



Universidade do Minho
Escola de Engenharia

DIAuth - Serviço de Autenticação Distribuído

André Filipe Ferreira de Mira Vieira A78322

Eduardo Gil Ribeiro Rocha A77048

Ricardo André Araújo Neves A78764

Junho 2019

Índice

1	Introdução	3
2	Motivações	3
3	Requisitos	4
4	Solução Desenhada	4
5	Diagrama de Gantt	6
6	Primeira Fase	6
7	Segunda Fase	7
8	Terceira Fase	7
9	OAuth 2.0	8
9.1	OAuth Flow	9
9.2	Grant Types	9
9.3	Esquema de Fluxo	10
10	Guia de Utilização	11
11	Exemplos de Páginas da Aplicação	13
12	Conclusão	14

1 Introdução

Este trabalho envolve a produção de um serviço de autenticação distribuído, no âmbito da Unidade Curricular de Laboratórios de Engenharia Informática. O projeto surge de um desejo de que exista uma aplicação de autenticação que as aplicações feitas no Departamento de Informática possam usar. O relatório irá explicar as motivações por trás do projeto, e as medidas tomadas durante o desenvolvimento do mesmo.

O serviço de autenticação criado deve fornecer processos de registo e *login* aos seus utilizadores, deixando depois que clientes autorizados acessem a parte da informação armazenada, para uso próprio. Para além dos serviços normais de registo e *login*, foi proposta também o uso de contas terceiras para acesso à aplicação; por exemplo, um utilizador poder associar informações de *login* Google, Facebook, GitHub ou Twitter à sua conta para poder usar essa conta terceira para acessos futuros.

2 Motivações

Existem dois motivos principais por trás deste desejo: o primeiro é o facto de cada nova aplicação precisar dos seus próprios métodos de autenticação e autorização, necessitando que os criadores destas gastem tempo adicional a desenvolver estas funcionalidades. O nosso servidor pode então realizar alguns destes processos por parte das aplicações.

O segundo motivo é facilitar o uso da aplicação por parte do cliente. Não só seria preciso apenas decorar um par *username/password* para aceder a todas as aplicações do Departamento de Informática, mas também facilitaria o uso dos mesmo, sendo que não seria preciso fazer repetidamente *login*. Após a entrada em um *site*, o nosso sistema irá permitir a entrada do mesmo a partir de uma outra aplicação, não sendo necessário repetir o processo de *login*.

3 Requisitos

Os serviços oferecidos pelo servidor podem ser divididos em três partes: deve haver forma de um utilizador criar uma conta e um processo para entrar na mesma; devemos também fornecer métodos de *login* alternativos, usando contas terceiras como atalhos; finalmente, devemos usar o protocolo *OAuth 2.0* para permitir a outras aplicações reencaminhar os seus utilizadores para os nossos processos de autenticação.

Com isto dito, concluímos que os requisitos para o nosso servidor são:

- Servidor a correr sobre *HTTPS*
- Registo de novas contas
- *Login* para contas existentes
- *Login* por contas terceiras (*Google*, *Facebook*, entre outros.)
- Base de dados para guardar informação sobre utilizadores e clientes
- Página de visualização de informação dos utilizadores
- Página de registo de clientes por parte dos utilizadores
- Página de visualização de informação dos clientes
- Implementação de protocolo *OAuth 2.0* para autorização
- Página de diálogo para utilizadores darem autorização a clientes
- Processo de criação e armazenamento de códigos de acesso
- Processo de criação e armazenamento de *tokens*
- *API* com acesso protegido por *tokens*

4 Solução Desenhada

Para assegurar todos os requisitos definidos, decidimos usar um servidor construído em *Node.JS*, usando a *framework* para aplicações *Express*. Esta *framework* foi escolhida pois deixa-nos resolver os problemas enfrentados enquanto que ao mesmo tempo é uma ferramenta familiar a dois dos elementos do grupo, minimizando tempo perdido a aprender como trabalhar com a ferramenta. Este servidor ligar-se-á a uma base de dados *mongoDB*, que guardará toda a informação necessária.

