

Architecture and Design Document

Healthy Belly

Mohammad-Murtuza Bharoocha

Josue Rodriquez

Hassan Ishmam

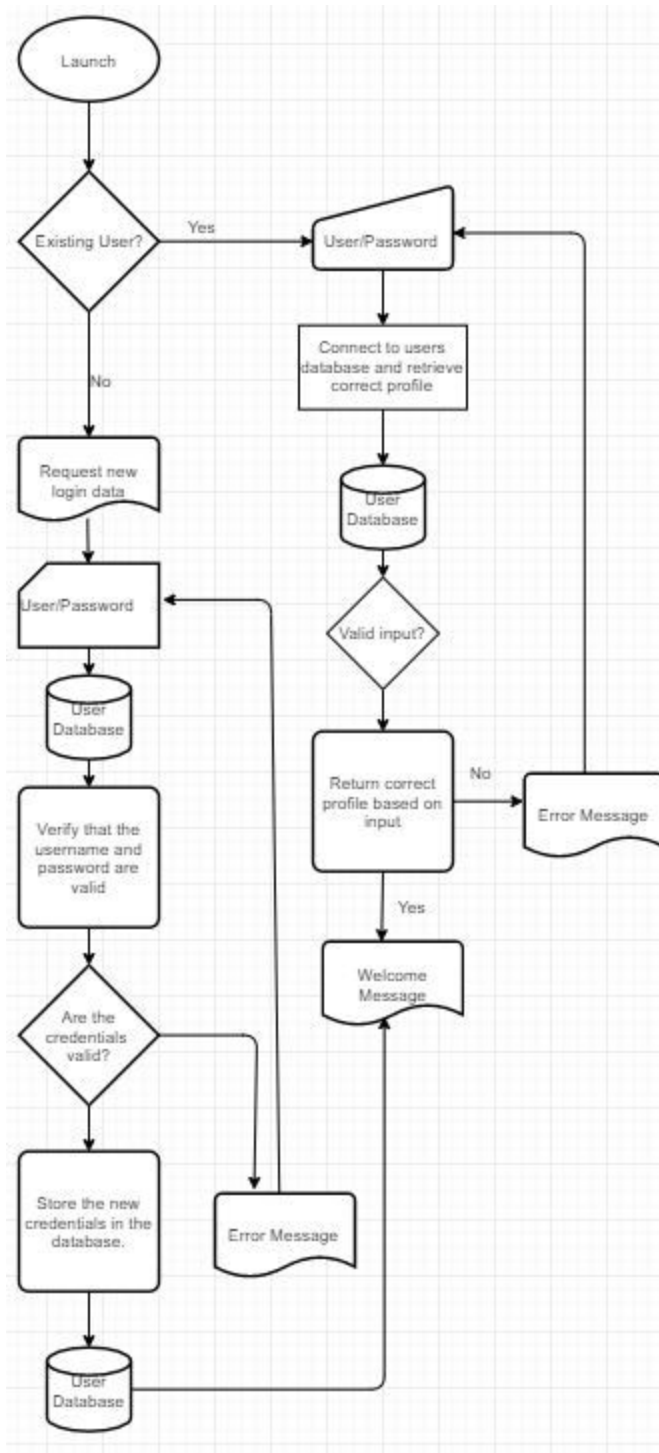
Mateo Perez

