High-Level Design

October 14, 2022

Issued by:

Algorithmic Alchemist

Team Lead

Sierra Harris

Team Members

Bryant Lam

Abhay Solanki

Faisal Al Muharrami

David Chan

Github: https://github.com/abhay772/AA_Senior_Project/

Version History

Version #	Date	Reason for Change
Version 1	10/5/2022	Original Document
Version 2	10/12/2022	Updated design and added more details.

Table of Contents

Glossary (definitions/abbreviations)

- Property Manager: A property owner, commercial property manager, and landlord, anyone that manages the property.
- Service Provider: An organization or company that provides a service.
- PMtoGo: Algorithmic Alchemist proposed single-page web application for Property Maintenance and Renovation.
- DAL: Data Access Layer
- DS: Data Store
- DSL: Data Store Layer
- SL: Service Layer
- PL: Presentation Layer
- TLS: Transport Layer Security

Overview

The high-level design document will describe the general system and architecture of PMtoGo. This document will address the system as a whole and present a visual estimate of the system's architecture. In addition, this includes how the system handles logging, errors, and input validation at a high-level design.

We are going to use a layered architecture. A layered architecture is an architectural pattern that is composed of several isolated layers, each layer only has access to the layer next to them, which makes it easier for testing and reusability in the future at the cost of reducing performance. The layers are separated into 5 layers: presentation, manager, service, data access, and data store.

- Presentation Layer:
    - This project will be presented as a Single Page Web app.
    - All the UI and UX components will be encapsulated by this layer.
    - This layer will be mainly client-facing Architecture
- Manager Layer:
    - This layer will control which microservice will be called.
    - It will receive input from the user's interaction with the presentation layer and decide which microservice will be called to properly complete the given task.
- Service Layer:
    - Service layer will encapsulate all the business logic for the domain model.
    - This layer handles all data manipulation, accessing, and storing by calling the DAL.
    - And the processing, analysis, and calculation for all the services will be passed on to the Presentation Layer.
- Data Access Layer:
    - DAL or Data Access Layer will facilitate the extraction of data, from the Datastore, in the form of objects. Instead of data structured as rows, and tables
    - This allows for the abstraction of Business Logic in the service layer.

    - This layer makes adding and implementing new data stores easier.
- Data Store Layer:
    - Datastore layer will consist of one or more types of databases.
    - This will be used to store and access data.

