Team: The Musketeers
Andrew Soth 014248453
Hardit Singh 011635245
Jennifer Nguyen 013012543
Luis Gonzalez (team lead) 014707301
11/1/2018

High-Level Design Document

ParkingMaster

Table of Contents

1. Overview

Purpose

This document provides an overview of the high level design of the ParkingMaster application. The Musketeers will maintain a high level design document so that the stakeholders and new onboarding developers can easily digest the flow and architecture of the ParkingMaster system.

Scope

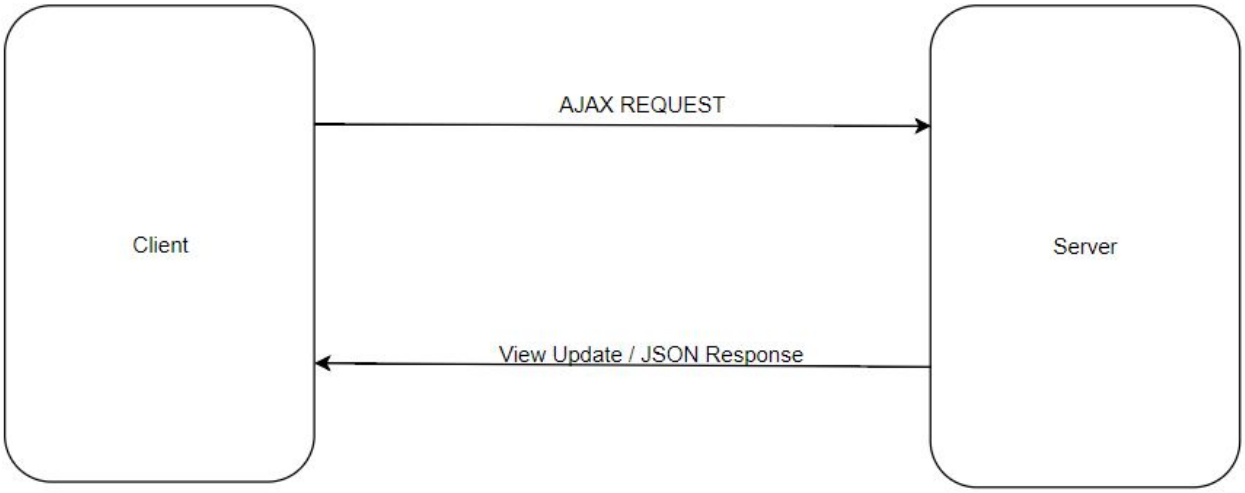
ParkingMaster is required to run on both desktop and mobile devices, so we have decided to design the project as a single page application. This high level design documentation will focus on single page architecture, which will allow our web application to be more responsive.

2. Design

Guidelines

To keep our code base organized and easier to test, we will implement the SOLID programming guidelines. Implementing these principles in our programming will make our code base more maintainable and require less code modification when applying new classes or features.

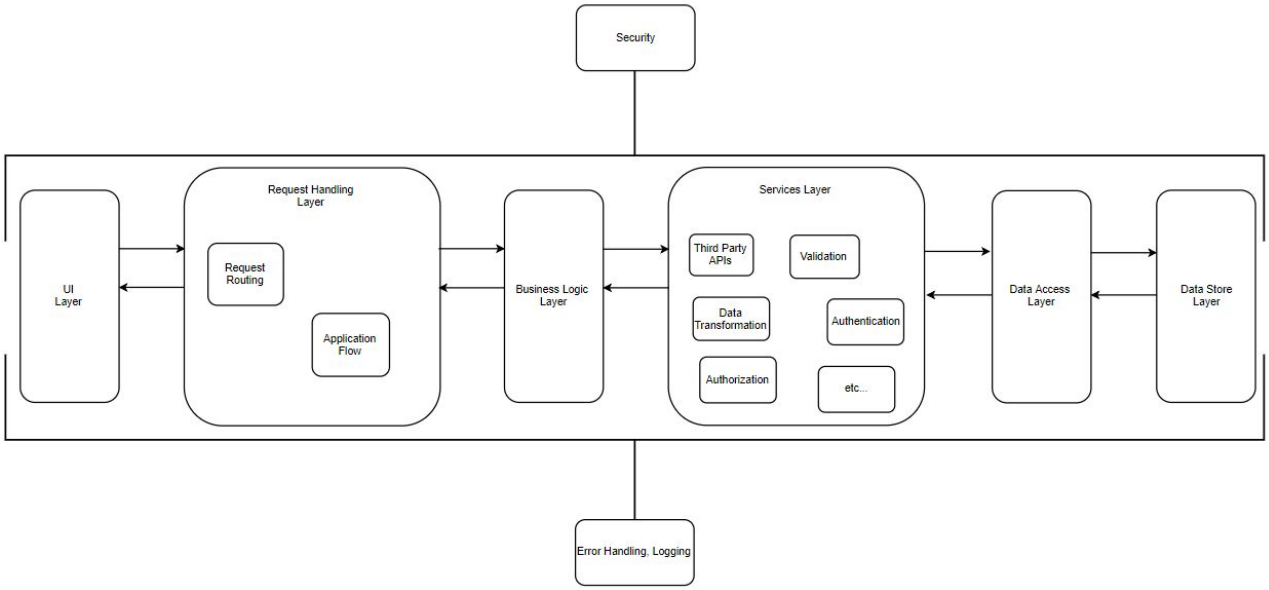
Another design choice we will be making early on in the project is to follow the Web API architecture. We decided to go with WebAPI because our application does not require any HTML/View rendering directly from the controller, just constant data requests using AJAX. We also need to provide for both desktop and mobile compatibility, so WebAPI will be beneficial for this application.



