Universidade do Minho

Desenvolvimento de Aplicações WEB

Trabalho Prático

ViewBook

Autores:
Carlos Gonçalves - a77278
José Ferreira - a78452
Jorge Oliveira - a78660

Supervisor:
José Carlos Ramalho

29 de Janeiro de 2019

Conteúdo

Capítulo 1

Introdução

O presente relatório apresenta uma descrição da estrutura da aplicação **ViewBook**, esta tem um conceito parecido com a rede social *facebook*, onde há um "eu digital"de um utilizador.

Na nossa aplicação um utilizador poderá visualizar publicações de outros, criar publicações consoante um tipo, poderá ser do tipo desportivo para uma atividade física, do tipo culinária onde poderá inserir uma receita, do tipo casamento para postar uma evento único entre dois nubentes ou então criar uma publicação livre onde apenas exprima os seus pensamentos. Para além disto, pode comentar publicações ou responder a comentários, colocar gostos nestes mesmos, ou então criar/remover amizades com outros utilizadores.

Esta aplicação foi codificada com diferentes tecnologias, *Mongo* para o armazenamento de dados persistentemente, *NodeJS-Express* para o servidor de *backend* e o *Vue* para o *frontend*.

Capítulo 2

Desenvolvimento

2.1 Modelos

Decidimos representar 3 "entidades"que existem no sistema. São elas a **Publicacao**, **Utilizador** e ainda a **Hashtag**.

2.1.1 Publicacao

Na publicação guardamos o seu *status*, que nos indica o grau de visibilidade da mesma, poderá ser publica, privada ou só para amigos. Para tratar do problema do **tipo** da publicação decidimos criar um esquema para o mesmo. Neste esquema temos, como *required*, a informação que está presente em qualquer tipo (título, data e descrição), e depois os diferentes objetos de cada tipo, para um evento desportivo, casamento ou então o casamento.

Para além destes dados, temos também a data de quando a mesma foi publicada, o seu autor, onde incluímos o identificador único e o seu nome e ainda possivelmente a sua fotografia. Para os ficheiros e fotos decidimos criar dois *arrays* diferentes, porque facilitaria a distinção dos diferentes formato. Temos ainda um *array* com os identificadores das *hashtags* associadas à pu-

blicação e ainda outro que identifica os utilizadores que colocaram gosto na mesma.

Para os comentários decidimos que a melhor estratégia passaria por criar um esquema já que é um componente com informação diversificada. Guardamos a data em que o mesmo foi realizado, o corpo que contém a mensagem, o autor e ainda os utilizadores que colocaram um gosto. Decidimos ainda permitir responder a estes comentários, e para tal criámos outro esquema para o comentário aninhado que é igual ao comentário mas sem a possibilidade de existir outra resposta.

2.1.2 Utilizador

No utilizador obrigatoriamente guardamos o email do mesmo que é único em todo o sistema, o nome e ainda a sua *password* que no momento em que é guardada é cifrada para aumentar a segurança. Temos ainda outras informações pessoais que poderão ser opcionais como o sexo, a sua data de nascimento, morada e o seu telefone.

Temos ainda um *array* com a informação a cerca das fotos de perfil do utilizador, os identificadores únicos das publicações que estão associados ao utilizador, os amigos, onde para cada um são guardados o identificador único e o nome deles e ainda também um *array* com todos os pedidos de amizade onde guardamos o utilizador que efetuou o pedido e para quem é que foi direcionado.

Por fim temos ainda um *array* para controlar as notificações. Para guardar toda a informação relativa a uma notificação decidimos criar um esquema onde obrigatoriamente mantemos os dados sobre se a mesma já foi vista ou não, qual o tipo da mesma (gosto, comentário ou pedido de amizade) e ainda o texto da mesma, que é a mensagem que aparecerá para o utilizador. Temos ainda os identificadores do utilizador que gerou a notificação, e ainda da publicação a que a mesma está associada. Por ventura pode ainda estar associada a um comentário ou a uma resposta de um comentário.

