

# FINAL PROJECT DAW 2023/2024

## Trabalho realizado por:

Alexandre Carvalho a75727

Miguel Santos a74863

Pedro Colaço a75538

Tiago Bota a75487

## Criação da aplicação:

Esta aplicação desenvolvida, tem o principal intuito de enviar emails seja para contactos ou para emails que não estejam guardados.

Começamos por criar um login que permite o utilizador entrar na sua conta de email, esse utilizador pode criar a sua lista de contactos onde vai guardar o email e o nome da respetiva pessoa (cada utilizador logado possui a sua própria lista de contactos).

Adicionamos uma nova funcionalidade que permite o utilizador mandar um email para mais do que uma pessoa ao mesmo tempo e por fim para sair da sua conta basta clicar no botão "logout".

# Funcionalidades e requisitos nos lados:

## Cliente:

### 1ª Funcionalidade:

Na resolução deste projeto achamos importante adicionar duas novas funcionalidades à API.

A primeira foi a possibilidade de enviar a mesma mensagem, para vários destinatários sem ser necessários copiar a mesma mensagem e enviá-la manualmente para cada um. Verificámos inicialmente que ao inserirmos mais do que um endereço de email no destinatário da mensagem, a mesma era apenas enviada para o primeiro endereço. E também verificámos que na variável do destinatário era guardada uma string, tal como a inseríamos na opção destinatário, portanto tivemos de fazer uma pequena mudança.

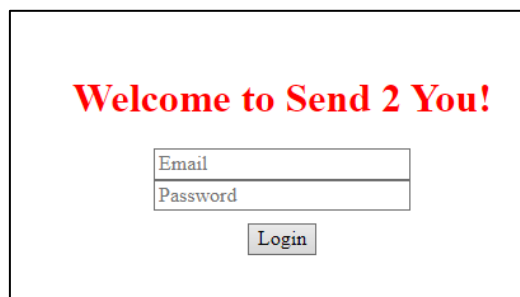
```
*/  
public async sendMessage(inTo: string, inFrom: string, inSubject: string, inMessage: string): Promise<void> {  
    console.log("SMTP.Worker.sendMessage()");  
    console.log(inTo);  
    const toAddresses: string[] = inTo.split(/\s+/);  
  
    for (const toAddress of toAddresses) {  
        console.log(toAddress);  
        await axios.post(`${config.serverAddress}/messages`, {  
            to: toAddress,  
            from: inFrom,  
            subject: inSubject,  
            text: inMessage,  
        });  
    }  
}  
/* End sendMessage(). */
```

Adicionámos a constante “toAddresses” que será um array de string correspondente à string do nosso destinatário, separado por espaços. Por fim temos um *for* que vai executar o processo de enviar a mensagem, por todos os endereços.

### 2ª Funcionalidade:

#### 2ª parte

Com base na informação da primeira parte e com o objetivo também referido na mesma parte criámos um login na primeira página (onde utilizador iria meter os seu email/password), onde estas variáveis serão enviadas do cliente para o servidor, através do middleware sendo então substituídas automaticamente no código (lab6/server/ServerInfo.json).



Welcome to Send 2 You!

Email

Password

Login

Não chegámos a implementar nenhuma forma de validar as credencias do utilizador então é possível entrar no site sem que as credenciais estejam corretas, porém para enviar um email é obrigatoriamente necessário que as credenciais estejam validas.

### 3ª parte

Uma vez logado queremos ver os contactos a nós associados e a forma que utilizámos para mostrar os contactos na base de dados pelo email associado foi a seguinte:

```
doLogin: async function (): Promise<void> {
  console.log(
    "state.doLogin()",
    this.state.userEmail,
    this.state.userPass
  )

  this.state.messageFrom = this.state.userEmail;

  this.state.showHidePleaseWait(true);
  const loginWorker: Logins.Worker = new Logins.Worker();
  await loginWorker.doLogin(
    this.state.userEmail,
    this.state.userPass
  );

  this.state.showHidePleaseWait(false);
  this.setState({ currentView: "compose" });
  this.state.loadContacts();
}.bind(inParentComponent)
```

No final do login, chamamos o método responsável por carregar os contactos "this.state.loadContacts()".

```
loadContacts: async function(): Promise<void>{
  const contactsWorker: Contacts.Worker = new Contacts.Worker();
  const contacts: Contacts.IContact[] = await contactsWorker.listContacts(this.state.userEmail);
  this.state.setContactList(contacts);
  console.log(this.state);
}.bind(inParentComponent),
```

Esta função vai criar um array de IContact que no final vai ser atribuído a nossa variável contacts (o array que guarda a informação da lista de contactos). Mas para vermos o que está a ser atribuído ao array temos de ver a função "listContacts".

```
public async listContacts(owner: string): Promise<IContact[]> {

  console.log("Contacts.Worker.listContacts()");

  const response: AxiosResponse = await axios.get(`${config.serverAddress}/contacts?owner=${owner}`);
  console.log(response.data);
  return response.data;
} /* End listContacts(). */
```

Esta função faz um request ao servidor, no entanto agora vai dar também o atributo owner, que é dado como atributo da função, no caso é a nossa variável "this.state.userEmail".

