MEet and You

Brent Nishioka (Leader)
Gideon Essel
Joshua Ramos
Raymond Guevara
Vivian Dinh
Team Pentaskilled
October 6th, 2021

Table of Contents

Overview

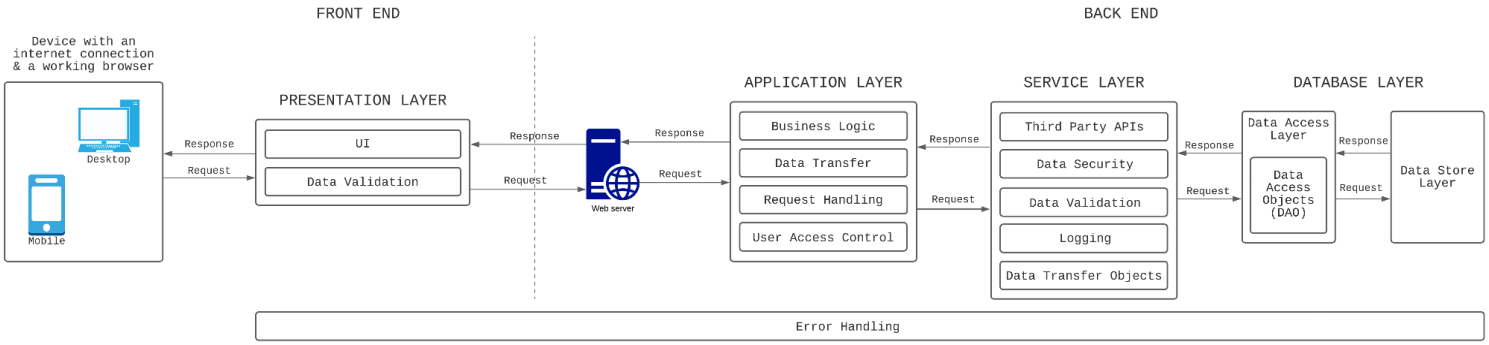
Purpose

The purpose of this high-level design (HLD) document is to provide an overview of the MEet & You application. This document also gauges insight into the structure of our application.

Scope

MEet & You is intended to be a web application. In order to use MEet & You, a sufficient internet connection and access to a web browser are necessary. MEet & You will function as a single-page application (SPA), dynamically loading the content as the user navigates the website.

System Architecture



General Overview

Our application will be split into a front end and a back end which will support different architectures. The front end of our system will support the Model, View, ViewModel (MVVM) architecture. The back end of our system would implement a microservice architecture.

Presentation Layer

The presentation layer will house the front end of our application; this is where the user will be able to interact with and carry out functions like creating an itinerary or accessing their user dashboard. The presentation layer will also take any user input, send that input through our system, and receive a corresponding output.

User Interface (UI)

The user interface is the visual component shown to the user as they interact with our application. It displays the different functionalities of our application in a simple way for the user to understand.

Data Validation in the Presentation Layer

Any input provided by the user in the presentation layer will be checked before sending it to the other layers in our system. This ensures proper formatting of the input, which will maintain the integrity of data going to our back end. Suppose the user enters

misformatted data. In that case, our presentation layer will communicate to the error handling layer, provide the user with a user input error, and allow the user to re-enter their input.

Web Server

The web server will serve as our hosting platform for our web application. It will process HTTP requests and deliver them to the back end of our system. It will also receive any HTTP responses and send them to the presentation layer.

Application Layer

Our application layer is the entry point of our web application. It contains the business logic, data transfer, request handling, and user access control components of our system. The application layer is responsible for managing the business requests originating from and gathering all the information business requests require.

Business Logic

Business logic is the custom rules utilized in handling the exchange of data between both a database as well as the user interface. Business logic is an essential part of defining the constraints in regards to how the application operates. The business logic of our application will be maintained in our application layer, which will notify the service layer to perform the necessary services to validate the business requirements stated in our Business Requirements Document.