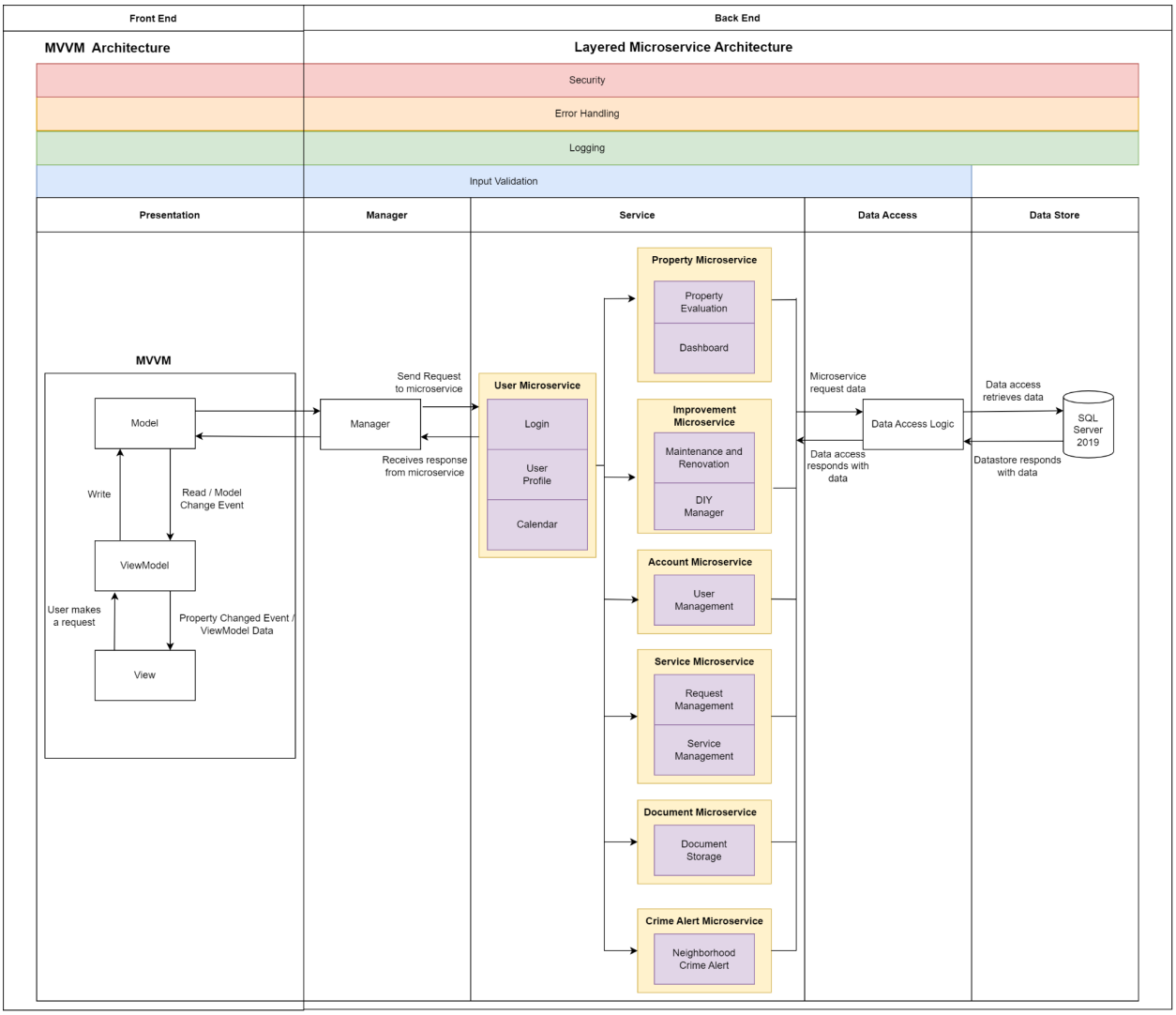
Architecture

This project will follow a Layered Microservice Design Pattern. The reason we choose this type of layered architecture is due to every layer being isolated which makes it easier for testing and reusability in the future at the cost of reducing performance.

This Project will use MVVM or Model - View - ViewModel for the front end. This structure allows for the separation of the Model and View. Where if the Model is changed the View does not need to be updated, and vice versa. The ViewModel separates the View from the Model so that the view is not dependent on any specific model form. This makes the MVVM model easily maintainable and scalable. And MVVM is also a good tool while testing services and functionalities during development.

Over every layer, we have security and error handling to keep data secure and input validation over the presentation, manager, services, and data access layer. We chose to handle errors in every layer because an error can happen at any and all layers. By doing so we are able to tend to errors on each layer and notify users of any errors. We also chose to secure each layer of the architecture to ensure the proper authorization and authentication of the users and system is appropriate to their layer. This ensures no user can access data they do not have authorization for and that the system is secure from outside attacks. We decided to only validate input on the first four layers because the only input to the data store will be by the DAL.

High-Level Diagram



Bounded Context of Microservices

- User Microservice
    - We purposely decided it was best to chain this microservice in front of all the others due to the fact it will be needed in all other microservices. If we were to duplicate the code inside all microservice it would still require the user to log in each time they switch the microservices. We chose the simpler route of having it chained in front even though it breaks conventional microservices rules and causes another major point of failure if User Microservice goes down but it will allow simpler user usage.

