Department of Computer Science & Engineering
The University of Texas at Arlington

Detailed Design Specification
CSE 4317: Senior Design II
Fall 2023

Team O
GourmetBook

An Hyeonjun
Ayalew Bereket
Shrestha Bhumika
Gyawali Reety
Rakshav Patel

Revision History

Revision	Date	Author(s)	Description
0.1	10.23.2023	BS	Document creation with system overview correction
0.2	10.23.2023	RP	Section 5
0.3	10.24.2023	HA	Section 1-5; edit and layout adjustments
0.4	10.23.2023	HA	System layer diagrams added
0.5	10.25.2023	BS	Presentation Layer Subsystem, Section 3

CONTENTS

List of Figures

List of Tables

1 Introduction

Our product is a mobile based application built to enhance the dining experience for customers and to be an efficient way for restaurant to provide service. With a user-friendly interface and innovative features included the application is truly an upgrade to the hospitality industry. This app aims to streamline the reservation process, offer convenient online pre-ordering, provide event information nearby, and ensure efficient restaurant management.

The growing demand and curiosity in experiencing well-treated and high-end quality cuisines are something that has recently caught people's attention through various sources of media. And there is no single application that aggregates all those fancy restaurants in one place. Moreover, Fine-dining restaurants tend to stay close to the traditional ways of reservations such as phone and website. In addition, there are people with credit cards who get priority to reserve, Since not everyone has access to this privilege, we are here building this application for the rest of the people who could enjoy fine dining without having specific credit cards. This app will be a convenient and user-friendly.

The following sections are detailed descriptions and explanations of each architectural layer and highlighting their subsystems.

2 System Overview

The application has three complementary layers and each layer is divided into different subsystems and functions. The layers are presentation, application, and data/backend.

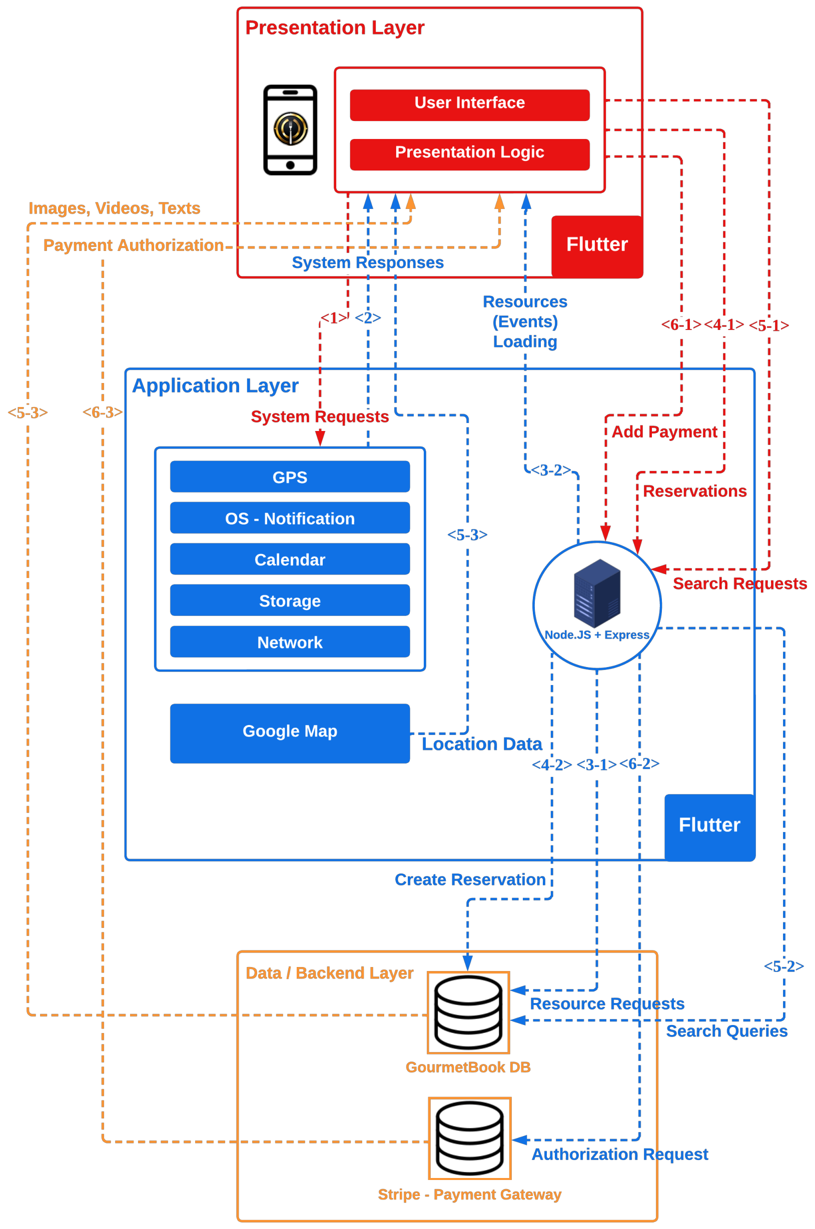

Figure 1: System Architecture Diagram with numbered directed data/request flow

2.1 Presentation Layer Description

The presentation layer serves as the user-facing interface of the application. It encompasses the visual elements, user interactions, and overall user experience. This layer focuses on providing a visually appealing and intuitive interface for users to interact with the app's features. It includes components like screens, buttons, menus, and other user interface elements that allow users to navigate through the application and perform various actions. The presentation layer is responsible for capturing user input and displaying relevant information in a coherent and user-friendly manner.