O nosso servidor oferece também serviços de associação de contas terceiras às nossas contas, o que permite usar contas de plataformas como a *Google* para fazer *login* no nosso serviço. Esta associação serve então apenas como um atalho de *login* para uma conta pré-existente, deixando um utilizador realizar o seu *login*

Por último, queremos usar o protocolo *OAuth 2.0* para deixar clientes nossos usarem a nossa *API* e para poderem autenticarem os seus utilizadores.

O seguinte esquema traduz de uma forma simples, o que foi abordado até aqui:

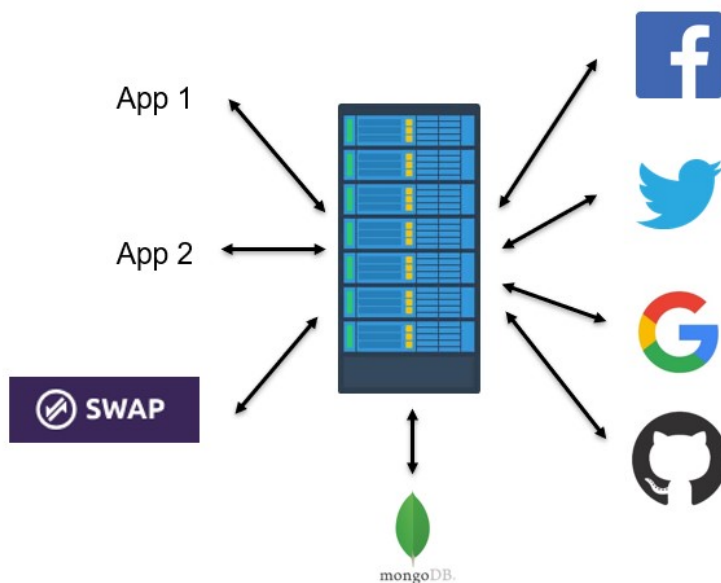


Figure 1: Esquema do projeto

Como podemos observar, do lado esquerdo temos aplicações que se podem vincular ao nosso serviço. A plataforma *SWAP*, por exemplo, da Universidade do Minho é uma potencial aplicação que pode usufruir das vantagens da nossa implementação.

Do outro lado, temos as 4 das maiores redes sociais, utilizadas quase diariamente por estudantes, e não só: o *Facebook*, o *Twitter*, o *Google* e o *GitHub*. Escolhemos estes quatro serviços porque qualquer possível utilizador da nossa aplicação terá, pelo menos, uma conta numa destas aplicações. Para além deste facto, todas estas aplicações usam também o protocolo *OAuth* para autorização. Iremos explorar o protocolo *OAuth 2.0* mais à frente neste relatório.

Na parte inferior da imagem, está presente o *mongoDB*, a base de dados usada.

5 Diagrama de Gantt

Um Diagrama de *Gantt* é um gráfico utilizado para ilustrar os intervalos de tempo relativos ao início e finalização das diferentes etapas de um projeto. Esta informação é organizada numa linha de tempo, onde podemos verificar, por exemplo, a data de entrega ou dependências entre as etapas.

Na fase inicial do trabalho, grupo debateu e desenvolveu o seguinte diagrama, de forma a funcionar como uma ferramenta de controlo para o trabalho realizado:

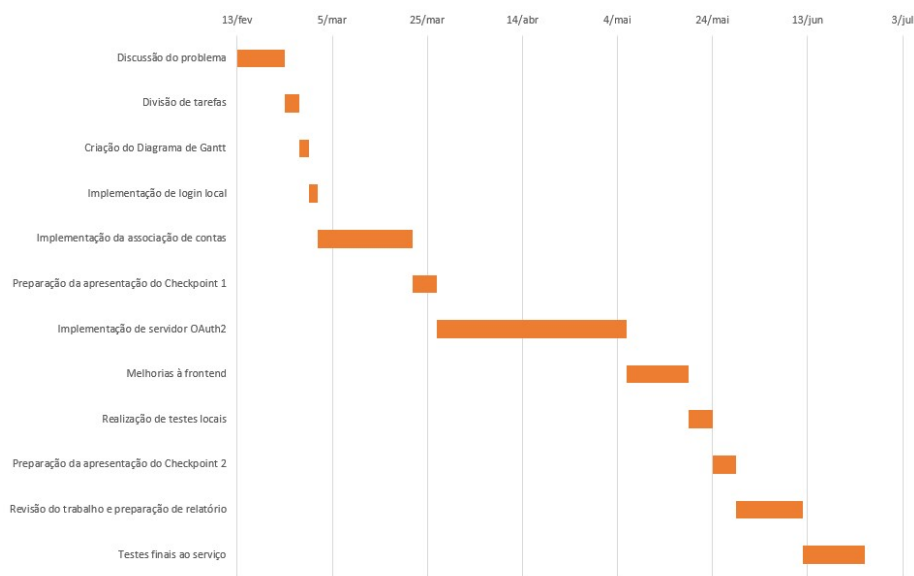


Figure 2: Diagrama de Gantt

De seguida, iremos entrar em mais detalhe sobre as diferentes etapas do nosso projeto, e os passos tomados para as concluir.

6 Primeira Fase

Durante a primeira fase do trabalho, o grupo decidiu planejar exatamente quais as tecnologias que iriam ser usadas para a realização do trabalho e como iríamos dividir as tarefas entre os elementos do grupo. Após isto, dedicamos o resto da fase à implementação de processos de registo e *login* locais, assim como a associação de contas terceiras para usar como atalhos ao *login*. Durante esta fase foi planeado também o tipo e a estrutura da base de dados que iríamos usar.

Como referido anteriormente, foram usadas quatro aplicações para associação às nossas contas: *Google*, *Facebook*, *Twitter* e *GitHub*. Este processo

necessitou primeiro um registo como clientes nas respetivas páginas para *developers* destas aplicações. Após isto, importamos os dados necessários para o nosso servidor, como o nosso *ID* de cliente. Adicionamos *links*, na página de informação do utilizador, para adicionar estes atalhos a uma conta. Em conclusão, os objetivos definidos para esta fase, foram, na maioria, atingidos. Todos os tipos de atalhos inicialmente planeados foram adicionados até ao fim da primeira fase, porém, haviam problemas de segurança com a implementação nesta altura, que foram resolvidos na fase seguinte.

7 Segunda Fase

A segunda fase focou-se quase exclusivamente na implementação do protocolo *OAuth*. Após um novo período de pesquisa, em que o grupo procurou confirmar se este protocolo era a escolha correta, procuramos encontrar a melhor forma de o implementar no nosso projeto. Como referido no capítulo anterior, usamos a segunda fase também para refinar os processos de associação de contas terceiras e de *login* com as mesmas.

Descobrimos o *middleware* conhecido por *OAuth2orize*, criado para uso com a linguagem *Node.JS*, que fornece uma *framework* de autorização, que, combinado com outras tecnologias como o *Passport*, um *middleware* de autenticação, pode ser usado para construir um servidor que implemente o protocolo *OAuth 2.0*.

O objetivo principal para esta fase era ter um servidor com um serviço de autorização funcional, um objetivo que acabamos por conseguir cumprir. Nesta fase, implementamos um processo de tipo de resposta *Authorization Code*, que será explicado em maior detalhe numa secção a seguir.

No fim desta fase, o nosso servidor era então capaz de responder a pedidos de autorização com uma página de diálogo que pedia ao utilizador se permite dar acesso à sua conta ao cliente. Caso aceite, é reencaminhado com um código de autorização, que o cliente pode trocar por um *token* de acesso, que lhe permite aceder à *API* do servidor, para obter a informação que deseja.

8 Terceira Fase

Para a última fase, definimos os objetivos melhorar vários detalhes da aplicação e começar a testar o nosso serviço de autorização usando outras aplicações.

No que toca a melhorias da aplicação, fizemos mudanças a várias partes da *back-end* da aplicação. Mudamos também alguns aspetos da *front-end*. Adicionamos uma página para registar novos clientes e uma página para ver os clientes existentes, e a informação específica a cada um.

