UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and Information Technology

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS301 - Software Engineering

Coviduous

Architectural Requirements Document

September 18, 2021

Team Name: CAPSlock
Team Members: * - *indicates team leader*

Name	Surname	Student Number
Njabulo*	Skosana*	u18089102*
Rudolf	van Graan	u16040865
Clementime	Mashile	u18139508
Peter	Okumbe	u18052640
Chaoane	Malakoane	u18202374

Contents

1 Architectural Design Strategy

For this system, we have chosen to follow the strategy of designing to fit architecturally significant requirements. The six quality requirements we have chosen as our most important ones have defined our choice of architectural styles and patterns. For example, our model-view-controller architecture satisfies the requirement of usability by providing the user with a defined point through which they can access the system.
In terms of the requirements which are less significant, we will still take them into account, but only after our system has satisfied our most significant requirements.
We will also try to build the system in a way which satisfies as many significant requirements at once as possible, rather than focusing on the requirements one at a time.

2 Architectural Styles

2.1 Data-centered architectural style

- In data-centered architecture, the data is centralized and accessed frequently by other components, which modify data. The main purpose of this style is to achieve integrality of data.
- Data-centered architecture consists of different components that communicate through shared data repositories. The components access a shared data structure and are relatively independent, in that, they interact only through the one data store.
- The data store in the file or database is occupied at the center of the architecture. Stored data is accessed and updated continuously by the components of the system that interact with it.
- Our system will be storing user data, office floorplan data, and other system data that will be continuously accessed and updated.
- Data-centered architecture helps integrity.
- All the processes are independently executed by the client components.

2.2 REST architectural style

- Representational state transfer (REST) is a software architectural style that was created to guide the design and development of the architecture for the World Wide Web. It is a set of constraints and rules that creates a software architectural style, which can be used for building distributed applications.
- Defines an architectural style for building an application program interface (API) that uses the primary HTTP protocol to send requests to access and use data. That data can be used to GET, PUT, POST and DELETE data types, which refers to the reading, updating, creating and deleting of operations concerning resources. The identification of each resource is performed by its unique Uniform Resource Identifier (URI).

- REST describes simple interfaces that transmit data over a standardized interface such as HTTP and HTTPS. The consumer will access REST resources via a URI using HTTP methods. After the request, a representation of the requested resource is returned. The representation of any resource is, in general, a document that reflects the current or intended state of the requested resource.

- The REST architectural style describes six constraints to define how data is transferred between components, our systems focuses on two of these constraints:

  - Client-server architecture: The REST style separates clients from a server. In short, whenever it is necessary to replace either the server or client side, things should flow naturally since there is no coupling between them. The client side should not care about data storage and the server side should not care about the interface at all.
  - A layered system: Each layer must work independently and interact only with the layers directly connected to it. This strategy allows passing the request without bypassing other layers.

2.3 Layered architectural style

- Modules or components with similar functionalities are organized into horizontal layers, therefore, each layer performs a specific role within the application (presentation logic or business logic).

- Applied with most n-tier architectural design patterns where each tier represents a layer, the most common being the 3-tier architectural pattern consisting of a presentation tier, application tier and database tier.

- The layered architecture style does not define how many layers are in the application and developers can have as many layers as required when developing the application.

- Each layer is independent of the other layers, thereby having little or no knowledge of the inner workings of other layers in the architecture.

- Advantages: Security, Usability, and Reliability.

- Disadvantages: Scalability

2.4 Component-based architectural style

- Allows for building applications with independent, modular, and reusable pieces known as the components.

- Each component will be developed, tested, scaled, and maintained independently. The REST architectural style is used to employ a REST API to forward requests to the appropriate services on the back-end.

