High Level Design
Hubba

Version 1.2
Prepared By: Development Hell
Class: CECS 491A-04
Date: October 5, 2022

Github Repository:
https://github.com/DevelopmentHellaHell/SeniorProject

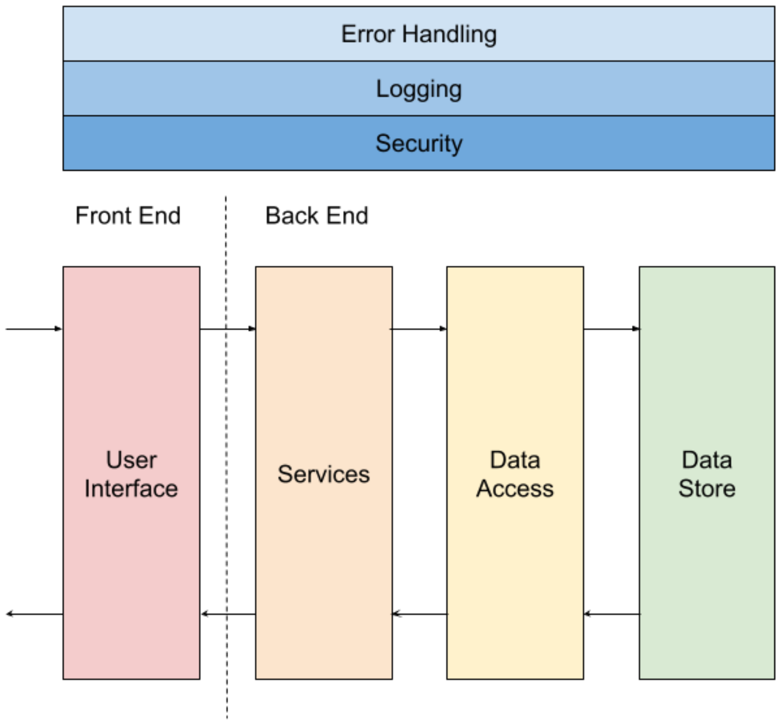
Team Leader
Kevin Dinh
Members
Garrett Tsumaki
Bryan Tran
Jett Sonoda
Tien Nguyen
Darius Koroni

Table of Contents

Revision History

Version	Overview	Date
v.1.0	Initial HLD	October 1, 2022
v.1.1	Added and revised all sections	October 3, 2022
v.1.2	Revisions to all sections	October 5, 2022

Abstraction Layers



User Interface Layer

The user interface layer will handle all user interactions and will house the UI components to render elements of the website. This layer will also communicate with the service layer through api calls over HTTP.

Services Layer

Business logic will be handled in the service layer and is used as an intermediary step between the User Interface layer and the Data Access layer. With a RESTful environment, the HTTP methods (GET, POST, PUT, DELETE) will be used as they correlate with the respective CRUD operations.

Data Access Layer

Data access is abstracted from the data store layer to maintain loose coupling between components of the system. Creating this abstraction layer requires a create, read, update, and delete (CRUD) paradigm which will be responsible for housing any data methods that would be used to communicate with the data store. Using a code-first object mapping approach will allow for easy creation of data models in a database.

Data Store Layer

Any information that needs to be stored will be stored within the data store layer. The data store contains multiple databases that store the domain objects.

Application Flow

Data input will follow through the layers sequentially from the user interface to the data store and back to the user interface. Development of features will also reflect this and all features will be built from the user interface to data store while keeping user interactions as a core priority.

Cross-Cutting Concerns

Error Handling

Errors will be handled primarily in each abstraction layer. Exception management will be best implemented at the contact point between layers to support debugging and system maintenance. Besides that, Error Handling as a cross-cutting concern will be the last catch for any missing scenarios.

Once handling an error, developers will evaluate the scenario and consider if the caller can handle it. If the caller has no way to handle it and it's not critical, log the error and let the application throw an exception. If it's critical, try to have a return value that still makes sense to the caller while indicating the exception[1].

Logging

System wide logging will be used for system monitoring and user analytics. Each layer will communicate with a logging service that is stored separately, ensuring access when a service is down. This logging architecture will require a separate backend to store and query logs. The logging service will have different log types and levels for categorizing the log data input stream.

Security

Each layer will have a separate security measure to ensure that only people with access should have access to each layer.

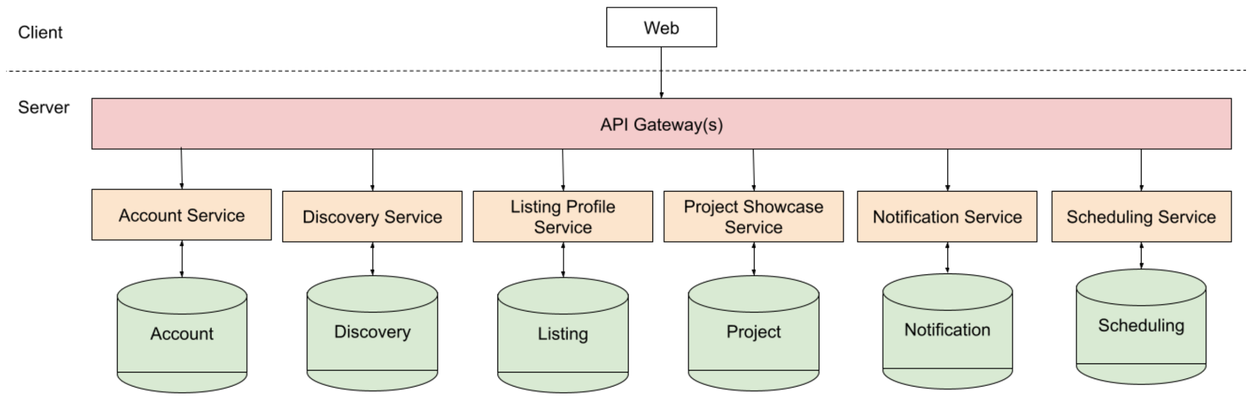
Since the user interface layer will handle most of the user interactions, it will be the layer that will be the most heavily guarded. Any type of user input will be sanitized and filtered. Any communication between the client and server should use appropriate response headers that can be verified on the service layer.

In the data access layer, to connect to the database, an authorization token will be provided to ensure people with the credentials are able to access the database.

On the development side, deployment keys will be created to prevent unauthorized deployments of the webapp to the server.

[1] Tao, L. (2019)

Application Architecture



Each of the microservices are self-contained and running in parallel. This allows for horizontal scalability for each of the microservices and dispersed deployment. Horizontal scalability provides an infrastructure which is flexible enough to adapt to seasonal changes in server requests while dispersed deployment ensures that each service is not impacted in the case of another server's outage or disruption. Running services in parallel also ensures reliability and resiliency as a service failure may result in degraded performance, however a process can still continue to function relying on other microservices to handle the bigger load.

In Hubba, all server and client communication is handled through API gateways which have endpoints for communication between the client web interface and the server's microservices. Each of these microservices handles specific tasks related to the service. Microservices that have data to be stored will have an additional database for write and read access.

References

Approved Request for Proposal Document, version 1.4 by professor Vong, 10/01/202

Tao, L. (2019, July 15). *The error handling done right*. Medium. Retrieved October 5, 2022, from https://medium.com/swlh/the-error-handling-done-right-d19ffca2656f