CraftVerify

High-Level Design Document

Team Wanderer

GitHub Link: https://github.com/JPJ-5/Senior-Project

Team Members:

Jason Jitsiripol (Team Leader)

Steven Hall

Parth Thanki

An Nguyen

Khuong Nguyen

Joshua Reyes

Shane Kerr

11/17/2023

Version	Description	Date
1.0	HLD Creation	11/08/2023
1.1	Added some information on front-end and back-end	11/13/2023
1.2	Revise the high-level design.	11/17/2023
1.3	Add architecture to the back end	11/22/2023

Table of Contents

Product's Overall Architecture

Client-side/Front End

We will use HTML5, CSS 3 with Bootstrap 5.1.0 framework, and ReactJS library for JavaScript language following ECMAScript 6 standard to build the front end. The website will be hosted using an AWS EC2 server.

- MVVM
    - View
        - The View portion of CraftVerify will show elements such as buttons, a search bar, and pop-ups for the user to interact.
        - Our View is also responsible for recording the events and passing those events to ViewModel.
    - ViewModel
        - The ViewModel will take the events from View and check for logic to trigger the appropriate action between the ViewModel and the Backend or Server-side.
        - View Model will handle the UI logic, Handle Data from Server, and check user inputs.

Server-side/Back End

We will use C# 10+, .NET 8, ASP.NET Core 8, MariaDB 10.x (MySQL fork), and IIS 10+ for the back end.
We will adhere to the Microservice architecture, which comprises 5 services as follows: User Administration Service, Product Management Service, Analytic Service, Notification Service, and Log Service.

- Microservices:
    - User Administration Service: In charge of the user account.
        - Account Creation, User Account Deletion, User Account Recovery, User Profile Update, User Management, User Privacy Control,
    - Product Management Service: In charge of craft items, and trading.
        - Item Listing + Creation, Add to Wishlist, Remove from Wishlist, List and Sort Wishlist, Shopping Cart, Price Range Sorting, Product Editing, Product Deletion, Inventory Stock List, Search Capability, Autocomplete Suggestions, Results Sorting, Score System, Feedback System, Feedback Sorting, Report System, Auction Initiation View, Auction Completion View, Item Offer.
    - Analytic Service: In charge of analytic reports.
        - Usage Dashboard Analytics, Financial Progress Report,

    - Notification Service: In charge of informing, and emailing users.
        - Send emails to users in needed situations, such as account registration, OPT, profile change, email change, and trading info.
    - Log Service: In charge of keeping users and system events.
        - In charge of Logging and Recording.

For each microservice, we will have the following layers to keep it organized:

- Manager Layer: This layer is for enforcing all our business rules as written in the BRD. Errors will be handled here most of the time.
- Services Layer: This layer will consist of a collection of all reusable methods, and classes. These reusable modules will not implement business error handling.
- Data Access Layer: This layer is in charge of reading and/or writing data to the data store for all needed operations. We will use ISQLDAO to adhere to the SOLID principles across all the connections between each microservice and the data store to help increase the robustness of the system.
- Datastore Layer: The layer represents the users not have direct access to the stored data, and only users with proper access which is granted from the user role can use the methods to interact with the stored data. The database engine that we will use to manage and interact with our data store is the MariaDB.

Abstractions and flow of control for cross-cutting concerns

Error Handling

- UI Layer
    - We will inform the user with a detailed message when an error occurs during their session of using services.
- Manager Layer
    - Error handling within the manager layer will revolve around business rules being violated. The bulk of the errors that are created here will be taken care of with try/catch blocks as well as data entry checks in the code.
- Services Layer
    - For the reusable generic methods and classes, we don't apply business rules here but will handle other types of error if possible.
- Data Access Layer
    - Error handling within the data access layer will include catching and handling time-out errors, invalid data to be entered into the data store, or invalid queries to the data store.

- Data Store
    - Error handling within the data store will include I/O with the data store itself. Examples are errors encountered when interacting with the data store with valid input or valid queries. The errors are with the data store, not the access to it. We can try to mitigate these errors using try/catch blocks within the code, narrowing down the cause.

Logging

- UI Layer
    - Log all of the events performed by the users or the system.
    - Any logging of user data given by the user will need permission to view.
- Manager Layer
    - The system will log and keep track of the user Authentication and Authorization attempts.
    - The system will log and keep track of the errors within the manager layer for Analytic Reports and future debugging.
- Service Layer
    - The system will log and keep track of the user Authentication and Authorization attempts.
    - The system will log and keep track of the errors within the service layer for Analytic Reports and future debugging.
- Data Access Layer
    - The system will log all services that attempt to access the data store.
    - The system will log any time-out error in the case of the services cannot reach the data store.
- Data Store
    - The system will log all of the actions along with the data that is being used or affected.

Security(Authentication & Authorization)

- UI Layer
    - Only the public UI elements will be available to be viewed by unauthenticated users. The majority of features will require authentication to access.
- Manager Layer
    - We will check username and password for authentication and authorization before processing the microservices. Authenticated users may use the features within the business layer based on their user role.
- Service Layer
    - We will check username and password, authenticated users may use the features within the business layer based on their user role only.
- Data Access Layer

    - We continue checking usernames and passwords and provide/restrict services based on user roles. This is a safety feature to ensure that users who don't have the authorization to access data don't even get the option to interact with the data store or send queries to it.
- Data Store
    - Define the read-write permission for each service based on the user roles.