Data Transfer

The purpose of data transfer inside our application layer is to dispatch any data requests from the presentation layer to the service layer. It will also return data responses from the service layer to the presentation layer.

Request Handling

The purpose of our request handling functionality is to process business requests. It will accomplish this by first consuming the request from the presentation layer. Once our request handler receives the request, it then gathers the appropriate information necessary to fulfill the request. After that information is collected, the fulfilled request is sent back to the presentation layer.

User Access Control

Adding user access control functionality is necessary to issue limitations to users using our system. We want to ensure that only certain users have access to certain data and functionalities of our application.

Service Layer

The service layer of our application provides access to the microservices required to run our application. Data that has been processed is sent to the service layer in order to get both authorized and validated to ensure that the user is allowed access to specific information, preventing the user from performing invalid actions. If access is unauthorized, then this particular session will be recorded and stored in our log database.

Third-Party APIs

Any third-party APIs utilized in our application will be called in our service layer. We intend for each API to be accompanied by a microservice, which would process business requests calling the API, and create responses to send back to our application layer.

Data Security

The purpose of data security in the service layer is to provide additional protection of the data before utilizing an API or a microservice. We intend to accomplish this through the authorization of the user and their session, ensuring they are both valid before permitting the user to perform a microservice.

Data Validation

If a certain business request requires database access, the data inside the request must be validated before giving database access. This is to ensure the data's integrity before entering the data access layer. Data returned from API calls would also be validated to ensure correctness and completeness.

Data Transfer Objects (DTOs)

The purpose of DTOs in our system is to exchange and share data between our microservices. We will also implement DTOs to regulate the flow of requests coming from our application layer that require access to the databases in our system.

Logging in the Service Layer

Any logs created in our system would utilize a logging microservice found in our service layer. This would also be where the logs in our system would be located. A centralized logging system is vital to solving issues within our system in the shortest time possible.

Data Access Layer

The data access layer would process any request from the user which requires information to be accessed via a database. It will act as a liaison between our data requests and the data store layer and administer the transfer of data. Our application will utilize Data Access Objects to retrieve and manipulate database entities. Any requests passing through this layer will need to be authenticated by our system to confirm the request sender's identity and if they have a valid user session.

Data Access Objects (DAOs)

Our application will utilize DAOs to enable business requests to perform data manipulation without revealing database details. To accomplish this, we will model DAO interfaces for business requests to perform operations on our database. These will work in conjunction with the DTOs to fulfill any business database request.

Data Store Layer

This layer of our architecture contains all the database tables and entities required for our application. It can only be accessed through the data access layer using a data access object.

Security Layer

Security features will be centrally located in the service layer of our application. However, we are aware of the potential security issues throughout our application and will handle those issues accordingly. Security inside our application will take the form of:

- Authentication
    - Checking the identity of the user and comparing that to who they claim to be.
- Authorization
    - Access control and regulating what parts of our application are visible to certain users and which parts of our application certain users can interact with.
- Data Security
    - Ensures the integrity of data going in and out of our system.

The protection of our users is one of our priorities as data leaks or breaches violate our users' privacy instated in our privacy policy.

Error Handling Layer

Error handling layers will be implemented throughout our entire system. This is because exceptions can be thrown in a handful of locations. Errors would be handled and logged where they originated. Potential errors in our system include:

- Server Request Timeouts
- Invalid Server Request
- Server Error
- Invalid User Input
- Unauthorized Access
- Required Contact Administrator

Our system supports multiple server-side errors. These include:

- Invalid Server Request
- Server Error
- Unauthorized Access
- Required Contact Administrator

Logging Layer

Our logging layer will note specific events which occur in our system. Events in our application which we intend to log are:

- User logins/logouts
- User registration is successful
- User itinerary registration is successful
- Security breaches
- Errors in our system
- Usage of application features by users

Logs can be created anywhere in our system. When the system needs to create a log, it will be created and written in the exact location where the log function was called. Once the log is completely written, it is then transported to our service layer, where all the logs in our system will be centrally located.