Modificamos também a nossa implementação do protocolo *OAuth*, adicionando suporte para o tipo *Implicit*. Este tipo foi adicionado para o servidor ser compatível com a plataforma *Odoo*, que requer um tipo de resposta específica para permitir autenticação através de uma conta terceira.

Na área de testes, começamos por criar um cliente usando o mesmo tipo de tecnologia usado no nosso servidor: uma aplicação criada com *Express*. Este cliente é muito simples, apresentando apenas um *link* que redireciona esse cliente para a página de diálogo do nosso servidor. Após o diálogo, o utilizador é redirecionado de volta para o cliente com um código de acesso, que o cliente imediatamente troca por um *token*. Este cliente usa o tipo de resposta *Authorization Code*.

Tentamos também realizar testes com a plataforma *Odoo*, que não foram finalizados por causa de problemas com os nossos certificados *SSL*. Apesar disto, o servidor implementa o tipo de resposta necessário para ser compatível com aplicações *Odoo*, ou seja, o tipo de resposta *Implicit*.

9 OAuth 2.0

A nossa escolha de usar este protocolo baseou-se no facto de ser a tecnologia mais prevacente nestas áreas de autorização e autenticação de clientes. Outro benefício desta escolha é haver já implementações de *OAuth* em *Express*, a *framework* sobre a qual foi construída o nosso servidor, que estudamos para entender como implementar a nossa solução. O *OAuth* fornece fluxos de autorização para aplicações *web*, *desktop* e para dispositivos móveis. A versão *OAuth 2.0* oferece várias melhorias relativas ao seu antecessor, o *OAuth 1.0*, em termos de segurança e em número de funcionalidades.

O *OAuth 2.0* é uma estrutura de autorização, lançada em Outubro de 2012, que permite que as aplicações obtenham acesso limitado às contas dos usuários em serviço HTTP, tendo como exemplos mais conhecidos o *Google* e o *Facebook*. Por acesso limitado, queremos dizer que a aplicação apenas tem acesso a informação pessoal, como o nome, o email ou a fotografia, se for necessário, e nunca a acesso privado, ou seja, a password do utilizador para aquele serviço. O servidor que implementa o protocolo *OAuth* é capaz de determinar os tipos de *scope* que fornece, ou seja, quais partes da informação do utilizador fornece. Podemos, por exemplo, definir um *scope* para dar acesso apenas ao *email* do utilizador e outro para dar acesso a toda a informação que temos.



Figure 3: Logótipo do OAuth 2.0

Logo, quando falamos em "Servidor", queremos dizer um servidor como o nosso, que tem um serviço de autenticação e oferece um processo de autorização de uso de contas. "Cliente" é uma aplicação que usa um Servidor para realizar autenticação. Finalmente, um "Utilizador" é o utilizador que faz uso da aplicação do Cliente e que tem uma conta registada no Servidor.

Podemos até usar uma analogia para especificar mais facilmente este protocolo: imaginemos que temos um carro de luxo, e com este, uma "*valet key*". Esta chave pode ser utilizada para um arrumador de carros poder estacionar um automóvel. No entanto, por se tratar de uma chave especial, este manobrista só pode dirigir no carro até uma distância máxima de, por exemplo, 2 quilómetros. Assim, tiramos daqui a ideia que, se queremos fornecer a alguém um acesso limitado, usamos a chave especial; se queremos um acesso total e sem limitações, utilizamos a chave normal.

9.1 OAuth Flow

No primeiro passo do processo de autenticação *OAuth*, um utilizador é reenaminhado para o servidor, que autentica o utilizador, caso não tenha já o *login* feito. Após isto, é-lhe mostrado uma página de diálogo, onde o utilizador pode ver alguma informação sobre o cliente, e sobre o tipo de informação a que o cliente quer ter acesso. É dada então ao utilizador a escolha entre deixar o cliente ter acesso às partes da sua conta definidas, ou recusar a autorização, o que irá terminar o processo de autorização imediatamente.