2.1.3 Hashtags

A *hashtag* de todos os modelos foi a que se apresentou mais simples onde guardamos qual o seu nome, que por conseguinte deverá ser unico para todo o sistema e ainda temos uma lista com os identificadores das publicações a que estas estão associadas.

2.1.4 Redundância de Informação

Como se pode observar temos alguma redundância nos casos das publicações e quais os utilizadores e *hashtags* estão associadas. Mas era crucial para otimizar as pesquisas, já que por exemplo, quando procurámos por uma *hashtag* temos logo todos os identificadores das publicações que contêm aquela *hashtag* podendo aceder diretamente às mesmas e não tendo de percorrer todas as publicações e para cada uma verificar se a *hashtag* está presente ou não.

Como a informação guardada no utilizador era o identificador único e ainda o seu nome e a foto tivemos de ter em atenção que sempre que o utilizador alterar estes últimos dois tínhamos de efetuar uma atualização a toda a base de dados onde este utilizador estava envolvido. Apesar de ser uma operação custosa, é pouco frequente e compensava nas operações mais efetuadas que são apresentar o nome e foto do utilizador nas publicações e comentários.

2.2 Caminhos

2.2.1 API de Dados

feed - '/api/feed'

GET '/' - é retornado todas as publicações que o utilizador autenticado tem acesso (só as dos seus amigos e cujo o *status* não seja privado).

Pode ainda ser passado na *query* o limite, para limitar o número de publicações apresentadas. As mesmas são ordenadas consoante a sua data de criação da mais recente para a mais antiga.

pubs - '/api/pubs'

GET '/' - tem um comportamento parecido com a do *'api/feed/'*. No entanto, existe ainda a possibilidade de inserir na *query* o tipo e serão apresentadas apenas as publicações do tipo indicado

GET '/:id' - retorna a publicação que possui o identificador passado nos parâmetros

POST '/' - cria um novo documento na nossa base de dados referente à publicação passada no *body*. Utilizámos o *formidable* já que poderão ser passados ficheiros. No final da inserção é necessário criar uma pasta da publicação na pasta do autor para poder guardar todos os ficheiros. Para além disto, é acrescentado o identificador da publicação no utilizador e, caso existam, associadas as *hashtags* para esta publicação. É retornada a publicação criada.

POST '/comment' - acrescenta um objeto "comentario"ao *array* de comentarios da publicação. Por fim gera uma notificação para todos os utilizadores que devem de ser notificados (esta lista é passada no *body*). Neste caso os utilizadores notificados são todos os utilizadores que já efetuaram um comentário naquela publicação, sem existir repetidos e sem que o próprio autor do comentário receba uma notificação, e ainda o autor da publicação. Retorna a publicação completa associada ao comentário.

POST '/nestedComment' - muito parecido ao referido anteriormente, mas acrescenta um objeto "resposta"ao *array* de comentários anhinhados do comentário. Neste caso os utilizadores notificados são os que efetuaram uma resposta aquele comentário, o autor do comentário "pai"e ainda o autor da publicação.

POST '/gosto' - acrescenta um objeto "gosto"(que contém a informação do autor) à publicação ou comentário ou comentário aninhado. É efetuada uma validação dos identificadores existentes no *body* e consoante

os mesmos é atualizado corretamente a estrutura indicada. No final são ainda geradas notificações para os utilizadores associados. Neste caso é apenas o respetivo autor da publicação/comentário/comentário aninhado.

POST '/desgosto' - o comportamento é parecido ao referido anteriormente, mas ao invés de acrescentar, retira o objeto e não gera qualquer notificação.