Priyanshi Vaishnav

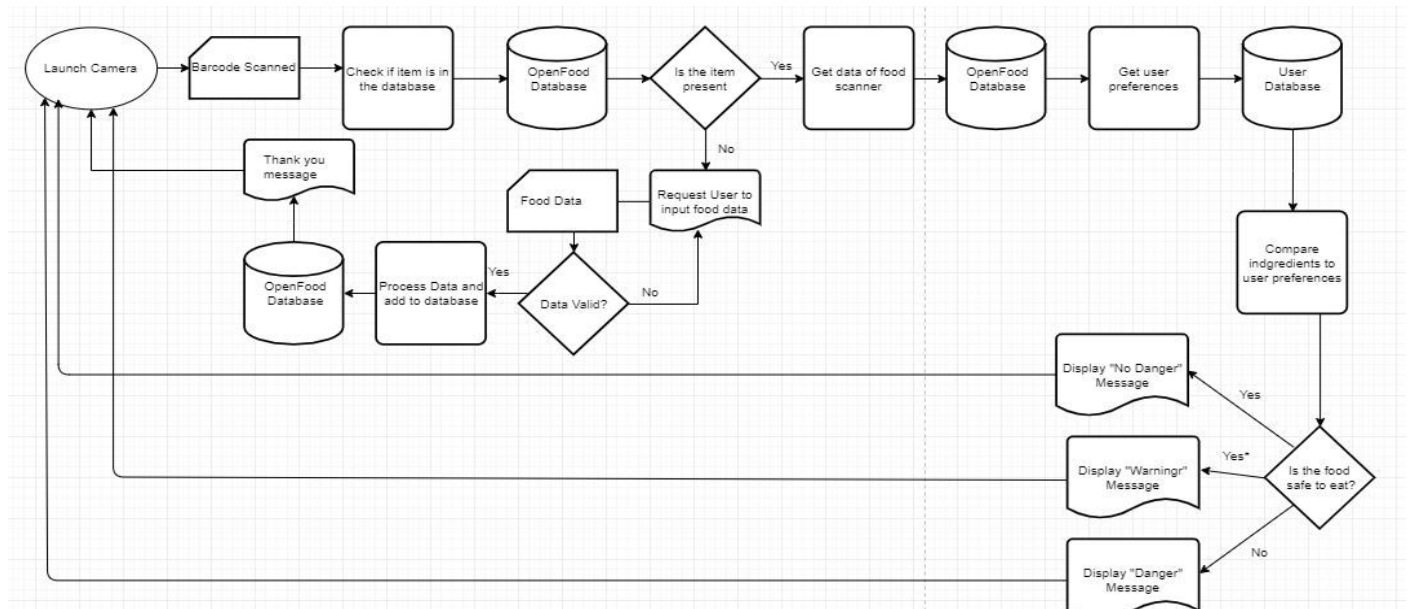
Nisal Gamage

Information Flow Diagrams

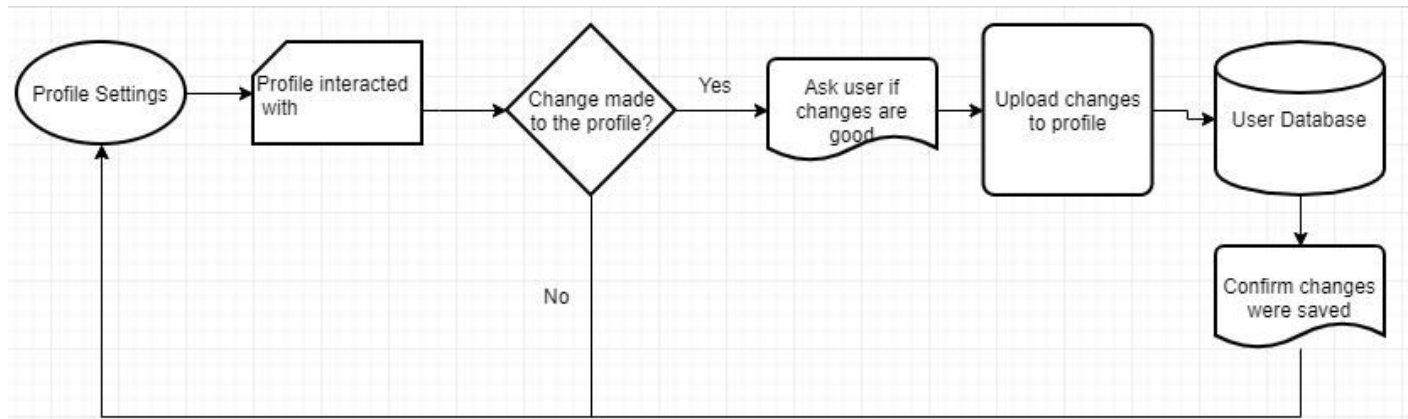
User Login:



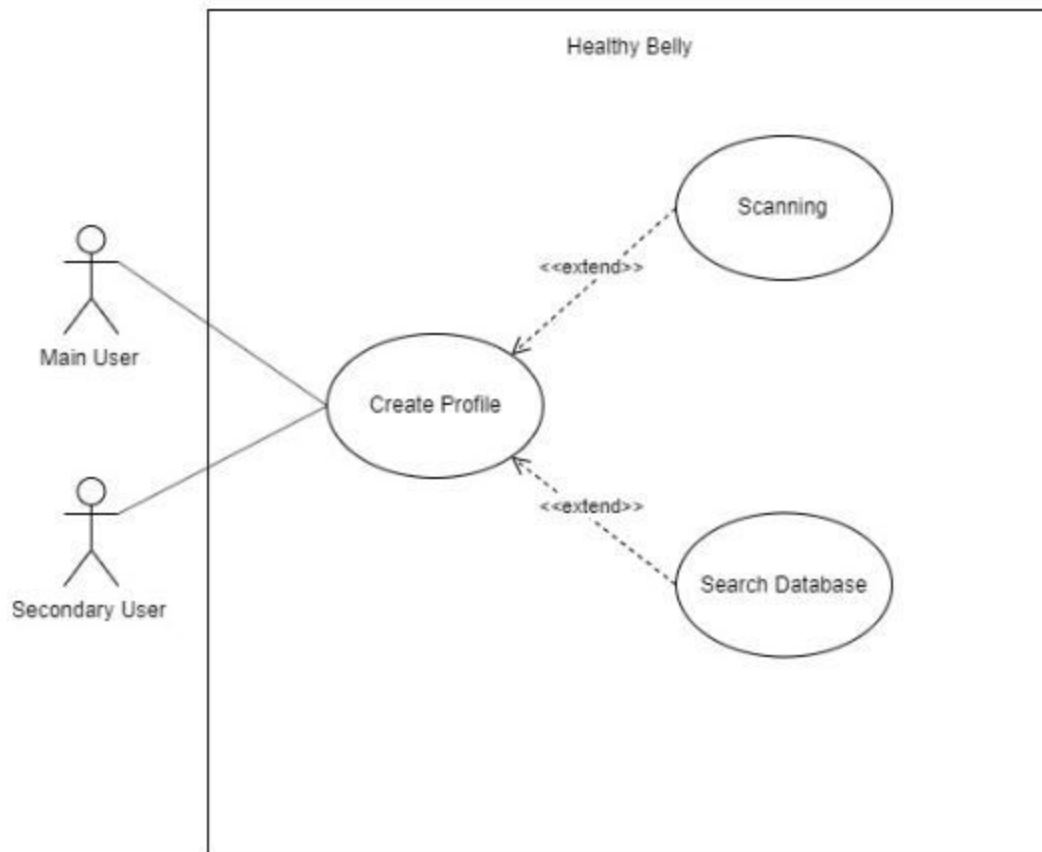
Scanner:



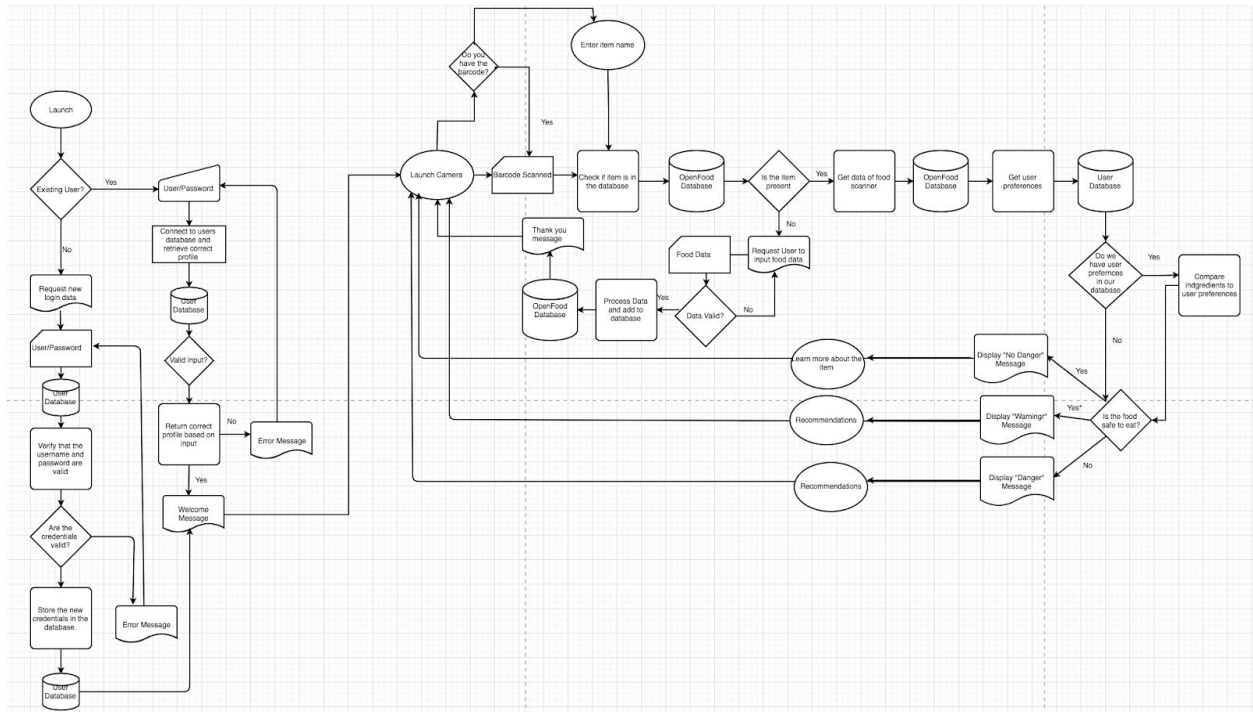
User Profile Change:



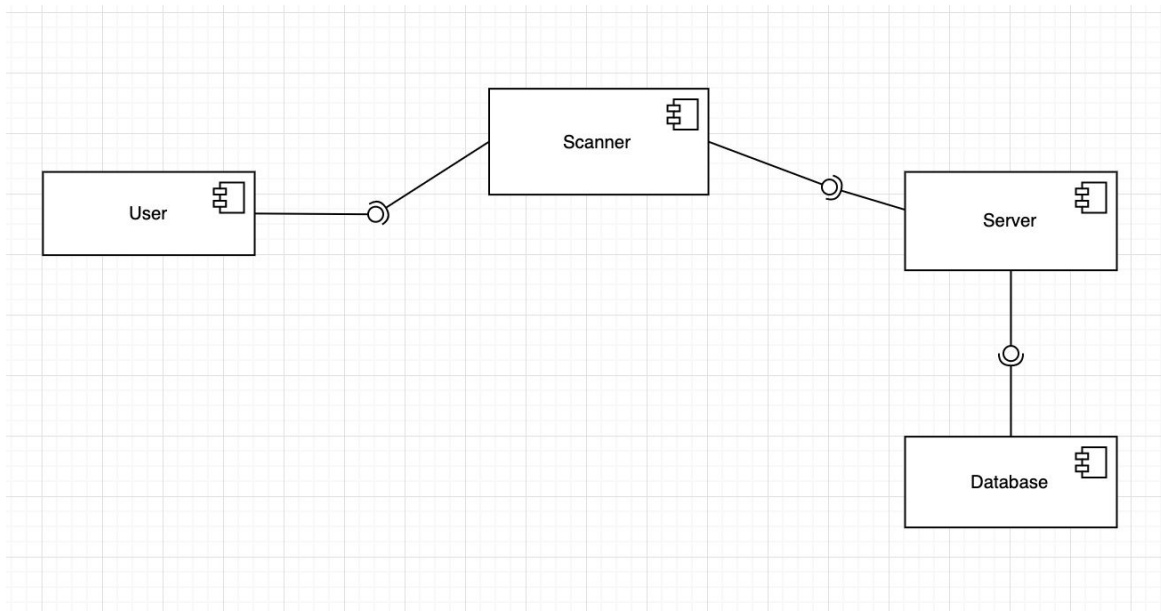
Use Case Diagram



Activity Diagram

