EPI-USE LABS
value through innovation

Architecture

Contents

1. Architectural Design Strategy

It was decided that a Decomposition design strategy would be the best for this project. The Decomposition design strategy integrates seamlessly with the technology used. The Front-end of the product uses Ionic React, which prefers the pure component implementation approach. The database and Server (Heroku) uses a ORM (Object-Relational Mapping) design, which treats each table in the database as a unique entity, and ARCore (Android) and ARKit (iOS), which are separate tools and need individual work. The Decomposition Design Strategy is efficient, ensures that components can be reused and allows for improved error-management.

2. Architectural Styles

2.1. Client-Server Architecture (multi-tier)

The Gym King server will be hosted on the cloud service Heroku to provide a service to many different users. The Client-Server architecture is a style that consists of one server that provides for a number of users which is what is required for the Gym King server. The server does not need to know each of the clients and the clients do not need to know each other. The clients are completely dependent on the server. The client-server architecture was chosen to reduce the storage space and computational power needed on the client device. This will save the device's power and will allow lower end devices to be supported as they do not need to process information themselves. This is as long as the lower end devices understand what is being sent to them from the server.

2.2. Component-based Architecture

The Gym King application is programmed using Ionic React. The application will be broken down into components that will then be used together to provide the required functionality. This follows the Component-based architecture style as the different components accomplish different tasks of the Gym King application.

2.3. Cloud Computing Architecture

As said above with the client-server architecture, we have to use a cloud-based platform to allow the server to always be available and online. Inside the cloud platform, we run our client-server architecture. Our server now acts as a cloud service which the Gym King application will use on behalf of the users to minimise use of the devices' resources. The application can make calls to the persistent server to ask for resources that it needs to serve the user. An example of this would be the Augmented Reality models. The Gym King application does not have these models saved locally to save space so it would rather ask the server for the models that are needed. The server is persistent on the internet and access to it is gained from the Heroku cloud platform. This means Gym King applications should be able to reliably receive resources from it.

3. Architectural Quality Requirements

3.1. Availability

The system is expected to have at least a 95% uptime. Deployed with no single point of failure.

3.2. Deploy-ability

The Gym King application must be deployed, working, and supported on both Android and IOS devices.

3.3. Maintainability

The system should be easily understood by future developers. The system should use pure components, such that an individual component can be improved upon without affecting the system. The system should last for several years after deployment, meaning that it should not rely on any packages or services that will become deprecated in the near future. The system should be fully documented, and the at least 50% of the project must have comments detailing their functions.