PUT '/status/:id/:status' - altera o estado de uma publicação passada como parâmetro

perfil - '/api/perfil'

GET '/:id' - retorna o utilizador que possui o identificador passado nos parâmetros

item[GET '/foto/:id'] - retorna a lista de todas as fotos do utilizador que possua o identificador único passado como par âmetro

GET '/infoTotal/:id' - retorna as informações pessoais do utilizador cujo o identificador único seja igual ao passado nos parâmetros e o utilizador autenticado tem de ser amigo do mesmo

GET '/info/:id' - parecido ao referido anteriormente mas apenas apresenta o nome, a data de nascimento e morada caso existam

POST '/novaPub' - adiciona o identificador da publicação ao utilizador correspondente, ambos os identificaores são passados no *body*

POST '/adicionaPedido' - é adicionado um pedido de amizade na lista dos utilizadores envolvidos (utilizador autenticado e o utilizador cujo o pedido foi direcionado), é ainda gerada uma notificação para o utilizador ao qual o pedido foi destinado para que possa ser informado de que recebeu um pedido de amizade. No *body* é apenas passado o identificador único e o nome do utilizador para qual o pedido foi enviado, e depois é criado o objeto pedido contendo os dois utilizadores

POST '/aceitaPedido' - retira da lista de pedidos, o respetivo pedido onde mais uma vez segue no *body* a informação sobre o destinatário, mas para

além disso é acrescentado a cada um, na lista de amigos, o respetivo utilizador e é gerada uma notificação para quem gerou o pedido para o informar de que o pedido foi aceite e que tem uma nova amizade

POST '/recusaPedido' - tem um comportamento parecido com o indicado anteriormente, porém não adiciona os utilizadores à lista de amigos

POST '/removeAmizade' - remove da lista de amigos o utilizador correspondente

POST '/veNotificacoes' - altera o estado de visibilidade da lista de notificações (passadas no *body*) para "vista"

POST '/novaFoto' - adiciona uma nova foto de perfil à lista do utilizador. É sempre acrescentada no inicio, e ficará definida como a foto atual do mesmo

POST '/alteraInfo' - atualiza a informação do utilizador. Caso altere a *password* esta é cifrada antes de ser atualizada, para aumentar a segurança

GET '/nome/:nome' - retorna todos os utilizadores cujo o nome deles contenha o "nome"passado como parâmetro

hashtag - '/api/hashtags'

GET '/:id' - retorna todas as publicações que estão associadas ao nome da *hashtag* que é passado como parâmetro

POST '/' - adiciona um novo documento *hashtag*

GET '/novaPub' - adiciona à lista de publicações o identificador da publicação para um nome de *hashtag* especifico (ambos são passados no *body*). Caso a *hashtag* não exista, então é criado o respetivo documento.

De salientar que todas as rotas indicadas anteriormente estão protegidas, para que não exista um acesso indevido à nossa *api* de dados.

users - '/api/users'

POST '/registo' - cria um novo documento referente ao utilizador. O *body* é tratado com o *formidable* já que é submetido uma foto para o perfil. É ainda criado uma pasta para o utilizador, nomeada com o identificador único do utilizador, e ainda uma pasta "Fotos"para se colocarem todas as fotos de perfil do utilizador.

POST '/login' - é validado os dados de *login* do utilizador e ainda acrescentada toda a informação para a correta identificação do utilizador autenticado, que será explicado mais em detalhe adiante.

2.2.2 Interface

'/feed' e '/' - carrega todas as publicações que um utilizador pode visualizar (as que são dos amigos e que não tenham um estado privado). POderá ainda inserir uma nova publicação consoante o tipo desejado, pesquisar por um utilizador e ainda pesquisar por uma *hashtag*

'/profile/infoPessoais' - apresenta as informações pessoais de um utilizador, que poderam ser alteradas. Apenas apresenta as informações do utilizador autenticado

'/profile/:id' - apresenta o perfil de um utilizador com o identificador igual ao do passado como parâmetro

'/nome/:nome' - apresenta uma lista com os utilizadores que fazem *match* com o nome passado como parâmetro

'/pedidosAmizade' - apresenta uma lista com os pedidos de amizade direcionados para o utilizador que se encontra autenticado

'/amigos/:id' - apresenta a lista de utilizadores que são amigos do utilizador cujo seu identificador único é igual ao id passado como parâmetro