- The system programming logic will be divided into 7 components in the service layer, representing each service as a subsystem component consisting of the appropriate controllers to perform the required actions.
- Also used in the MVC architecture to separate and divide the presentation tier system into 3 independent components (Model, View, Controller) that communicate with each other.
- Advantages: High cohesion, Adaptability, Maintainability, Faster development process

3 Architectural Quality Requirements

3.1 Security

- The Coviduous application should separate admin rights and user rights to prevent unauthorized users from making unnecessary and unintentional changes to the functionality of the system.
- Users should be authorized before they can gain access to the system and its services.
- Each user has a username and password which will be used to authenticate and give them access to the system. For example, users that are not employees of the office will not have access to the system. Use of the MVC architecture makes this easier to implement as the user has to go through the view to access the system's data.
- The system is split into user types so role-based access control is used. This means that a user's access is limited to what they are authorized to do as a specific user type. For example, normal users can only view announcements, they cannot create or delete an announcement, only admin users can.
- Hashing and salting algorithms must be applied to all sensitive data, such as passwords, before storage. This will mainly be done using an external authentication service such as Firebase authentication where passwords are automatically encrypted and not stored to the database.
- It is important that the system is secure so that users' data is protected and inaccessible to unauthorised users. The use of JWT tokens will help enforce security for the sending of API requests. Unauthorized users will be restricted from sending API requests.

3.2 Scalability

- The system database must be able to cater for organisations of varying user base sizes.
- The system must be able to withstand a growing user base, allowing a number of new user requests per day and this should not have a noticeable effect on daily operations.
- As the amount of end users increases, load balancing can be implemented. The client-server architecture makes this implementation easier. Since the client has to make http requests to the server in order to retrieve information or update information, it is clear to see that a load

balancer can be placed in the server layer and the load balancer would direct requests to the least busy request handling server components. Use of an API gateway service in a Backend as a Service (BaaS) application platform (such as AWS cloud services or google Firebase) would help to automatically manage and scale the load balancing of requests.

- By use of cloud storage for the database, it allows for the ever expanding size of the database which stores the data of users that use the application. The application should also cache data needed later on, which would lead to less requests sent to the database, allowing more concurrent users being able to use the application as there is a lessened load of requests on the server.

- The system also uses a layered architecture. Multiple instances of each layer can be run when necessary in order to handle large user base sizes.

- **Throughput**: X = N/R, where X is the throughput (measured in transactions per second, or TPS), N is the number of users accessing the system, and R is the amount of time on average that they spend using the app. The app should handle initial scaling of about 200 user requests. The average amount of time they could spend is 10 milliseconds per transaction, so the throughput would be 200/0.1 = 2000 TPS.

3.3 Integrability

- The system is broken into different subsystem components which have their own specialised functionality. Subsystems need to be designed to easily integrate with each other and other external system control modules.

- By use of the component based architectural style, components of the system can be easily removed or added that integrate with each other or external systems such as the South Africa COVID-19 government API to retrieve data relating to the current COVID-19 alert level allowed mass percentage.

- The component-based design of our system allows for every current or new subsystem component to be tested independently for correct functionality before being integrated into the system and thus also allows for easily picking up in which components errors may reside and working on those specific components to resolve the error instead of traversing through the entire system.

3.4 Testability

- The Coviduous application will have to undergo various testing methods to ensure that all processes and tasks perform correctly and to the best of it's ability. Considering security being a core quality requirement, correct testing and error handling management such as null input field checks, throwing of exceptions for invalid operations, and proper database management storage checks will need to be enforced to strengthen validity and security of data.

- Testing of the system should be done weekly to solidify continuous correct implementation and integration of operations from both the front-end and back-end, this will comprise of automated test-driven development unit and integration tests for each end of the stack.

- Monthly integration testing with the application hosted server will also be enforced to ensure continuous availability and reliability of the application to all users, especially during peak working hours of the day for workers working in an office space. All tests should verify that a service is provided if all pre-conditions are met, and that post-conditions are true once the service has been provided.

3.5 Reliability

- Reliability is measured as the probability of a system not entering a fail state for a set time under specified conditions. The system needs to be reliable to gain user trust and confidence in the system. The system must present the correct information to users and provide a high uptime for users.

- The system should not be affected or shut down when any components are swapped in or out, it should continue to operate as usual.

- For every user interaction, the system must produce appropriate and correct results, responses, feedback, and error messages 100% of the time where necessary.