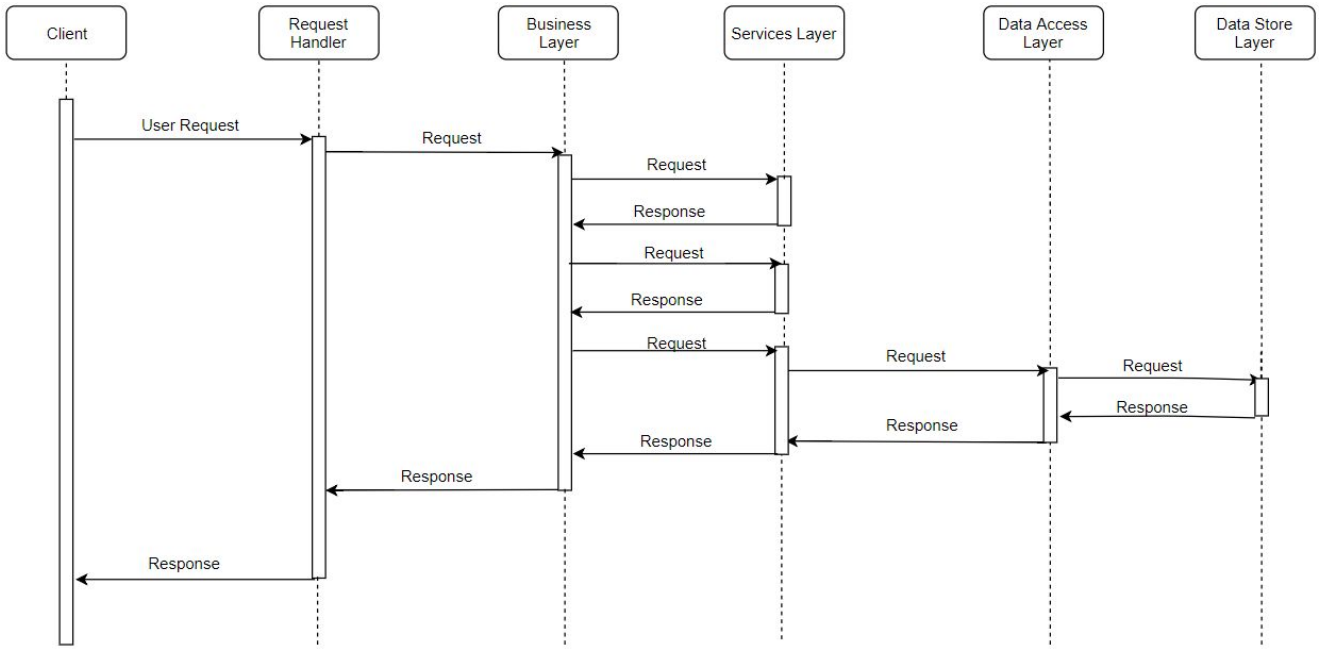
General Design

Our system mainly consists of a presentation/UI layer, request handling layer, a business logic layer, services layer, data access layer, and data store layer. This design will allow us to be more scalable and less susceptible to technology change, because if we need to make any code or technology changes, it will not require each layer to be modified.

Abstraction Flow Diagram



User Request Sequence Diagram



In this sequence diagram, we are displaying the flow of a single user request with respect to the usage of the layer. The business layer can call one or more services to satisfy the user request, hence the multiple requests in between the business layer and services layer.

An error could occur in each of the layers except the data store. If an error occurs in any of the other layers, it will pass the exception object starting from the point of failure, which would pass each layer and eventually get to the RequestHandler to be logged.

User Interface Layer

Goal -
The goal of our application is to make parking a lot less stressful, and to do that we need a UI that will make it convenient for our users to perform inputs and view data.