'/register' - permite que um utilizador se registe na nossa aplicação, após o preenchimento correto dos dados

'/login' - permite que um utilizador efetue uma autenticação com sucesso, caso o utilizador e a *password* coincidam

2.3 Autenticação

Para a autenticação utilizamos o *JWT*. Para gerar o *token* usamos os campos: **nome,email e id** para que pudessemos retirar todos estes campos a partir do *token* no futuro. Em todos os caminhos em que é necessária a autenticação, é verificado se o *token* existe e caso exista então a autenticação é realizada com sucesso, caso contrário não é autorizado. A verificação do *token* é sempre realizada em 2 locais diferentes, no *Authorization header* e na *Session*, para compatibilidade com aplicações como o *Postman*. De referir que no *login* a autenticação verifica se os campos *email* e *password* coincidem na base de dados.

Do lado do *FrontEnd* o *token* é guardado na *store* e sempre que se realiza um pedido é enviado no *Authorization header*. Para verificar a autenticação do lado do *FrontEnd* as rotas são protegidas e é verificado se existe o *token* na *store*.

2.4 Extras

Apresentámos agora alguns extras em relação aos requisitos pedidos no enunciado:

Gostos - os utilizadores podem efetuar um "gosto"nas publicações, comentários ou respostas aos comentários. Ainda foi pensada a hipótese de colocar diferentes reações, mas não foi conseguido, pelo que seria um melhoramento num futuro na nossa aplicação

Amigos - permitimos ainda que os utilizadores adicionam/removam amizades com outros utilizadores. O que implicou que uma publicação tivesse um estado adicional, "só para amigos", onde apenas é visivel para os amigos dos utilizadores

Notificações - quando é realizada alguma das atividades indicadas anteriormente os respetivos utilizadores são notificados para que não percam

qualquer atividade. Para receber as notificações devem de atualizar a página, já que apesar de utilizarmos uma interface "reativa"a mesma não é reativa ao atualizar a informação na base de dados. Mas mesmo assim, dá para simular a apresentação das notificações, apesar de não as receber *inTime*.

Apesar de não ser bem considerado um extra, decidimos frisar a utilização do *Vue* para a interface gráfica, ao invés do uso do *PUG* que foi recorrente durante todo o semestre. Utilizámos por ser reativa e porque apresenta uma série de funcionalidades que nos facilitaram em vários aspetos para a ligação e construção de diferentes elementos na interface.

Capítulo 3

Conclusão

O processo de desenvolvimento desta aplicação foi uma tarefa desafiante para todos os elementos do grupo. Uma das principais razões para isso foi a adoção de uma *framework* de *frontend*, o *Vue* visto ser a primeira experiência com esta tecnologia para todos. Permitiu aprender mais sobre a mesma, e facilitou em vários casos para apresentar os dados dinamicamente sem que seja necessário recarregar a página, facilitando a produção de uma aplicação *single-page*. Sabíamos que iamos enfrentar algumas dificuldades, já que durante todo o semestre trabalhamos com *PUG* para produzir a interface gráfica, mas desta forma permitiu-nos aproximar um pouco mais dos casos reais, e ainda ganhar um maior conhecimento. De salientar, que com mais experiência nesta tecnologia alguns pormenores poderiam ter sido realizados de uma forma diferente.

A maior dificuldade na nossa aplicação foi a autenticação. Por norma é sempre díficil, e no ínicio apreceram vários problemas que o grupo teve de corrigir, sendo que no final conseguimos colocar tudo funcional e todas as rotas protegidas devidamente.

Na comparação entre a produção do *frontend* e *backend* foi mais "fácil"codificar o segundo, não só porque no *frontend* utilizámos uma tecnologia nova, que é o *Vue*, mas também a parte da lógica de negócio já é mais intuitiva para os elementos do grupo.

Em suma, o grupo ficou agradado com o trabalho produzido. Conseguimos realizar os requisitos básicos propostos no enunciado bem como aplicámos outros extras que se aproximam um pouco mais da aplicação que nos serviu de base.