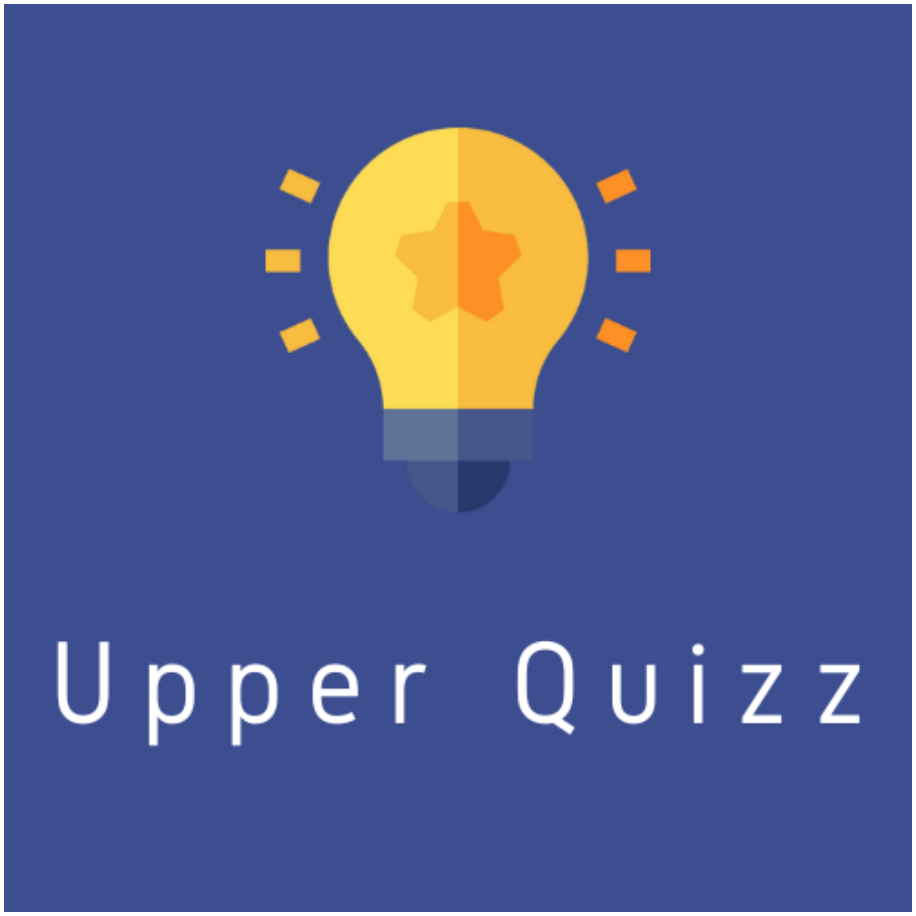
UpperQuizz

Rodrigo Francisco Pablo, Emanuel Flores Martínez, Galileo Garibaldi Cabrera

<2021-09-25 Sat>



Notas:

- En este repositorio se aloja el código de la API REST. Que a su vez se aloja en Heroku
- La documentación de la api se encuentra aquí
- El código de la aplicación iOS se encuentra en el siguiente link
- La documentación técnica se puede leer en este *README* o en el siguiente link
- La presentación del proyecto se puede leer aquí

Introducción

UpperQuizz esta orientada para personas (en su mayoría estudiantes) que se encuentran en el proceso de admisión a educación superior, es decir, que quieren ingresa a la universidad. Como objetivo base, nos enfocamos en el mercado usuarios cuyo objetivo se entrar a universidades como la UNAM, IPN, UAM o BUAP.

La aplicación se plantea como una herramienta de apoyo que complemente los estudios y métricas de los estudiantes. En otras palabras, la aplicación, en su primera versión no pretende sustituir un curso completo de ingreso a nivel superior. Esta aplicación ayudará a realizar exámenes diágnostico y dar seguimiento para medir el progreso.

Actualmente existen aplicaciones similares en el mercado, cómo las que se enlistan a continuación.

Reglas de negocio

Un estudiante debe registrarse a la aplicación. Una vez registrado podrá acceder, en principio a 3 evaluaciones. Dicha evaluaciones, también pueden llamarse *exámenes de simulación*.

Las evaluaciones puede contestarse las veces que sean necesarias con la finalidad de que el usuario practique y domine las preguntas de cada una de las materias.

Un examen de simulación esta conformado por un conjunto de preguntas (puede variar conforme la evaluación). Dicho conjunto de preguntas se dividen en preguntas por materia, las materias consideradas son:

- Matemáticas
- Español
- Física
- Química
- Biología
- Historia universal
- Historia de México
- Literatura
- Geografía
- Filosofía

La pregunta se conforma de un texto (el cuál describe la pregunta) y 4 opciones por pregunta. Dichas opciones son texto plano.

Cuando un estudiante termine de realizar el examen se le califica, se guardan los siguientes datos: fecha de realización, puntaje total, puntaje por cada materia y respuestas del estudiante. Además también se guarda el identificador del examen que se realizó. La interfaz despliega los puntajes gráficamente, de tal forma que el usuario se puede dar cuenta de su progreso.

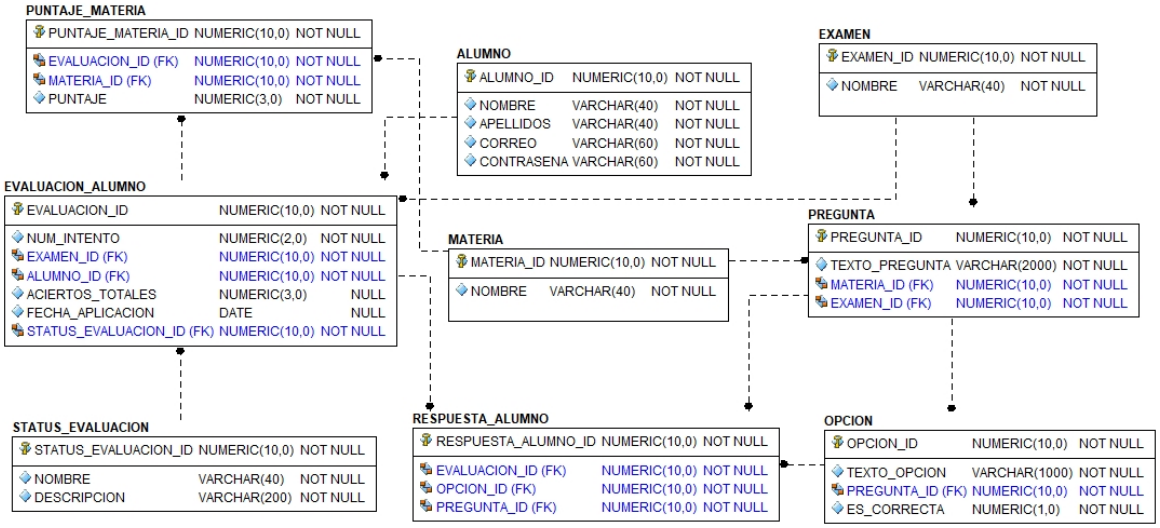
Además de la sección de las evaluaciones, el usuario puede ver su historial, en donde se le desplagan las evaluaciones que ha realizado, con sus aciertos generales y aciertos por cada materia. En esta misma sección se le muestra un promedio general en escala del 0 a 10.

Por último, se tiene una sección llamada "Recodatorios", en donde se le muestra al usuario información las convocatorias de las universidades más importantes, así como la fecha de registro al examen y fecha tentativa de realización del mismo.

Características del backend

Para almacenar toda la información mostrada en la aplicación, así como la información generada por el usuario se hizo lo siguiente

Modelo de datos



Tecnologías utilizadas

PostgreSQL

PostgreSQL es un gestor de base de datos, que se reconoce por ser un gestor de tipo entidad-relación *open source*. Se distingue por ser robusto, confiable y tener buen rendimiento. Además de ser soportando en muchas plataformas de *hosting*.

Python

Python es un lenguaje de programación interpretado de alto nivel y de propósito general. Esta diseñado con la fisolofía de ser fácil de leer, crear código limpio y código limpio para proyecto pequeños y de gran escala. Además de ser muy versátil ya que sus aplicaciones son desde la creación de aplicaciones de escritorios, aplicaciones web, sistemas embebidos, etc.

Flask

Flask es un *micro* framework web escrito en Python. Se clasifica como microframework porque no necesita de ningua herramienta o librería en particular para poder funcionar. No tiene abstración de capa de base de datos, validación de formularios o cualquier otro componente provisto por librerías de tercer. Todo lo anterior se agrega por medio de extensión o dependencias manejadas por el usuario según su conveniencia. Para el caso de este proyecto se utilizó Flask para crear la API REST

Heroku

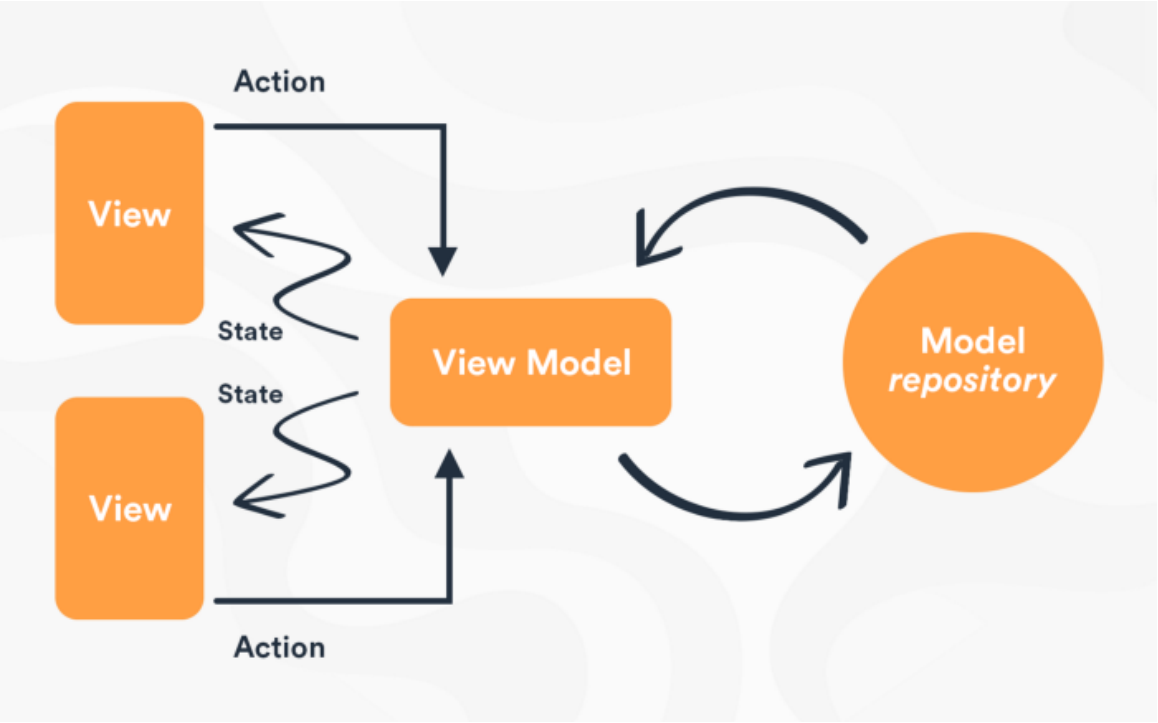
Es una plataforma conocida como plataforma como servicio (PaaS, por sus siglas en inglés), permite a los desarrolladores construir, correr y operar aplicaciones enteramente en la nube. Soporta múltiples lenguajes de programación, como Ruyb, Java, Node.js, Scala, Clojure, Python, PHP y Go. Además, gracias a los *add-ons* podemos agregar servicios de base bases de datos y algunas otras tecnologías que nuestra aplicación pueda ocupar.

Características de la aplicación

A continuación se exponen la arquitectura MVVM, así como algunas herramientas para la creación de la aplicación

MVVM (Model - View - View Model)

MVVM es la abreviación de 3 palabras: Model, View y View Model. El concepto principal de MVVM es construir un *view model*, que pueda representar los datos a través de la vista.



En la imagen anterior se observa que *view model* o la vista-modelo ocupa la posición central, por lo que se encarga de envíar y recibir los datos del *modelo* y proveerlos a la *vista*

Modelo

Representa los datos (en tiempo real), que serán utilizados en la aplicación. El modelo se utiliza principalmente para separar de los datos de la lógica de negocios.

Vista

La vista representa la interfaz con la que el usuario interactua en la aplicación. Este elemento también posee propiedades para utilizar comportamientos asociados con el modelo, como identificar y actuar conforme la entrada del usuario.

Vista-Modelo

Es la parte más esencial de la arquitectura MVVM, el *viewModel* presenta la parte de la vista separada del modelo. Hace que la vista solo sostenga la parte **formateada** de los datos, además se encarga de la comunicación entre el modelo y la vista.

Gestor de dependencias

Como gestor de dependencias se hizo uso de **Carthage**.

Carthage, es un gestor de dependencias que nos permite agregar frameworks a nuestra aplicación de *Cocoa*. Este gestor se distigue por compilar la dependencias y proveer frameworks en forma de binarios sin modificar la estructura original del proyecto.

Ejecución la app

Requisitos:

- *Carthage*
- *XCodde* en su versión más reciente

El primer paso es descargar el repositorio, utilizando la opción de *zip* o vía *git*

```
git clone https://github.com/e-muf/UpperQuizz-iOS
```

En caso de no tener Carthage, se debe instalar con `brew`

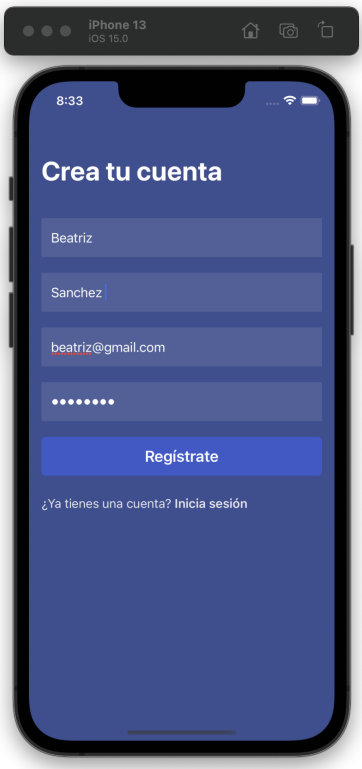
```
brew install carthage
```

Dentro del directorio del repositorio ejecutamos los siguiente

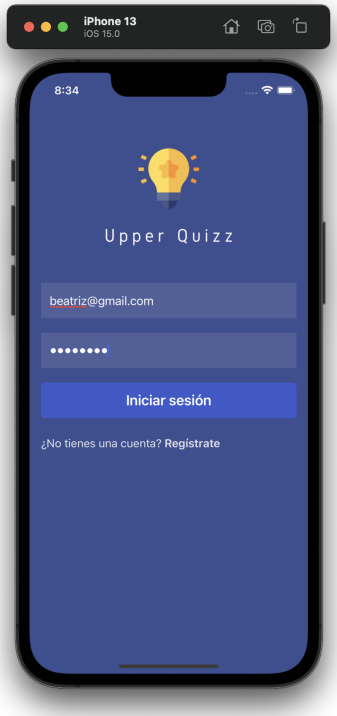
```
carthage bootstrap --platform iOS --use-xcframworks --no-use-binaries
```

Screenshots

Registo



Login

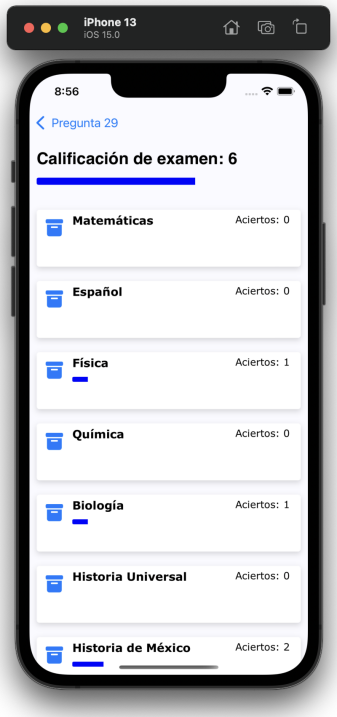


Evaluaciones (pantalla principal)

Quizz

Recordatrios

Resultados examen



Conclusiones

En este proyecto se utilizaron los conocimientos adquiridos en los tres módulos de diplomado, por ejemplo, del módulo uno se rescatan todas las herramientas que se tienen para programar en `Swift`, como son extensiones, getters, setters, genéricos, etc. Del segundo módulo se rescata la importancia de construir una arquitectura robusta que permita separar bien la lógica de los datos, así como también varios conceptos esencial sobre el *backend* de nuestras aplicaciones. Por último, para el tercer módulo se rescata la importancia de manejar adecuadamente la memoria, evitando ciclos de retención y manejando adecuadamente las tareas ejecutadas en el hilo principal y/o los demás hilos. Así mismo, a lo largo de la creación de esta aplicación aprendimos a aterrizar las reglas de negocio en modelos de datos que posteriormente se convirtieron en información.