Instituto Politécnico de Viana do Castelo

Instituto Politécnico de Viana do Castelo

Escola Superior de Tecnologia e Gestão

## Graduation in Informatics Engineering

# Course unit : Sistemas de Informação em Rede


**Hélder Gonçalves, n.º 22067**

**Hugo Pereira, n.º 20790**


2020 - 2021


Coordinator: Prof. Hugo Barbosa | hugob@estg.ipvc.pt

# Índice

**Introdução** **5**

**Tecnologias, Ferramentas, Bibliotecas, Metodologias e Gestão de Projeto** **5**

API Desenvolvida e Pedidos Resolvidos à mesma 6

**Backend** **6**

3.1 Firebase Auth 6
3.2 Signup Function 7
3.3 Login Function 9
3.4 Atualização de dados do utilizador 10
3.5 Busca de dados de um utilizador 12
3.6 Buscar todos os posts / screams 13
3.7 Fazer um post novo 14
3.8 Busca de um post específico 16
3.9 Criação de um comentário 17
3.10 Criação e atribuição de um Like 18
3.11 Remover o Like (Unlike) 19
3.12 Apagar um post/scream 20
3.13 Notificações 21
3.14 Atualização de imagem 24

**4. Frontend** **27**

4.1 Reducers 30
4.1.1 UserReducer 30
4.1.2 UiReducer 31
4.1.3 DataReducer 32
4.2 React Redux - Em Prática 33
4.2.1 Login 33
4.2.2 Signup 34
4.2.3 Logout Upload Edit e Ler Notificações 35
4.2.4 Buscar posts 36
4.2.5 Like Dislike e delete 37
4.2.6 Postagens 38
4.3 Páginas 39
4.3.1 Login 39
4.3.2 SignUp 40
4.3.3 HomePage 42
4.3.4 Página de Utilizador 43
4.4 Componentes 44
4.4.1 Scream / Post component 44

4.4.2 ScreamDialog 45
4.4.3 Comments component 49
4.4.4 CommentForm component 50
4.4.5 Botão de Like 51
4.4.6 Perfil 52
4.4.7 Edit Details 55
4.4.8 Perfil estático 56
4.4.9 Barra de Navegação 57
4.4.10 Post Scream / Post Component 58
4.4.11 Componente Notificações 59
4.5 Heroku Deploy 61

**5. Dificuldades e Future Work 62**
5.1 Dificuldades 62
5.2 Future Work 63

**6. Conclusão 64**

**7. Bibliografia 65**

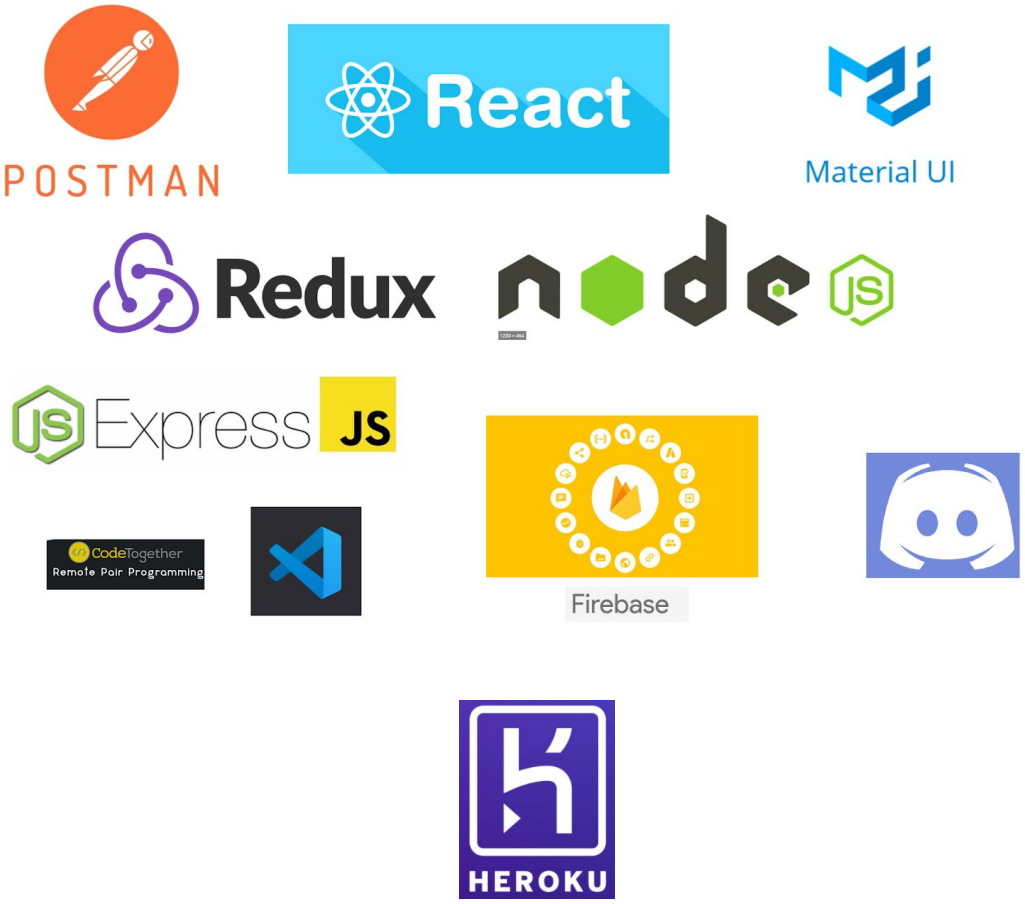
# 1. Introdução

Este trabalho tem como objetivo a realização de uma rede social do estilo do twitter. Para alcançar o nosso objetivo utilizamos Firebase , Node.js , Express , React , Redux , Postman , etc.

O nosso Backend usa maioritariamente Node.js e Express com Javascript para escrever as funções, depois deu-se deploy no firebase para guardar as funções na Cloud. Usou-se também a Storage e a Fire Store do Firebase.

Quanto ao Frontend usamos a framework React para a construção da estrutura da app ( components / pages / etc ) escritos em javascript, react redux para os eventos e interação com o backend usamos Axios para fazer pedidos à API desenvolvida, estes previamente testados com o Postman.

# 2. Tecnologias, Ferramentas, Bibliotecas, Metodologias e Gestão de Projeto



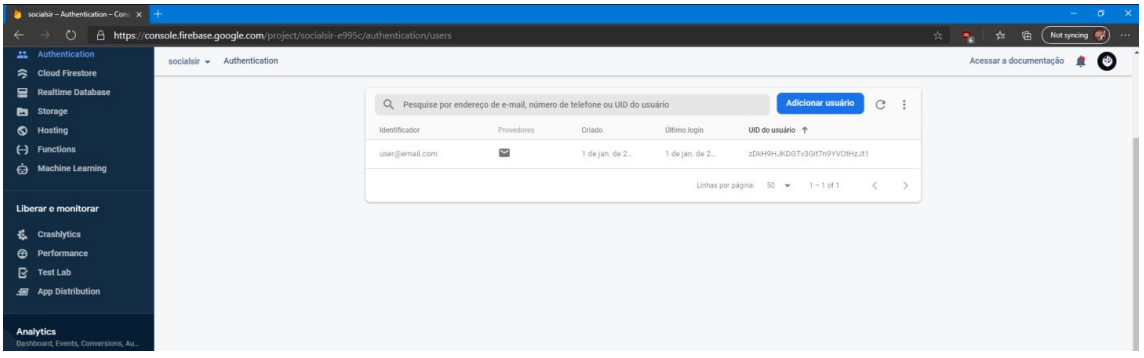
## *API Desenvolvida e Pedidos Resolvidos à mesma*

```
const FBAuth = require('./util/fbAuth');
const cors = require('cors');
app.use(cors());

const {db} = require('./util/admin');
//imports
const { getAllScreams ,postOneScream,getScream,commentarioEmScream,likeScream,unlikeScream,apagarScream} = require('./handlers/screams');
const { signup, login, uploadImage,addUserDetails,getUserAutenticado,getDetalhesUser,markNotificationsRead} = require('./handlers/users');



//util routes
// foram refactored para handlers
app.get('/screams', getAllScreams);
//função middleware pra auth (FBAuth) serve para podermos aceder a dados protegidos
app.post('/scream' , FBAuth ,postOneScream);
app.get('/scream/:screamId',getScream);
app.post('/scream/:screamId/comment',FBAuth,commentarioEmScream);
app.get('/scream/:screamId/like',FBAuth,likeScream);
app.get('/scream/:screamId/unlike',FBAuth,unlikeScream);
app.delete('/scream/:screamId',FBAuth,apagarScream);
      You, 19 days ago • Initial commit
// user routs
app.post('/signup', signup);
app.post('/login',login);
app.post('/user/image',FBAuth,uploadImage);
app.post('/user',FBAuth,addUserDetails);
app.get('/user',FBAuth, getUserAutenticado);
app.get('/user/:handle',getDetalhesUser);
app.post('/notifications', FBAuth, markNotificationsRead);
// express permite exemplo hhtps//blabla.com/API/screams em vez de https//blabla.com/Screams
exports.api = functions.https.onRequest(app);
```

# 1. Backend

Nos testes do backend usou-se firebase deploy este serviu  para depois realizar-se testes com o Postman para verificar se tudo estava a correr como planeado quanto aos pedidos à API.

## *3.1 Firebase Auth*




O firebase provide de um sistema de autenticação com o email e password. Os nossos utilizadores têm os fields Email , Password , Confirm Password e Nome ( handle ) que são obrigados a preencher para criarem a sua conta.

Ao realizar Login o utilizador gera uma token, utilizamos esta para efetuar a sua autenticação, no caso de testes em Postman devemos colocar no header Authorization “Bearer ${token}” usamos o split para remover o Bearer, isto foi aconselhado pela documentação do firebase por ser uma boa prática de programação.

```
module.exports =

//            request , response , next devolve respota ao handler
 (req,res,next) => {                        //Bearer e um standart de programação para vereficar a condição
  let idToken;
  if(req.headers.authorization && req.headers.authorization.startsWith('Bearer ')){
    idToken = req.headers.authorization.split('Bearer ')[1] //array de 2 strings Bearer e a token o [1] vai buscar a token
  } else {
    console.error('No token found')
    return res.status(403).json({error : 'Unauthorized'}) //403 erro de autorização
  }

  admin.auth().verifyIdToken(idToken)
  .then(decodedToken => {
    req.user = decodedToken; // para eliminar data extra a fim de nao afetar proximos requests
    console.log(decodedToken);
    return db.collection('users')
    .where('userId', '==', req.user.uid)
    .limit(1) // 1 resultado
    .get();
  })
  .then(data => {
   req.user.handle = data.docs[0].data().handle; // busca do handle e attach ao user.handle
   req.user.imageUrl = data.docs[0].data().imageUrl;
   return next();
  })
  .catch(err => {
    console.error('Erro ao verificar a token', err)
    return res.status(403).json(err);
  })
}

      You, 12 days ago • Refactor + FBAuth
```

## 3.2 Signup Function

Nós utilizamos uma combinação de regras para verificar se o utilizador inseriu a informação e a inseriu de forma correta.

```
const isEmail = (email) => {
    //reg expression retirada da net para verificar se o email e valido
    // https://emailregex.com/
    const regEx = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(
    if(email.match(regEx)) return true; //esta string está formatadaa como um email regular
    else return false;
  }

  const isEmpty = (string) => { //verificar se esta a ser enviado algo vazio
    if(string.trim() === '') return true;
    else return false;
  };


  exports.validateSignupData = (data) => {

  let errors = {};

  if(isEmpty(data.email)){
    errors.email = 'Email é um campo obrigatório'
  } else if(!isEmail(data.email)){
    errors.email = 'O email introduzido não é valido'
  }

  if(isEmpty(data.password)) errors.password = 'Password é um campo obrigatório';
  if(data.password !== data.confirmPassword) errors.confirmPassword = 'As passwords não são iguais ';
  if(isEmpty(data.handle)) errors.handle = 'Não pode estar vazio';


  return {
      errors,
      valid: Object.keys(errors).length === 0 ? true : false // no keys fica true se tiver fica false (0 erros a data e valida)
  }

  }
```

Para o signup em si a função deve receber o email , password , confirm password e handle depois confirma se são válidos, de seguida verifica se o utilizador em questão já existe caso não exista utilizamos a função predefinida auth().createEmailWithPassword para criar um utilizador de firebase automaticamente depois com promessas atribuímos a token ao utilizador para login imediato e criamos o utilizador em si para guardar na Firebase Firestore.

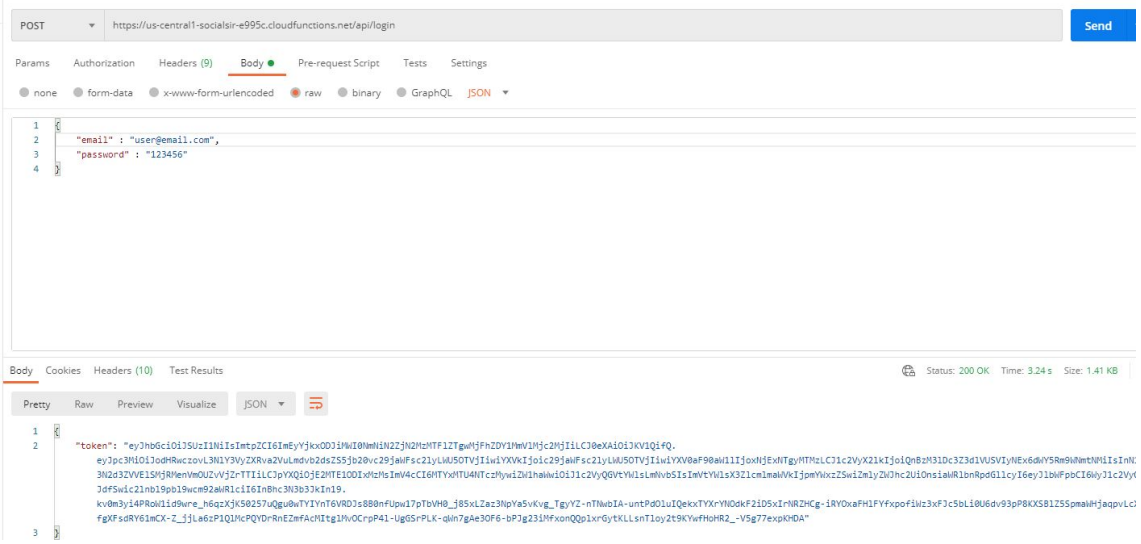
```
exports.signup = (req,res) => {
    const newUser = {
      email: req.body.email,
      password: req.body.password,
      confirmPassword: req.body.confirmPassword,
      handle: req.body.handle,
    };


    const{ valid,errors } = validateSignupData(newUser);
    if(!valid) return res.status(400).json(errors); // se não for valido devolvemos erros

    const noImg = 'no-img.png' // imagem  vazia

    let token, userId; //user ID como variavel para ser acedivel
    db.doc(`/users/${newUser.handle}`).get()
    .then(doc => {
      if(doc.exists){ //user ja existe
        return res.status(400).json({ handle: 'Este nome já foi escolhido'});      You, seconds ago • Uncommitted changes
      } else {
      return firebase.auth().createUserWithEmailAndPassword(newUser.email, newUser.password)
      }
    })
  .then(data =>{ // login depois do signup
    userId = data.user.uid;
   return data.user.getIdToken();
  })
  .then(idToken => {
    token = idToken;
    const userCredentials = { // credenciais para o firebase
      handle : newUser.handle,
      email: newUser.email,
      createdAt: new Date().toISOString(),
      imageUrl:  `https://firebasestorage.googleapis.com/v0/b/${config.storageBucket}/o/${noImg}?alt=media`, //imagem vazia default
      userId
    };
    db.doc(`/users/${newUser.handle}`).set(userCredentials); //criar o documento
  })
  .then(() =>{
    return res.status(201).json({ token });
  })
  .catch(err => {
    console.error(err);
```