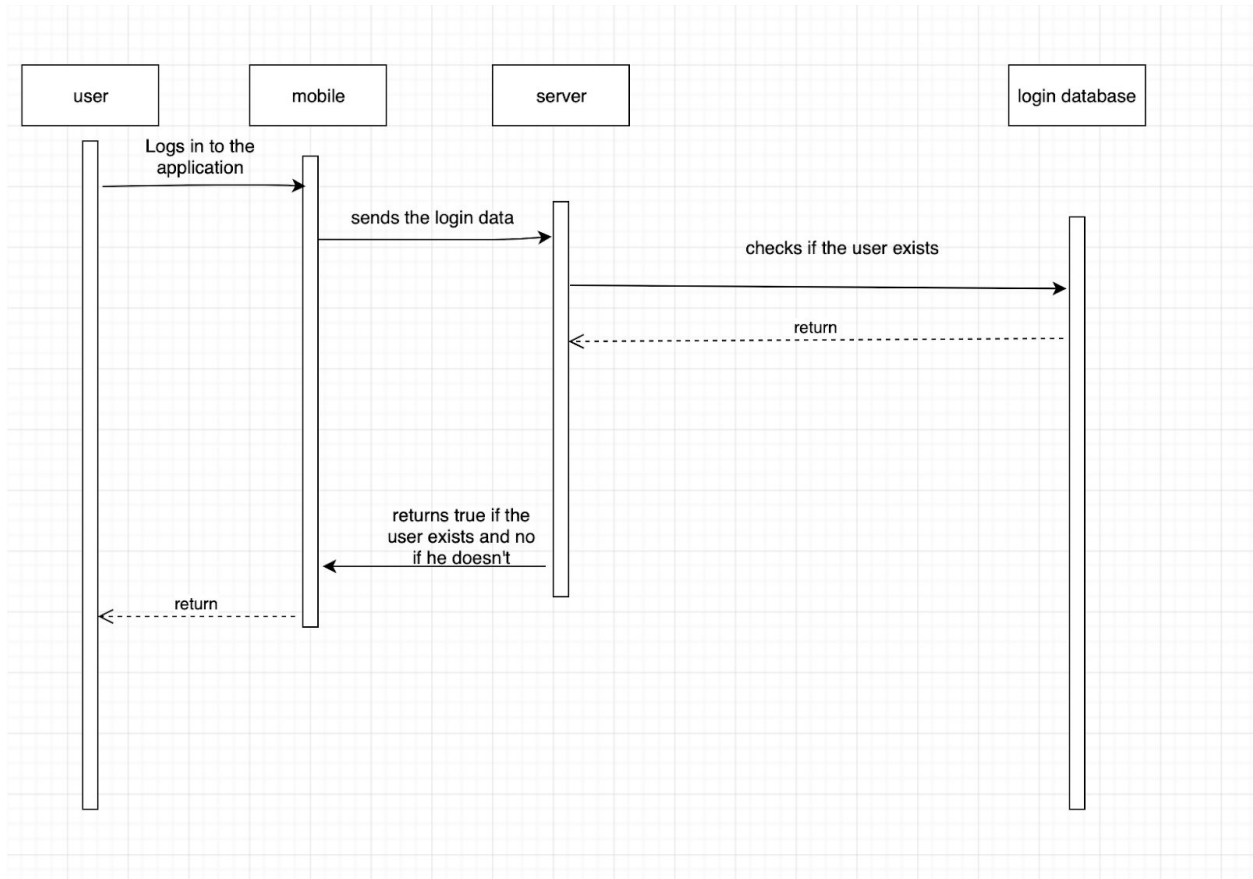


Component Diagram:

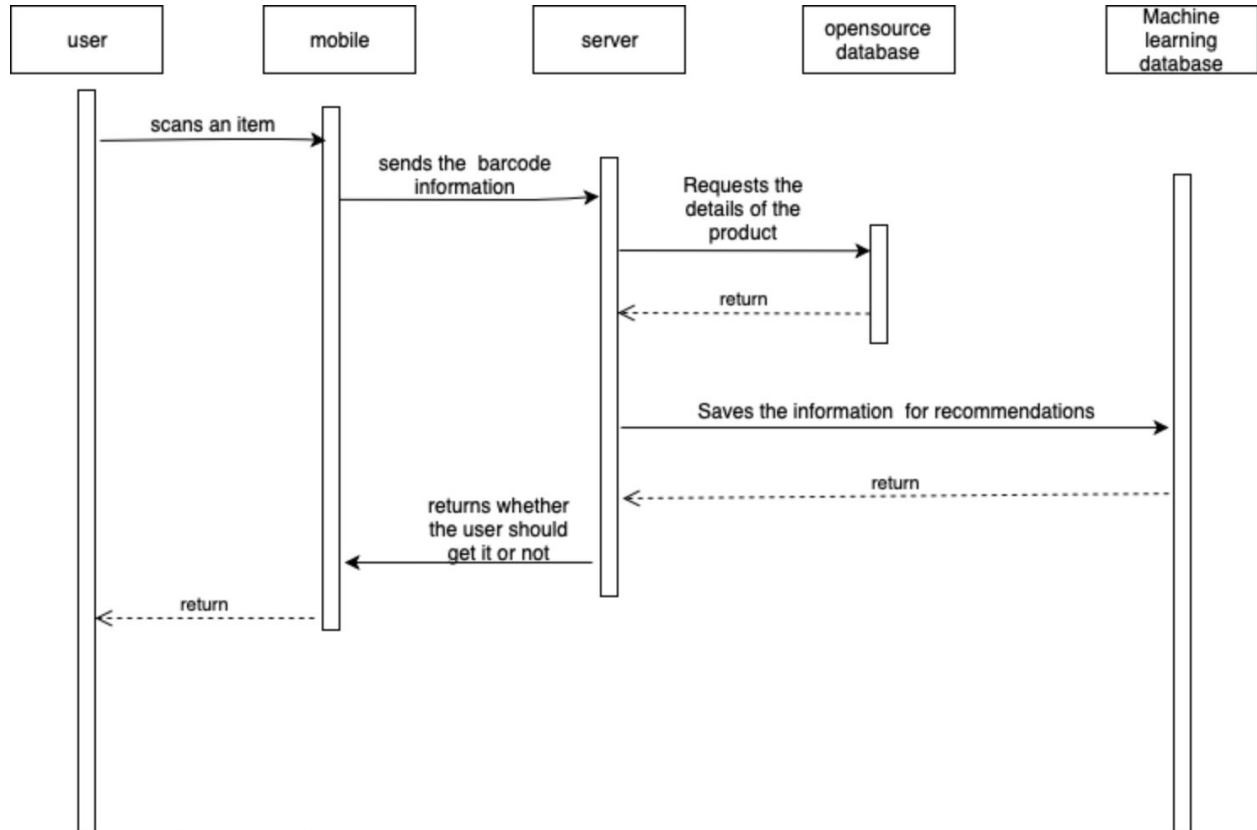


Sequence diagrams

User login sequence diagrams:



Scanning sequence diagrams:



Tradeoff Analysis:

Major Architecture Patterns:

After carefully looking at some of the major software architectural patterns in software engineering, we came to the conclusion that these two listed below are the best ones for the purposes of our Application.

Layered Architecture Pattern:

This pattern breaks down a program into groups of subtasks, and each layer provides services to the next higher layer. Most common found layers are UI Layer, Service Layer, Domain Layer, and Persistence Layer.

Pros	Cons
Reusability- Being able to reuse components if needed for a similar Application or for future such as exporting to different O.S.'s	Performance- Extra overhead when calling a component directly
Independent- Components of the Application can be independently deployed and maintained. Also allows to test components individually and independently	Time- Typically applications take longer to program when layering configurations are off since combining the said separated layers could result in mismatches
Secure- Layered architecture allows flexibility to configure different layers of security to different components.	Complexity- great for big projects, but can make simple projects harder
Flexibility- Allows team to work on different parts of the Application since every layer is separated from one another	

Client-Server Pattern:

C-S Pattern consists of two parties; a server and multiple clients. Servers will provide services to many clients. Client requests services from server and server handles the said requests.

Pros	Cons
Centralization- Servers help administer the whole setup and the access rights and resource allocation is done by servers	Costs- Client-Server network is very expensive
Management- All files are stored in the same place. Also allows for easy backup. And since the data is stored on the servers, it reduces the amount of data replication	People- Need PROFESSIONAL IT people to maintain the servers and other details of the network
Security- Rules and access rights are defined at the time of server creation	Failure- Server failure leads to whole network failure

Performance- Does not slow down with heavy usage	Congestion- Too many requests from clients may lead to overload, which can lead to breaking down of servers
--	---

The Architectural pattern we considered implementing but chose not to is listed below:

Model-View-Controller Pattern

This pattern divides an interactive application into 3 parts as: Model (contains the core functionality and data), View (displays the information to the user), and Controller (handles the input from the user).

Pros	Cons
Clean design- Good module design keeps classes separate and clean	Complexity- VERY complex if the entire idea is not completely grasped
Views- Multiple views of the same model	Workload- workload is unevenly distributed between everyone
SOLID- it satisfies the Single Responsibility Principle of SOLID	Outdated- Modern user interface tools are hard to use with MVC
	Development issues- Each and Every team needs to be working at the same speed.

Component Choices

Components are something we were unable to analyze and complete a Trade-off analysis based on our progress. We have not yet made enough progress for us to properly research which components would be best for our Application.