    - This microservice will consist of the functionality of login, user profile, and calendar. Login is required for all other microservices for authentication, a user profile will be needed to access data from other microservices, and a calendar will also be required to access data from other microservices.
- Property Microservice
    - Property evaluation and dashboard will be combined in a single microservice because the dashboard will need to use property evaluation to give an up-to-date estimate of the value of the property.
    - The Property Microservice is dependent on the user microserver. Refer to “Bounded Context of User Microservice”.
- Improvement Microservice
    - Maintenance and renovation is combined with the DIY manager because the implementation of a smart saver will be used in the DIY manager for marking the prices of materials.
    - The Improvement Microservice is dependent on the User Microserver. Refer to “Bounded Context of User Microservice”.
- Account Microservice
    - The user management functionality is not dependent on other features except User Microservice so it will not be combined with other features.
    - The Account Microservice is dependent on the User Microserver. Refer to “Bounded Context of User Microservice”.
- Service Microservice
    - Service management is the property manager users setting up services and the request management is the other side which is the companies who will interact with the service request. Since both features need to interact with each other they will be grouped together.
    - The Service Microservice is dependent on the User Microserver. Refer to “Bounded Context of User Microservice”.
- Document Microservice

    - The document storage with optical text search functionality is not dependent on other features except for User Microservice so it will not be combined with other features.
    - The Document Microservice is dependent on the User Microserver. Refer to “Bounded Context of User Microservice”.
- Crime Alert Microservice
    - The crime alert functionality is not dependent on other features except User Microservice so it will not be combined with other features.
    - The Crime Alert Microservice is dependent on the User Microserver. Refer to “Bounded Context of User Microservice”.

Why Microservices architecture?

We chose to go for a layered architecture because it would work best as the architecture for PMtoGO and here are the advantages:

- Horizontal scaling is a lot easier if a single server becomes overloaded it can just open another server containing the same microservice.
- As the services are independent, increasing the scope is easier as a service can be easily added/updated.
- Website stays online even after a failure of a service because of loose coupling.
- Scope of each microservice is defined making the codebase easier to understand.
- Isolated microservices make testing a lot easier.
- Security can be implemented at every layer.

We acknowledge the drawbacks of said architectures in terms of

- Communication between services is more complex.
- Adding a service requires adding data stores and logging.
- If the microservice servers are far away from each other then it can increase the delay from traveling server to server
- It will require more hardware and be more complex to implement
- If one layer doesn't work the whole system will have difficulties.

Security:

- A website firewall will be used to filter invalid requests.
- Users will be advised to enter strong passwords and that will be validated.
- Every layer will perform authentication to determine access.
- Users' access will be limited to their authorized functions according to the business rules.
- The site will have a daily backup to minimize damage in case of errors or failures.
- The data between the host and client will be encrypted during transit.
- During the testing phase, the website will undergo security tests to ensure no exploits or vulnerabilities are present.
- Passwords and all sensitive details will be stored in an encrypted database.
- TLS 1.3 will be used to encrypt data between the client and server.

Input Validation:

- Input validation will be done at all the layers, except Data Store.
- This makes the layers reusable in future projects. As the input validation is done locally on every layer, the layer will not be dependent on adjacent layers for input validation.
- This does make processing slower with repeated data validation checks at each layer.
- We chose not to handle input validation in the datastore because only the data access layer will have access to it.
- The Data Access Layer will be designed to work with a relational database and will handle the input validation for the Data Store.
- Therefore, if a different kind of data store is used in the future, the data access layer will have to be updated to do the input validation for that kind of data store.
- A whitelist will be applied for input validation, which checks for,
    - Correct format,
    - input data type,
    - data consistency,
    - Constraints, i.e. required by the business rule

    - and input range allowed.

Error Handling:

- As all the layers are prone to errors, we will be implementing error handling in each layer.
- Although it involves repetition of code and reduced performance, this ensures a better understanding when troubleshooting.
- It allows the layers to be independent and reusable.