## 3.3 Login Function

Para o login recebe-se apenas o email e a password, usa-se a função predefinida do Firebase signinwith password() para o login em si, caso tenhamos uma password errada devolvemos erro com aviso de credenciais erradas. Caso não as promessas sejam compridas vamos buscar a token e devolvemos-a.

```
exports.login = (req,res) => {

  const user = {
    email: req.body.email,
    password: req.body.password
  };

  const{ valid,errors } = validateLoginData(user);
  if(!valid) return res.status(400).json(errors);


  firebase.auth().signInWithEmailAndPassword(user.email, user.password)
  .then(data => {
    return data.user.getIdToken(); //buscar a token
  })
  .then(token => {
    return res.json({token}); //passar a token      You, seconds ago • Uncommitted changes
  })
  .catch(err =>{
    console.error(err);
    // auth/wrong-password
    if(err.code === 'auth/wrong-password'){
      return res.status(403).json({ general: "Credenciaís errados. Por favor tente de novo"}); // 403 = nao autorizado
    } else {
      return res.status(500).json({
        error: err.code
      });
    }
  });

};
```

Login em postman

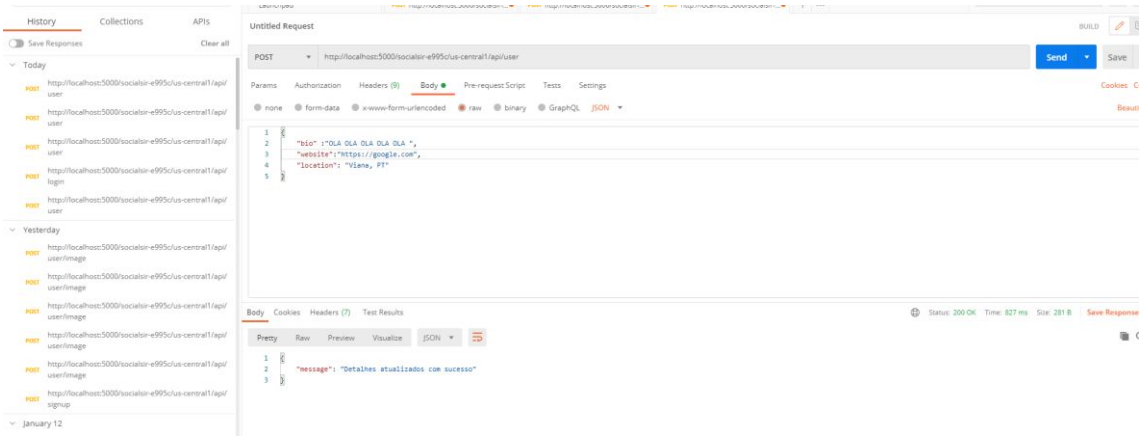


## 3.4 Atualização de dados do utilizador

Para a utilização dos dados do utilizador tal como a bio , website ou localização usou-se uma função para tratamento destes dados (reduceUserDetails) em que se usou o .trim() para apagar espaços vazios e verificou-se que o website teve http introduzido no nome senão introduz-se manualmente no final usa-se o .update do Firebase para atualizar os dados do utilizador em questão.

```
//update details do user
exports.addUserDetails = (req,res) => {
  let userDetails = reduceUserDetails(req.body);

  db.doc(`/users/${req.user.handle}`).update(userDetails)
  .then(() =>{
    return res.json({message : "Detalhes atualizados com sucesso"});
  })
  .catch(err => {
    console.error(err);
    return res.status(500).json({error: err.code});
  });
}
```

```
//react manda isto empty caso utilizador n preencha
exports.reduceUserDetails = (data) =>{
  let userDetails = {};
  if(!isEmpty(data.bio.trim())) userDetails.bio = data.bio;
  if(!isEmpty(data.website.trim())){
    // https://website.com
    // pegamos em h t t p se nao tiver acvresentamos
    if(data.website.trim().substring(0,4) != 'http'){
      userDetails.website = `http://${data.website.trim()}`;
    } else userDetails.website = data.website;
  }
  if(!isEmpty(data.location.trim())) userDetails.location = data.location;
  return userDetails;
}
```

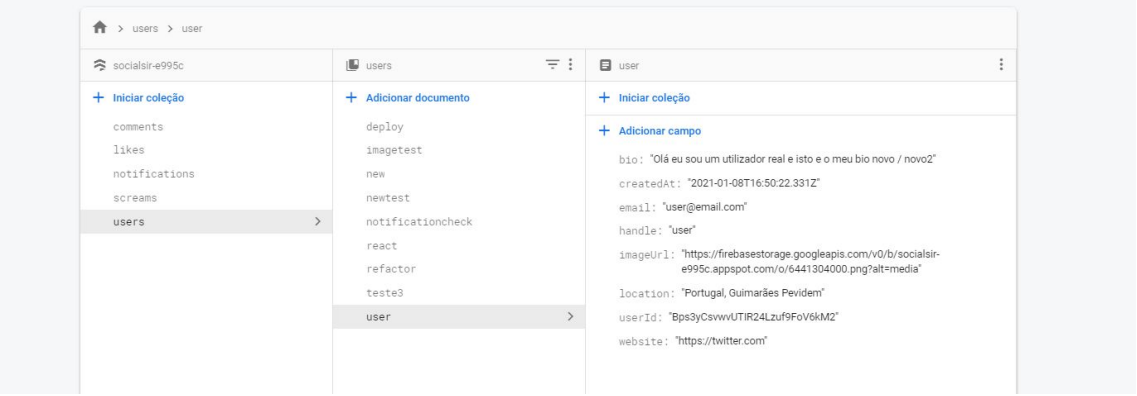

History
Collections
APIs
Save Responses
Clear all
Today
POST http://localhost:5000/socialsir-e995c/us-central1/api/user
POST http://localhost:5000/socialsir-e995c/us-central1/api/user
POST http://localhost:5000/socialsir-e995c/us-central1/api/user
POST http://localhost:5000/socialsir-e995c/us-central1/api/login
POST http://localhost:5000/socialsir-e995c/us-central1/api/user
Yesterday
POST http://localhost:5000/socialsir-e995c/us-central1/api/user/image
POST http://localhost:5000/socialsir-e995c/us-central1/api/user/image
POST http://localhost:5000/socialsir-e995c/us-central1/api/user/image
POST http://localhost:5000/socialsir-e995c/us-central1/api/user/image
POST http://localhost:5000/socialsir-e995c/us-central1/api/user/image
POST http://localhost:5000/socialsir-e995c/us-central1/api/signup
January 12
Untitled Request
BUILD
POST
http://localhost:5000/socialsir-e995c/us-central1/api/user
Send
Save
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
1 {
2 "bio" :"OLA OLA OLA OLA OLA ",
3 "website":"https://google.com",
4 "location": "Viana, PT"
5 }
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 827 ms
Size: 281 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "message": "Detalhes atualizados com sucesso"
3 }

## 3.5 Busca de dados de um utilizador

Para a busca de dados de um utilizador específico.

```
//detalhes de utilizador
exports.getDetalhesUser = (req, res) => {
  let userData = {};
  db.doc(`/users/${req.params.handle}`)
    .get()
    .then((doc) => {
      if (doc.exists) {
        userData.user = doc.data();
        return db
          .collection("screams")
          .where("userHandle", "==", req.params.handle)
          .orderBy("createdAt", "desc")
          .get();
      } else {
        return res.status(404).json({ error: "Utilizador não foi encontrado" });
      }
    })
    .then((data) => {      You, 9 days ago • working on user and notifications
      userData.screams = [];
      data.forEach((doc) => {
        userData.screams.push({
          body: doc.data().body,
          createdAt: doc.data().createdAt,
          userHandle: doc.data().userHandle,
          userImage: doc.data().userImage,
          likeCount: doc.data().likeCount,
          commentCount: doc.data().commentCount,
          screamId: doc.id,
        });
      });
      return res.json(userData);
    })
    .catch((err) => {
      console.error(err);
      return res.status(500).json({ error: err.code });
    });
};
```

Aparência de um utilizador na base de dados firebase - Documentos, Coleções



## 3.6 Buscar todos os posts / screams

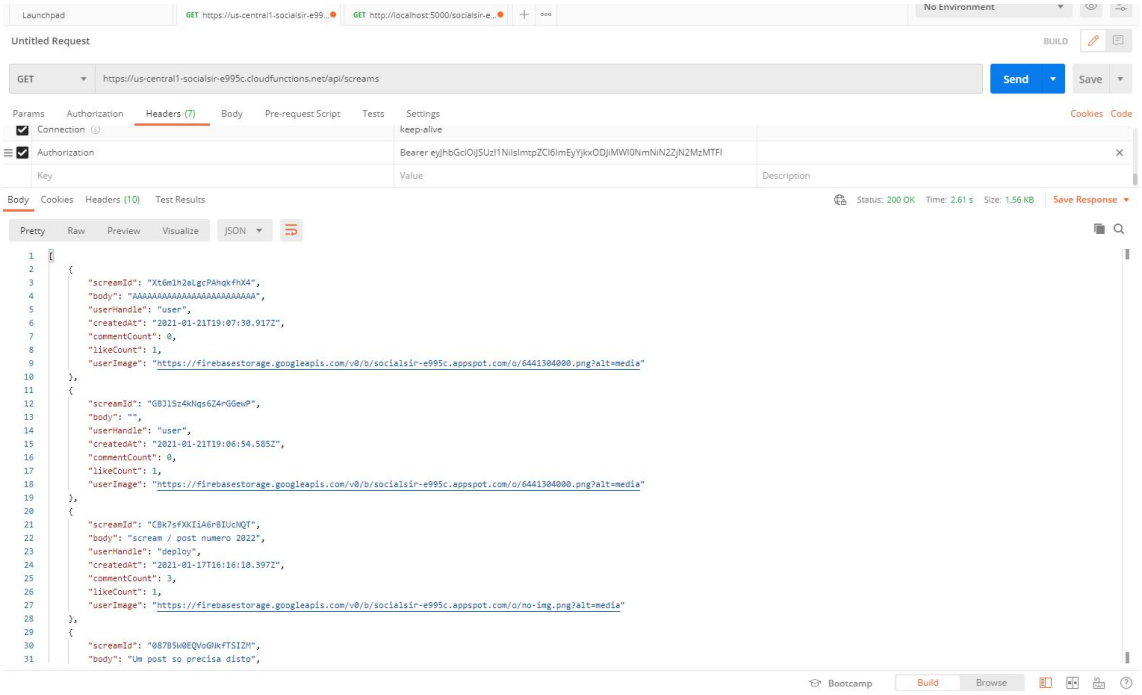
Enquanto seguíamos um tutorial eles chamavam aos posts de screams e acabamos por dar o mesmo nome também.

A variável db está guardada numa função chamada de admin que inicia a Firebase, esta variável escrita em longo seria admin.firestore() que daria então acesso à base de dados do firebase.

```
You, 13 days ago | 1 author (You)
const admin = require('firebase-admin');
admin.initializeApp();
const db = admin.firestore();



module.exports = { admin ,db}; // exportar pros outros ficheiros
```

```
(property) getAllScreams: (req: any, res: any) => void
exports.getAllScreams = (req,res) => {
    db        You, 13 days ago • Refactor + FBAuth
      .collection('screams')
      .orderBy('createdAt','desc')
      .get()
      .then((data) =>{
            let screams = [];
            data.forEach((doc) =>{
            screams.push({
              screamId: doc.id, //screamID
              body: doc.data().body,
              userHandle: doc.data().userHandle,
              createdAt: doc.data().createdAt,
              commentCount: doc.data().commentCount,
              likeCount: doc.data().likeCount,
              userImage: doc.data().userImage
            });
            });
            return res.json(screams);
          })
          .catch((err) => console.error(err));
    };
```

## Teste de postman do get all screams

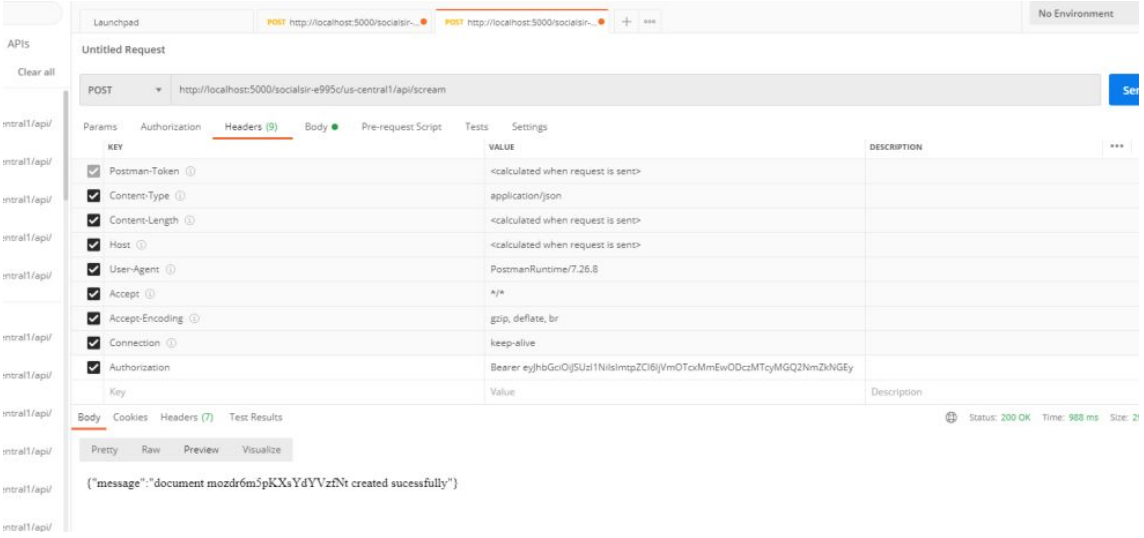


## 3.7 Fazer um post novo

Para fazer um post um utilizador deve fazer login para passar o seu handle no request depois apenas precisa de um body para realizar o post em questão.

```
exports.postOneScream = (req, res)=>{
    if(req.body.body.trim() === ''){
      res.status(400).json({body: 'Body nao pode estar vazio'});
    }

  const newScream = {
    body: req.body.body,
    userHandle: req.user.handle,
    userImage: req.user.imageUrl,
    createdAt: new Date().toISOString(),
    likeCount: 0,
    commentCount: 0
  };

  db
        .collection('screams')
        .add(newScream)
        .then(doc => {
          const resScream = newScream;
          resScream.screamId = doc.id;
          res.json(resScream);

        })
        .catch(err => {
          res.status(500).json({ error: 'something went wrong'});
          console.error(err);
        });
    };
```

**Primeiro post criado com o postman :**

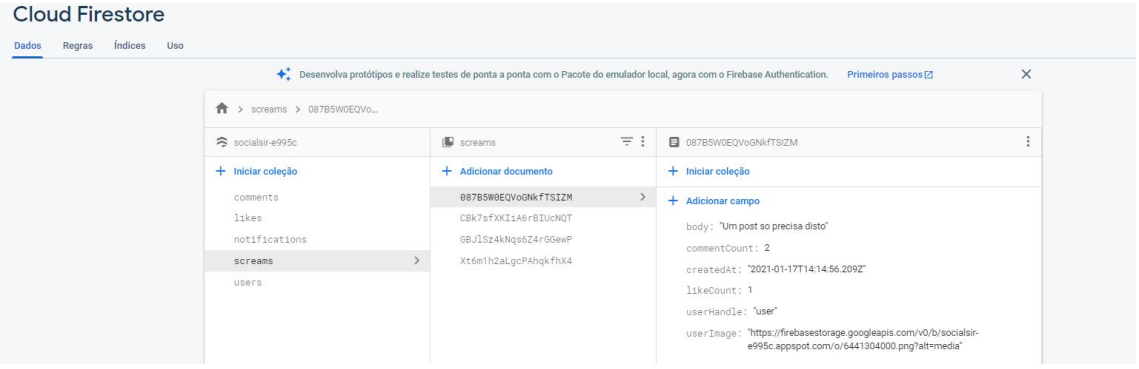


## 3.8 Busca de um post específico

Para podermos mais tarde abrir um post/scream específico no frontend decidimos criar uma busca por um post específico pelo id.

```
// buscar um scream
exports.getScream = (req,res) => {
  let screamData = {};
  db.doc(`/screams/${req.params.screamId}`).get()
  .then(doc => {
    if(!doc.exists){
      return res.status(404).json({error: 'Scream não encotrado'})
    }
    screamData = doc.data();
    screamData.screamId = doc.id;
    return db
    .collection('comments')
    .orderBy('createdAt','desc')
    .where('screamId','==',req.params.screamId)
    .get();
  })
  .then(data => {
    screamData.comments = [];
    data.forEach(doc => {
      screamData.comments.push(doc.data())
    });
    return res.json(screamData);
  })
  .catch(err => {
    console.error(err);
    res.status(500).json({error: err.code});
  })
};
```

Os documentos e coleções de um post presentes no Cloud Firestore.