- The system should be available 95% of the time. The 5% unavailable down time percentage caters for system maintenance, data backups, and recovery from system failure.

- The MVC architectural pattern enables the use of messaging queues which provide a guarantee that all requests will be handled even when the Controller is too busy to process new requests.

- In the event of the system being down, users are informed either by dialogues on the application or through email. However, use of a serverless system where our server is managed and hosted by a backend cloud computing service provider would help to ensure that the system stays up and running as much as possible.

3.6 Usability

Usability can be described as the capacity of a system to provide a condition for its users to perform the tasks safely, effectively, and efficiently while enjoying the experience.

- The system should be easy for intended users to use by making use of readable fonts, clearly labelled components, appropriate hints and error messages to improve user experience. Error messages must describe the issue and, where relevant, indicate rectifying steps that need to be taken.

- Users should be able to master the use of the main system functionality within 24 hours.

- Implementing useful tools such as a date and time picker, drop down lists, and suggested entries in the UI interface will help to ensure validity of user's input.
- The possibility of user error must be minimised, with the monthly average of user errors that result from a lack of understanding of the system being 10 errors per individual.
- Users must be able to navigate from one point to another easily. It must take a maximum of 3 - 5 clicks or taps before the user is able to get to the desired page or view from the current page.
- A 90% average user base satisfaction rate should be aimed for by developers.

The following requirements are crucial:

- Employees with a low computer literacy rate can use the system without documentation.
- Users should not find any aspect of the system cumbersome, non-initiative or frustrating.
- The colours used in Coviduous should not be evasive and distracting.
- Our user interfaces should not be cluttered and hard to use.

By using flutter and making sure we follow the mobile/website UX/UI design guidelines, this will ensure that innovative and novel approaches are used for this system. There should be usability reasons if we decide to deviate from any user interface design aspects from accepted norms and standards.

4 Architectural Design and Patterns

- Design Issue: A core architectural design pattern must be chosen that best suits quality requirements of Coviduous.
- Decision: Our project stakeholders have quality requirement preferences. The design of the system will be focused on presenting the system with regards to our quality requirements.

1. **Client-Server Architecture**:

- At the highest level of granularity, we have our client-server architecture of which consists of the two layers: Client and Server layer.
- Client layer: The client side of the application where users interact with the user interface, view data, and send requests to the server.
- Server layer: The server side of the application which users never directly interact with. It consists of all the business logic in the system. It will process incoming requests, handle user input and interactions, and execute appropriate application logic. A user will only be able to interact with the server through sending requests to it. The server will perform business logic operations and send data back to the client.
- The client communicates with the server using requests and responses. There may be more than one client, more than one server, or more than one of both, but the process remains the same due to how scalable each component can be made.
- Advantages: Security, Scalability, Usability, Reliability, Testability
- Disadvantages: Performance

2. **3-tier Architecture**:

- The 3-tier architecture is used to keep the separation of concerns, where the client, server and database are developed and maintained separately.
- Client tier: The client side of the application where users interact with the user interface, view data, and send requests to the server.
- Server tier: The server side of the application which users never directly interact with. It consists of all the business logic in the system. It will process incoming requests, handle user input and interactions, and execute appropriate application logic. A user will only be able to interact with the server through sending requests to it. The server will perform business logic operations and send data back to the client.
- Database tier: The database tier comprises of a centralized database through which data objects can be retrieved and model states can be stored. Components that contain business logic and database operations interact with the centralized data store contained in the database tier.
- Advantages: Security, Scalability, Usability, Reliability, Testability

- Disadvantages: Performance

3. **Model View Controller**:

   - Within our presentation (client) tier, we utilize an MVC architecture for a well-organized, structured workflow in the client tier of our system.
   - Model: The model processes commands and performs calculations.
   - View: The user interface.
   - Controller: The controller controls flow of information between model and view.
   - The MVC architecture divides the system into three components: the model, view and controller. The communication between components is through requests and response objects and flow is controlled by the controller. Communication flow is triangular.
   - The model is responsible for managing application data. It responds to the request from view and to the instructions from the controller to update itself.
   - The controller handles the business logic to be performed on the data models. They will help process incoming requests, handle user input and interactions, and then modify the state of the data models.
   - Advantages: Security, Scalability, Usablity, Intergrability, Testability, and Reliability.
   - Disadvantages: Flexibility, Performance

