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

August 1, 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

- The data store in the file or database is occupying at the center of the architecture.
- Stored data is accessed continuously by the other components like an update, delete, add, modify from the data store.
- Our system will be storing Floor-plans, User data and calendar 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.
- An architectural style for an application program interface (API) that uses HTTP 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 REST architectural style describes six constraints, But our systems focuses on two:
  - 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

3 Architectural Quality Requirements

3.1 Security

- The main core quality requirement of the system. 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. Sensitive user data like passwords will make use of a hashing function for extra protection. This will ensure that even if the hashed password is leaked, it will remain useless to an attacker. Furthermore, a user will be prompted to create a strong password including special characters to guarantee a unique hash so that it can not be reversed by using a rainbow table.

- Confidential data such as CompanyIDs and AdminIDs will be required when trying to attempt performing administrative tasks (e.g. deletion of user accounts) or registering as a User (Employee / Visitor) for a specific company. This will revoke power from random, uninvited users in trying to perform these tasks without having the correct credentials.

- Each registered User will require confirmation from an Admin of a specific company when trying to update their company user details in order to prevent users from easily updating their CompanyID to another without having being employed at or invited to visit that particular company. A requested RFC access code with a 24hr expiry timespan will also be required on entry into the office for each user to further secure access into the office and prevent unauthorized and COVID-19 contracted users that will not be granted an access code, from entering the office space.

3.2 Scalability

- **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. I think the app should scale to about 10,000 users at first. The average amount of time they could spend is 10 milliseconds per transaction, so the throughput would be 10,000/0.1 = 100,000 TPS.

- **Resource usage**: How much CPU, memory, disk, bandwidth, etc. is being used. I think that about 25% of the CPU should be used at once, since Coviduous is a simple booking app. Disk will only store floorplan data, bookings, and minimal user data, so at most I see about 50% of memory and disk would be used. Bandwidth usage would slow down as users increase, possibly taking the 10 milliseconds per transaction figure up to 1000 milliseconds per transaction, if the app reaches 1,000,000 concurrent users.

3.3 Integrability

- The system is broken into different subsystems which have their own specialised functionality. Subsystems need to be designed to easily integrate with each other and system control modules.
- Coviduous needs to be able to integrate with with external systems. E.g. The system should interface with the South African government COVID-19 API to retrieve the current alert level capacity percentage.
- The system will interface with more than one external system for example the AWS dynamo db database server and also interface with the South African government API.

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, Exception throws 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 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 must be available for 24 hours because if it happens the Admin wants to check the report after working hours. The system must ensure that in the case of system failure, data integrity is maintained and critical functions such as the number of workers allowed in the office premises.

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 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. **Model View Controller**:
   - 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.
   - Advantages: Security, Scalability, Usablity, Intergrability, Testability, and Reliability.
   - Disadvantages: Flexibility, Performance
2. **Layered-based Architecture**:
   - Presentation Layer: Contains all files responsible for presenting UI to the end user and client.
   - Application Layer: Contains all logic required by the application to meet functional requirements and use cases.
   - Infrastructure Layer (Persistence Layer): Performs technical processing of data and database access.
   - Requests are sent downward through the layers and communication between layers uses the up and downward flow.
   - Advantages: Security, Usability, and Reliability.
   - Disadvantages: Scalability
3. **3-Tier Client-Server Architecture**:
   - 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. They will only be able to access the server through sending requests to it. The server will perform calculations and send data to the client.

- Database Tier: The database side of the application comprises of a centralized database through which we can retrieve objects and store model states.
- The client communicates with the server using requests and responses. There may be more than one client, or more than one server, or more than one of both, but the process stays the same due to how scalable it is.
- Advantages: Security, Scalability, Usability, Reliability, Testability
- Disadvantages: Performance

4. **Component-based Architecture**:

- 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 programming logic is divided amongst 7 components, representing each subsystem 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

Resolution: We have decided to use a combination of an n-tier Client-Server architecture (with an added database it connects to), Layered architecture, component-based architecture and Model View Controller. The Client-Server architecture forms the system's structure at the highest layer. Then, within the Presentation tier (client) is the MVC architecture, where the user views and controls the model of the application; and within the Application tier (server) tier is where the server's logic performs calculations and sends back responses to the Client tier. A component based structure is used to ensure that, for example, a notification's requests and responses do not interfere with a booking's requests and responses.