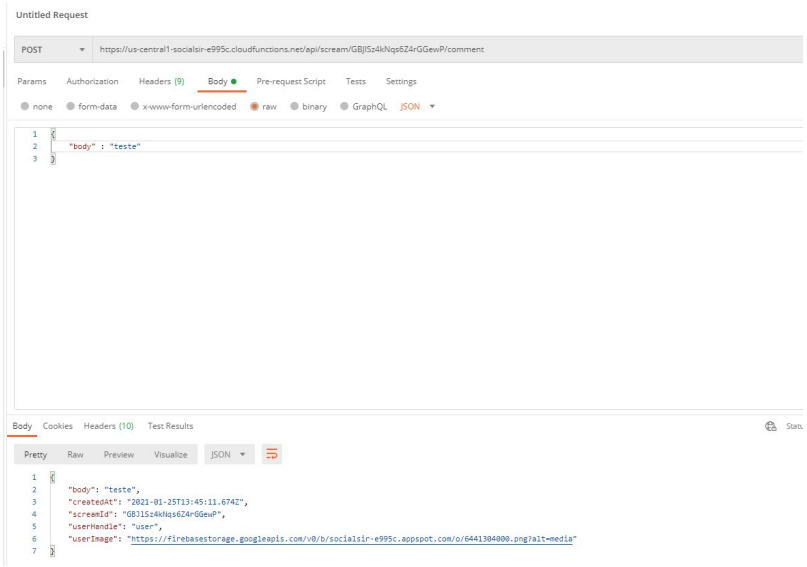


## 3.9 Criação de um comentário

Para se criar um comentário tem se que passar o ID do post em questão. Usou-se a função predefinida toISOString para processar a data em uma string, se o post pedido existir então incrementa-se o commentCount do post em questão e prossegue-se em adicionar o comentário novo à base de dados.

```
//comment em scream
exports.commentarioEmScream = (req,res) => {
  if(req.body.body.trim() === '') return res.status(400).json({ comment : 'Não deve estar vazio'});

  const newComment = {
    body: req.body.body,
    createdAt: new Date().toISOString(),
    screamId: req.params.screamId,
    userHandle: req.user.handle,
    userImage: req.user.imageUrl
  };

  db.doc(`/screams/${req.params.screamId}`).get()
  .then(doc => {
    if(!doc.exists){
      res.status(404).json({error: 'Scream não existe'});
    }
    return doc.ref.update({ commentCount: doc.data().commentCount + 1});
  })
  .then(() =>{        You, 10 days ago • Likes/Unlike + Delete Scream(post) + CommentCount
    return db.collection('comments').add(newComment);
  })
  .then(() => {
    res.json(newComment);
  })
  .catch(err => {
    console.log(err);
    res.status(500).json({error: 'Algo está errado :)'});
  });
};
```

**Teste de postman de criar um comentário**



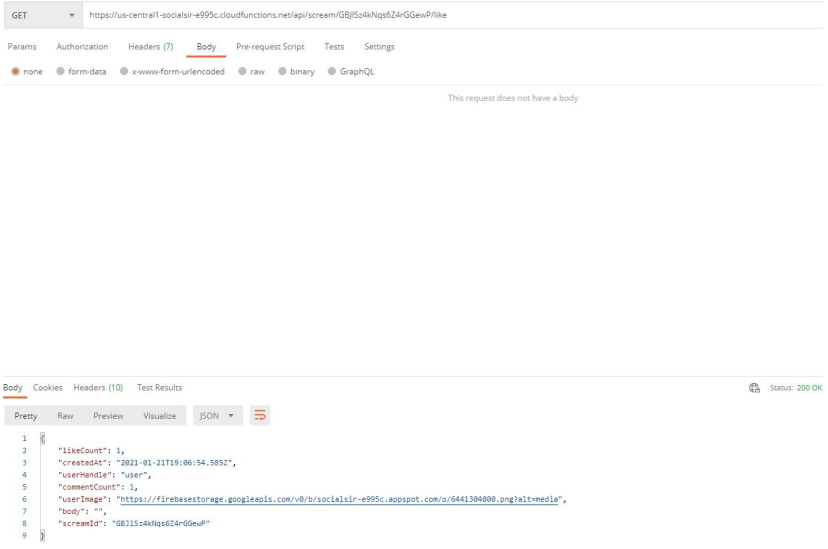
## 3.10 Criação e atribuição de um Like

Para se atribuir um like vai-se necessitar do ID do post em questão e claramente do utilizador a fazer o like.

Primeiro verifica-se se o post/scream  a que se está a atribuir o like existe, se existir guarda-se então a data  depois na promessa verifica-se se o like já existe caso não exista incrementamos o like count e cria-se o like na coleção da base de dados Likes.

```
//likes
exports.likeScream = (req, res) => {
  const likeDocument = db
    .collection('likes')
    .where('userHandle', '==', req.user.handle)
    .where('screamId', '==', req.params.screamId)
    .limit(1);

  const screamDocument = db.doc(`/screams/${req.params.screamId}`);

  let screamData;

  screamDocument
    .get()
    .then((doc) => {
      if (doc.exists) {      You, 10 days ago • Likes/Unlike + Delete Scream(post) + CommentCount
        screamData = doc.data();
        screamData.screamId = doc.id;
        return likeDocument.get();
      } else {
        return res.status(404).json({ error: 'Scream não encontrado' });
      }
    })
    .then((data) => {
      if (data.empty) {
        return db
          .collection('likes')
          .add({
            screamId: req.params.screamId,
            userHandle: req.user.handle
          })
          .then(() => {
            screamData.likeCount++;
            return screamDocument.update({ likeCount: screamData.likeCount });
          })
          .then(() => {
            return res.json(screamData);
          });
      } else {
        return res.status(400).json({ error: 'Scream já tem like' });
      }
    })
    .catch((err) => {
      console.error(err);
      res.status(500).json({ error: err.code });
    });
};
```

**Like no postman**

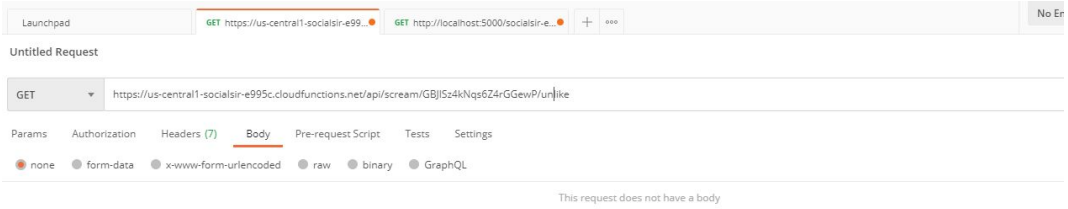


## 3.11 Remover o Like (Unlike)

Para retirar o like usa-se basicamente a mesma função anterior com a exceção na parte do que fazemos se o like já existir. Neste caso a data foi devolvida como vazia, dizemos que o like não existe senão removemos o like da collection e reduzimos o likecount.

```
exports.unlikeScream = (req, res) => {
  const likeDocument = db
    .collection('likes')
    .where('userHandle', '==', req.user.handle)
    .where('screamId', '==', req.params.screamId)
    .limit(1);

  const screamDocument = db.doc(`/screams/${req.params.screamId}`);

  let screamData;

  screamDocument
    .get()
    .then((doc) => {
      if (doc.exists) {
        screamData = doc.data();
        screamData.screamId = doc.id;
        return likeDocument.get();
      } else {
        return res.status(404).json({ error: 'Scream não encontrado' });
      }
    })
    .then((data) => {
      if (data.empty) {
        return res.status(400).json({ error: 'Scream ainda não levou like' });
      } else {
        return db
          .doc(`/likes/${data.docs[0].id}`)
          .delete()
          .then(() => {
            screamData.likeCount--;
            return screamDocument.update({ likeCount: screamData.likeCount });
          })
          .then(() => {
            res.json(screamData);
          });
      }
    })
    .catch((err) => {
      console.error(err);
      res.status(500).json({ error: err.code });
    });
};
```

**Unlike no postman**

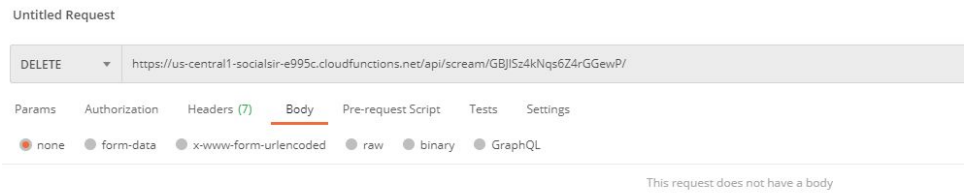


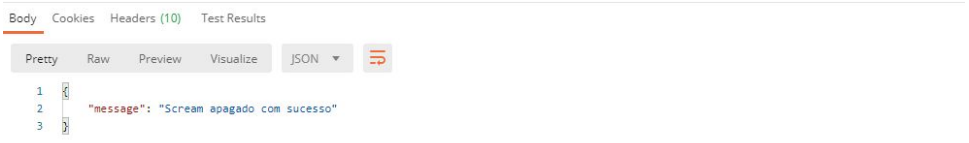
## 3.12 Apagar um post/scream

Para apagar o post vamos precisar do ID do post e do utilizador que está a realizar a ação verificamos se o utilizador a realizar a ação é o mesmo que está a tentar apagar o post e caso seja usamos o delete para apagar o post .

```
//apagar scream
exports.apagarScream = (req,res) => {const document = db.doc(`/screams/${req.params.screamId}`);
document
  .get()
  .then((doc) => {
    if (!doc.exists) {
      return res.status(404).json({ error: 'Scream não encontrado' });
    }
    if (doc.data().userHandle !== req.user.handle) {
      return res.status(403).json({ error: 'Não autorizado' });
    } else {
      return document.delete();
    }
  })
  .then(() => {
    res.json({ message: 'Scream apagado com sucesso' });
  })
  .catch((err) => {
    console.error(err);
    return res.status(500).json({ error: err.code });
  });
};
```

**Apagar um scream no postman**




**Trigger ao apagar um scream/post**

Usando o onDelete quando for detectado que um post tenha sido apagado procuramos então todos os likes, comentários e notificações que tenham o id do scream igual ao que foi apagado e apagamos estes documentos também.

```
exports.onScreamDelete = functions
.region('europe-west1')
.firestore.document('/screams/{screamId}')
.onDelete((snapshot, context) => {
  const screamId = context.params.screamId;
  const batch = db.batch();
  return db
    .collection('comments')
    .where('screamId', '==', screamId)
    .get()
    .then((data) => {
      data.forEach((doc) => {
        batch.delete(db.doc(`/comments/${doc.id}`));
      });
      return db
        .collection('likes')
        .where('screamId', '==', screamId)
        .get();
    })
    .then((data) => {
      data.forEach((doc) => {
        batch.delete(db.doc(`/likes/${doc.id}`));
      });
      return db
        .collection('notifications')
        .where('screamId', '==', screamId)
        .get();        You, 8 days ago • finalização do backend
    })
    .then((data) => {
      data.forEach((doc) => {
        batch.delete(db.doc(`/notifications/${doc.id}`));
      });
      return batch.commit();
    })
    .catch((err) => console.error(err));
});
```

## 3.13 Notificações

Nós queríamos informar o utilizado quando receber likes e comentários no seu post post por isso decidimos usar triggers na base de dados para gerar informação para uma nova coleção designada notifications para depois aceder-mos e mostrar ao utilizador a informação desejada

Usamos on on create para quando for criado um novo like na coleção iniciarmos o trigger, verificamos se o scream em questão existe (pode ter sido apagado) e caso sim então procedemos à criação de uma notificação na coleção notification usamos o snapshot para guardar o id do like para mais tarde ser acedido.

```
// DONE - FAZER FUNÇOES PARA AS NOTIFICAÇÕES
// Sem solução cloud functions são lentas - RESOLVER O DELAY ENORME DOS TRIGGERS
exports.createNotificationOnLike = functions
.region('europe-west1')
  .firestore.document('likes/{id}')
  .onCreate((snapshot) => {
    return db
      .doc(`/screams/${snapshot.data().screamId}`)
      .get()
      .then((doc) => {
        if (
          doc.exists
        ) {
          return db.doc(`/notifications/${snapshot.id}`).set({
            createdAt: new Date().toISOString(),
            recipient: doc.data().userHandle,
            sender: snapshot.data().userHandle,
            type: 'like',
            read: false,
            screamId: doc.id
          });
        }
      })
      .catch((err) => console.error(err));
  });
```

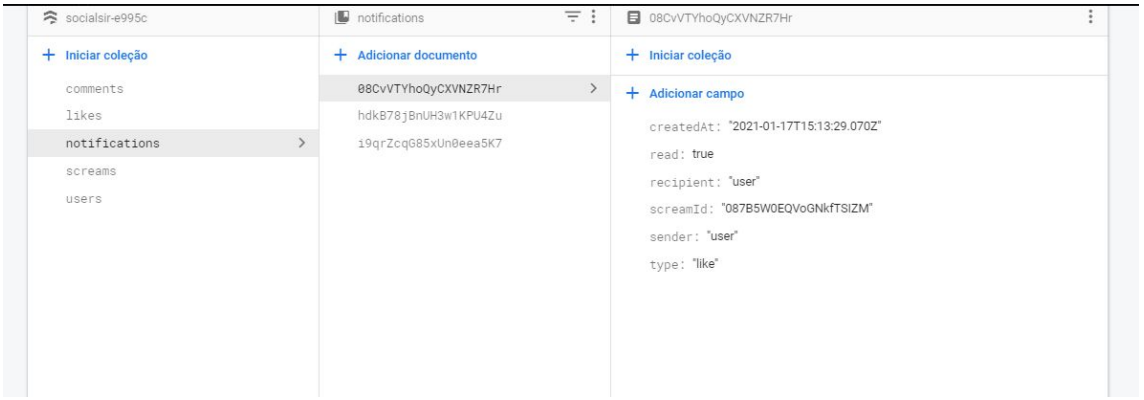
Para apagarmos a notificação iniciamos um trigger na coleção likes só que desta vez onDelete.

```
exports.deleteNotificationOnUnLike = functions
.region('europe-west1')
  .firestore.document('likes/{id}')
  .onDelete((snapshot) => {
    return db        You, 9 days ago • working on user and notifications
      .doc(`/notifications/${snapshot.id}`)
      .delete()
      .catch((err) => {
        console.error(err);
        return;
      });
  });
```

Para a notificação dos comentários é a mesma coisa dos likes só que desta vez verificamos se o utilizador a criar o comentário é o mesmo que está a receber a notificação para não o notificar da acção.

```
exports.createNotificationOnComment = functions
.region('europe-west1')
  .firestore.document('comments/{id}')
  .onCreate((snapshot) => {
    return db
      .doc(`/screams/${snapshot.data().screamId}`)
      .get()
      .then((doc) => {
        if (
          doc.exists &&
          doc.data().userHandle !== snapshot.data().userHandle
        ) {
          return db.doc(`/notifications/${snapshot.id}`).set({
            createdAt: new Date().toISOString(),
            recipient: doc.data().userHandle,
            sender: snapshot.data().userHandle,
            type: 'comment',
            read: false,
            screamId: doc.id
          });
        }
      })
      .catch((err) => {
        console.error(err);
        return;
      });
  });
```

Notificação na base de dados



## 3.14 Atualização de imagem

Para atualizar a imagem usamos a module busboy que serve para parsing de data. path e os que são default packages instaladas em projetos de node.js.