Caso o utilizador deixe o cliente usar a sua conta, será reenaminhado de volta para um *URL* predefinido, com um código de acesso ou *token*. Caso seja retornado um código, é preciso trocar este código por um *token*, que se faz num *endpoint* de troca, através de um pedido *HTTP* em método *POST*. Caso o código seja considerado válido, será enviada uma resposta de volta que irá conter o *token* de acesso. Com o *token* em mão, o cliente poderá aceder à *API* do servidor, e obter assim a informação do utilizador. Cada *token* tem registado o utilizador que permitiu o acesso ao servidor, logo não será possível invadir a privacidade de outros utilizadores e utilizar os dados deles sem permissão. Caso seja retornado um *token* imediatamente, não será preciso este processo de troca e poderá haver acesso à *API* imediatamente.

9.2 Grant Types

Existem vários tipos de acesso disponibilizados pelo *OAuth*, que decidem a forma como um cliente interage com um servidor. O nosso servidor, atualmente, usa dois *Grant Types* diferentes: *Authorization Code* e *Implicit*. Estes dois tipos distinguem-se pelo facto do primeiro tipo envolver o retorno de um código de autorização após o diálogo, enquanto que o segundo retorna um *token*. O tipo *Authorization Code* é o tipo mais comum, pois envolve um passo extra de segurança, requerindo um código de autorização válido para que um cliente tenha acesso a um *token*. Alguns servidores de *OAuth* proíbem até o uso do tipo *Implicit* por causa de falta de segurança. Apesar disto, implementamos este processo para o nosso servidor poder ser compatível com algumas tecnologias com as quais pretendemos realizar testes, como o *Odoo*.

A distinção entre os *Grant Types* é feita através de um parâmetro durante o processo de autorização, especificamente, o parâmetro "*Response Type*". Os parâmetros necessários durante o processo de autorização são os seguintes:

- *Response Type* - tipo de resposta esperada (código de autorização ou *token*)
- *Redirect URI* - endereço para qual será redirecionado o utilizador após o diálogo
- *Client ID* - ID do cliente registado no servidor
- *Scope* - gama de dados a que o cliente deseja ter acesso

9.3 Esquema de Fluxo

Iremos agora apresentar um diagrama que ilustra, de uma forma geral, as interações entre as entidades do *OAuth*, no tipo *Authorization Code*:

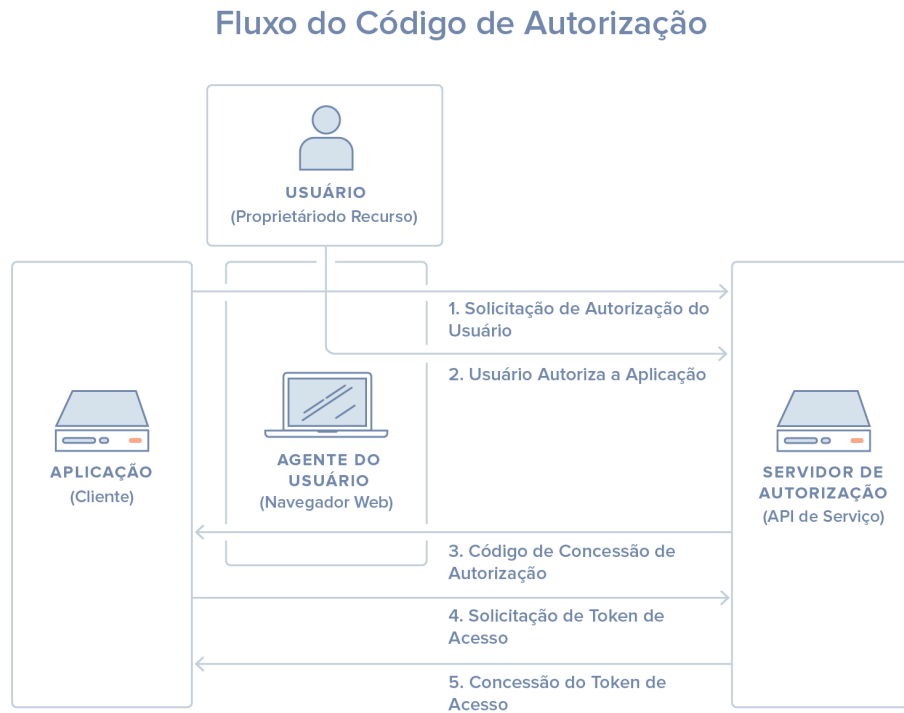


Figure 4: Esquema de Fluxo do OAuth

1. A aplicação solicita autorização para aceder aos recursos do utilizador, através de um *pop-up* ou redirecionamento.
2. O utilizador decide se autoriza o cliente a utilizar os seus dados.
3. O servidor envia o código de acesso ao cliente.

4. A aplicação solicita agora um *token* de acesso ao servidor de autorização. Para isto, terá de enviar a autorização dada pelo servidor.

5. O servidor de autenticação trata de validar a identidade da aplicação e a autorização recebida. Se tudo estiver correto, emite um *token* de acesso e passa-o para a aplicação. A autorização está agora completa.

6. Com o *token* de acesso em sua posse, a aplicação pode enviar esse *token* para o servidor, pedindo acesso às informações do utilizador a que tem acesso.

7. Se o *token* recebido for válido, o servidor fornece o recurso protegido para a aplicação. Agora, a aplicação tem acesso à informação que desejava do utilizador.

10 Guia de Utilização

Nesta secção iremos fornecer um guia de utilização do nosso servidor como opção de *login* para uma aplicação existente. Este guia irá basear-se no nosso primeiro teste: um cliente desenvolvido em *Node.JS*.

1. O primeiro passo é registar uma conta no nosso serviço.
2. De seguida, navegue para a página de criação de clientes e crie um, escolhendo um nome para ele.
3. Copie a informação dada, ou seja, o *ID* e segredo do cliente, para um ficheiro a partir do qual possa aceder à informação por dentro da página.

```
module.exports = {  
  'name': 'NAME',  
  'clientId': 'CLIENT_ID',  
  'clientSecret': 'CLIENT_SECRET'  
};
```

Figure 5: Exemplo de formatação de dados

4. Implemente algum tipo de *link* que reencaminhe o utilizador para o endereço `"/oauth/dialog/authorize"`. Este *link* terá de enviar os seguintes parâmetros: *redirect_uri*, *response_type*, *client_id* e *scope*. Este endereço irá apresentar o diálogo de autorização ao utilizador. Use o *client_id* que lhe foi dado no servidor. O *scope* tem o valor `"*"` por predefinição, que lhe dará toda a informação pública do utilizador.

5. Receba o código de autorização, enviado como um parâmetro com o nome `"code"` para o endereço de redireccionamento.

6. Envia um pedido com o método *POST* para o endereço de troca `"/oauth/token"`, para trocar o seu código de autorização por um *token* de acesso. Eis um exemplo de pedido de troca usando o *middleware Axios*:

```

axios({
  method: 'post',
  url: 'https://localhost:3000/oauth/token',
  data: {
    code: req.query.code,
    redirect_uri: redirectURI,
    client_id: clientID,
    client_secret: clientSecret,
    grant_type: 'authorization_code'
  }
})
.then(result => {
  console.log(result.data)
})

```

Figure 6: Exemplo de troca de código

7. Após a troca, irá receber na resposta um *token*. Pode então usar esse *token* para aceder à *API*. Eis um exemplo de um pedido à *API*, usando o *token* para autorização:

```

axios({
  method: 'GET',
  url: 'https://localhost:3000/api/user',
  headers: {'Authorization': 'Bearer ' + token},
})
.then(response => console.log(response))

```

Figure 7: Exemplo de uso de *token*

8. Com o passo anterior concluído, terá agora as informações do seu utilizador e deverá então autenticá-lo e guardar a conta dele como entender.

11 Exemplos de Páginas da Aplicação

Neste capítulo iremos mostrar algumas páginas da aplicação, para exibir algumas das funcionalidades a que os utilizadores têm acesso.

