o bundle.js composto pelo seguinte código

```
const provider = new HocuspocusProvider({
    url: 'ws://127.0.0.1:1234',
    name: postID,
  })
```

cria uma nova ligação ao servidor hocuspocus configurado no código do lado do servidor, acedendo ao documento especifico ao post via o id do post.

Instancia-se um novo editor via o seu construtor fornecido pelo TipTap

```
const editor = new Editor({
    element: document.querySelector('.element'),
    extensions: [
      StarterKit.configure({
        // The Collaboration extension comes with its own history handling
        history: false,
```

```
    }),
    // Register the document with Tiptap
    Collaboration.configure({
      document: provider.document,
    }),
    CollaborationCursor.configure({
      provider: provider,
      user: {
        name: session.username,
        color: randomColor(),
      },
    }),
  ],
})
```

Verifia-se o uso do *StarterKit* que providencia umas funções básicas e o *Collaboration* e *CollaborationCursor* que potenciam a edição colaborativa.

O servidor hocuspocus trata da persistência dos documentos através de um servidor SQLite, não sendo necessário tratar deste aspecto, apenas é preciso passar o id do post.