Após recebermos a imagem iniciamos o busboy.on que precisa de fieldname , ficheiro , nome do ficheiro e mimetype que um identificador para formatos.

No nome da imagem (ImageExtension) damos split para tratar pontos finais no nome da imagem porque caso contrário dava erro.

Damos um nome random ao nome para ficarmos com o nome da imagem

para o path da imagem usamos o os.tmpdir() para ir buscar o namespace do sistema operativo tmpdir porque isto não é um server mas uma função da cloud. e juntamos o image.filename

depois temos o filepath e o mymetype como o final para a imagem que vai receber upload.

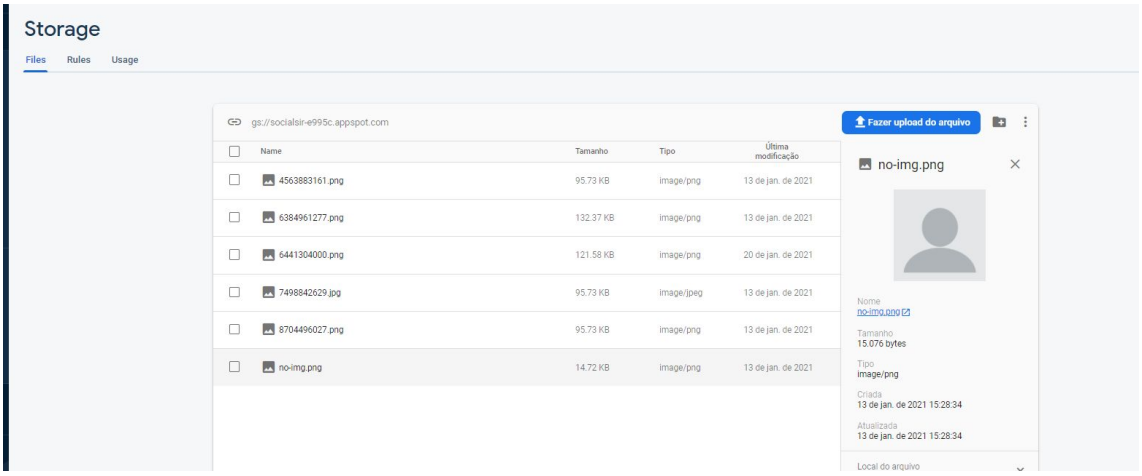
de seguida usamos o a libraria de file para criar o ficheiro

Quando isto estiver completo o busboy on finish começa e fazemos o upload para a storage do firebase.

Nas promessas vamos buscar o URL da imagem criada e depois fazemos update ao user em si

```
//profile image para o utilizador
//module busboy com npm install --save busboy
  exports.uploadImage = (req,res) => {
const BusBoy = require('busboy');
const path = require('path');
const os = require('os');
const fs = require('fs');

const busboy = new BusBoy({ headers: req.headers});

let imageFileName;
let imageToBeUploaded = {};


busboy.on('file',(fieldname,file,filename,encoding,mimetype) => {
  console.log(fieldname);
  console.log(filename);
  console.log(mimetype);
  // my.image.png iria causar problemas por isso dividimos por .
  const imageExtension = filename.split('.')[filename.split('.').length - 1];
  //7543267345643756.png
  imageFileName = `${Math.round(Math.random()*10000000000)}.${imageExtension}`;
  const filepath = path.join(os.tmpdir(), imageFileName);
  imageToBeUploaded = {filepath,mimetype};
  file.pipe(fs.createWriteStream(filepath));
}); //firebase doc sdk documentação
busboy.on('finish', () =>{
admin.storage().bucket().upload(imageToBeUploaded.filepath, {
  resumable: false,
  metadata:{
    metadata: {
      contentType: imageToBeUploaded.mimetype
    }
  }
})
.then(() =>{
  const imageUrl = `https://firebasestorage.googleapis.com/v0/b/${config.storageBucket}/o/${imageFileName}?alt=media`;
  return db.doc(`/users/${req.user.handle}`).update({imageUrl});
})
.then(() => {
  return res.json({message: 'Imagem uploaded com sucesso'});
})
.catch(err =>{
  console.error(err);
  return res.status(500).json({error: err.code});
});
});
busboy.end(req.rawBody);
```

**Imagens guardadas na storage**




## Trigger na mudança de imagem

No trigger seguinte verificamos se a imagem foi alterada por uma diferente da anterior. procuramos pelos posts que tenham o handle do utilizador que está a mudar de imagem e para cada um passamos a nova imagem

```
exports.onUserImageChange = functions
.region('europe-west1')
.firestore.document('/users/{userId}')
.onUpdate((change) => {
  console.log(change.before.data());
  console.log(change.after.data());
  if (change.before.data().imageUrl !== change.after.data().imageUrl) {
    console.log('a imagem mudou');
    const batch = db.batch();
    return db
      .collection('screams')
      .where('userHandle', '==', change.before.data().handle)
      .get()
      .then((data) => {
        data.forEach((doc) => {
          const scream = db.doc(`/screams/${doc.id}`);
          batch.update(scream, { userImage: change.after.data().imageUrl });
        });
        return batch.commit();
      });
  } else return true;
});
```

## Imagens para o Post / Postar imagens

Mesmo código com exceção do caminho

```
    }
  })
  .then(() =>{
    const imageUrl = `https://firebasestorage.googleapis.com/v0/b/${config.storageBucket}/o/${imageFileName}?alt=media`;
    return db.doc(`/screams/${req.params.screamId}`).update({imageUrl});
  })
  .then(() => {
    return res.json({message: 'Imagem uploaded com sucesso'});
  })
  .catch(err =>{
    console.error(err);
    return res.status(500).json({error: err.code});
  });
```

Untitled Request

POST http://localhost:5000/socialsir-e995c/us-central1/api/scream/087B5W0EQVoGNkfTSIZM/image

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

| KEY | VALUE | DESCRIPTION |
|---|---|---|
| ☑ | Untitled.png | |
| Key | Value | Description |

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "message": "Imagem uploaded com sucesso"
}
```

socialsir-e995c

+ Iniciar coleção

- comments
- likes
- notifications
- screams
- users

screams

+ Adicionar documento

- 087B5W0EQVoGNkfTSIZM
- CBk7sfXKIiA6rBIUcNQT
- LSLw6RdPJ7jICaHJJnEx
- Xt6m1h2aLgcPAhqkfhX4
- iDeTlW0wIVUzfcVZ1hpu
- kLuADfACIYYBYkS4WhSx

087B5W0EQVoGNkfTSIZM

+ Iniciar coleção

+ Adicionar campo

```
body: "Um post so precisa disto"
commentCount: 2
createdAt: "2021-01-17T14:14:56.209Z"
imageUrl: "https://firebasestorage.googleapis.com/v0/b/socialsir-e995c.appspot.com/o/2743514678.png?alt=media"
likeCount: 2
userHandle: "user"
```

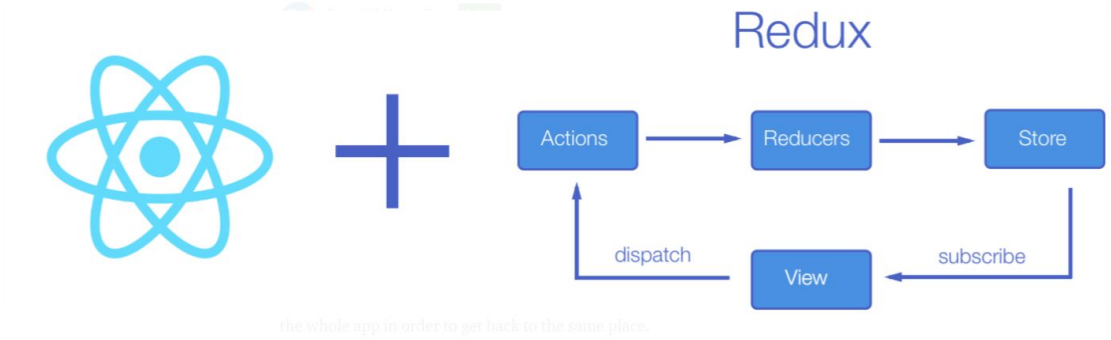
# 4. Frontend

O nosso front end foi desenvolvido com react. Já estávamos a usar React para a disciplina de Projeto III, onde se está a desenvolver uma plataforma "clone" de zoom - RemoteSports - com mais funcionalidades (ex TensorFlowJS) daí já exercer alguma experiência desta framework. Para correr o programa basta fazer um npm install (para instalar todos os módulos - node_modules) e de seguida de npm start para iniciar o script do react. Usamos o material UI para facilitar o design e o redux para tentar melhorar a performance e também nos permitir ver as alterações sem precisar de refrescar a página, por exemplo ver o post que nós fizemos aparecer sem ser preciso f5

Frameworks utilizadas para a construção do front end.

**Material UI :**

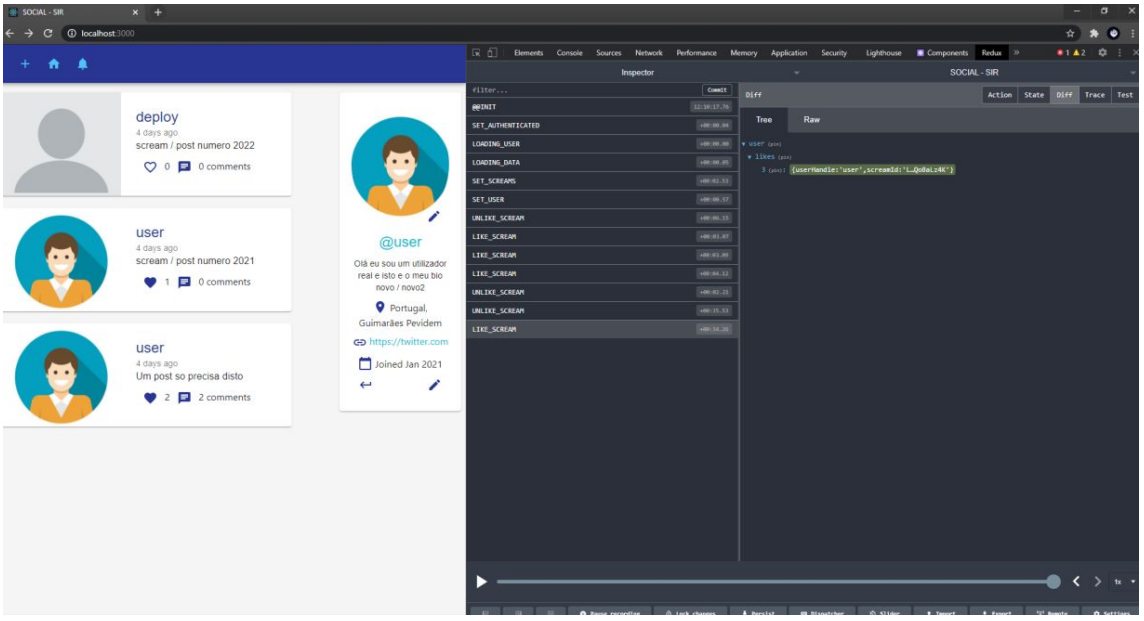
**React Redux :**

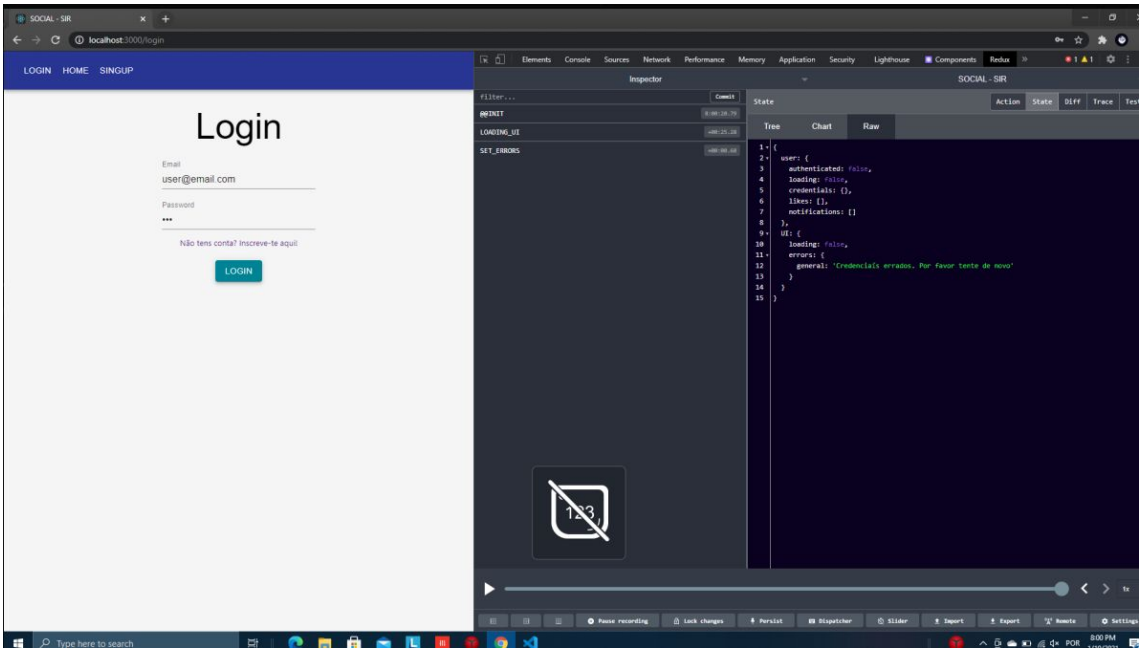


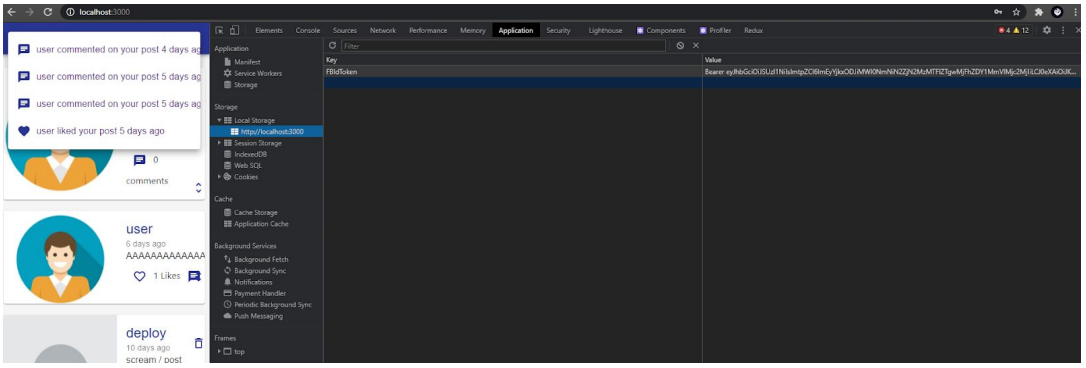

Rotas das páginas que vamos ter no website:

```
function App() {
  return (
    <MuiThemeProvider theme ={theme}>
      <Provider store={store}>
      <div className="App">
     <Router>
     <NavBar/>
       <div className ="container">
     <Switch>
       <Route exact path ="/" component = {home}/>
       <AuthRoute exact path ="/login" component = {login}/>
       <AuthRoute exact path ="/signup" component = {signup}/>
       <Route exact path ="/users/:handle" component = {user}/>
       <Route exact path ="/users/:handle/scream/:screamId" component = {user}/>
     </Switch>
       </div>
    </Router>
   </div>
      </Provider>

   </MuiThemeProvider>
  );
}
```

**Usamos também as extensões do react e do redux para google chrome para ajudar o nosso desenvolvimento.**



## *4.1 Reducers*

Reducers são ações que pegam no state fazem uma ação e depois devolvem o novo state

### 4.1.1 UserReducer

```
export default function(state = initialState, action) {
  switch (action.type) {
    case SET_AUTHENTICATED:
      return {
        ...state, //devolve o estado
        authenticated: true //metemos como autenticado
      };
    case SET_UNAUTHENTICATED:
      return initialState; //estado inicial tem autenticado como false
    case SET_USER:
      return {
        authenticated: true,
        loading: false,
        ...action.payload //payload e o authenticated loading credential ... etc desta maneira ligamos os likes aos likes as notifications as notifications
      };
      case LOADING_USER:
        return {
          ...state,
          loading: true
        };

  case LIKE_SCREAM:
    return {
      ...state,
      likes: [
        ...state.likes,
        {
          userHandle: state.credentials.handle,
          screamId: action.payload.screamId
        }
      ]
    };
  case UNLIKE_SCREAM:
    return {
      ...state,
      likes: state.likes.filter(
        (like) => like.screamId !== action.payload.screamId //para n alterar os outros apenas o que queremos
      )
    };
    case MARK_NOTIFICATIONS_READ:
    state.notifications.forEach((not) => (not.read = true)); //mudar o atributo read de false para true      You, 4 days ago • Finishing notifications profil
    return {
      ...state
    };
  default:
    return state;
}
```

### 4.1.2 UiReducer

```
};

export default function(state = initialState, action) {
  switch (action.type) {
    case SET_ERRORS:
      return {
        ...state,
        loading: false,
        errors: action.payload
      };
    case CLEAR_ERRORS:
      return {
        ...state,
        loading: false,
        errors: null
      };
    case LOADING_UI:
      return {
        ...state,
        loading: true
      };
      case STOP_LOADING_UI:
        return {
          ...state,
          loading: false
        };
    default:      You, 7 days ago • Initial commit
      return state;
  }
}
```

### 4.1.3 DataReducer

```
switch (action.type) {
    case LOADING_DATA:
        return {
            ...state,
            loading: true
        };
    case SET_SCREAMS:
        return {
            ...state,
            screams: action.payload,
            loading: false
        };
    case SET_SCREAM:
        return {
            ...state,
            scream: action.payload
        };
    case LIKE_SCREAM:
    case UNLIKE_SCREAM:
        let index = state.screams.findIndex(
            (scream) => scream.screamId === action.payload.screamId
        );
        state.screams[index] = action.payload;
        if (state.scream.screamId === action.payload.screamId) {
            state.scream = action.payload;
        }
        return {
            ...state
        };
    case DELETE_SCREAM:
        index = state.screams.findIndex(
            (scream) => scream.screamId === action.payload  //encontrar um scream com o mesmo id com o que enviamos
        );
        state.screams.splice(index, 1); //splice remove o scream
        return {
            ...state
        };
    case POST_SCREAM:
        return {
            ...state,
            screams: [action.payload, ...state.screams]
        };
    case SUBMIT_COMMENT:
        return {
            ...state,
            scream: {
                ...state.scream,
                comments: [action.payload, ...state.scream.comments]
```

## *4.2 React Redux - Em Prática*

### 4.2.1 Login

Recebemos os dados do utilizador que vai conter o email e o login e realizamos uma ação de post login depois atribuímos a token ao utilizador e manda-o para a página home, o dispatch realiza a ação e é a única maneira de alterar o state, os tipos são as ações do reducer

```
import axios from 'axios';
  //asyncrono por isso e preciso dispatch
export const loginUser = (dataUtilizador,history) => (dispatch) => {
    dispatch({type: LOADING_UI});
  axios.post('/login',dataUtilizador)
  .then(res=>{
    setAuthorizationHeader(res.data.token);
      dispatch(getUserData());
      dispatch({type: CLEAR_ERRORS});
      history.push('/'); //push state redireciona a homepage
  })
  .catch(err => {
      dispatch({
          type: SET_ERRORS,
          payload: err.response.data
      })
  });      You, 7 days ago • Initial commit
}
```

A função seguinte é chamada para atribuirmos o token ao utilizador

```
const setAuthorizationHeader = (token) => {
  const FBIdToken = `Bearer ${token}`;
  localStorage.setItem('FBIdToken', FBIdToken);
  axios.defaults.headers.common['Authorization'] = FBIdToken;
};
```

A função seguinte é chamada para buscar a data do utilizador

```
29    }
30
31    export const getUserData = () => (dispatch) => {
32      dispatch({type: LOADING_USER});
33      axios.get('/user')
34      .then(res => {
35          dispatch({
36              type: SET_USER,
37              payload: res.data
38          })
39      })
40      .catch(err => console.log(err));
41    }
42
```

### 4.2.2 Signup

```
export const signupUser = (newUserData, history) => (dispatch) => {
  dispatch({ type: LOADING_UI });
  axios
    .post('/signup', newUserData)
    .then((res) => {
      setAuthorizationHeader(res.data.token);
      dispatch(getUserData());
      dispatch({ type: CLEAR_ERRORS });
      history.push('/');
    })
    .catch((err) => {
      dispatch({
        type: SET_ERRORS,
        payload: err.response.data
      });
    });
};
```

### 4.2.3 Logout Upload Edit e Ler Notificações

```
export const logoutUser = () => (dispatch) => {
  localStorage.removeItem('FBIdToken');
  delete axios.defaults.headers.common['Authorization'];
  dispatch({ type: SET_UNAUTHENTICATED });
};


export const uploadImage = (formData) => (dispatch) => {
  dispatch({type: LOADING_USER});
  axios.post('/user/image',formData)
  .then(res => {
    dispatch(getUserData());
  })
  .catch(err => console.log(err));
}

export const editUserDetails = (userDetails) => (dispatch) => {
  dispatch({ type: LOADING_USER });
  axios
    .post('/user', userDetails)
    .then(() => {
      dispatch(getUserData());
    })
    .catch((err) => console.log(err));
};


export const markNotificationsRead = (notificationIds) => (dispatch) => {
  axios
    .post('/notifications', notificationIds)
    .then((res) => {
      dispatch({
        type: MARK_NOTIFICATIONS_READ
      });
    })
    .catch((err) => console.log(err));
};
```

### 4.2.4 Buscar posts

```
  // Get all screams
export const getScreams = () => (dispatch) => {
    dispatch({ type: LOADING_DATA });
    axios
      .get('/screams')
      .then((res) => {
        dispatch({
          type: SET_SCREAMS,
          payload: res.data
        });
      })
      .catch((err) => {
        dispatch({
          type: SET_SCREAMS,
          payload: []
        });
      });
  };

//para abrir 1 unico post em uma pagina
  export const getScream = (screamId) => (dispatch) => {
    dispatch({ type: LOADING_UI });
    axios
      .get(`/scream/${screamId}`)
      .then((res) => {
        dispatch({
          type: SET_SCREAM,
          payload: res.data
        });
        dispatch({ type: STOP_LOADING_UI });
      })
      .catch((err) => console.log(err));
  };
```

### 4.2.5 Like Dislike e delete

```
// Like a scream
export const likeScream = (screamId) => (dispatch) => {
    axios
      .get(`/scream/${screamId}/like`)
      .then((res) => {
        dispatch({
          type: LIKE_SCREAM,
          payload: res.data
        });
      })
      .catch((err) => console.log(err));
  };
  // Unlike a scream
  export const unlikeScream = (screamId) => (dispatch) => {
    axios
      .get(`/scream/${screamId}/unlike`)
      .then((res) => {
        dispatch({
          type: UNLIKE_SCREAM,
          payload: res.data
        });
      })
      .catch((err) => console.log(err));
  };

  export const deleteScream = (screamId) => (dispatch) => {
    axios
      .delete(`/scream/${screamId}`)
      .then(() => {
        dispatch({ type: DELETE_SCREAM, payload: screamId });
      })
      .catch((err) => console.log(err));
  };
```

### 4.2.6 Postagens

Criamos a função clearErros() porque quando cometemos um erro a postar ficava sempre atribuído esse erro e não podíamos continuar. Por exemplo, se tentasse enviar um comentário vazio avisava sempre que estava vazio mesmo quando já tínhamos preenchido.

```
  // Post scream
export const postScream = (newScream) => (dispatch) => {
  dispatch({ type: LOADING_UI });
  axios
    .post('/scream', newScream)
    .then((res) => {
      dispatch({
        type: POST_SCREAM,
        payload: res.data
      });
      dispatch({type: CLEAR_ERRORS});
    })
    .catch((err) => {
      dispatch({ // Podemos ter erros de validação
        type: SET_ERRORS,
        payload: err.response.data
      });
    });
};

// submet comentario
export const submitComment = (screamId, commentData) => (dispatch) => {
  axios
    .post(`/scream/${screamId}/comment`, commentData)
    .then((res) => {
      dispatch({
        type: SUBMIT_COMMENT,
        payload: res.data
      });
      dispatch(clearErrors());
    })
    .catch((err) => {
      dispatch({
        type: SET_ERRORS,
        payload: err.response.data
      });
    });
};
```

```
export const clearErrors = () => (dispatch) => {
  dispatch({ type: CLEAR_ERRORS });
};
```

## *4.3 Páginas*

**Para a construção das páginas usamos as seguintes ferramentas:**

React prototypes para verificar se estamos a receber o tipo de dados corretos no componente; handle(submit,change,etc) para realizar ações.

### 4.3.1 Login

***Design:***

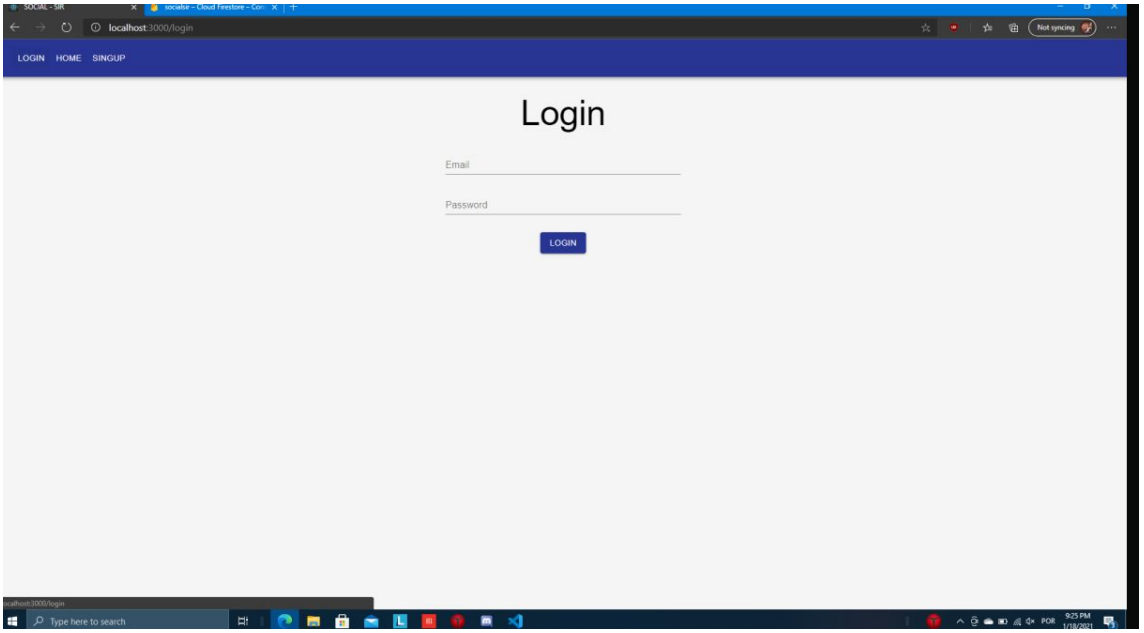
```
//materialui - textfield para documentação do textfield
render() {
    const  {classes,UI:{loading}} = this.props;
    const {errors} = this.state;
    return ( //segundo item sm para centrar
        //helpertext mostra o aviso para utilizador escrever o email
        //TODO descobrir porque e que nao avisa se a password estiver vazia
        //se tivermos a key error significa que existe erro no email ou password (error = {error.email ? ....})
       <Grid container className={classes.form}>
        <Grid item sm/>
        <Grid item sm>
            <Typography variant="h2" className={classes.pageTitle}>
                Login
            </Typography>
            <form noValidate onSubmit={this.handleSubmit}>
                <TextField id="email" name="email" type="email" label="Email" className = {classes.textField} helperText = {errors.email} error= {errors.email
                value={this.state.email} onChange={this.handleChange} fullWidth/>

                <TextField id="password" name="password" type="password" label="Password" className = {classes.textField} helperText = {errors.password} error
                value={this.state.password} onChange={this.handleChange} fullWidth/>

                    {errors.general && (
                        <Typography variant="body2" className={classes.customError}>
                            {errors.general}
                        </Typography>
                    )}
                     <small><Link to="/signup" color="primary">Não tens conta? Inscreve-te aqui!</Link></small>
                     <br/>
                <Button type="submit" variant="contained" color="primary" className={classes.button} disabled={loading}>
                    Login
                {loading && (
                    <CircularProgress size={30} className={classes.loading}/>
                )}
                </Button>

            </form>
        </Grid>
        <Grid item sm/>
       </Grid>
    )
}
```

***HandleSubmit:***

Prevent default porque senão a password aparecia no endereço, pegou-se no email e password e realizou-se a ação loginUser

