SDIS Millions

Relatório final

Mestrado Integrado em Engenharia Informática e Computação

Sistemas Distribuídos

05-06-2015

Grupo T5G03:

Eduardo Almeida – 201204989 - ei12018@fe.up.pt
João Almeida – 201206113 - ei12053@fe.up.pt
Miguel Fernandes – 201105565 - ei12137@fe.up.pt
Pedro Santiago – 201201737 - ei12044@fe.up.pt

Tabela dos conteúdos:

1. Introdução

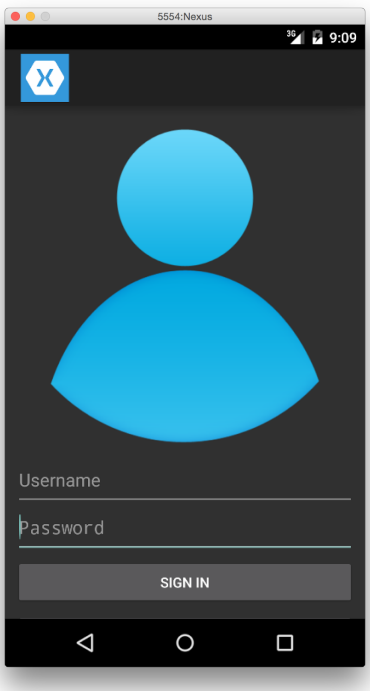
Para a concretização do segundo projeto de SDIS, resolvemos desenvolver uma aplicação para Smartphone, com um servidor REST para a suportar.

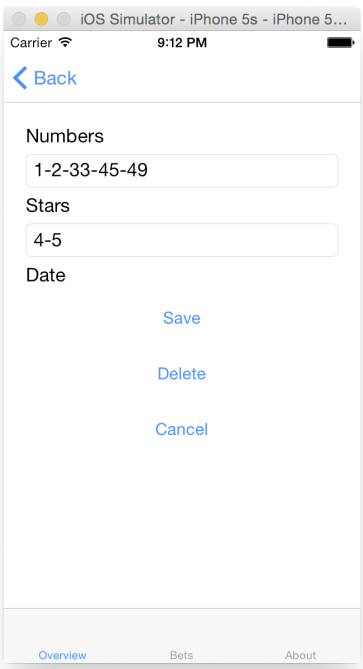
A *SDIS Millions* tem a capacidade de facilitar a interação com o famoso jogo de apostas "Euro Milhões". Através de uma simples fotografia do bilhete de apostas, a chave apostada é registada na aplicação, numa base de dados que é sincronizada com um servidor remoto. Mal o sorteio ocorre, o utilizador é notificado se a sua chave é a vencedora (ou não).

Conseguimos também observar as chaves apostadas anteriormente, assim como o sucesso ou falhanço destas.

Em suma, estas são as principais funcionalidades da aplicação:

- Adicionar uma aposta através de fotografia;
- Verificação automática de prémio;
- Listagem de apostas passadas;
- Sincronização destas apostas com um servidor central.





Arquitetura

Existem vários componentes importantes na nossa aplicação, das quais algumas foram desenvolvidas por nós, para este projeto, e outras são componentes já existentes desenvolvidos por terceiros.

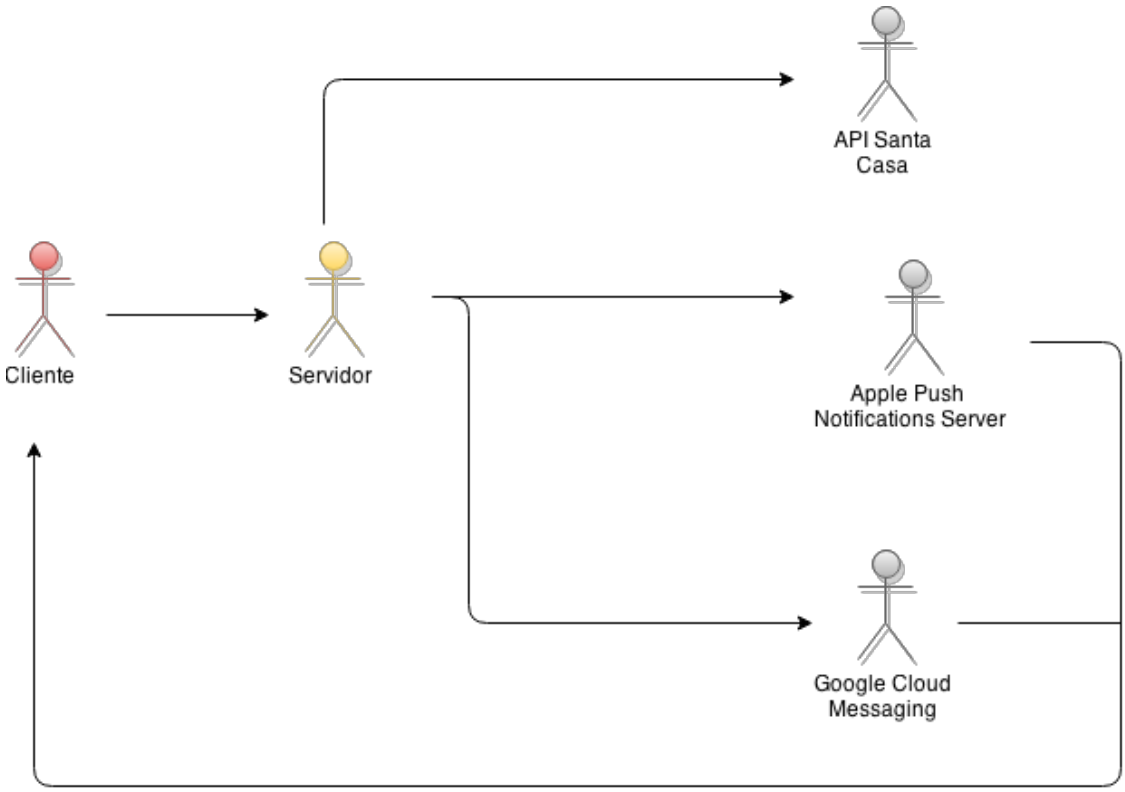
Componentes *"1st-party"*:

- Cliente
  - Desenvolvido em C#
  - Aplicação Android, iOS
- Servidor
  - Desenvolvido em PHP e mySQL, apoiado em nginx

A comunicação entre estes dois componentes é estabelecida através de uma API REST, cuja documentação está em anexo.

Componentes *"3rd-party"*:

- APNS (Apple Push Notifications Server)
  - Envia notificações *push* para dispositivos a correr o sistema operativo iOS.
  - A comunicação é estabelecida através de uma interface binária, utilizando um design que assenta sobre TCP.
- GCN (Google Cloud Messaging)
  - Envia notificações *push* para dispositivos a correr o sistema Android.
  - A comunicação é assegurada através de uma interface JSON que assenta sobre HTTP.
- API Santa Casa
  - De acesso aberto, desenvolvido por Nuno F. C. Guerreiro, e disponível através de um request GET no endereço https://nunofcguerreiro.com/api-euromillions-json.


API Santa
Casa
Cliente
Servidor
Apple Push
Notifications Server
Google Cloud
Messaging

3. Implementação

São de notar dois grandes componentes na nossa aplicação: o(s) clientes, e o servidor.

3.1 Cliente

A parte de cliente da aplicação é implementada em C#, utilizando a plataforma Xamarin.Forms, que permite usar a framework .NET para desenvolver aplicações para os iOS, Android e Windows Phone.

São apenas utilizadas frameworks externas para permitir a utilização de uma base de dados SQLite interna (SQLite.Net-PCL) e para tratar as notificações push de uma maneira mais fácil (Xam.Plugin.PushNotification).

Foi utilizado um modelo MVC (Model View Controller) para desenvolvimento da aplicação móvel, de modo a facilitar a programação cross-platform devido ao nível de abstração que providencia.

Estrutura do Cliente

- EuroMillions
  - Código multiplataforma.
- Euromillions\Database
  - Código referente à implementação da base de dados SQLite.
- Euromillions\Helpers
  - Contém uma classe com um conjunto de funções auxiliares úteis.
- EuroMillions\Notifications
  - Código referente à implementação das *push notifications*
- EuroMillions\Pages
  - Código referente ao layout,design e controlo das diversas páginas da aplicação.
- EuroMillions\Security
  - Código referente à implementação dos métodos de encriptação utilizados.
- EuroMillions\Webservice
  - Código referente à implementação dos métodos que comunicam com o servidor através das *APIs* desenvolvidas.
- EuroMillions.Droid
  - Código gerado pela framework relacionado com código nativo Android. Existe porções de código nativo adicionado por nós para implementar funcionalidades, nomeadamente, base de dados (SQLite_Android).
- EuroMillions.iOS
  - Código gerado pela framework relacionado com código nativo Android. Existe porções de código nativo adicionado por nós

para implementar funcionalidades, nomeadamente, base de dados (SQLite_iOS).

3.2 Servidor

O servidor desta aplicação é implementado em PHP, com recurso a uma base de dados mySQL e ao servidor web nginx.

É utilizada a ferramenta TesseractOCR, em conjunto com um wrapper para PHP, para proceder ao reconhecimento dos "tickets" de Euro Milhões.

De modo a ajudar no design da API REST, baseamo-nos num exemplo disponível em http://coreymaynard.com/blog/creating-a-restful-api-with-php/ para começar o desenvolvimento, que foi fortemente modificado à medida que o projeto ia ficando cada vez mais maduro.

Todos os outros ficheiros foram inteiramente desenvolvidos por nós.

A concorrência dos requests é tratada automaticamente ao nível do servidor web, sendo apenas dependente do número de worker threads do mesmo.

As mensagens do protocolo são tratadas inteiramente pela classe EuroMillionsAPI, localizada em /api/v1/endpoint.php, que trata de chamar as funções responsáveis e devolver as respostas apropriadas, cuja documentação está localizada no Anexo 1.

Estrutura do Servidor

- api/v1/endpoint.php
  - Processa todos os requests enviados ao servidor.
- cron/cron_draw_check.php
  - Contacta uma API externa de modo a verificar qual o resultado do último sorteio.
- inc/config.inc.php
  - Configurações gerais do servidor.
- inc/database.inc.php
  - Classe que gere a ligação à base de dados.
- inc/ocr.inc.php
  - Trata todos os aspetos relacionados com o reconhecimento do ticket.
- inc/push.inc.php
  - Trata do envio de notificações PUSH para os dispositivos móveis.
- inc/sync.inc.php
  - Trata da sincronização dos dados do cliente com o servidor.
- inc/users.inc.php
  - Trata de todos os pormenores relacionados com o login/registo dos clientes.

4. Detalhes Relevantes

Segurança

A segurança foi, sem dúvida alguma, o aspeto que mais relevância teve ao longo do desenvolvimento da aplicação.

A aplicação oferece a opção de sincronização da base de dados de apostas para um servidor, de modo a poder ter acesso à mesma informação em todos os seus dispositivos.

Esta base de dados é completamente encriptada no lado do cliente, nunca sendo possível ler os conteúdos da mesma sem os dados de acesso à conta do utilizador. Cada utilizador têm um *salt* diferente associado (salt guardado na *cloud*), e todos os dados são encriptados com o algoritmo SHA-512 e são *encoded* para *base*64 para posteriormente serem enviados de forma segura para a *API*.

Quanto à conta do utilizador, os dados de acesso à mesma também são completamente encriptados do lado do cliente, apenas tendo o servidor acesso à hash da palavra-passe, modificada com uma salt também gerada no cliente.

Tolerância a Falhas

A aplicação cliente consegue realizar consultas sobre as apostas passadas sem efetuar qualquer tipo de comunicação com o servidor, estando minimamente funcional mesmo que o servidor fique offline.

As funções são retomadas na sua totalidade assim que uma ligação ao servidor é corretamente estabelecida.

Consistência

Sempre que a base de dados local de apostas é modificada, a mesma é atualizada no servidor de modo a que outros dispositivos associados à mesma conta de utilizador tenham acesso à mesma informação.

5. Conclusão

No geral, o grupo está muito satisfeito com o produto final desenvolvido. Gostaríamos, no entanto, de ter desenvolvido outras características que só iriam valorizar o projeto, como a realização de apostas dentro da aplicação. Apesar de ambicioso, seria uma excelente experiência dar continuidade ao que foi desenvolvido até ao momento.

5.1 Trabalho Realizado por cada Membro do Grupo

- Eduardo Almeida
    - Implementação da API REST;
    - Implementação do código PHP que suporta a API REST;
    - Configuração do servidor.
    - Contributo para o resultado final: 35%
- João Almeida
    - Implementação da aplicação móvel.
    - Contributo para o resultado final: 35%
- Miguel Fernandes
    - Desenho e implementação da base de dados do cliente;
    - Desenho e implementação da base de dados do servidor;
    - Desenvolvimento de todos os assets gráficos.
    - Contributo para o resultado final: 15%
- Pedro Santiago
    - Desenho e implementação da base de dados do cliente;
    - Desenho e implementação da base de dados do servidor;
    - Desenho da API REST.
    - Contributo para o resultado final: 15%

6. Bibliografia

http://en.wikipedia.org/wiki/Representational_state_transfer

https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html

https://developers.google.com/cloud-messaging/server-ref

http://xamarin.com/forms

https://nunofcguerreiro.com/API-Interface-de-Programacao-de-Aplicacoes

7. Anexo - Documentação da API REST

GET /api/v1/login

+username: text; +password: text
{result: bool, ?key: text, ?error: text}

GET /api/v1/salt/[Resource Identifier]

(No Body Required)
{result: bool, ?salt: text, ?error: text}

POST /api/v1/register

+username: text; +password: text; +salt: text; +email: text
{result: bool, ?error: text}

GET /api/v1/ID/[Resource Identifier]

+key: text
{result: bool, ?id: int, ?error: text}

GET /api/v1/OCR/[Resource Identifier]

+key: text
{result: bool, ?response: {numbers: array, stars: array}, ?error: text}

POST /api/v1/OCR

+key: text; +blob: data/base64
{result: bool, ?request: int, ?error: text}

DELETE /api/v1/OCR/[Resource Identifier]

+key: text
{result: bool, ?error: text}

GET /api/v1/data/[Resource Identifier]

+key: text
{result: bool, ?data: data/base64, ?error: text}

PUT /api/v1/data/[Resource Identifier]

+key: text; +blob: data/base64
{result: bool, ?error: text}

GET /api/v1/gameResults

+key: text
{result: bool, ?drawn: {+date: date, +numbers: array, +stars: array}, ?error: text}

GET /api/v1/push/[Resource Identifier]

+key: text
{result: bool, ?result: array, ?error: text}

PUT /api/v1/push/[Resource Identifier]

+key: text
Body: +data: text
{result: bool, ?error: text}

DELETE /api/v1/push/[Resource Identifier]

+key: text
{result: bool, ?error: text}