3.4. Security

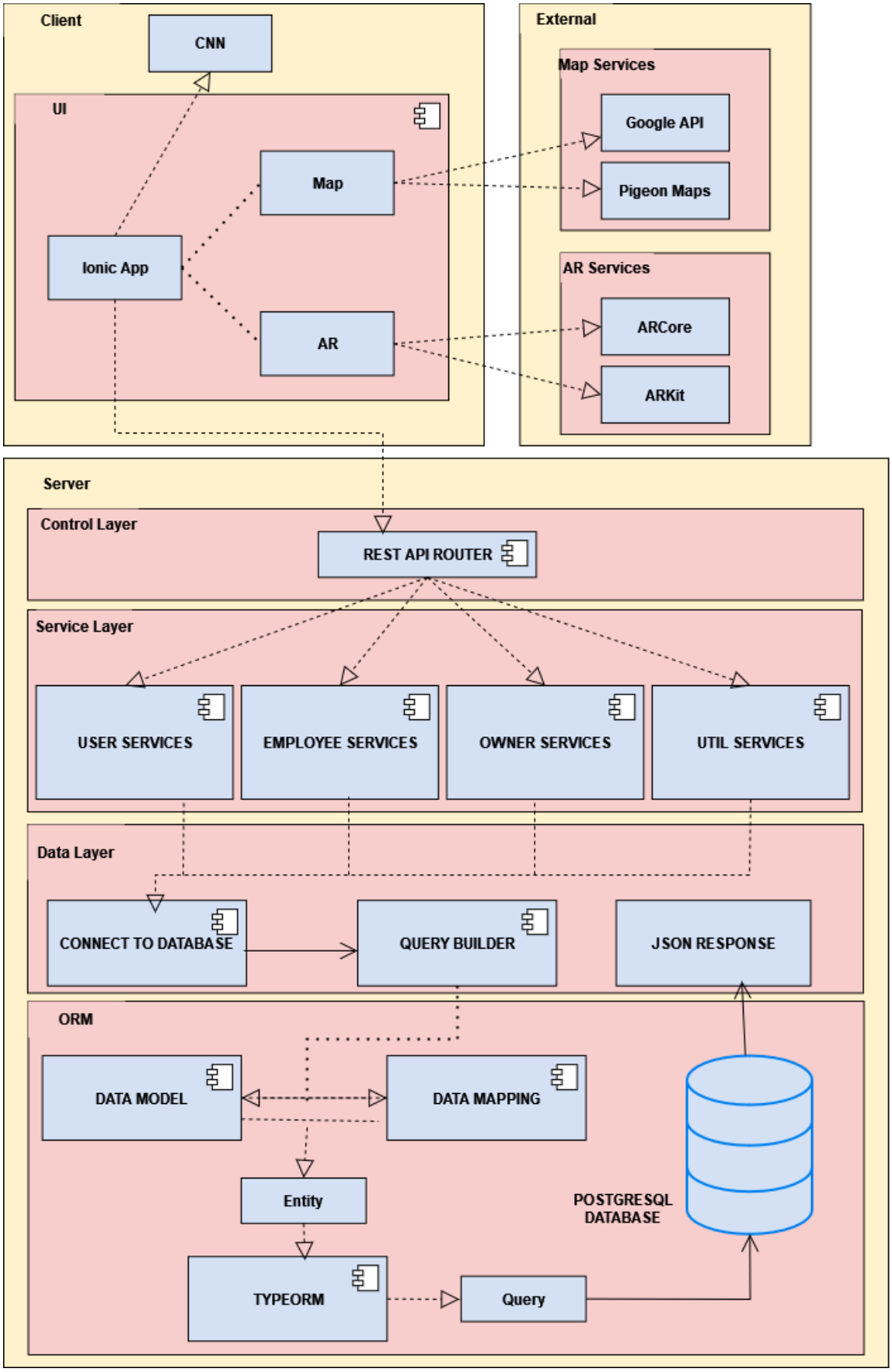
The system will use role-based access control. Unregistered users should not be able to make use of services offered by the Gym King application. Sensitive user information should be encrypted when stored and should not be accessible by unauthorized parties. The system should not be liable to SQLi attacks, and the components used should not be single-points of failure.

3.5. Scalability

The system is designed to handle 50 user requests per second. The system should allow for the addition of extra functionality and/or components. The Gym King server can be scaled up by purchasing more dyno hours, higher tier Heroku server for more connections and better PostgreSQL tier for more connections and storage size.

4. Architectural Design and Pattern

As explained above, Gym King uses the Client-server pattern, the Cloud-computing pattern and the component-based pattern. The application also uses the ORM (Object-Relational Mapping) pattern. The Architecture is based on the Decomposition design strategy, where all the various components and systems with the application are broken down into smaller and more manageable parts. The architectural patterns where chosen to reflect this. The component-based pattern and the ORM pattern both break down the various components and tables and makes them separate entities. These separate entities are mostly independent of others and are able to be reused in other parts of the application. The Client-server pattern and the Cloud-computing pattern where chosen to reduce the strain that might be placed on the client device and will allow for improved scalability and manageability as there is a central control of the application.