A seguinte página é a de registo. Nesta página o utilizador deve introduzir o seu *email*, que deve ser do domínio ”@uminho.pt” ou ”@alunos.uminho.pt”, o seu nome, número mecanográfico e *password*.

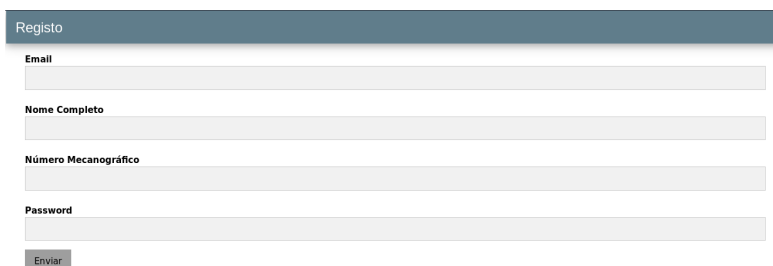


Figure 8: Página de registo de utilizadores

Com registo concluído e sessão iniciada, o utilizador tem acesso à página de informação de utilizador. É nesta página que o utilizador pode também associar contas de aplicações terceiras.

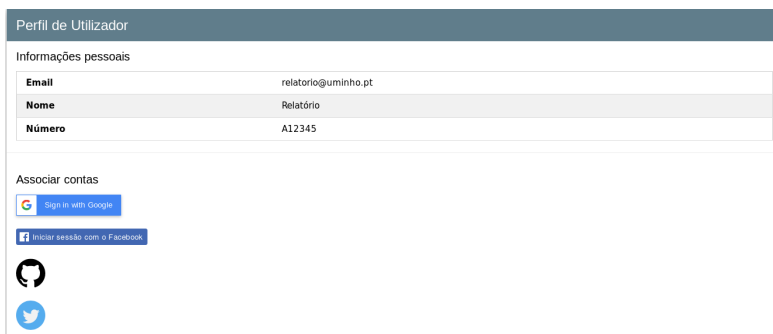


Figure 9: Página de informação de utilizador

Esta próxima página é a página a partir de qual se registam novos clientes. É preciso apenas colocar o nome do cliente a registar.




Figure 10: Página de informação de utilizador

A última página que vamos demonstrar aqui é a página com a lista de informação sobre clientes. Mostra, para cada um, o seu nome, o ID de cliente, segredo de cliente e o ID do utilizador que o registou.

Aplicações Registadas
Registrar nova aplicação
Nome: App1
ID de cliente: 13cdf9cd-7c82-408b-ae4a-872271e272b0
Segredo de cliente: b9535ff0-90a5-46de-95bd-120cbe3b6f66
ID de utilizador: relatorio@uminho.pt

Figure 11: Página de informação de utilizador

12 Conclusão

Em conclusão, o grupo está satisfeito com o trabalho feito. Procuramos descobrir como se implementa um servidor central de autenticação, não tendo experiência na área, e acabamos por produzir um servidor que, como intencido, pode ser usado para autenticação em outras aplicações. Este trabalho aprofundou o nosso conhecimento sobre *Web Development* em geral, mas especialmente na área de *back-end*.

Apesar disto, existem partes do trabalho que beneficiariam de mais tempo de trabalho. O grupo lamenta não ter sido possível realizar testes com mais tecnologias e tipos de aplicações. Os outros grupos que desenvolveram aplicações, tal como nós, tiveram também muito trabalho a fazer na última etapa do projeto, e então não tiveram muito tempo para dispensar para ligar as suas aplicações ao nosso servidor. Claro que podemos realizar testes sozinhos, criando novas aplicações feitas em diferentes linguagens e *frameworks*, o que tentamos, mas limitações de tempo impediram-nos de concluir mais testes. Por exemplo, o fato de o servidor não estar hospedado em nenhum *site* publicamente disponível impede de que sejam obtidos certificados *SSL* válidos, o que impediu a realização de testes com *Odoo*.

A interface do servidor é muito simples, e poderia também ter sido mais trabalhada. Existem também funcionalidades que podiam ter sido implementadas, como uma opção para remover atalhos de contas.