2.2 Application Layer Description

The application layer acts as the logic and control center of the application. This layer handles the processing of user inputs, interactions, and business logic. It coordinates the different functionalities and services provided by the app, ensuring that the appropriate actions are taken in response to user requests. This layer is also responsible for managing the flow of data between the presentation layer and the data/backend layer. It encompasses various subsystems like the server that perform specific tasks, such as authentication, authorization, data processing, and communication with the backend.

2.3 Data/Backend Layer Description

The data/backend layer is responsible for managing the storage, retrieval, and manipulation of data used by the application. It communicates with remote servers and databases to retrieve and update information. This layer ensures data integrity, security, and efficient access. It includes subsystems related to data storage, retrieval, caching, and synchronization. Additionally, it handles communication protocols, APIs, and network interactions to seamlessly connect the mobile application with the remote server and databases.

3 Presentation Layer Subsystems

The Presentation Layer is responsible for managing the user interface (UI) and handling the presentation logic. It controls how the app's user interface is structured, styled, and displayed to users, as well as how user interactions are processed and responded to. This layer ensures a seamless and user-friendly experience by connecting the visual elements with the underlying logic and data stored in backend.

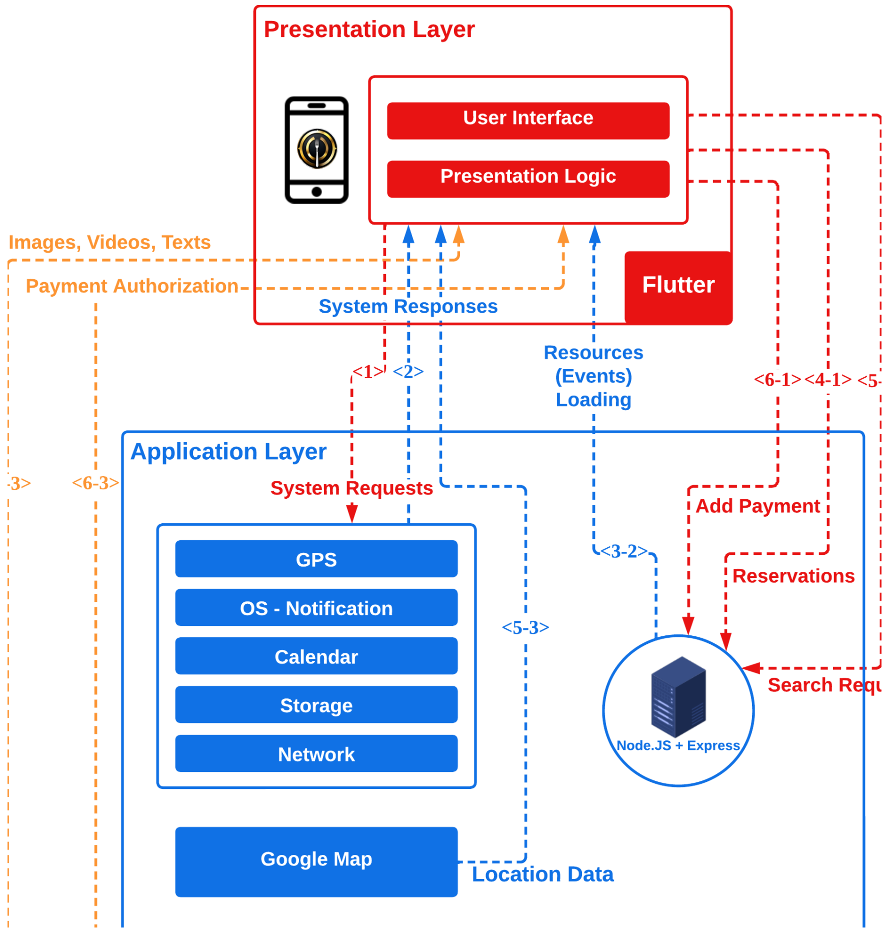

Figure 2: Presentation layer diagram with data flows

3.1 Layer Hardware

The Presentation Layer typically runs on a variety of consumer devices, including smartphones, tablets, and desktop computers. Any Android or iOS device or emulator on a Desktop computer to test the app. Our team is mainly using emulators/simulators provided by Android Studio to run Android and IOS apps for testing purpose.

3.2 Layer Operating System

This layer can run on various operating systems, including Android and iOS for mobile devices, and Windows, Mac OS, and Linux for desktop devices.

3.3 Layer Software Dependencies

The Presentation layer relies on software dependencies, including the Flutter framework for building the user interface and application logic. It requires platform-specific SDKs for mobile devices, such as

Android and iOS SDKs, and utilizes NodeJS + Express for data services and authentication.

3.4 User Interface Subsystem

One of the two subsystems displayed in the diagram below is the User Interface. This Subsystem is responsible for the visual representation of the application that the user interacts with. It includes all the elements and components that user see and interact with on their devices. This subsystem includes features such as a List of Restaurants, Events, Menu, Search and Filters, Profile, Purchase History etc.