4. **2-tier architecture**

   - Within our application (server) tier, we utilize a 2-tier layered architecture for a well-organized, structured workflow in the application server.
   - API Gateway Layer: Acts as an entry point and reserves proxy for all the services. The external authentication service resides in this layer for user authentication before sent requests are processed.
   - Service Layer: Each service represents serverless Functions as a Service (FaaS) that run in independent containers and are event-triggered. These functions are fully managed and hosted by 3rd party BaaS (Backend as a Service) providers.
   - Services are divided into subsystems as components that contain their respective serverless functions.
   - Calls and requests sent from the client land in the API gateway which then decides which service to redirect the request to, based on routing rules.
   - Advantages: Security, Usability, and Reliability.
   - Disadvantages: Scalability

Resolution: Our system uses a combination of a client-server architecture, n-tier architecture, model view controller (MVC), layered architectural style, and component-based architectural style. The Client-Server architecture forms the system's main structure at the highest level. Following our client-server architecture, we have our 3-tier architecture consisting of the presentation tier, application tier, and database tier. The presentation layer communicates with the server layer through REST API calls. The API then queries our database, which is our database layer.
Within the client (presentation) tier is the MVC architecture, where the user views and controls the model of the application; and within the server (application) tier is a 2-tier architecture where the server's business logic resides in the service layer components that receive requests through the API gateway layer and sends responses back to the client tier. A component based structure is used to ensure separation of concerns between subsystem components and that each subsystem component's business logic functionality does not interfere with that of another subsystem.