Client
CNN
UI
Map
Ionic App
AR
External
Map Services
Google API
Pigeon Maps
AR Services
ARCore
ARKit
Server
Control Layer
REST API ROUTER
Service Layer
USER SERVICES
EMPLOYEE SERVICES
OWNER SERVICES
UTIL SERVICES
Data Layer
CONNECT TO DATABASE
QUERY BUILDER
JSON RESPONSE
ORM
DATA MODEL
DATA MAPPING
Entity
TYPEORM
Query
POSTGRESQL DATABASE

5. Architectural Constraints

1. Limited number of Geo calls per month for the Map component.
2. PostgreSQL database is situated on a cloud platform.
3. NodeJS server works in TypeScript and is situated on a cloud platform.
4. The Heroku Server does not have any South African servers.
5. The Heroku PostgreSQL database has limits on simultaneous connections, row count and total storage size.
6. The Heroku Server has a dyno hour (uptime) limit of 550 hours per month.
7. The system is limited to open-source technology.
8. The system must run smoothly on a mid-range smartphone/smart device.
9. The system should ensure that battery drainage is minimised as far as possible.
10. AR capabilities are limited to devices that support ARCore (Android) and ARkit (iOS).

6. Technology Choices

6.1. Heroku

6.1.1. Overview

Heroku is a cloud platform that allows us to run a server on the web in its environment. We add our server code with profile to tell Heroku how to start our server.

6.1.2. Pros

- Free to host a server
- Free PostgreSQL database
- Easy to use
- Lots of online help and documentation

6.1.3. Cons

- Limited dyno hours (uptime) per month.
- Does not have South African based servers
- Limits on simultaneous connections, row count and total storage size.
- Only one server running

6.1.4. Reasoning

Heroku allows NodeJS and its various libraries to operate on it. This allows it to work seamlessly within the client-server architecture. It also incorporates libraries to assist with the architecture, such as the express library to make a REST API and Type ORM which will allow the product to make entities of tables for better use of the PostgreSQL database. Heroku is an open source technology that does not have the same constraints that many other cloud service providers offer. Heroku was chosen since it had a wide range of supported languages and had a good uptime.

6.2. Ionic React

6.2.1. Overview

Ionic React is a native React version of the Ionic Framework. It is open-source and allows for development for both iOS and Android devices.

6.2.2. Pros

- Open Source
- Compatible with iOS and Android.

6.2.3. Cons

- Requires numerous plug-ins to properly function.
- Debugging can be a tedious task as error messages are sometimes vague.

6.2.4. Reasoning

Ionic React works well with the component-based pattern, since the technology works best by splitting components into pure components that can be reused throughout the system. The technology works well the client-server pattern, since it does not need massive local storage space. The technology is also open source which is a key constraint on the development of the product. Ionic React was the only option for the front end component as it was specifically asked for by the clients.

6.3. ARCore and ARKit

6.3.1. Overview

ARCore is designed to assist with the creation of augmented reality (AR) components that seamlessly integrate the digital and physical worlds. We are using it for Android AR components.

ARKit is an iOS specific tool kit designed to create augmented reality (AR) components that brings the digital world to the physical world.

6.3.2. Pros

- ARCore: Works with multiple development environments, such as Android and iOS.
- ARCore: Large development and support community.
- ARCore: Maps the surroundings relatively accurately.
- ARKit: The best AR software for iOS devices.

6.3.3. Cons

- ARKit: Only works with iOS devices.
- ARCore: Scanning accuracy could use improvement.

6.3.4. Reasoning

ARCore and ARKit work well with the client-server architecture and the cloud-based architecture. This is because the AR components are stored on the cloud/server side and then downloaded to the client side when the badge is viewed and/or earned. ARCore and ARKit work well with their respective operating systems and both allow for components to be separated from the rest of the product and features, which allows for improved error-management. It is an external technology that is important to the overall product, however, it does not have much of an impact on the core architecture. ARCore and ARKit were chosen as they were the best AR tool kits available that were largely open source. Since open source is an important requirement of the project, no other technology was possible.