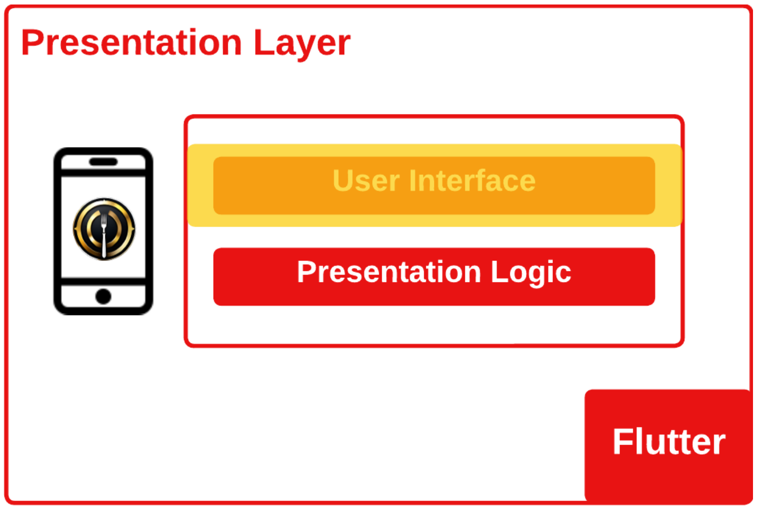

Figure 3: Presentation layer diagram with its subsystems

3.4.1 UI Hardware

The User Interface Subsystem operates on the hardware components of the device on which the app is running. This includes devices like smartphones, tablets, or desktop computers, each with its own set of hardware components like processors, memory, display, and input mechanisms.

3.4.2 UI Operating System

The User Interface Subsystem relies on the operating system of the device. It requires Android or iOS for mobile devices, or other operating systems such as Windows, mac-OS, or Linux for desktop devices, depending on the target platform.

3.4.3 UI Software Dependencies

The deployment environment must have Flutter framework as core front-end development. Platform-specific SDKs for mobile platforms (Android and iOS). Android Studio for testing the application.

3.4.4 UI Programming Languages

The User Interface Subsystem primarily uses the Dart programming language for implementing the visual representation and user interaction components.

3.4.5 UI Data Structures

This subsystem utilizes data structures to manage and display information to the user, such as lists for restaurant and menu items, user profile data, and purchase history. Data are exchanged between the application and databases in JSON format.

3.4.6 UI Data Processing

This subsystem data processing applies common algorithms for rendering UI elements including layout algorithms, text rendering, and graphics rendering. Interaction processing involves event handling, user input validation, and transitioning between app screens. The main feature i.e. Reservation tab is handled by Node.Js and Firebase for real-time data management. Similarly, any image, videos and text displayed will be handled by Back-end layer which deals with NodeJS and Express.

3.5 Presentation Logic Subsystem

Another Subsystem is the Presentation logic which is responsible to control the behavior and flow of the UI. It handles and integrates user inputs, process them, and coordinates with the back-end to retrieve and display the proper data. This subsystem includes Data Validation, updating UI, Interaction with Application Layer, user inputs and navigation.

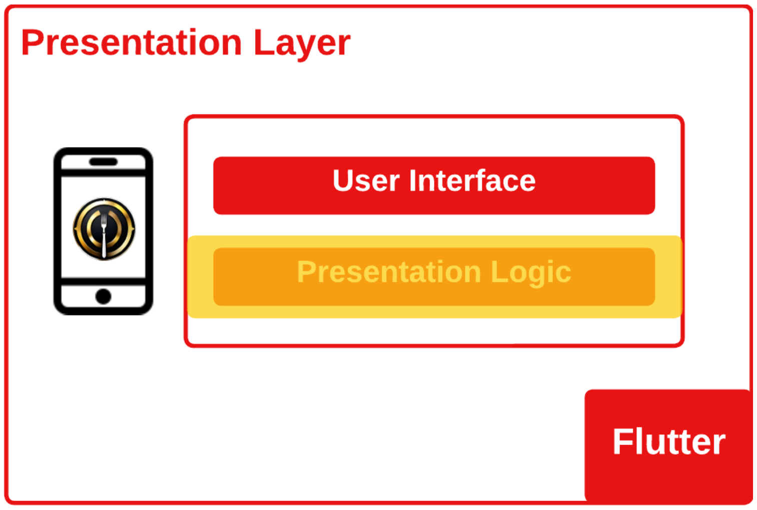

Figure 4: Presentation layer diagram with its subsystems

3.5.1 Presentation Logic Hardware

The Presentation Logic Subsystem operates on the same hardware components as the User Interface Subsystem, which are the devices running the app, including smartphones, tablets, or desktop computers. However, in a restaurant reservation app, it might access the device's GPS to provide location-based restaurant recommendations to users.

3.5.2 Presentation Logic Operating System

The Presentation Logic Subsystem relies on the underlying operating system of the device, such as Android, iOS, or desktop operating systems like Windows, mac-OS, or Linux, depending on the platform.

3.5.3 Presentation Logic Software Dependencies

The deployment environment must have Flutter framework with JavaScript integrated as core front-end development. Stripe API is used as payment gateway which handles payment option under Reservation and Menu feature. The deployment environment Must have Node.js installed on the system, which includes npm (Node Package Manager).

3.5.4 Presentation Logic Programming Languages

The Presentation Logic Subsystem primarily uses the Dart programming language to implement and control the app's business logic and data flow.

3.5.5 Presentation Logic Data Structures

Data structures used in the Presentation Logic Subsystem include variables, data models, and structures like lists, maps, and custom classes to manage application data and state. These structures help in processing and organizing data for presentation.