4.1 Architectural Design Diagram

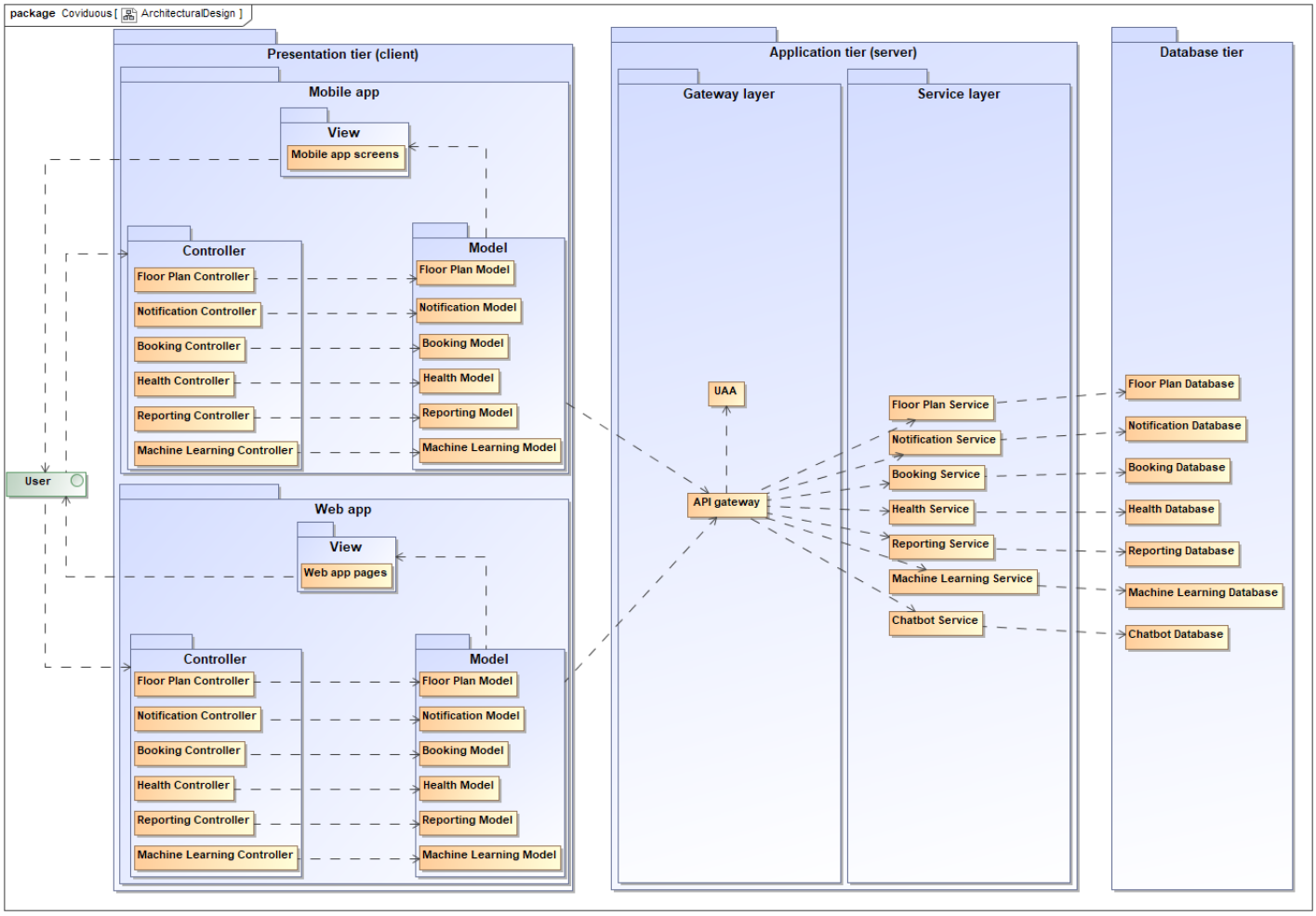

Figure 1: Architectural design diagram for Coviduous

5 Architectural Constraints

5.1 Operating System

The system's mobile application will be developed for an Android mobile device, hence will be limited to the Android operating system.

5.2 Hardware

The system should work on a mobile device - android smartphones for the mobile application, and any devices that can connect to the internet as the dashboard functionality will be hosted on a website.

5.3 Cost of Development

We are complied to use mostly open source frameworks and software as no fixed budget was provided. A lot of open-source frameworks come with a free tier usage for a certain time period, like Firebase, which is used to host our firestore database and serverless cloud functions. Once this time period is over the client will have to pay to continue usage.

5.4 Testing

The selected serverless architecture is used and extensively tested over multiple languages and frameworks. It might be difficult for our group members as we may be unfamiliar with some languages but it is beneficial to us in our growth of knowledge since it is a learning curve.

6 Technological Requirements

Coviduous is an open-source application, which means that it must employ use of any proprietary technologies. Technologies have been selected with this specification in mind and are, thus, all open-source. Furthermore, Coviduous is intended to be a platform-independent system in order to reach a wide user base. Selected technologies have been selected to comply with these requirements.

For each component layer of our main 3-tier architectural design pattern (Presentation tier, Application tier, Database tier) of our system, various technologies were considered. We compiled a list of technology choices per component based on our architectural strategy and design, and arrived at final decisions that would best suit our architectural quality requirements, design, and constraints. Each component with their chosen technology choices and final decisions are listed below:

Presentation Tier

- **Angular** - A web framework developed by Google. Angular can be viewed as a complete framework as all the required modules such as router management, state management, UI library is provided by the framework. Angular is a Model-View-Controller (MVC) focused web application framework, implying the all three components are located in the web browser. Angular allows for efficient and rapid development of Single Page Applications (SPA) for both the mobile and web applications. Since Angular supports HTML and CSS which are well-known scripting and styling languages, it makes it easier to learn, and also provides functionality for progressive web apps.

- **React** - A JavaScript library developed by Facebook for the building of Single Page Applications (SPA) for both web and mobile. React is only concerned with the rendering of components to the Document Object Model (DOM) in a browser; as such other community libraries must be utilized to assist with in application state management and routing, Redux and the React Router being examples of these. One can implement various MV* (MVC, MVVM, MVP, MVA, MVI) architectures using React, depending on how and what libraries are used to support the development.