Language Choices:

Since we are making the Application for Android O.S. we have two options, either Java or Kotlin. In future iterations we will move towards iOS but for our first iteration of the Application it will be for Android. Out of the two options we chose Java based on our Analysis done below between the two languages

Java

Pros	Cons
Popular- Every member on the team is familiar with the coding language	Performance- It is more memory consuming than Kotlin
Cross-Platform- The ability to use Java on multiple different platforms.	Security- Older versions of Java have a reputation of being Vulnerable
Object Oriented- Language is focused on creating objects, manipulating said objects, and making objects work together	

Kotlin

Pros	Cons
Interoperable- It is possible to switch back and forth between Kotlin and Java since it is consistent with Java and all related tools and frameworks	Limited- Used only for Android unlike Java.
Future- Android development seems to be moving towards Kotlin, as seen by Google who are treating it as a preferred Programming language for Android	Learning- Although it is an easy language to learn, there is a learning period needed for using Kotlin, since every member needs to spend time learning about the language
Simplified- Coding in Kotlin is shorter compared to Java codes	Performance- It is noted to be slower than Java in compilation speed.

Framework Choices

Two of the choices for frameworks we considered are analyzed below:

Node.js

Pros	Cons
Asynchronous event driven IO helps concurrent request handling.	Every time using a callback end up with tons of nested callbacks.
Share the same piece of code with both server and client side.	Node.js is not suited for CPU-intensive tasks. It is suited for I/O stuff only (AKA us).
Active and vibrant community, with lots of code shared via GitHub	Learning curve for those who are not familiar with JavaScript
Node packaged modules has already become huge, and still growing.	Node.js doesn't provide scalability.

FireBase

Pros	Cons
Javascript Object Notation storage means no barrier between data and objects	Limited querying and indexing
Minimal setup	No Aggregation
Easy access to data, files, auth, and more	Cannot query or list users/stored files
Secure, AND ITS GOOGLE	

Databases Choices

The three databases we looked into for our app are SQLite, OrmLite and Realm. Based on the Trade-off analysis, we have decided to work with SQLite.

SQLite

Pros	Cons
Lightweight so easy to use it as an embedded software	Used to handle low to medium traffic HTTP requests

Great performance only loads data when needed	Database is restricted to 2GB
Reliable: constantly updates content so no work lost if app crashes	

OrmLite

Pros	Cons
Can use any programming language	Slow compared to other databases
Adding new data and updating data is really easy since the databases treats them as the same	Some queries are limited forcing you to use raw SQL
Will pull the data automatically for you	Requires lots of optimizing to make database smooth

Realm

Pros	Cons
Simple: fewer lines of codes for similar methods in other databases.	Does not work well for large scale projects
Faster than other databases	Takes up more memory than other databases

Server vs Serverless Choices

Based on our Analysis, we find that choosing either server/serverless Architecture would work best for our Application. We will decide which would be better for our App as we continue with the Development of the Application. Here are the two trade-off Analysis we conducted:

Servers

Pros	Cons
------	------

Reliability- Proper servers would maintain redundant hardware for critical internal devices in case a component fails. One server failing would not crash the entire program	Update- Requires constant updating and maintaining
Collaboration- Servers allow us to divide use-cases into multiple different servers so each server handles one specific task	Cost- Server based software as well as server based network will cost money as the company increases. Can't run on student options forever

Serverless

Pros	Cons
Cost- Pay for what you use not for any hardware or services that you are not utilizing	Commitment- once you commit to a service provider, you are stuck with them till the contract ends.
User- since we do not have to worry about servers, devs can focus on developing and improving Customer-facing elements	Complexity- learning curve is huge if people have not used a serverless architecture. Will require extra time to go into organizing the functions

Front-end Framework Choices

There are a lot of different front-end frameworks we can use for our app. The three we looked into are Android studio, Corona SDK, and Appcelerator Titanium.

Android Studio

Pros	Cons
Supported by google	High memory use
Easy to integrate with google services	Android development only
Allows GPU tracking	Can be slow sometimes
Supported by many platforms	

Corona SDK

Pros	Cons
Uses C#	Does Not support 3rd party libraries
Easy to create prototypes quickly	Not suitable for apps with heavy graphics
Can create for multiple platforms at the same time	Larger app size

Appcelerator Titanium

Pros	Cons
Easy to create prototypes quickly	Web oriented
Can create for multiple platforms at the same time	Uses javascript
	Not smooth compared to other front end frameworks
	Limited compared to other front end frameworks.