```
//todo make errors show on screan
componentWillReceiveProps(nextProps){ //resolve os erros não aparecerem
    if(nextProps.UI.error){ //estamos sempre a receber props / melhora performance
    this.setState({errors: nextProps.UI.errors});
    }
}
handleSubmit = (event) =>{
    event.preventDefault(); // para nao mostrar a pass na parte de cima e nao dar reload a pagina
    const dataUtilizador = {
        email: this.state.email,
        password: this.state.password
    };
    this.props.loginUser(dataUtilizador,this.props.history);


};
```

***Resultado***




## 4.3.2 SignUp

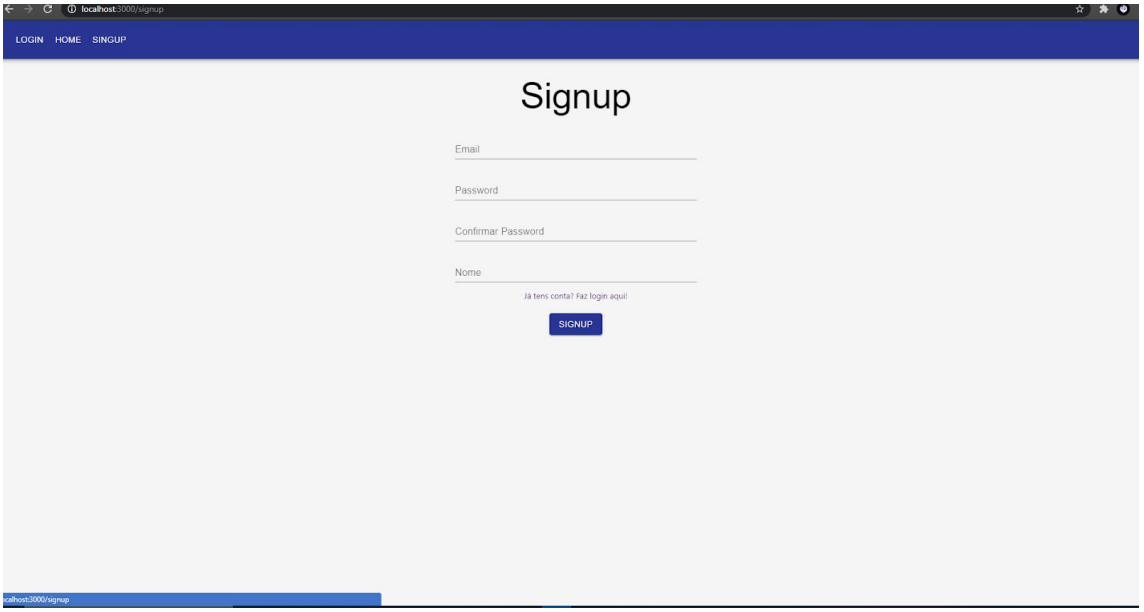
**Handlers Basicamente igual ao login**

```
    //todo make errors show on screan
    componentWillReceiveProps(nextProps){ //resolve os erros não aparecerem
        if(nextProps.UI.error){ //estamos sempre a receber props / melhora performance
        this.setState({errors: nextProps.UI.errors});
        }
    }

    handleSubmit = (event) =>{
        event.preventDefault(); // para nao mostrar a pass na parte de cima e nao dar reload a pagina
        this.setState({
            loading:true
        });
        const newUtilizador = {
            email: this.state.email,
            password: this.state.password,
            confirmPassword: this.state.confirmPassword,
            handle: this.state.handle
        };

        this.props.signupUser(newUtilizador, this.props.history);
    };

    handleChange = (event) => {
        this.setState({
            [event.target.name]: event.target.value //serve para o email e password
        });
    };
        //materialui - textfield para documentação do textfield
```

***Design da pagina***

```
handleChange = (event) => {
    this.setState({
        [event.target.name]: event.target.value //serve para o email e password
    });
};
    //materialui - textfield para documentação do textfield
    render() {
        const  {classes,UI:{loading}} = this.props;
        const {errors} = this.state;
        return ( //segundo item sm para centrar
            //helpertext mostra o aviso para utilizador escrever o email
            //TODO descobrir porque e que nao avisa se a password estiver vazia
            //se tivermos a key error significa que existe erro no email ou password (error = {error.email ? ....})
          <Grid container className={classes.form}>
           <Grid item sm/>
           <Grid item sm>
               <Typography variant="h2" className={classes.pageTitle}>
                   Login
               </Typography>
               <form noValidate onSubmit={this.handleSubmit}>
                   <TextField id="email" name="email" type="email" label="Email" className = {classes.textField} helperText = {errors.email} error= {errors.email ? tru
                   value={this.state.email} onChange={this.handleChange} fullWidth/>

                   <TextField id="password" name="password" type="password" label="Password" className = {classes.textField} helperText = {errors.password} error= {erro
                   value={this.state.password} onChange={this.handleChange} fullWidth/>

                       {errors.general && (
                           <Typography variant="body2" className={classes.customError}>
                               {errors.general}
                           </Typography>
                       )}
                        <small><Link to="/signup" color="primary">Não tens conta? Inscreve-te aqui!</Link></small>
                        <br/>
                   <Button type="submit" variant="contained" color="primary" className={classes.button} disabled={loading}>
                       Login
                   {loading && (       You, 7 days ago • Initial commit
                       <CircularProgress size={30} className={classes.loading}/>
                   )}
                   </Button>

               </form>
           </Grid>
           <Grid item sm/>
          </Grid>
```

***Resultado***

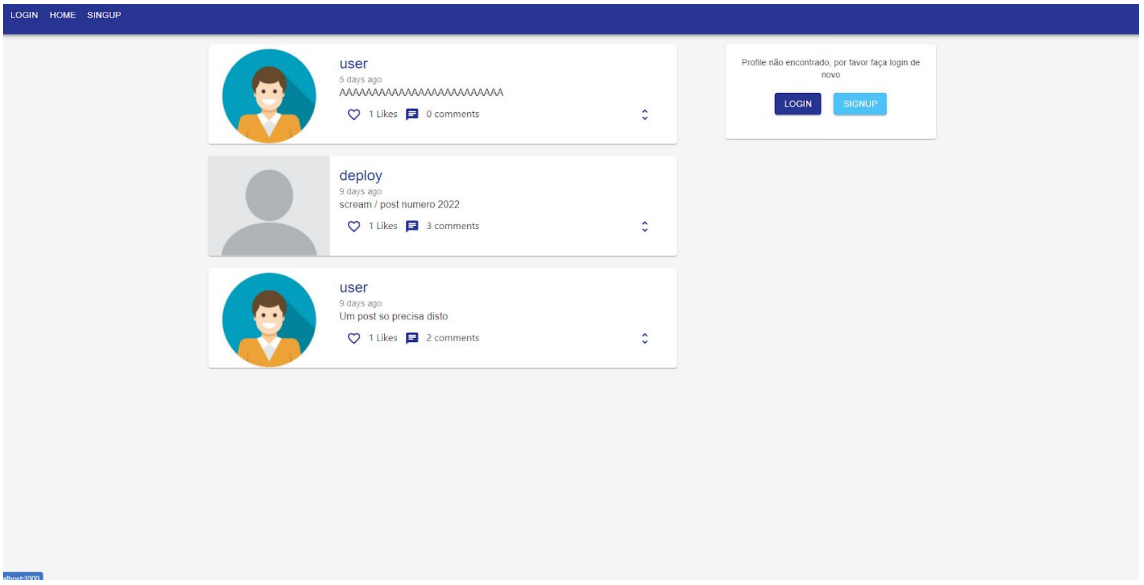


### 4.3.3 HomePage

Para o Home simplesmente vamos buscar os screams ao componente Scream (ver 4.4 componentes) e o Profile ( ver 4.4 componentes )

```
componentDidMount(){
this.props.getScreams();
  /* axios.get('/screams')
        .then((res) => {
                this.setState({
                    screams: res.data
                })
        })
        .catch(err => console.log(err));*/
}


render() {
    const { screams, loading } = this.props.data;
    let recentScreamsMarkup = !loading ? (
      screams.map((scream) => <Scream key={scream.screamId} scream={scream} />) //se não for null significa que já temos a data
    ) : <p>Loading....</p> //senão loading
    return (
        <Grid container spacing={10}>
        <Grid item sm={8} xs={12}>
          {recentScreamsMarkup}
        </Grid>
        <Grid item sm={4} xs={12}>
          <Profile/>
        </Grid>
      </Grid>
```

***Resultado***

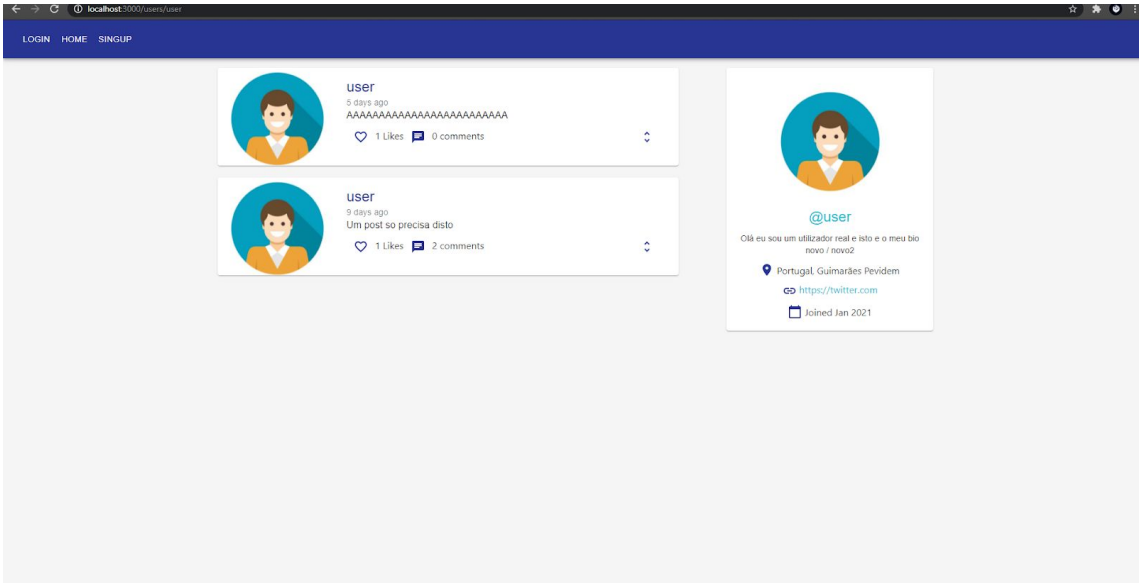


## 4.3.4 Página de Utilizador

Na página de utilizador vamos buscar os screams do mesmo e utilizamos o Component profileEstatico

```
src > pagina1 > JS User.js
        profile: null,
        screamIdParam: null
      };
      componentDidMount() {
        const handle = this.props.match.params.handle; //buscar a handle do user (nome ) ao url
        const screamId = this.props.match.params.screamId;  // para irmos buscar o post

        if (screamId) this.setState({ screamIdParam: screamId }); // se tivermos scream id entao o state screamidparam passa a ser scream ID

        this.props.getUserData(handle); //temos acesso ao handle
        axios //mesmo request da action
          .get(`/user/${handle}`)
          .then((res) => {
            this.setState({
              profile: res.data.user
            });
          })
          .catch((err) => console.log(err));
      }
    render() {
        const { screams, loading } = this.props.data;
        const {screamIdParam} = this.state;
        const screamsMarkup = loading ? (
            <p>LOOOOOADING IN MY SCREEEEEEEEEEEEEN</p>
        ) : screams === null ? (
            <p>ESTE USER NAO TEM POSTS</p>
        ) : !screamIdParam ? (
             screams.map(scream => <Scream key={scream.screamId} scream={scream}/>) // se nao tivermos id param estamos a visitar pagina user
        ) : ( //senao estamos a visitar um post /scream
            screams.map(scream => {
                if(scream.screamId !== screamIdParam) //nao é este o post que queremos abrir
                return <Scream key={scream.screamId} scream={scream}/>
                else return <Scream key={scream.screamId} scream={scream} openDialog/> // scream que nos queremos abrir openDialog com valor de TRUE
            })
        )
        return (
            <Grid container spacing={10}>
            <Grid item sm={8} xs={12}>
              {screamsMarkup}
            </Grid>
            <Grid item sm={4} xs={12}>
                {this.state.profile === null ? (
                    <p>Loading perfil</p>
                ): (
                    <ProfileEstatico profile ={this.state.profile}/>
                )}
            </Grid>
          </Grid>
        )
```

**Resultado**



## *4.4 Componentes*

### 4.4.1 Scream / Post component

Cada post será guardado numa carta na carta vamos ter o nome do poster ligado com o LINK para a sua página botão de like ligado à respetiva ação (Like component) , verificação de ownership do post para botão de apagar , Counts de likes e comentários e um botão (scream dialog) para abrir o post escolhido(ver scream dialog component) , também temos o body e a data do post!

```
render() {
    dayjs.extend(relativeTime)
    const {classes, scream:{ body,createdAt,userImage,userHandle,screamId,likeCount,commentCount},
                    user: {authenticated,credentials: {handle}}
} = this.props;
    const likeButton = !authenticated ? (
        <MyButton tip ="Like">
            <Link to ="/login">
                <FavoriteBorder color ="primary"/>
            </Link>
        </MyButton>
    ) : (       You, 6 days ago • Postar novos posts
        this.likedScream() ? (
            <MyButton tip="Unlike" onClick={this.unlikeScream}>
                <FavoriteIcon color="primary"/>
            </MyButton>
        ) : (
            <MyButton tip="Like" onClick={this.likeScream}>
                <FavoriteBorder color="primary"/>
            </MyButton>
        )
    );
                                                                    //se nao for post do ultizador com login nao tem delete button
    const deleteButton = authenticated && userHandle === handle ? (
        <DeleteScream screamId={screamId}/>
    ) : null
```