3.5.6 Presentation Logic Data Processing

Data processing in the Presentation Logic Subsystem involves implementing algorithms for handling data retrieval, manipulation, and business logic. This includes authentication and authorization processes, data validation, and communication with Data/Back-end layer.

4 Application Layer Subsystems

The application layer serves as the logic and control center of the application. The application layer processes the information it receives from the presentation layer and sends it to the data/back-end layer. Below are the several subsystems that work together to handle user inputs, interactions, business logic, and data flow.

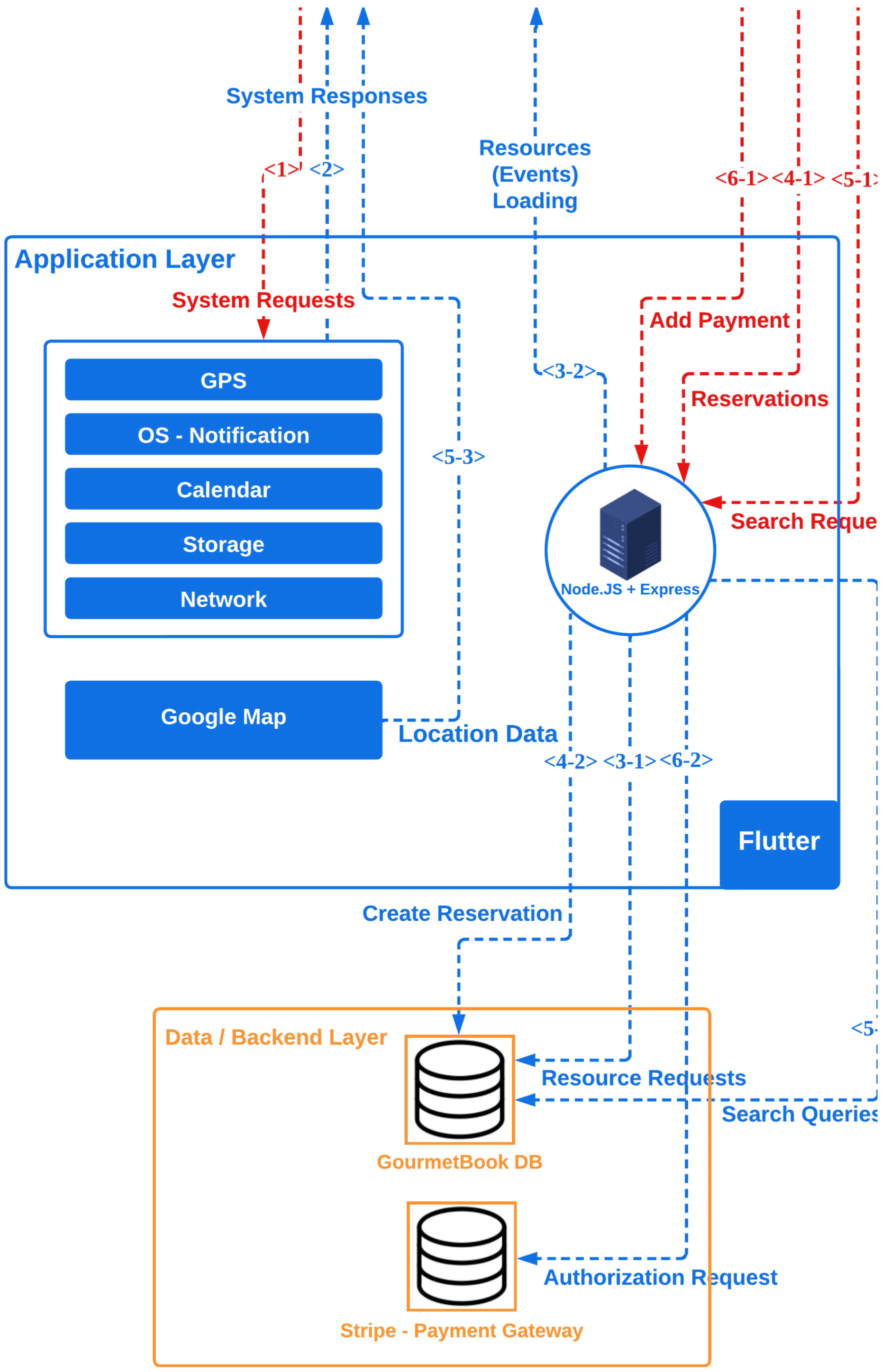

Figure 5: Application layer diagram with data flows

4.1 Layer Hardware

A physical device or emulator to run and test the application if the communication is well set up between the application and the databases through the server. Emulators/simulators are provided by Android Studio to run Android and IOS applications. Device can also be physically connected using USB cable.

4.2 Layer Operating System

The deployment environment for the server can be any Linux-based operating system. The server will be deployed on Firebase web-hosting service, which runs on a Linux operating system. The application will handle both Android and iOS operating systems with configurations done by Flutter.

4.3 Layer Software Dependencies

The deployment environment Must have Node.js installed on the system, which includes npm (Node Package Manager). Other dependencies include Flutter SDK, corresponding IDE, and either API platform or command-line network tools such as "curl".