Logging:

- Logging will be performed in each layer, to make debugging faster.
- In the presentation layer, we will be logging any errors and invalid inputs.
- In the manager layer, we will be logging any errors and invalid inputs.
- In the service layer, we will be logging any errors and invalid input along with how much each microservice is used and so if there is any bottleneck we can expand and open a new server assisting in that microservice. In addition to successful and unsuccessful login attempts
- In the data access layer we will be logging any errors, invalid inputs, and status of requests.

References

"Application-Manager-Driver Architecture." *F´*, https://nasa.github.io/fprime/v1.5/UsersGuide/best/app-man-drv.html#:~:text=The%20manager%20layer%20manages%20a,at%20all%20by%20the%20Application

Baeldung. "Layered Architecture." *Baeldung on Computer Science*, Baeldung, 11 Nov. 2021, www.baeldung.com/cs/layered-architecture#:~:text=What%20Is%20a%20Layered%20Architecture%3F&text=In%20these%20frameworks%2C%20components%20that,part%20of%20the%20overall%20system

"C5: Validate All Inputs¶." *C5: Validate All Inputs - OWASP Proactive Controls Documentation*, https://owasp-top-10-proactive-controls-2018.readthedocs.io/en/latest/c5-validate-all-inputs.html.

Dang, Anh T. "MVC VS MVP VS MVVM." *Medium*, Level Up Coding, 14 Oct. 2020, https://levelup.gitconnected.com/mvc-vs-mvp-vs-mvvm-35e0d4b933b4

"Data Validation." *Wikipedia*, Wikimedia Foundation, 22 July 2022, https://en.wikipedia.org/wiki/Data_validation#Data-type_check

"Exception Handling." *Wikipedia*, Wikimedia Foundation, 17 July 2022, https://en.wikipedia.org/wiki/Exception_handling#Exception_handling_based_on_design_by_contract

GeeksforGeeks. (2022, June 9). MVVM (Model View ViewModel) Architecture Pattern in Android. Retrieved October 14, 2022, from https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/

"Input Validation." *Input Validation - an Overview | ScienceDirect Topics*, https://www.sciencedirect.com/topics/computer-science/input-validation

Lteif, Georges. “High-Level Solution Design Documents: What Is It and When Do You Need One.” *Operational Excellence in Software Development*, 8 Sept. 2022, https://softwaredominos.com/home/software-design-development-articles/high-level-solution-design-documents-what-is-it-and-when-do-you-need-one/

MildWolfieMildWolfie 64511 gold badge55 silver badges1111 bronze badges, et al. “How to Clearly State to the User What Characters Are Valid.” *User Experience Stack Exchange*, 1 Aug. 1961, https://ux.stackexchange.com/questions/57491/how-to-clearly-state-to-the-user-what-characters-are-valid.

“MVVM – Advantages.” *Tutorials Point*, https://www.tutorialspoint.com/mvvm/mvvm_advantages.htm

Publications, Manning. “The Layers of a Cloud Data Platform.” *Manning*, 9 Oct. 2020, https://freecontent.manning.com/the-layers-of-a-cloud-data-platform/

Published By - Kelsey Taylor. “What Are Microservices and Their Advantages and Disadvantages?” *HitechNectar*, 5 Oct. 2022, https://www.hitechnectar.com/blogs/5-pros-and-cons-of-microservices-explained/

Richards, Mark. “Software Architecture Patterns.” *O'Reilly Online Learning*, O'Reilly Media, Inc., https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html

“Service Layer.” P Of EAA: Service Layer, https://martinfowler.com/eaaCatalog/serviceLayer.html

Smith, Jon, et al. “Using Entity Framework with an Existing Database: User Interface.” *Simple Talk*, 17 May 2021, https://www.red-gate.com/simple-talk/development/dotnet-development/using-entity-framework-with-an-existing-database-user-interface/

Website security guide: Secure & protect your website. Sucuri. (2022, July 27). Retrieved October 5, 2022, from https://sucuri.net/guides/website-security/

Vatanik Vong, Lecture on MV* structures, CECS 491A Sec 04, CSULB, November 28, 2022.