```
        <DeleteScream screamId={screamId}/>
    ) : null
    return (
        <Card className={classes.card}>
        <CardMedia
          image={userImage}
          title="Profile image"
          className={classes.image}
        />
        <CardContent className={classes.content}>
          <Typography
            variant="h5"
            component={Link}
            to={`/users/${userHandle}`}
            color="primary"
          >
            {userHandle}
          </Typography>
          {deleteButton}
          <Typography variant="body2" color="textSecondary">
            {dayjs(createdAt).fromNow()}
          </Typography>
          <Typography variant="body1">{body}</Typography>
          <LikeButton screamId={screamId} />
          <span>{likeCount} Likes</span>
          <MyButton tip="comments">
            <ChatIcon color="primary" />
          </MyButton>
          <span>{commentCount} comments</span>
          <ScreamDialog
            screamId={screamId}
            userHandle={userHandle}
            openDialog={this.props.openDialog} // para abrirmos o scream na pagina do user se não tivermos passa um valor nao defenido e não faz nada
          />
        </CardContent>
      </Card>
```

### 4.4.2 ScreamDialog

Usamos o history para quando fecharmos o dialog voltarmos à página anterior onde estávamos.

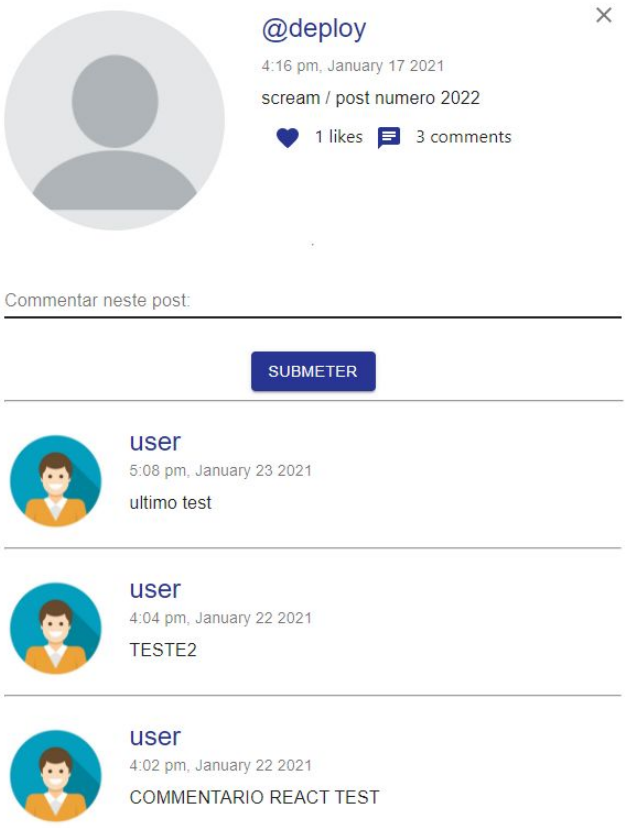
```
class ScreamDialog extends Component {
  state = {
    open: false,
    oldPath: '',        You, 5 days ago • Finishing notifications profile and general - fin…
    newPath: '' //caminhos do url
  };
  componentDidMount() {
    if (this.props.openDialog) { //se tivermos o open dialog
      this.handleOpen(); //abrimos
    }
  }
  handleOpen = () => {
    let oldPath = window.location.pathname; //localizaçao atual

    const { userHandle, screamId } = this.props;
    const newPath = `/users/${userHandle}/scream/${screamId}`; //para quando abrirmos screams diretamente irmos a pagina do utilizador que o postou

    if (oldPath === newPath) oldPath = `/users/${userHandle}`; //se formos diretamente ao link ao postar-lo nao temos old path por isso temois que criar old path
                                  //sem data , sem data, url
    window.history.pushState(null, null, newPath);

    this.setState({ open: true, oldPath, newPath });
    this.props.getScream(this.props.screamId);
  };
  handleClose = () => {
    window.history.pushState(null, null, this.state.oldPath); // quando fecharmos volta ao path anterior
    this.setState({ open: false });
    this.props.clearErrors();
  };

  render() {
```

Botão para abrir o post

```
return (
  <Fragment>
    <MyButton
      onClick={this.handleOpen}
      tip="Expandir scream"
      tipClassName={classes.expandButton}
    >
      <UnfoldMore color="primary" />
    </MyButton>
    <Dialog
      open={this.state.open}
      onClose={this.handleClose}
      fullWidth
      maxWidth="sm"
    >
      <MyButton
        tip="Close"
        onClick={this.handleClose}
        tipClassName={classes.closeButton}
      >
        <CloseIcon />
      </MyButton>
      <DialogContent className={classes.dialogContent}>
        {dialogMarkup}
      </DialogContent>
    </Dialog>
  </Fragment>
```

Após clicar no botão , mostramos uma barra de loading enquanto não estiver pronto, depois mostramos o conteúdo do diálogo. Este contém o mesmo que o post em pequeno mas podemos ver os comentários escritos no mesmo (ver comments component) e também escrever o próximo comentário (ver commentForm component)

```
const dialogMarkup = loading ? (
  <div className={classes.spinnerDiv}>
    <CircularProgress size={200} thickness={2} />
  </div>
) : (        You, 5 days ago • Finishing notifications profile and general -
  <Grid container spacing={16}>
    <Grid item sm={5}>
      <img src={userImage} alt="Profile" className={classes.profileImage} /
    </Grid>
    <Grid item sm={7}>
      <Typography
        component={Link}
        color="primary"
        variant="h5"
        to={`/users/${userHandle}`}
      >
        @{userHandle}
      </Typography>
      <hr className={classes.invisibleSeparator} />
      <Typography variant="body2" color="textSecondary">
        {dayjs(createdAt).format('h:mm a, MMMM DD YYYY')}
      </Typography>
      <hr className={classes.invisibleSeparator} />
      <Typography variant="body1">{body}</Typography>
      <LikeButton screamId={screamId} />
      <span>{likeCount} likes</span>
      <MyButton tip="comments">
        <ChatIcon color="primary" />
      </MyButton>
      <span>{commentCount} comments</span>
    </Grid>
    <hr className={classes.visibleSeperator}/>
    <CommentForm screamId={screamId} />
    <Comments comments={comments}/>
  </Grid>
);
return (
```


@deploy
4:16 pm, January 17 2021
scream / post numero 2022
1 likes
3 comments
Commentar neste post:
SUBMETER
user
5:08 pm, January 23 2021
ultimo test
user
4:04 pm, January 22 2021
TESTE2
user
4:02 pm, January 22 2021
COMMENTARIO REACT TEST

### 4.4.3 Comments component

Utilizamos o index para a barra <hr> que foi estilada para dividir os comentários em si não aparecer no último comentário.

Cada comentário mostra a imagem do commenter e o body.

```
class Comments extends Component {
  render() {
    const { comments, classes } = this.props;
    return (
      <Grid container>
        {comments.map((comment, index) => { // index e para a barra nao parecer no ultimo commentario      You, 5
          const { body, createdAt, userImage, userHandle } = comment;
          return (
            <Fragment key={createdAt}>
              <Grid item sm={12}>
                <Grid container>
                  <Grid item sm={2}>
                    <img
                      src={userImage}
                      alt="comment"
                      className={classes.commentImage}
                    />
                  </Grid>
                  <Grid item sm={9}>
                    <div className={classes.commentData}>
                      <Typography
                        variant="h5"
                        component={Link}
                        to={`/users/${userHandle}`}
                        color="primary"
                      >
                        {userHandle}
                      </Typography>
                      <Typography variant="body2" color="textSecondary">
                        {dayjs(createdAt).format('h:mm a, MMMM DD YYYY')}
                      </Typography>
                      <hr className={classes.invisibleSeparator} />
                      <Typography variabnt="body1">{body}</Typography>
                    </div>
                  </Grid>
                </Grid>
              </Grid>
              {index !== comments.length - 1 && (
                <hr className={classes.visibleSeparator} />
              )}
            </Fragment>
          );
        })}
      </Grid>
    );
  }
}
```

### 4.4.4 CommentForm component

```
handleSubmit = (event) => {
  event.preventDefault();
  this.props.submitComment(this.props.screamId, { body: this.state.body });
};

render() {
  const { classes, authenticated } = this.props;
  const errors = this.state.errors;

  const commentFormMarkup = authenticated ? (
    <Grid item sm={12} style={{ textAlign: 'center' }}>
      <form onSubmit={this.handleSubmit}>
        <TextField
          name="body"
          type="text"
          label="Commentar neste post:"
          error={errors.comment ? true : false}
          helperText={errors.comment}
          value={this.state.body}
          onChange={this.handleChange}
          fullWidth
          className={classes.textField}
        />
        <Button
          type="submit"
          variant="contained"
          color="primary"
          className={classes.button}
        >
          Submeter        You, 5 days ago • Finishing notifications profile and general
        </Button>
      </form>
      <hr className={classes.visibleSeparator} />
    </Grid>
  ) : null;
  return commentFormMarkup;
}
```

### 4.4.5 Botão de Like

O MyButton foi um botão que nós fizemos com o objetivo de não estar sempre a repetir o código

```
import React from 'react';

import Tooltip from '@material-ui/core/Tooltip';
import IconButton from '@material-ui/core/IconButton';
//para poder utilizar em varias paginas sem estar a repetir codigo
                //children = o q esta dentro // tip e o titulo pro tooltip
export default ({ children, onClick, tip, btnClassName, tipClassName }) => (
  <Tooltip title={tip} className={tipClassName} placement="top">
    <IconButton onClick={onClick} className={btnClassName}>
      {children}        You, 7 days ago • My button + Atualização da NavBar quando login
    </IconButton>
  </Tooltip>
);
```

```
};
likeScream = () => {
  this.props.likeScream(this.props.screamId);
};
unlikeScream = () => {
  this.props.unlikeScream(this.props.screamId);
};
render() {
  const { authenticated } = this.props.user;
  const likeButton = !authenticated ? (
    <Link to="/login">
      <MyButton tip="Like">
        <FavoriteBorder color="primary" />
      </MyButton>
    </Link>
  ) : this.likedScream() ? (
    <MyButton tip="Undo like" onClick={this.unlikeScream}>
      <FavoriteIcon color="primary" />
    </MyButton>
  ) : (
    <MyButton tip="Like" onClick={this.likeScream}>
      <FavoriteBorder color="primary" />
    </MyButton>
  );
  return likeButton;
}
```

### 4.4.6 Perfil

Para o perfil caso estejamos autenticados mostramos a imagem do utilizador o handler que linka ao perfil do utilizador e a sua informação com opções para editar a imagem fazer logout ou editar a sua informação (ver component edit details)

```
let profileMarkup = !loading ? (authenticated ? (
  <Paper className={classes.paper}>
    <div className={classes.profile}>
        <div className={"image-wrapper"}>
            <img src={imageUrl} alt="profile" className="profile-image"/>
            <input type="file" id="imageInput" onChange={this.handleImageChange} hidden="hidden"/>
          <Tooltip title="Editar foto de perfil" placement="top">
          <IconButton onClick={this.handleEditImage} className="button">
              <EditIcon color = "primary" />
            </IconButton>
          </Tooltip>
        </div>
        <hr/>
        <div className={"profile-details"}>
    <MuiLink component={Link} to={`/users/${handle}`} color="primary" variant ="h5">
      @{handle}
    </MuiLink>
    <hr/>
    {bio && <Typography variant="body2">{bio}</Typography>}
    <hr/>
    {location && (
      <Fragment>
    <LocationOn color="primary"/> <span>{location}</span>
      <hr/>
      </Fragment>
    )}
    {website && (
      <Fragment>
        <LinkIcon color ="primary"/>
        <a href={website} target="_blank" rel="noopener noreferrer">
          {' '}{website}
        </a>
        <hr/>
      </Fragment>       You, 8 days ago • Initial commit
    )}
    <CalendarToday color="primary"/>{'  '}
    <span>Juntou-se em {dayjs(createAt).format('MMM YYYY')}</span>
        </div>
        <Tooltip title="Logout" placement="top">
          <IconButton onClick={this.handleLogout}>
            <KeyboardReturn color="primary"/>
          </IconButton>
        </Tooltip>

        <EditDetails/>
    </div>
  </Paper>
```

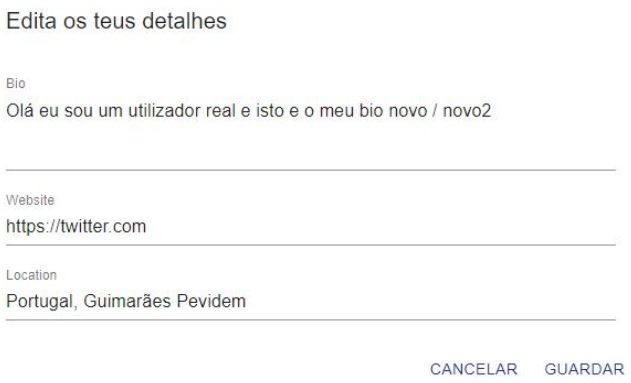
```
      </Paper>
    ) : ( //nao auntenticado
      <Paper className={classes.paper}>
          <Typography variant="body2" align="center">
            Profile não encontrado, por favor faça login de novo
          </Typography>
          <div className={classes.buttons}>
            <Button variant="contained" color="primary" component={Link} to="/login">
              Login
            </Button>
            <Button variant="contained" color="secondary" component={Link} to="/signup">
              Signup
            </Button>
          </div>
      </Paper>
```

**Resultado:**

Profile não encontrado, por favor faça login de novo
LOGIN
SIGNUP

## 4.4.7 Edit Details

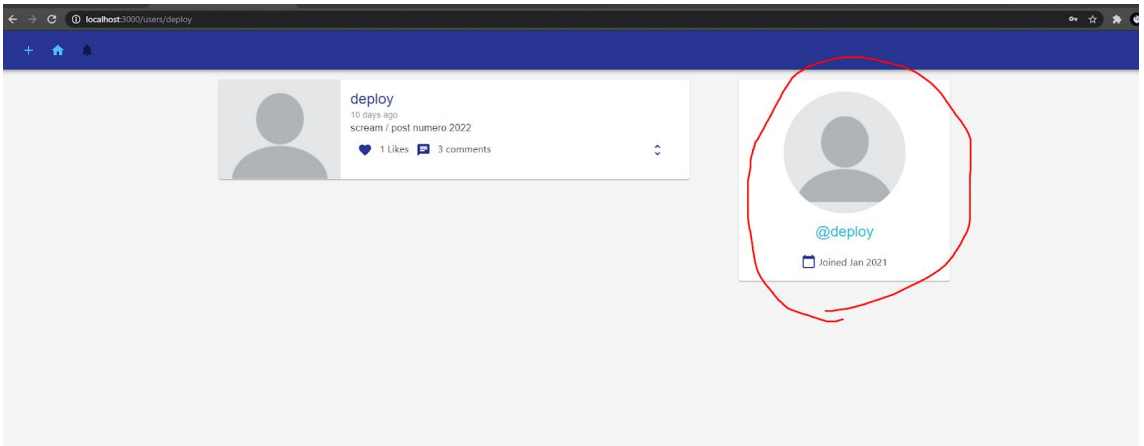
```
>
    <DialogTitle>Edita os teus detalhes</DialogTitle>
    <DialogContent>
      <form>
        <TextField
          name="bio"
          tpye="text"
          label="Bio"
          multiline
          rows="3"
          placeholder="Bio"
          className={classes.textField}
          value={this.state.bio}
          onChange={this.handleChange}
          fullWidth
        />
        <TextField
          name="website"
          tpye="text"
          label="Website"
          placeholder="Website"
          className={classes.textField}
          value={this.state.website}
          onChange={this.handleChange}
          fullWidth
        />
        <TextField
          name="location"
          tpye="text"
          label="Location"
          placeholder="A tua localização"
          className={classes.textField}
          value={this.state.location}
          onChange={this.handleChange}
          fullWidth
        />
      </form>
    </DialogContent>
    <DialogActions>
      <Button onClick={this.handleClose} color="primary">
        Cancel
      </Button>
      <Button onClick={this.handleSubmit} color="primary">
        Save
      </Button>
    </DialogActions>
  </Dialog>
</Fragment>
```



### 4.4.8 Perfil estático

Ao clicarmos em um utilizador some enviados para a sua página de perfil que nos mostra o seu profile e a sua post history este componente cobre a parte do perfil

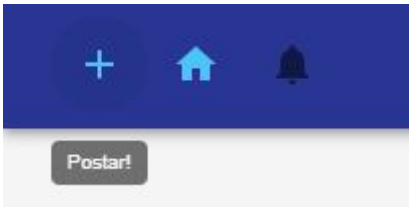
```
const ProfileEstatico = (props) => { //component
  const {
    classes,
    profile: { handle, createdAt, imageUrl, bio, website, location }
  } = props;

  return (
    <Paper className={classes.paper}>
      <div className={classes.profile}>
        <div className="image-wrapper">
          <img src={imageUrl} alt="profile" className="profile-image" />
        </div>
        <hr />
        <div className="profile-details">
          <MuiLink
            component={Link}
            to={`/users/${handle}`}
            color="primary"
            variant="h5"
          >
            @{handle}
          </MuiLink>
          <hr />
          {bio && <Typography variant="body2">{bio}</Typography>}
          <hr />
          {location && (
            <Fragment>
              <LocationOn color="primary" /> <span>{location}</span>
              <hr />
            </Fragment>
          )}
          {website && (
            <Fragment>
              <LinkIcon color="primary" />
              <a href={website} target="_blank" rel="noopener noreferrer">
                {' '}
                {website}
              </a>
              <hr />
            </Fragment>
          )}
          <CalendarToday color="primary" />{' '}
          <span>Joined {dayjs(createdAt).format('MMM YYYY')}</span>
        </div>
      </div>
    </Paper>
  );
};
```



### 4.4.9 Barra de Navegação

Na nossa barra de navegação mostramos home login e signup para utilizadores não autenticados e botão de home, post e notificações para utilizadores autenticados