4.4 Subsystem 1: Node JS + Express

It handles communication with back-end services, APIs, and databases to fetch and store data. We will be using Node JS - Express as our web application framework to connect to the back-end layer subsystems. This subsystem is in charge of sending or retrieving information from the GourmetBook DB, as well as sending payment information to our payment gateway for authorization. This subsystem will receive resources from the frontend (user interactions with UI components) and translate them into actions according to application logic.

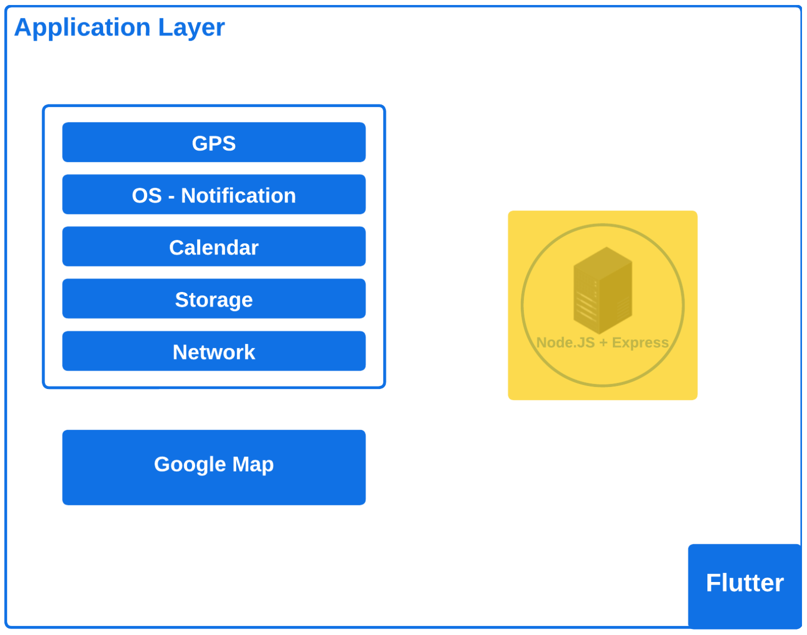

Figure 6: Application layer diagram with its subsystems

4.4.1 Subsystem Hardware

Any Android and IOS device/emulator with a processor, enough RAM, storage capacity, network connectivity, and a computer to deploy a web server, which in this case is a cloud environment.

4.4.2 Subsystem Operating System

Platform-independent can run on Windows, macOS, or Linux; however, the specific environment required for the application is Linux.

4.4.3 Subsystem Software Dependencies

The environment where the web server is deployed must have Node.js installed on the system, which includes npm (Node Package Manager).

4.4.4 Subsystem Programming Languages

JavaScript: Node.js and Express.js are JavaScript frameworks.

4.4.5 Subsystem Data Structures

Data are exchanged between the application and databases in JSON format. In JSON format, data is represented as key-value pairs enclosed in quotations. Keys are strings, while values can be any data type including null. JSON also allows for the nesting of objects and arrays in brackets.

4.4.6 Subsystem Data Processing

We will use Node.js for ETL pipeline

- Extract: Retrieve raw data from API.
- Transform: Renaming keys in the data if necessary, removing invalid or unnecessary data points, computing new values, or any other type of data processing
- Load: Move the formatted data (JSON) into GourmetBook DB where it can be stored

4.5 Subsystem 2: System Services (GPS, OS-Notifications, Calendar, Storage, Network)

It handles events from the Presentation Layer and processes them internally to update the system services of phones that have applications. The phone's system services' data is necessary for application logic to run according to a specific user's needs. GPS updates user location, OS notifications alert user, calendar is needed to mark important reservation dates, storage is needed to hold application data, and network provides internet access. This subsystem sends updated system service data to the Presentation Layer to display changes.

4.5.1 Subsystem Hardware

Any Android/IOS device with all the above system services, including GPS, notification capabilities, calendar, storage capacity, and network connectivity. Assumption that these services are up-to-date and accessible to device users. Also, users should have notifications allowed for the application.

4.5.2 Subsystem Operating System

The device OS should be either Android or IOS.

4.5.3 Subsystem Software Dependencies

Most system calls for mobile application development are handled through the built-in functions in Flutter; however, how it is handled in Android and iOS differs.

4.5.4 Subsystem Programming Languages

Dart: Flutter is a framework based on Dart and developed by Google.

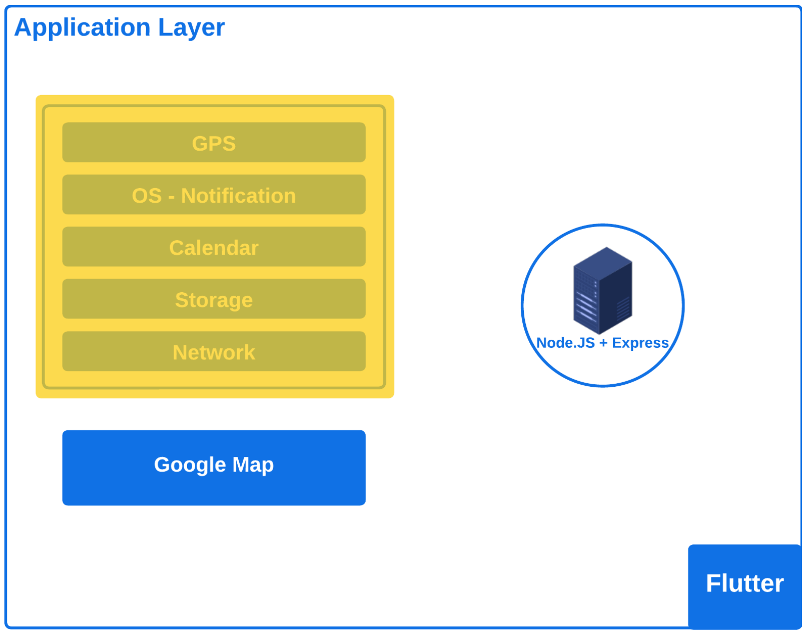

Figure 7: Application layer diagram with its subsystems

4.5.5 Subsystem Data Structures

We will use queues for notification pushing, and objects holding key-value pairs of data.

4.5.6 Subsystem Data Processing

The libraries and built-in APIs we will use offer various functions we can call to retrieve and send system data.

4.6 Subsystem 3: Google Maps

Google Maps is a GPS navigation application developed by Google for mobile devices. It provides various API services through Google Maps SDK for both Android and iOS. Once the user requests the location information of a restaurant they are interested in, Google Maps API will transmit the GPS coordinates of the restaurant to the presentation layer.

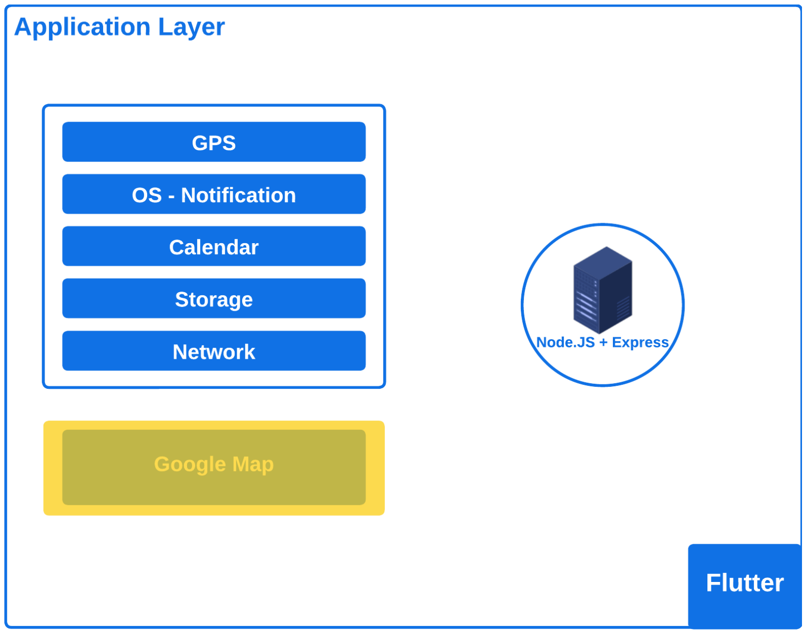

Figure 8: Application layer diagram with its subsystems

4.6.1 Subsystem Hardware

Any Android/IOS device with a processor, enough RAM, storage capacity, and network connectivity to receive and transmit the GPS data from Google Maps to the application.

4.6.2 Subsystem Operating System

It is Android and iOS, the operating systems that the application will be developed on with Flutter.

4.6.3 Subsystem Software Dependencies

Google Maps SDK and API secret keys for Android and iOS must be securely implemented to communicate through the API.

4.6.4 Subsystem Programming Languages

Dart: Flutter is a framework based on Dart and developed by Google.

4.6.5 Subsystem Data Structures

Google Maps uses JSON data format to transmit and receive data through communication. The JSON files contain location coordinates, marker coordinates if set up for a specific location, and etc..

4.6.6 Subsystem Data Processing

The data communication takes place directly in the background of the application. The application transmits HTTP request(s) to Google Maps API, and Google Maps API responses with a data set in JSON format. The application, then, extracts the necessary data out of the received JSON file so that it can be handled in the presentation layer.

5 Data/Backend Layer Subsystems

The data/backend layer serves as a secure data storage and end host system of the data communications between the application and necessary resources. Instead of directly communicating with and accessing the databases, the application requests the middle agent, which is the node.js server, to send request(s) on behalf of the authorized users, and to respond back to the application with the data accesses in the databases. In this manner, it can achieve the highly developed security of the application and data communication, and in case of failing to acquire the data needed, the middle agent can respond with a warning message instead of crashing the entire process of the application.