Architecture Advantages/Disadvantages

Front End

Architecture	Advantages	Disadvantages
MVC	Data can be returned without formatting, meaning that these components can be reused with any other interfaces. Multiple views can be created for a model restraining how often code duplication needs to be utilized. Multiple components can be developed at the same time such as the view and controller, supporting both rapid and parallel development.	The main disadvantage of the MVC architecture is its complexity to implement. Not suitable for small applications as it can have an adverse effect on both the application's design and performance. Huge amounts of resources are required to operate in a working environment.
MVVM	Segregates code across the user interface and application logic, thereby providing a modularised code structure. Dynamic changes to variables in the View Model also updates in the View due to	Reusability of views, as well as view model, is rendered difficult due to the segregation of these components. Due to data binding, relevant errors are not caught,

	data binding. Updating the user interface does not require an application redesign since the MVVM separates the user interface from the application logic.	resulting in a system that is difficult to debug. MVVM is difficult to integrate with larger applications due to the difficulty in generalizing the ViewModel
MVP	Isolated implementation means that each component of MVP can be tested separately. Division of responsibilities between its four components allows for code reusability.	Not ideal for smaller applications or solutions. It requires both a higher complexity and high learning curve to implement.

We plan to work with the MVVM architecture for the front end of our application. By making code segregated, the MVVM allows the reusability of modularised code structures. Where MVC has a triangular relationship with its component, MVVM has a chain relationship. MVVM flattens out the dependencies, which mean that other dependencies do not distract each other and you can do the isolated implementation. As the application itself grows in scale, via MVVM our business logic can be written in the View Model to restrict communication of the output to the view or controller, which keeps the controller from becoming bulky. A major component of MVVM which is shared by MVP is its test-driven approach where unit tests can be utilized on specific components. Due to the evolving nature of architectural patterns MVVM is the most useful to implement due to its potential usages where MVP has reached a level where it has not been taken further.

Back End

Architecture	Advantages	Disadvantages
Microservices	Encourages easy scalability of the application. Microservices are independent of each other, meaning the deployment of one component will not affect another. Provides necessary flexibility for testing out different technology stacks without needing to worry about dependency concerns.	The maintenance of a microservice architecture is increased due to its component deployment modularity. Any communication occurring between two services is complex. Debugging tends to be quite difficult because of the distinct logs produced by each microservice component.
Client-server	Application is easily scalable to accommodate a certain capacity. Centralized server where all business is conducted allowing for easy system management and modifying resources. Security is robust because if any data is lost, it can be	The architecture is proven to significantly slow down when too many clients make requests to the same server. Maintaining a client-server can be a nightmare if not given enough thought. Architecture lacks robustness; if the main server fails, then the entire system undergoes failure.

	recovered easily since data is only located in one place.	
N-tier architecture	The tiers function as independent components; any modifications made to one tier will not affect any other tier. Robust security control as security can be controlled on a tier-by-tier basis. Good reusability because components are loosely coupled.	High-cost overhead for hardware, network, maintenance, and deployment. Performance of the application may suffer if the hardware and network bandwidth are lacking.

We plan to work with a microservice architecture for the back end of our application. The main reason behind choosing this back-end architecture is its loose coupling and low dependencies between components. It will give our system more modularity and make maintenance of system components streamlined. The N-tier architecture offers this functionality but at the potential cost of performance. The client-server architecture does not offer any modularity, making it a maintenance nightmare, strengthening the choice of a microservice architecture. We also evaluated the disadvantage of debugging different microservice log types when deciding on the back-end architecture. This is a compromise we will have to work around; however, it is not a deal-breaking disadvantage. Both the N-tier architecture and microservice architecture offer high scalability; however, we acknowledged the microservice's main advantage of being able to use any technology as more important when choosing a back-end architecture.