```
You, 5 days ago | 1 author (You)
export class NavBar extends Component {
    render() {
        const { authenticated } = this.props
        return (
            <AppBar>
                <Toolbar>
                    {authenticated ? (

                        <Fragment>
                            <PostScream />

                        <Link to="/">
                        <MyButton tip = "Home" >
                                <HomeIcon color = "secondary"/>
                                </MyButton>
                            </Link>

                                <Notifications/>

                        </Fragment>


                    ) : (
                        <Fragment>
                    <Button color='inherit' component={Link} to="/login">Login</Button>
                    <Button color='inherit' component={Link} to="/">Home</Button>
                    <Button color='inherit' component={Link} to="/signup">Singup</Button>
                        </Fragment>


                    )}
                </Toolbar>
            </AppBar>
        )
    }       You, 8 days ago • Initial commit
}
```



## 4.4.10 Post Scream / Post Component

```
<MyButton onClick={this.handleOpen} tip="Postar!">
  <AddIcon color='secondary'/>
</MyButton>
<Dialog
  open={this.state.open}
  onClose={this.handleClose}
  fullWidth
  maxWidth="sm"
>
  <MyButton
    tip="Fechar"
    onClick={this.handleClose}
    tipClassName={classes.closeButton}
  >
    <CloseIcon />
  </MyButton>
  <DialogTitle>Postar</DialogTitle>
  <DialogContent>
    <form onSubmit={this.handleSubmit}>
      <TextField
        name="body"
        type="text"
        label="SCREAM!!"
        multiline
        rows="3"
        placeholder="TEXTO PLACEHOLDER"
        error={errors.body ? true : false}
        helperText={errors.body}
        className={classes.textField}
        onChange={this.handleChange}
        fullWidth
      />
      <Button
        type="submit"
        variant="contained"
        color="#fff"
        className={classes.submitButton}
        disabled={loading}
      >
        Submeter
        {loading && (
          <CircularProgress
            size={30}
            className={classes.progressSpinner}
          />
        )}
      </Button>
    </form>
  </DialogContent>
```

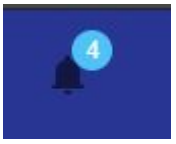
### 4.4.11 Componente Notificações

Vamos buscar as notificações não lidas e passamos para o utilizador

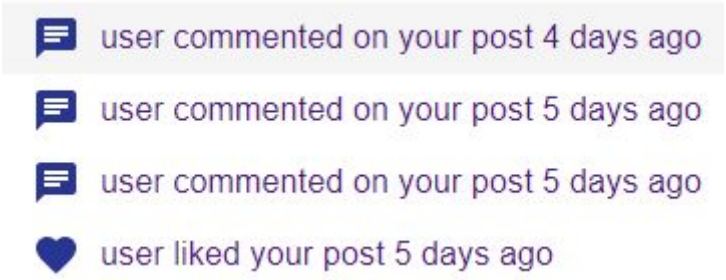
```
class Notifications extends Component {
  state = {
    anchorEl: null
  };
  handleOpen = (event) => {
    this.setState({ anchorEl: event.target });
  };
  handleClose = () => {
    this.setState({ anchorEl: null });
  };
  onMenuOpened = () => {
    let unreadNotificationsIds = this.props.notifications
      .filter((not) => !not.read)
      .map((not) => not.notificationId);
    this.props.markNotificationsRead(unreadNotificationsIds);
  };
  render() {
    const notifications = this.props.notifications;
    const anchorEl = this.state.anchorEl;        You, 5 days ago • Finishing notific

    dayjs.extend(relativeTime);
```

Caso o length seja mais que 0 mostramos e existam não lidas mostramos o número de notificações não lidas no ícone.

```
    dayjs.extend(relativeTime);

    let notificationsIcon;
    if (notifications && notifications.length > 0) {
      notif let notificationsIcon: any .read === false).length > 0
        ? (notificationsIcon = (
            <Badge
              badgeContent={
                notifications.filter((not) => not.read === false).length
              }
              color="secondary"
            >
              <NotificationsIcon />
            </Badge>
          ))
        : (notificationsIcon = <NotificationsIcon />);
    } else {
      notificationsIcon = <NotificationsIcon />;
    }
```

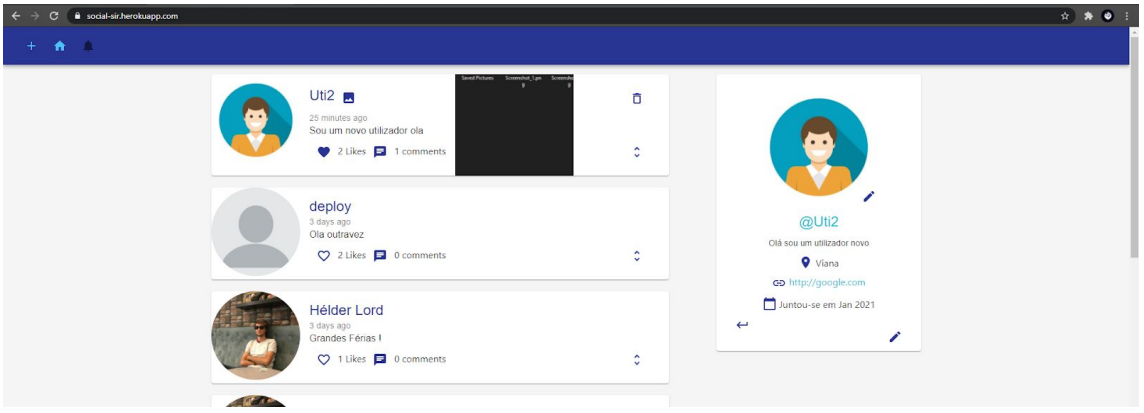


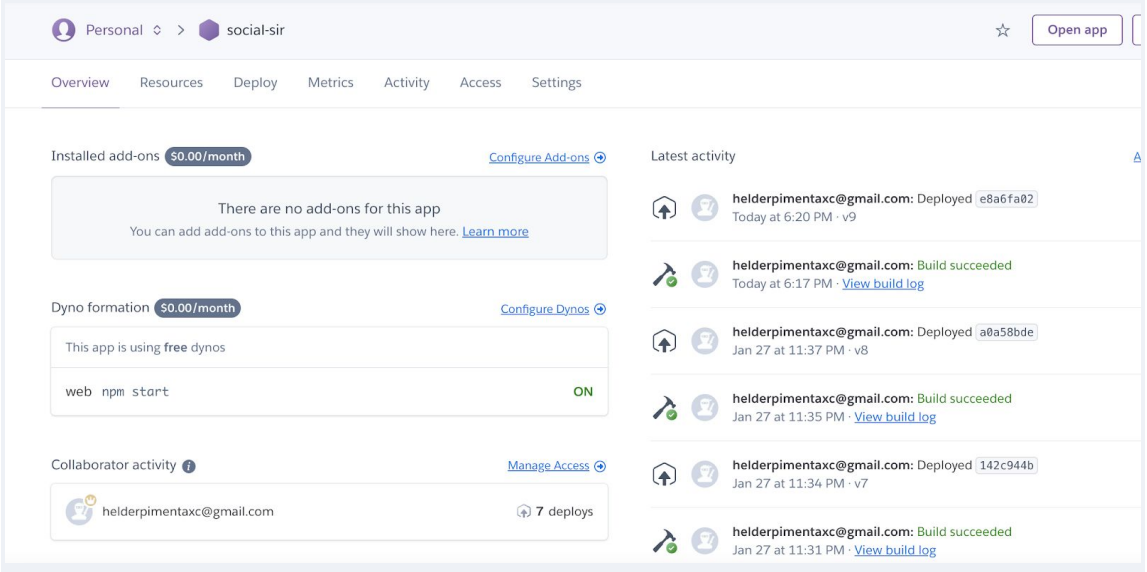
```
            );

            return (
              <MenuItem key={not.createdAt} onClick={this.handleClose}>
                {icon}
                <Typography
                  component={Link}
                  color="default"
                  variant="body1"
                  to={`/users/${not.recipient}/scream/${not.screamId}`}
                >
                  {not.sender} {verb} O teu post {time}
                </Typography>
              </MenuItem>
            );
          })
        ) : (
          <MenuItem onClick={this.handleClose}>
            Não tens notificações
          </MenuItem>
        );
      return (
        <Fragment>
          <Tooltip placement="top" title="Notifications">
            <IconButton
              aria-owns={anchorEl ? 'simple-menu' : undefined}
              aria-haspopup="true"
              onClick={this.handleOpen}
            >
              {notificationsIcon}
            </IconButton>
          </Tooltip>
          <Menu
            anchorEl={anchorEl}
            open={Boolean(anchorEl)}
            onClose={this.handleClose}
            onEntered={this.onMenuOpened}
          >
            {notificationsMarkup}
          </Menu>
        </Fragment>
      );
```



## 4.5 Heroku Deploy

Utilizamos o heroku para fazer o deploy do nosso website, o firebase também disponibiliza serviço de hosting mas este é pago.




```
npm install -g create-react-app
create-react-app social-sir
cd social-sir
git init
heroku create -b
https://github.com/helderpgoncalves/social-sir-buildpack.git
git add .
git commit -m "social-sir on Heroku"
git push heroku master
heroku open
```

https://social-sir.herokuapp.com/

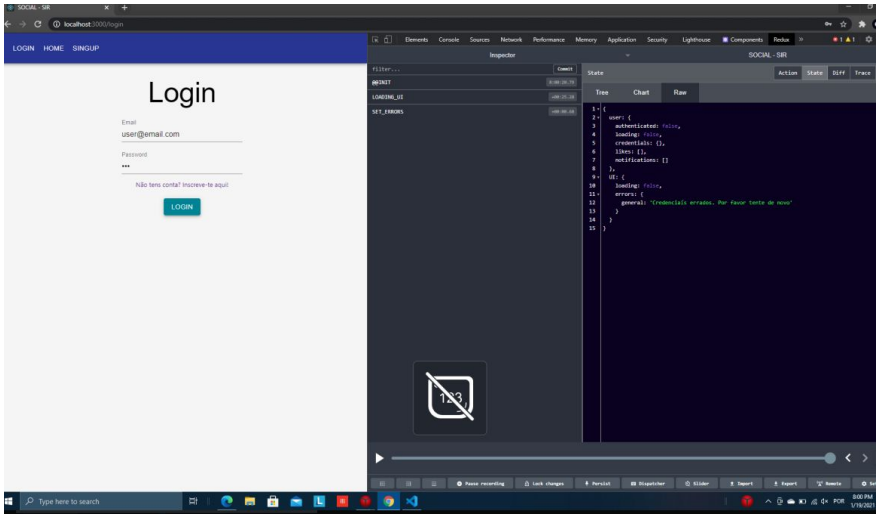
# 5. Dificuldades e Future Work

## *5.1 Dificuldades*

Tivemos alguns problemas na implementação das notificações, especialmente no backend uma vez que se o firebase estava sempre a dar erro quando fazíamos deploy:

```
10:06:37.053 PM   createNotificationOnLike
                  Function execution started
10:06:37.164 PM   createNotificationOnLike
                  TypeError: db.doc is not a function
10:06:37.164 PM   createNotificationOnLike
                  at /workspace/index.js:42:8
10:06:37.164 PM   createNotificationOnLike
                  at cloudFunction (/workspace/node_modules/firebase-functions/lib/cloud-functions.js:134:23)
10:06:37.164 PM   createNotificationOnLike
                  at /layers/google.nodejs.functions-framework/functions-framework/node_modules/@google-cloud/functions-framework/build/src/invoker.js:199:28
10:06:37.164 PM   createNotificationOnLike
                  at processTicksAndRejections (internal/process/task_queues.js:93:5)
10:06:37.262 PM   createNotificationOnLike
                  Function execution took 210 ms, finished with status: 'error'
```

***Alguns dos erros não aparecem no ecrã apesar de aparecerem na consola :***




***Deploy do backend:***

```
i  functions: updating Node.js 14 (Beta) function api(us-central1)...
i  functions: updating Node.js 14 (Beta) function createNotificationOnLike(europe-west1)...
i  functions: updating Node.js 14 (Beta) function deleteNotificationOnUnLike(europe-west1)...
i  functions: updating Node.js 14 (Beta) function createNotificationOnComment(europe-west1)...
i  functions: updating Node.js 14 (Beta) function onUserImageChange(europe-west1)...
i  functions: updating Node.js 14 (Beta) function onScreamDelete(europe-west1)...
!  functions[onScreamDelete(europe-west1)]: Deployment error.
Function deployment failed because the billing account is not available.
!  functions[createNotificationOnLike(europe-west1)]: Deployment error.
Function deployment failed because the billing account is not available.
!  functions[onUserImageChange(europe-west1)]: Deployment error.
Build failed: could not resolve storage source: googleapi: Error 403: The project to be billed is associated with an absent billing account., accountDisabled
!  functions[deleteNotificationOnUnLike(europe-west1)]: Deployment error.
Function deployment failed because the billing account is not available.
```

Erros no deploy do firebase. Nunca chegamos ao ponto de descobrir a razão destes erros em 1 dia o deploy corre perfeitamente no outro não. (Diz erro de billing mas se tentarmos mais 1 ou 2 vezes dá sem erro)

## *5.2 Future Work*

*1- Melhorar o design - tornar ainda mais responsivo*

*2- Signin com google & Facebook*

*3- Aceitar ficheiros .gif*

*4- AWS Deploy*

*5- Mais código limpo, refactor code*

# 6. Conclusão

Conseguimos alcançar todos os nossos objetivos tendo um website funcional que cumpre os requisitos do enunciado.

A nossa api foi desenvolvida utilizando algumas livrarias do firebase, o nosso website usando a api for desenvolvido usando o react e o utilizador tem acesso a uma área para ele fazer as mudanças do seu perfil e ver a sua atividade , mudar a foto , fazer posts comentários , etc.

No final demos deploy no Heroku.

# 7. Bibliografia

https://firebase.google.com/docs/web/setup?hl=pt-br

https://firebase.google.com/docs/firestore?hl=pt-br

https://nodejs.org/docs/latest-v13.x/api/

https://firebase.google.com/docs/auth

Day.js · 2kB JavaScript date utility library

https://www.npmjs.com/package/busboy

https://www.youtube.com/watch?v=8-RYuQIvwN8

https://reactjs.org/docs/getting-started.html

react-router-dom - npm (npmjs.com)

https://www.youtube.com/playlist?list=PLMhAeHCz8S38ryyeMiBPPUnFAiWnoPvWP

https://pastebin.com/f33g85pd

https://reactjs.org/docs/typechecking-with-proptypes.html

https://material-ui.com/pt/

https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers

https://stackoverflow.com/questions/55785316/how-would-i-see-updated-props-without-refreshing

https://react-redux.js.org/introduction/why-use-react-redux