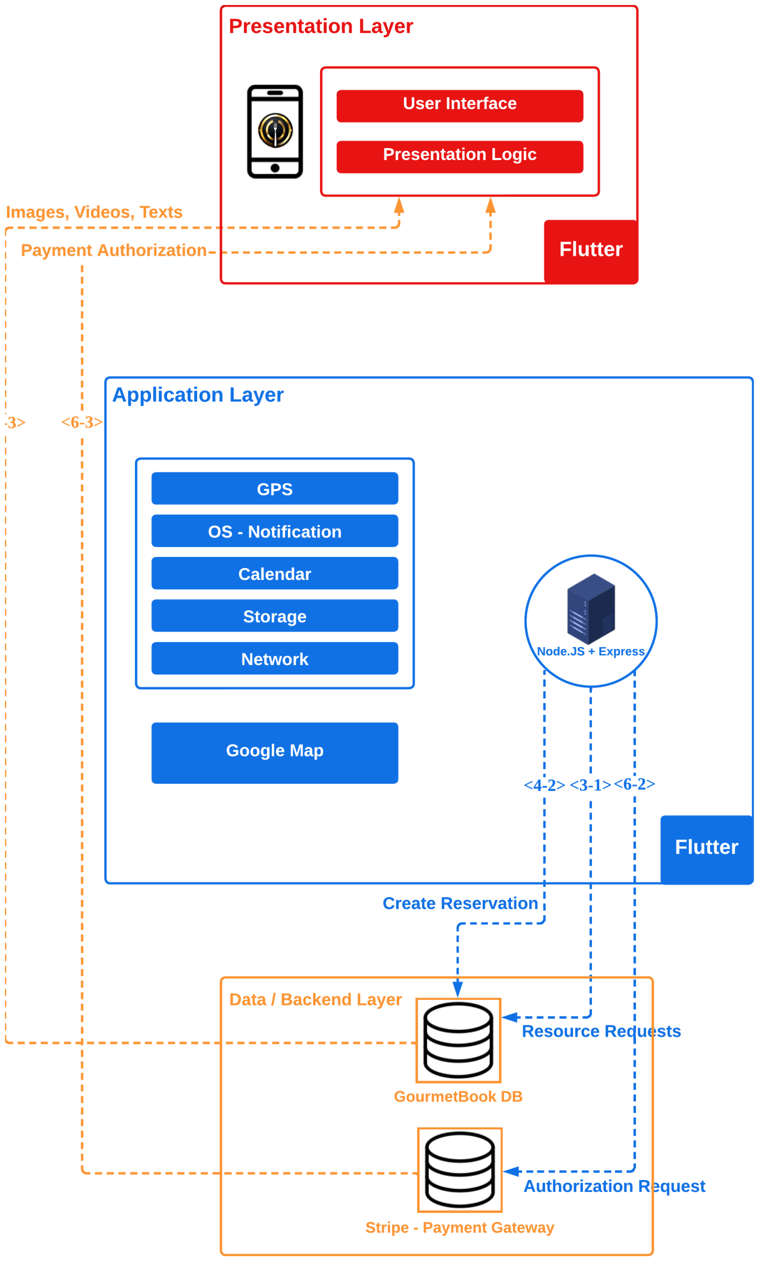

Figure 9: Data/backend layer diagram with data flows

5.1 Layer Hardware

This layer does not require any particular hardware resources for developers to integrate the databases. "GourmetBook Database" will be integrated through Google Firebase's real-time database service, which is a cloud database, and "Stripe Payment Gateway Database" has already been integrated by Stripe for whoever wants to access their database in an authorized manner.

5.2 Layer Operating System

In order to access each database system, the web server working as the middle agent between the application and the databases must be operating and available; therefore, refer to the operating system requirements for "Subsystem 1: Node JS + Express" of Application Layer.

5.3 Layer Software Dependencies

In order to access each database system, the web server working as the middle agent between the application and the databases must be operating and available; therefore, refer to the operating system requirements for "Subsystem 1: Node JS + Express" of Application Layer.

5.4 GourmetBook DB

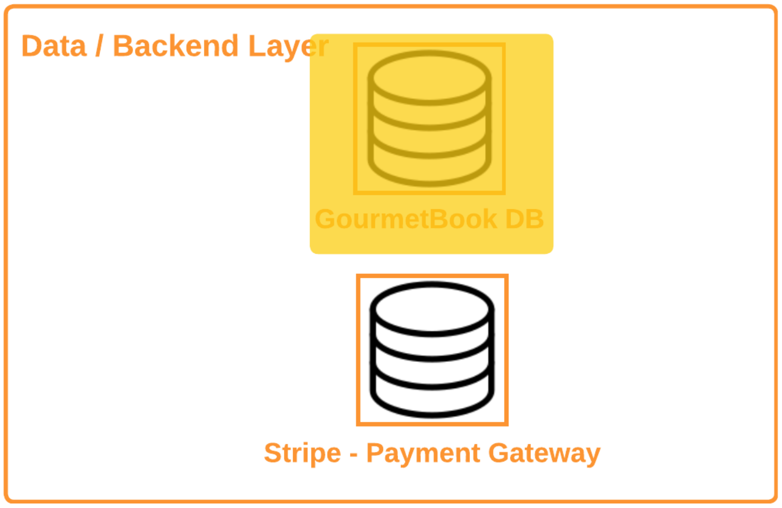

Figure 10: Data/backend layer diagram with its subsystems

5.4.1 Subsystem Hardware

Refer to the section 5.1.

5.4.2 Subsystem Operating System

Refer to the section 5.2.

5.4.3 Subsystem Software Dependencies

Refer to the section 5.3.

5.4.4 Subsystem Programming Languages

This subsystem does not require programming to build/integrate the database systems.

5.4.5 Subsystem Data Structures

The database transmits the requested resources and data in JSON format through HTTP messages. The database itself is also constructed in multi-layered JSON as well.

5.4.6 Subsystem Data Processing

The application sends signals to the server when some data-requiring event happens, then the server sends HTTP request(s) to the database system. The database will find the requested resources in the database, and if found, it will respond back to the server with the data set in JSON format. The application, then, receives the data set from the server and extracts only the necessary data.

5.5 Stripe Payment Gateway API

Describe at a high level the purpose and basic design of this subsystem. Is it a piece of hardware, a class, a web service, or something else? Note that each of the subsystem items below are meant to be specific to that subsystem and not a repeat of anything discussed above for the overall layer.