- **Flutter** - Open source software development kit (SDK) developed by Google which can be used to build applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia and web all from one single codebase. Flutter was released in 2015 at the Dart developer conference. Flutter uses the Dart programming language. Depending on the operating system targeted, Dart may be targeted to run inside the Dart Virtual Machine (DVM). It supports both Just-In-Time (JIT) and Ahead-Of-Time (AOT) compilation strategies. For mobile development the Dart framework contains widgets targeting both the Android's Material Design specification and Apple's iOS Human Interface guidelines.

Our final choice was Flutter. Although it is a relatively new technology, has a limited set of iOS feature support, and has massive file sizes, Flutter has no heap of completely incompatible design patterns, it supports a variety of plug-ins, cross-platform design from a single code-base, a high

performance of created applications, and has a powerful community support. It allows for faster app development and cost saving, given the one code-base functionality. This also allows for faster testing and debugging. Flutter also uses the Dart programming language which is similar to Java in syntax and operations that in turn, benefits the team in easily learning and understanding the language.

Application Tier

- **ASP.NET Core** - A free and open-source framework successor to ASP.NET, developed by Microsoft. It is a modular framework that runs on both the full .NET Framework, on Windows, and the cross-platform .NET Core. However, ASP.NET Core version 3 works only on .NET Core dropping support of .NET Framework. The framework is a complete rewrite that unites the previously separate ASP.NET MVC and ASP.NET Web API into a single programming model. It offers quick response times and is reliable in processing and handling of requests which is necessary in conforming to the reliability requirement of our system.

- **Node.js** - Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project. Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. This allows Node.js to be very performant. A Node.js app runs in a single process, without creating a new thread for every request. Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behavior the exception rather than the norm.

- **Express.js** - Express is a minimal and flexible backend Node.js web application framework that provides a robust set of features for web and mobile applications. It is designed for building web applications and APIs. It provides various features that make web application development fast and easy which could otherwise take more time using only Node.js. Express allows to set up middlewares to respond to HTTP Requests, defines a routing table which is used to perform different actions based on HTTP Methods and URL and allows creation of a REST API server structure with simple configuration and customization. It has been called the de facto standard server framework for Node.js.

- **Firebase** - Firebase is a Backend-as-a-Service (BaaS) app development platform that provides hosted backend services such as a real-time database, cloud storage, authentication, crash reporting, machine learning, remote configuration, and hosting for your static files. The products have backend components that are fully maintained and operated by Google. Client SDKs provided by Firebase interact with these backend services directly, with no need to establish any middleware between your app and the service. Using Cloud Functions, their serverless compute product, you can execute hosted backend code that responds to data changes in your database. It also has support for Flutter.

- **Cloud Functions** - Cloud Functions for Firebase is a serverless framework that lets you automatically run backend code in response to events triggered by Firebase features and HTTPS requests. JavaScript or TypeScript code is stored in Google's cloud and runs in a managed

environment. There's no need to manage and scale your own servers. The functions you write can respond to events generated by various Firebase and Google Cloud features, from Firebase Authentication triggers to Cloud Storage Triggers. Allows integration across Firebase features using the Admin SDK. Cloud Functions minimizes boilerplate code, making it easier to use Firebase and Google Cloud inside your function.

- **Microsoft Azure** - Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through Microsoft-managed data centers. It provides software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) and supports many different programming languages, tools, and frameworks, including both Microsoft-specific and third-party software and systems. Storage Services provides REST and SDK APIs for storing and accessing data on the cloud.

- **Azure Functions** - Azure Functions is a serverless solution that allows you to write less code, maintain less infrastructure, and save on costs. Instead of worrying about deploying and maintaining servers, the cloud infrastructure provides all the up-to-date resources needed to keep your applications running. Building APIs are implemented by a creating endpoints for your web applications using HTTP triggers.