Input -
The input to this layer is any form of user action or input.

Output -
The output of this layer is an updated view.

Security -
Authorization and sessions are major components in this layer. Security needs to monitor if the user is active and authorization decides what features the user can access on screen. Security should also handle sensitive information being displayed in the UI.

Error handling -
If an error occurs in this layer, the user will be prompted with a form of client or server error, depending on the issue, and the error will be handled without breaking the application.

Request Handling Layer

Goal -
This layer assists with application flow and request routing which ends up triggering our back end.

Input -
The input to this layer is the actual request triggered by a user action.

Output -
The layer routes the request and triggers the business logic.

Security -
Authorization and session checking

Error handling -
The request handler layer will deal with server errors and authorization errors.

Application Flow Component -
The application flow component contains function that are particular to direct the flow of the application, such as functions Application_Init or Application_Error

Business Logic Layer

Goal -
The business logic layer holds and handles the application's business logic.

Input -
The input for this layer is a trigger request from the Request Handling layer.

Output -
Based off the incoming request, the business logic layer sends one or multiple requests to the service layer. The business logic layer returns data back to the request handler.

Security -
The security checks performed in this layer is again session checking, since we will possibly be performing multiple requests to the services layer, we need to secure the data access between the requests and responses.

Error Handling -
The business logic layer needs to be able to handle no server response errors, internal server, and request errors.

Services Layer

Goal -
The goal of the services layer is to have a separation of concerns in the entire system, but also separation from other services. The different components in the services layer are called as needed, depending on the incoming request.

Input -
Data coming from the business logic, which is then routed to the appropriate service to handle the request.

Output -
Depending on the request, the output can be a direct response back to the business layer from the services, or to satisfy the business logic even further, the service component needs to request data from the data access layer.

Security -
Security checks performed in this layer need to secure the data being passed. Authorization checks, session checks.

Error Handling -
The errors occuring in the layer can range from server error to validation errors that a service component can return.

Data Access Layer

Goal -
Our application will need a way to work with the data that comes along with our classes, and our data access will allow us to manipulate objects instead of referencing the database tables and columns directly.

Input -
The input into the DAL is data from a service request.

Output -
An object that allows the data access layer to communicate with the data store directly..

Security -
The request needs permission to access data from the data store, we need to authorize and match sessions.

Error Handling -
No data errors, server errors can occur, so in our object that is returned back, it will contain the error message within the object, and get passed up towards the top layers.

Data Store Layer

Goal -
Contains the data sources that hold tables of data that are available on demand.

Input -
The input into this layer is an object coming from the DAL which is converted into executable SQL in the data store.

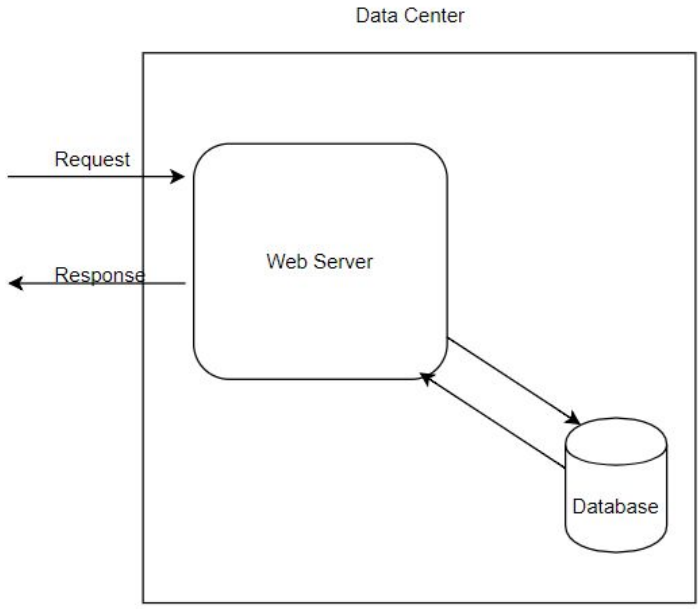
Output -
Some dataset that is passed back into the object of the data access layer.

Error Handling & Logging

Throughout the flow of our application, we decided to go with logging at a centralized location, rather than at logging at the direct source of the error. The centralized location would be within the request handling layer, as we perform application flow there. The error would bubble up to our top layers.

The way we will be passing errors is a JSON object that will have fields such as an array of data content and an error field. If the error field is not empty, then we can assume that an error has occurred.

Network Infrastructure



For our current system, we plan to have our web server and database contained within a data center. Requests would come into the web server, and depending on the request, it would require a transaction between the database. The database will exist outside of the web server; this will allow for greater separation of concerns and no overlapping procedures.

If future infrastructure upgrades are needed, our design will be capable of adding in multiple databases and load balancers to handle requests.