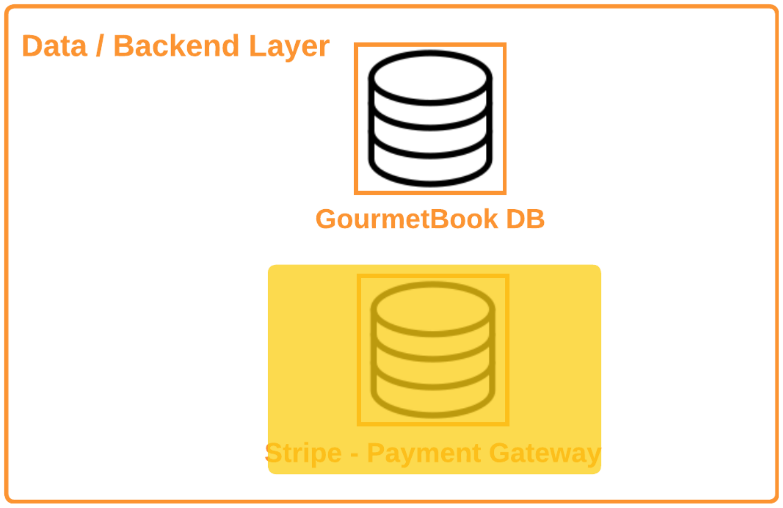

Figure 11: Data/backend layer diagram with its subsystems

5.5.1 Subsystem Hardware

Refer to the section 5.1.

5.5.2 Subsystem Operating System

Refer to the section 5.2.

5.5.3 Subsystem Software Dependencies

Refer to the section 5.3

5.5.4 Subsystem Programming Languages

This subsystem does not require programming to build/integrate the database systems. It is already built by Stripe and available for authorized users.

5.5.5 Subsystem Data Structures

The database transmits the requested resources and data in JSON format through HTTP messages. The database itself is also constructed in multi-layered JSON as well.

5.5.6 Subsystem Data Processing

A unique token-based authorization is required for each application user. A unique private key will be assigned to each user after the authentication process, and with the private token, the user will be able to access the Stripe API system and get their credit/debit card information verified.

6 APPENDIX A

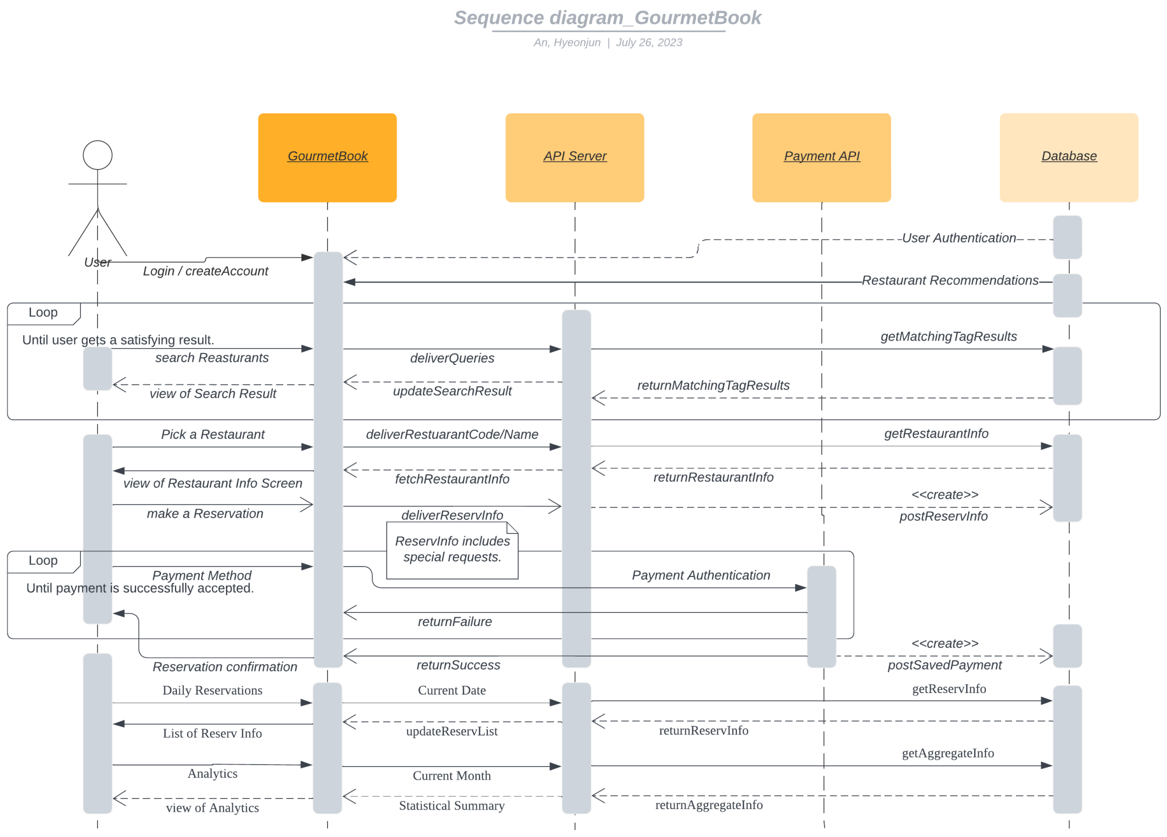

Figure 12: Sequence diagram of the overall system

References