- **AWS Lambda** - AWS Lambda is a serverless computer service that runs your code in response to events and automatically manages the underlying compute resources for you. You can use AWS Lambda to extend other AWS services with custom logic, or create your own back-end services that operate at AWS scale, performance, and security. Lambda runs your code on high-availability compute infrastructure and performs all the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code and security patch deployment, and code monitoring and logging. It natively supports Java, Go, PowerShell, Node.js, C#, Python, and Ruby code, and provides a Runtime API which allows you to use any additional programming languages to author your functions.

We concluded on using Firebase as a serverless application tier architecture using Firebase Cloud Functions as our FaaS (Functions as a Service) that will contain our server-side business logic. We will use Node.js as our runtime language with the Express.js framework to code and run our cloud functions. Node.js operates on a single-thread, using non-blocking I/O calls, which allows it to shine in building fast, scalable network applications, given it's capability of handling a huge number of simultaneous connections with high throughput, which equates to high scalability necessary for our system in handling the vast amount of a growing user base in the work space. Node.js also supports a wide variety of useable libraries for frameworks and tools to communicate with Firebase such as Express.js. Our team members are also familiar with the node.js javascript syntax which would help speed development. Firebase does not bring about much complexity in set-up and interfaces well with our presentation tier choice of Flutter, as both have been developed by Google.
As a serverless framework, there is no need to manage and scale our own servers as they are managed by the Firebase hosted backend services. JavaScript or TypeScript code is deployed to the servers with one command from the command line. After that, Firebase automatically scales up

computing resources to match the usage patterns of users. You never worry about credentials, server configuration, provisioning new servers, or decommissioning old ones. Firebase includes a variety of services, such as authentication, a NoSQL database, a command line interface (CLI), and templates for automated emails and communication with users.

Database Tier

- **PostgreSQL** - PostgreSQL is an advanced, enterprise class open source relational database that supports both SQL (relational) and JSON (non-relational) querying. It is a highly stable database management system, backed by more than 20 years of community development which has contributed to its high levels of resilience, integrity, and correctness. PostgreSQL is used as the primary data store or data warehouse for many web, mobile, geospatial, and analytics applications. PostgreSQL has a rich history for support of advanced data types, and supports a level of performance optimization that is common across its commercial database counterparts, like Oracle and SQL Server. AWS supports PostgreSQL through a fully managed database service with Amazon Relational Database Service (RDS).

- **MongoDB** - MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. The document data model is a powerful way to store and retrieve data that allows developers to move fast. MongoDB's horizontal, scale-out architecture can support huge volumes of both data and traffic. MongoDB, unfortunately, does not support transactions. So updating more than one document or collection per user request, poses for an overhead. Storage may lead to corrupted data, as there is no ACID guarantee and rollbacks have to be handled by the application.

- **DynamoDB** - Amazon DynamoDB is a fully managed proprietary NoSQL database service that supports key–value and document data structures and is offered by Amazon.com as part of the Amazon Web Services portfolio. DynamoDB exposes a similar data model to and derives its name from Dynamo, but has a different underlying implementation. Dynamo had a multi-leader design requiring the client to resolve version conflicts and DynamoDB uses synchronous replication across multiple data centers for high durability and availability.

- **Cloud Firestore** - Cloud Firestore is a flexible, scalable NoSQL database for mobile, web, and server development from Firebase and Google Cloud. Like Firebase Realtime Database, it keeps your data in sync across client apps through realtime listeners and offers offline support for mobile and web so you can build responsive apps that work regardless of network latency or Internet connectivity. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions.

Our final choice with regards to our data model settled on Cloud Firestore. The Cloud Firestore data model supports flexible, hierarchical data structures. It is included within Firebase as a fast, user friendly NoSQL database that allows for easy scaling and storage of data through the use of collections. Cloud Firestore stores data in documents, which are organized into those collections. Documents can contain complex nested objects in addition to subcollections. Cloud Firestore is also

available in native Node.js, Java, Python, Unity, C++ and Go SDKs, in addition to REST and RPC APIs to easily perform data storage operations in the supported programming languages using the respective SDKs. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions.