PATATES Architecture Notebook

1. Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation. Also, this document explains the design of the system, architectural requirements, architectural mechanisms and constraints. It gives context and so that it is a basic guide for developers to build the system. Architecture notebook instruct the team members how the system organized so the team can adapt itself to needs of developing the system. Also, it helps architects to collaborate with other team members.

The main goal of this software architecture document is to define design of Patates Online Book Store and help the developer to implement the system. It informs about architectural decisions and the architectural system designs.

2. Architectural goals and philosophy

The main purpose of the system is creating an online application which provides easy use, fastness and safety to users. This will be done by defining safety rules and implement them, making improvements on process completion time, cache management and designing the pages rest of the putative user experience rules.

- Patates will be a mobile application which works on Android phones. The application can also be integrated to iOS in the future.
- Application will use Zeplin and Figma to design the user interfaces and Flutter to implement the designs of frontend.
- Elasticsearch will be used to make search faster.
- Users will be able to use the application only if there is stable internet connection.
- The application will be interactive application. Users can interact with the system when searching or filtering by different parameters.
- All the passwords are encrypted to provide safety.
- Exception handling must be done properly so the application always works and never crash.

3. Assumptions and dependencies

- System should be working always, any time.
- User should be able to access application whenever s/he want if there is stable internet connection.
- Every week, there will be team meeting to discuss the process of development the application.
- All required information should be stored in Firebase Firestore. Also, passwords are encrypted.
- The system does not have any hardware devices.
- The system capacity should be high enough.

4. Architecturally significant requirements

- System is a mobile application works on both Android and iOS systems.
- The system should be fast enough. System should response at most 3 seconds in proper internet connection.
- Some operations should need confirmation like giving order.
- User has to be created a password et least 8 character for security.
- Users' private information cannot be seen by admins or developers.
- New updates will be properly testing before the deployment.

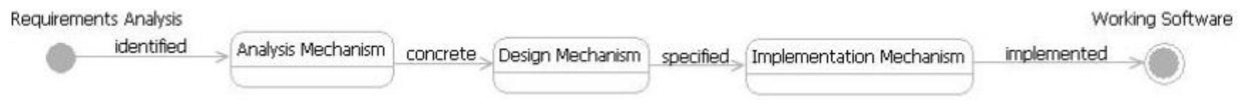
5. Decisions, constraints, and justifications

- Only Dart and Java languages will be used as programming languages.
- Codes must be clean and readable.
- The system should use as few resources as possible.
- Users who are not registered can view the product but cannot add to shopping cart and give order.
- Application only works on proper internet connection.
- System should be flexible in order to add new changes.
- Manipulations (books, shipping firms, campaigns etc.) are only can be done by admins.
- The system will use Firebase as Database.
- To provide good user experience, every screen has to similar looks and make feel good.

6. Architectural Mechanisms

Architectural Mechanisms are common solutions that can be used during development to minimize software complexity and prevent common problems. They represent the basic technical concepts to be standardized throughout the solution. Architectural mechanisms facilitate the development of architecturally important aspects of the system. It allows the working team to create and maintain a harmonious architectural structure, while helping to end all the features that will be added to the application.

Architectural Mechanisms are used to meet architecturally important requirements. These are often non-functional requirements such as performance and security issues. When these mechanisms are fully explained, they show the structure and behavior patterns in the software. They form the basis of the common software to be applied consistently during the development of the product to be revealed. They also form the basis for standardizing the way software works. Given all this, Architectural Mechanisms are an important element of general software architecture. Identification of Architectural Mechanisms also allows to decide whether existing software components can be upgraded to ensure the required behavior, whether new software should be purchased or created.



The value in defining architecture mechanisms is that they:

1- Explicitly call out aspects of the solution mechanics that are common across the system. This helps you plan.

2- Put down markers for the developers to build those aspects of the system once and then re-use them. This reduces the workload.

3- Promote the development of a consistent set of services. This makes the system easier to maintain.

Architectural Mechanisms	**Description**
Reporting	System provides some reporting features. Sales reports, profit reports etc.
Exception Handling	Exceptions will be handled. User will not see any system error etc.
Availability	System must be available always. Users must be able to access to system
Security	Provides services to protect access to certain resources or information.
Scheduling	Provides the ability to execute tasks at a specified time. It may be system backups, database cleanups, maintenances etc.
Archiving	Provides a means to move data from active storage when it has reached a specific state. System does not delete the deleted items. It stores them to use later if there is need.
Localization / Internationalization	Provides facilities for supporting multiple human languages and rendering the language preferred by the user. The system will provide different language choices.
Mail	Services that allow applications to send and receive electronic mail. System can send mail to reset password and confirm the registration.
Transaction Management	A mechanism for handling ACID (Atomicity, Consistency, Isolation, Durability) transactions.
Error Management	Allows errors to be detected, propagated, and reported.
Mega-data	Supports the runtime introspection of components and data.

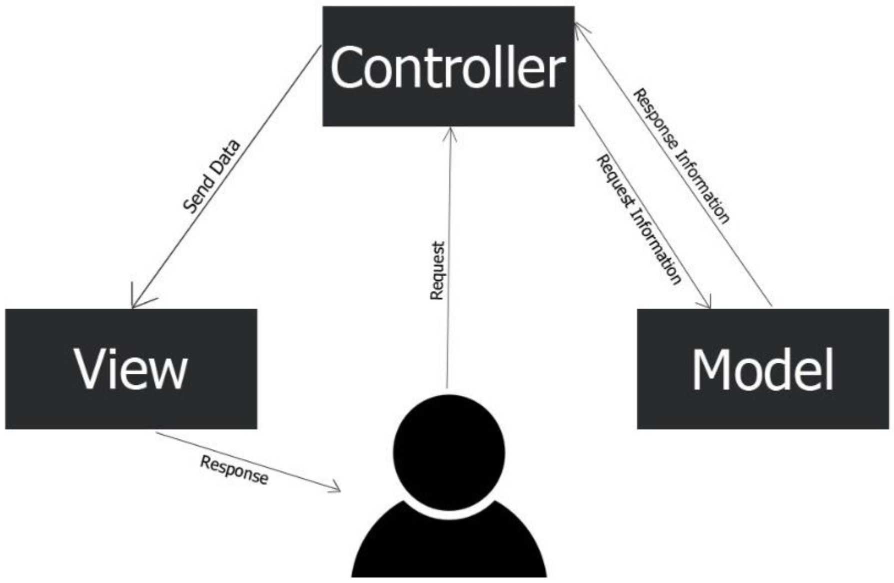
7. Key abstractions

- **Customer**: The abstraction of registered user, they can perform purchase products.
- **Guest**: The abstraction of unregistered user. They can only perform search product operation
- **Admin**: The abstraction of exclusive user. They can perform every operation that exist on system.
- **Database**: The abstraction of storing device which is cloud. (Firebase)

8. Layers or architectural framework

We developed our system based on MVC architecture. The points to choose the MVC architecture are those:

- MVC provides different views from model for different stakeholders.
- MVC provides support for rapid and parallel development.
- Model part does not affect from changes on view part.



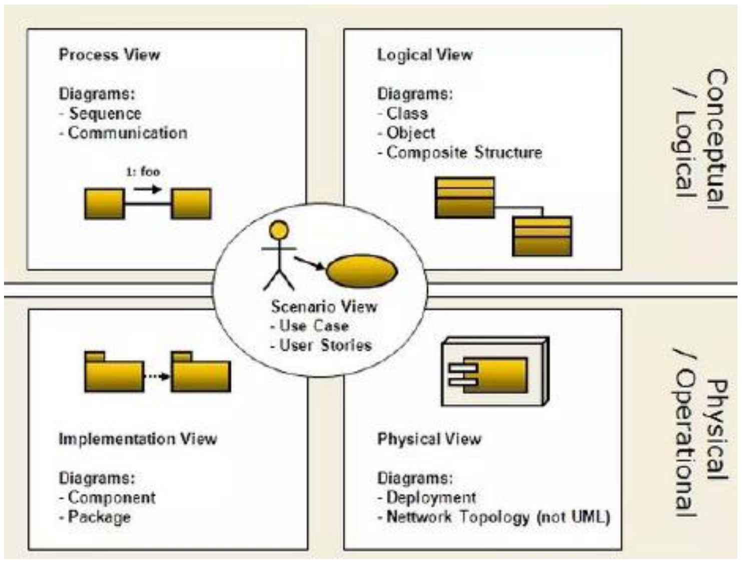
MVC architecture consist of 3 components.

1- Model
Model contains the main functionalities and data. Model is responsible for provide data from database and packaging the data as understandable for other components. It maintains the data of application.

2- View:
View displays the information (came from model) to users and also more than one view may be defined.

3- Controller:
Controller handles the inputs came from users. Take inputs from interfaces and calls the model. The controller renders the appropriate view with the model data as a response.

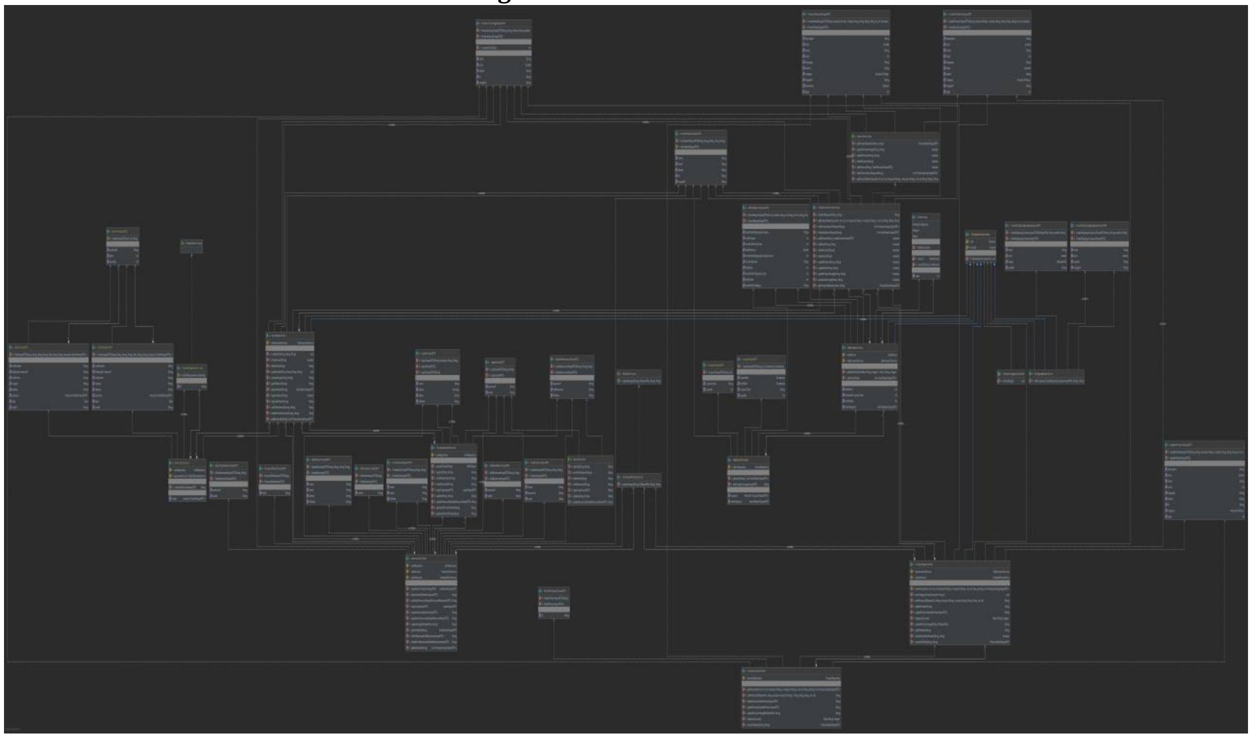
9. Architectural views



- **Logical:** Describes the structure and behavior of architecturally significant portions of the system. This might include the package structure, critical interfaces, important classes and subsystems, and the relationships between these elements. It also includes physical and logical views of persistent data, if persistence will be built into the system. This is a documented subset of the design.

  - **Logical View**
    The logical view is concerned with the functionality that the system provides to end-users. UML diagrams are used to represent the logical view and include class diagrams.

The Initial Class Diagram:



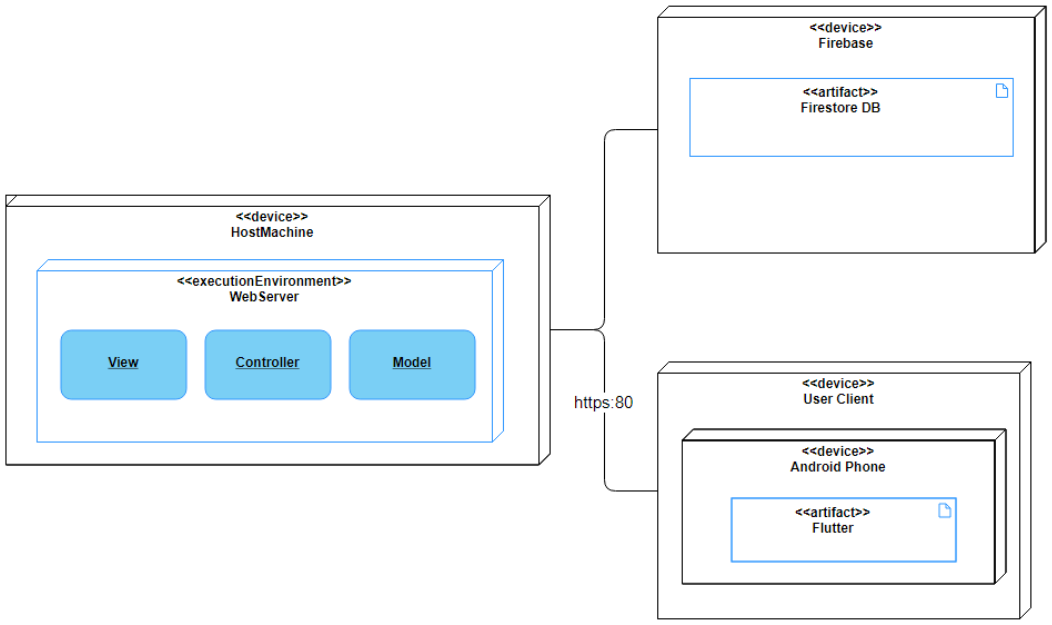
**Please click here to view the diagram more clearly.

  - **Process View**
    The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the run time behavior of the system. The process view addresses concurrency, distribution, integrator, performance, and scalability, etc.

- **Operational:** Describes the physical nodes of the system and the processes, threads, and components that run on those physical nodes. This view isn't necessary if the system runs in a single process and thread.

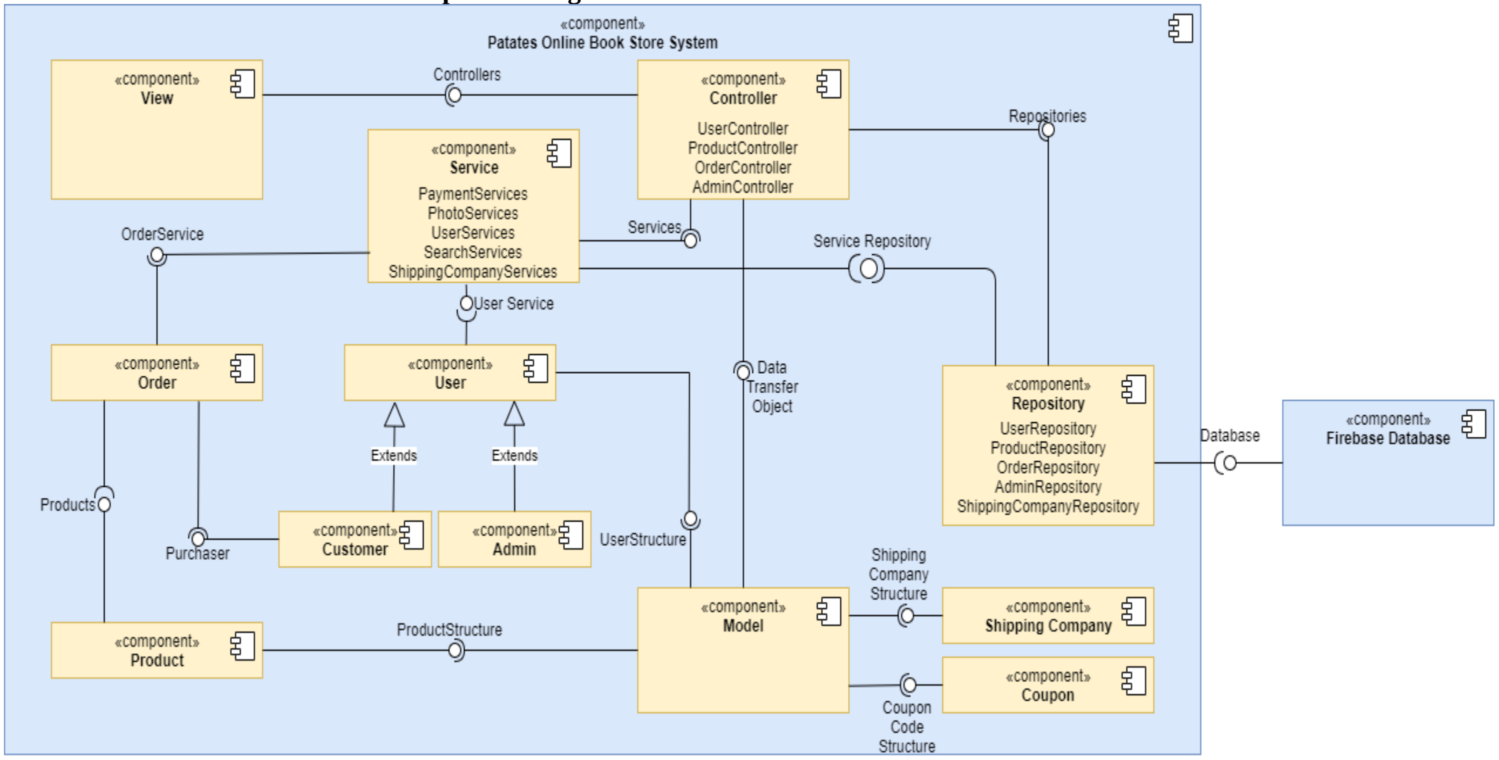
  - **Physical View**
    The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer as well as the physical connections between these components. This view is also known as the deployment view. UML diagrams used to represent the physical view include the deployment diagram.

Deployment Diagram:

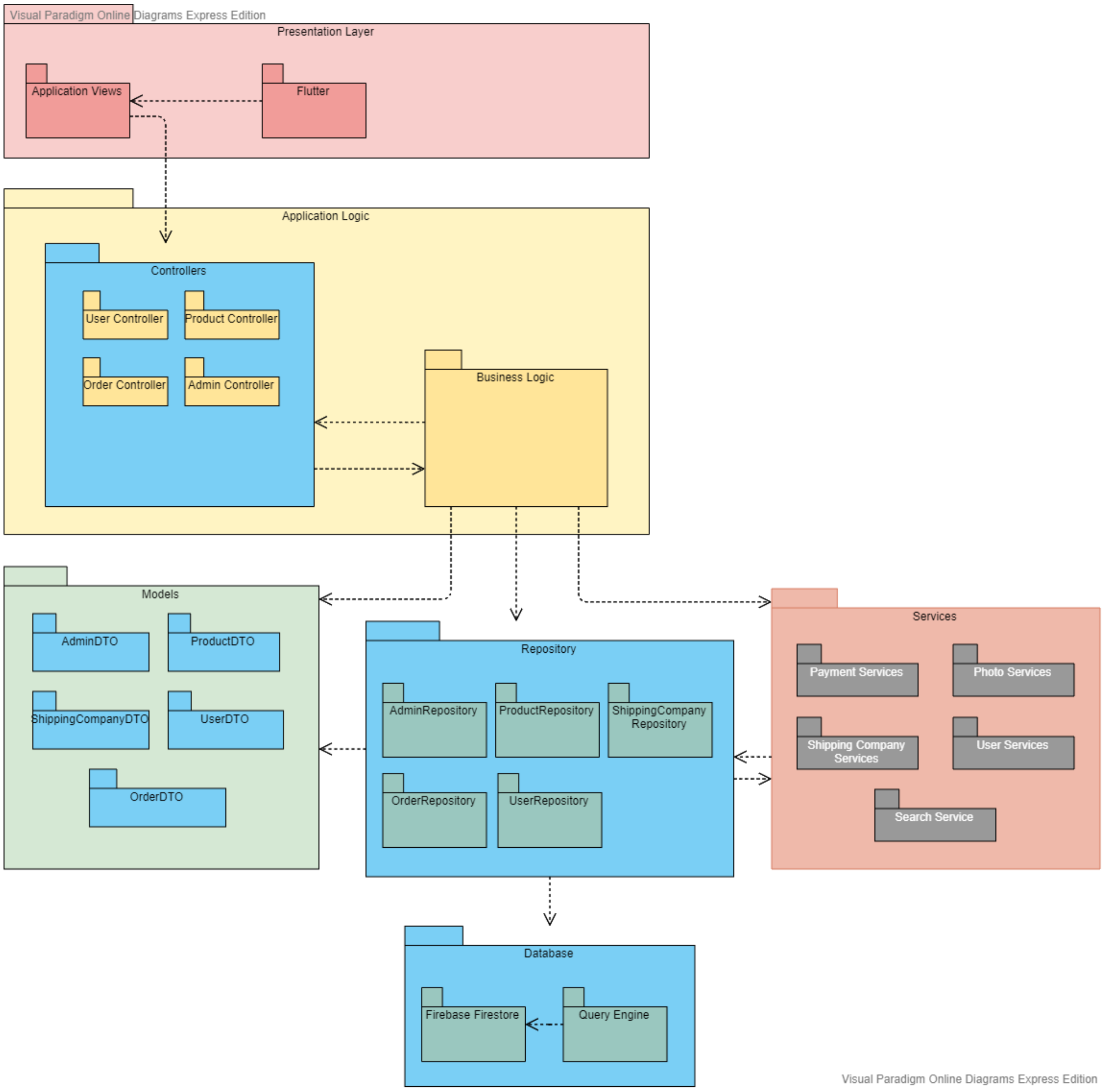


- **Implementation View (Development View)**
  The development view illustrates a system from a programmer's perspective and is concerned with software management. It uses the UML Component diagram to describe system components. UML Diagrams used to represent the development view include the Package diagram.

Component Diagram:



Package Diagram:



- **Use case:** A list or diagram of the use cases that contain architecturally significant requirements.
    - Scenario
      The description of an architecture is illustrated using a small set of use cases, or scenarios, which become a fifth view. The scenarios describe sequences of interactions between objects and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is also known as the use case view.