## Servidor:

### 2ª Funcionalidade:

#### 1ª parte

Já a segunda funcionalidade decidimos criar o sistema de login, isto serve para que mais do que uma pessoa possa usar a API no mesmo dispositivo, sem precisar de alterar no código os dados do utilizador (a variável “user” e “pass”) na pasta serverinfo no ficheiro serverInfo.json da parte do servidor.

```
1  {
2    "smtp": {
3      "host": "smtp.office365.com",
4      "port": 587,
5      "auth": {
6        "user": "a12345@ualg.pt",
7        "pass": "123"
8      }
9    }
10 }
```

Para além disso também não faz muito sentido a lista de contactos ser a mesma para todos os utilizadores, portanto fizemos uma alteração na base de dados para conseguirmos fazer a filtragem dos contactos por utilizador.

Como se pode ver nesta imagem

```
{ "name": "O Tal", "email": "4141", "owner": "a75727@ualg.pt", "_id": "1Ns2ipnBB49KycuU" }
{ "name": "MIGUEL ", "email": "123", "owner": "pedro@gmail.com", "_id": "3wRccTR87Rfeq7qq" }
{ "name": "Tiago Bota", "email": "123", "owner": "pedro@gamil.com", "_id": "ENE9S1Jac90VgaiH" }
{ "name": "Pedro Colaço", "email": "a75538@ualg.pt", "owner": "a75727@ualg.pt", "_id": "NQhagyooiI8eeHIE" }
```

Acrescentámos o atributo “owner”, que vai guardar o email do utilizador da API, portanto para este exemplo o contacto [a75727@ualg.pt](mailto:a75727@ualg.pt), tem dois contactos na sua lista “O Tal” e o “Pedro Colaço”, já o contacto [pedro@gmail.com](mailto:pedro@gmail.com) tem dois contactos “Miguel ” e “Tiago Bota”.

#### 4ª parte

Já em relação ao servidor na main, tivemos de fazer uma pequena alteração.

```
app.get("/contacts", async (inRequest: Request, inResponse: Response) => {
  try {
    const contactsWorker: Contacts.Worker = new Contacts.Worker();
    const contacts: IContact[] = await contactsWorker.listContacts(inRequest.query.owner as string); inResponse.json(contacts);
  } catch (inError) {
    inResponse.send("error");
  }
});
```

Onde é dado o owner como argumento.

```
public listContacts(owner: string): Promise<IContact[]> {  
  return new Promise((inResolve, inReject) => {  
    this.db.find({ "owner": owner }, (inError: Error | null, inDocs: IContact[]) => {  
      if (inError) {  
        inReject(inError);  
      } else {  
        inResolve(inDocs);  
      }  
    });  
  });  
}
```

Por fim precisamos de especificar no “this.db.find” que queremos apenas os contactos, onde o nosso o atributo “owner” é igual ao email que está a usar a API, caso este não tenha nenhum contacto adicionado, não será retornado nada.

```
export interface IContact { _id?: number, name: string, email: string, owner: string }
```

Para introduzirmos contactos na nossa base de dados, tivemos apenas de alterar o IContact, para receber mais um atributo “owner”.

## Layout:

O layout da nossa aplicação é composto por várias páginas em que todas elas são compostas por componentes. Cada componente tem uma função específica para apresentar visualmente as funcionalidades do código como por exemplo:

- A componente `ToolBar` representa a barra de navegação entre páginas através de botões;
- A componente `MessageView` representa o formulário para a inserção dos dados e para o envio desses dados como email e os botões respetivos para o envio do email;
- A componente `ContactView` representa o formulário para a criação (ou edição ou eliminação) de um contacto e os botões para essa mesma criação, eliminação ou edição;
- A componente `ContactList` representa a barra lateral onde são apresentados os contactos já presentes e os contactos adicionados pelo email logado;
- A componente `BaseLayout` representa a componente que une os outros componentes numa página de forma a ficarem organizados de acordo com o design escolhido;

## Project set up:

Para o desenvolvimento tivemos como bases o lab6 e o lab8 elaborados durante as aulas, logo todos os módulos utilizados serão os mesmos que os dos respetivos labs. Tendo como base essa informação para iniciar a nossa aplicação serão precisos os seguintes comandos:  
Para a parte servidora (Pasta Servidor):

- `npm run compile`

Para a parte cliente (Pasta Cliente):

- `npx webpack`

Para a visualização da aplicação:

- escrever `127.0.0.1:8080` no browser