Implications:

- Increased Complexity
- Decreased Performance

4.1 Architectural Design Diagram

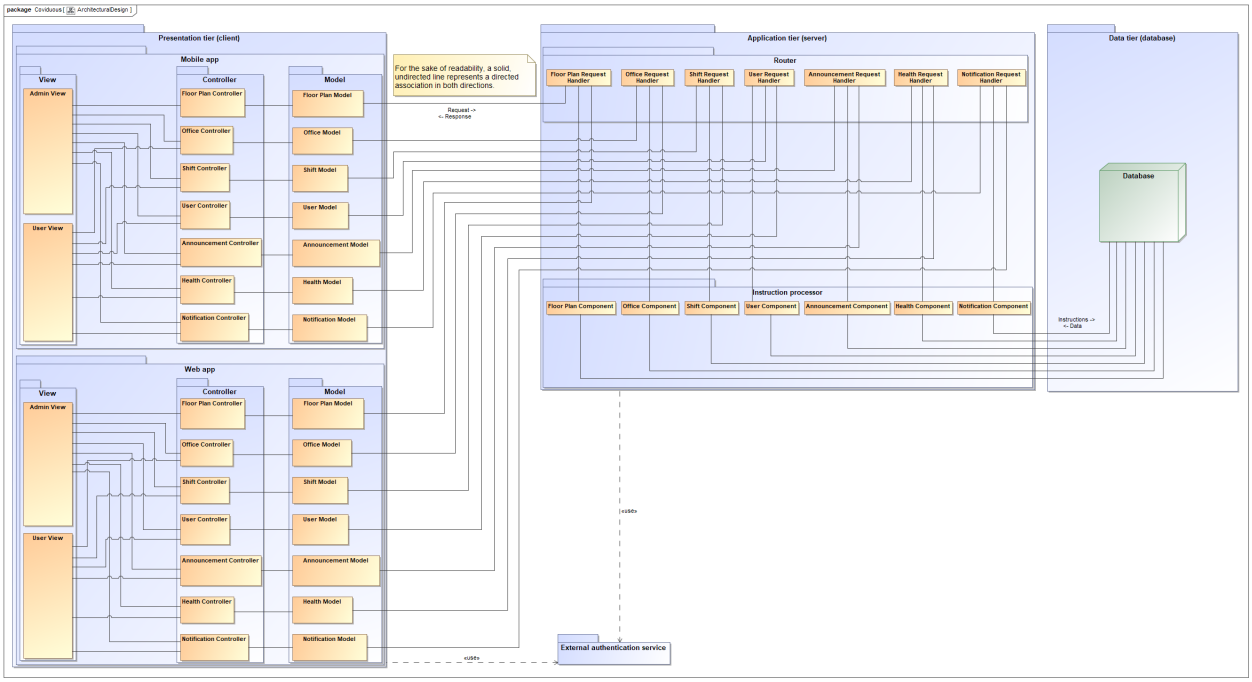

Figure 1: Architectural design diagram for Coviduous

5 Architectural Constraints

5.1 Cost of Services

The selected architecture requires maintenance to be able to update the application and developers to maintain these changes. Services from qualified developers may be expensive. Using open-source software tools for system development may help lower the cost of development.

5.2 Cost of Hardware

The selected architecture makes use of the data model that is expensive because of the multiple pairs of controllers and views based on the same data model.

5.3 Cost of Testing

The selected MVC architecture is used and extensively tested over multiple languages. 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 part (Model, View, Controller) 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.

- **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.

- **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.

- **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 Node.js to communicate with Firebase as our serverless application tier set-up. 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. Firebase does not bring about much complexity in set-up and interfaces well with our choice of Flutter, as they have both been developed by Google. It also 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, for the same reasons we decided on Firebase above. It is included within Firebase as a fast, user friendly NoSQL database, it allows for easy managing of collections, and it costs much less to store data than if we used an external database (in both a financial sense and a computational sense).