Breazy Fit

High-Level Design

BREAZY FIT

Github Link:
https://github.com/ChazArvizu/CECS491_Hexadecimators.git

Team Name: Hexadecimators
Team Leader: Chaz Arvizu
Team Members: Carlsean Claricia, Tania Adame, Tyler Kelsey, Andrew De La Rosa, Sean Iida

Submission Date: 10/05/2022

Version History

Version 1.0 - Initially Created: 09/28/2022

Table of Contents

Page Number

1. High-Level Design Scope

This document will explain the design architecture that we will use for our project. The information in here will describe how we will design our application and its components and will also go over the system we will use, how API's will be integrated, the application flow, technology architecture, layer and data models, and other types of requirements needed for our design.

2. Intended Audience

This document is intended for our client so that they may see exactly how we are going to design our project. Since the document will show the different types of designs and components we will use, as well as other very important design principles, it is important for our client to see this document so that they have a better understanding of what we are doing and how we will do it. Additionally, this document will also benefit our team as it is a resource for us to look back on when designing our project since it goes into detail of the different methodologies and technologies we will be using.

3. System Overview

Breazy fit is an application that is designed to give the user an all-encompassing tool for fitness and nutrition. The application is compatible with most modern browsers, with a recommendation of Chrome 104.x (64 bit). Security will be implemented through each of the layers within our layered architecture. A Relational Database (RDBMS) will be used to store the user's data. Breazy Fit's interface takes in numerous user inputs (i.e. calorie intake, workout regimine, etc.), parsing and analyzing the data to give the user updated information. User Access Control changes the user's experience depending on the type of account created.

4. Application Design

The application will be designed around a main homepage. After authenticating, the dashboard will change to user-specific data. A menu at the border of the screen will allow the user to navigate between different functions of the application. Users will be able to edit their profile-specific data by clicking their icon in the top right corner. The bottom of each page will have a list of resources to help get in contact with the developers along with links to find out more information about the team and the application. The application will follow a SPA (Single Page Application) architecture to stay more responsive and lightweight. Majority of the logic will be done through the browser and not the server. Because of the profile-driven nature of the application,

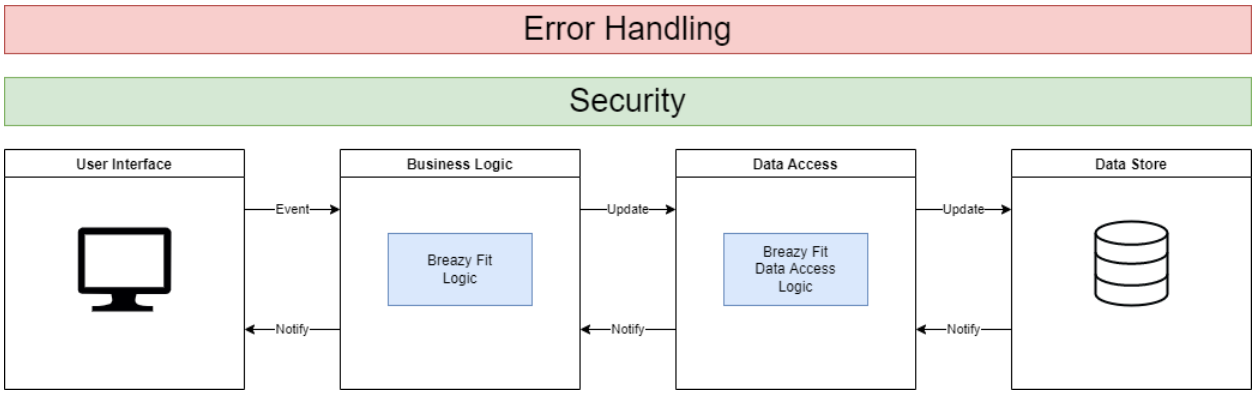
features such as authentication and registration will be conducted through a separate page to increase security.

5. Architecture

The architecture will describe the layout of how the application will be constructed and how data will flow.

5.1 Layered Architecture

Breazy fit will utilize a layered architecture in the design of the application where the User Interface, Business Logic, Data Access, and Data Store layers are split from each other.



5.1.1 User Interface Layer

Function - This layer is responsible for displaying the applications data and all the elements that a user can interact with. An initial HTML file payload will be received by the browser that will fill the browser page with the Breazy Fit UI. Updates to the page will be made with AJAX requests which are smaller payloads that don't require the page to reload.

Error Handling - Any error that is encountered by a user will have a message displaying what specifically went wrong. Some potential errors could be invalid user input, email or password.

Input Validation - Check to see if the input fields have appropriate characters and gives immediate messages to the user to fix the affected fields. This includes adding constraints to what is valid as a password and a username.

Security - The UI will have security implemented within the features to not allow any unwarranted access to data. One of the security features within the UI will be a forced logout after 15 minutes of inactivity.

5.1.2 Business Logic Layer

Function - The business logic layer is the middleman between the user interface and the data access layer. Its function is to calculate and hold the rules for what type of requests are allowed to be made to the databases. The business logic layer is also in charge of making sure the right data is given back to the UI layer to create a seamless user experience.

Error Handling - Will use try/catch and throw exceptions so that in the event that an error were to happen, the user and developers know what happened and where it happened and it will prevent the application from completely crashing.

Input Validation - Checks to make sure that the right functions and classes are called when selecting certain parts from the user interface. Also ensures that it accesses the right datasets within our data access layer.

Security - To ensure security, this layer will not be visible to the users wanting to use the application and will only be able to view and access what it allows them to access. This layer adds a level of security due to its regulation of the input data. Exploits become less likely the more constraints are added.

5.1.3 Data Access Layer

Function - The data access layer provides a consistent way to select, insert, update, and delete data from our database in a uniform manner. This allows for the business logic layer to make requests without having to deal with differing structures in the databases.

Error Handling - Errors that will be monitored and handled will include situations such as invalid data entry or a query that takes too long to return. Errors in the data access layer help mitigate the severity of the errors that would potentially be found in the data store layer.

Input Validation -In the data access layer, the input validation’s purpose is to make sure that entries into the databases adhere to their respective schemas. To create as much abstraction as possible, methods can be made to create and update profiles. This allows us to add to the database using the same procedure with every creation.

Security - We will make sure that any data that our program is trying to grab is not visible to the user. Also, any personal data that a user submits for their profile will only be accessible to the specific user.

5.1.4 Data Store Layer

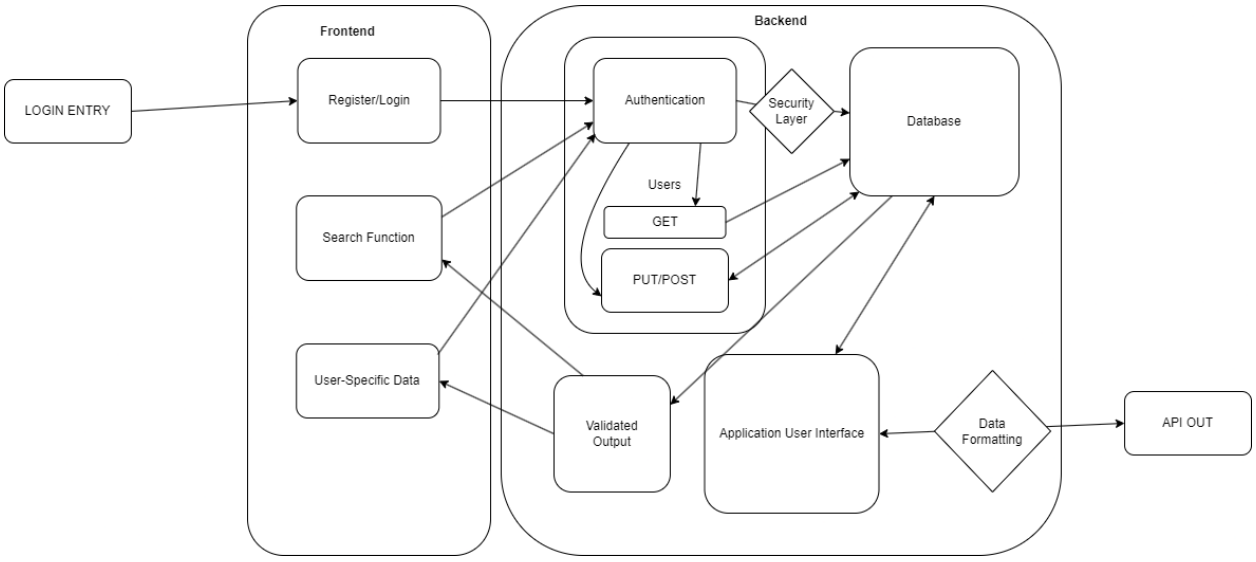
Function - This layer will be the database where data can be selected, inserted, updated, or deleted depending on queries provided by the Data Access Layer. The main database used will be a relational database, utilizing SQL.

Error Handling - Since we are using an SQL database, we will use “try and catch” along with using transactions to decide whether the database will commit or rollback depending on the circumstances and how bad the error is.

Input Validation - Checks to ensure data uses characters relevant to its category in the database by testing whether the database will commit or rollback certain additions and updates of data input using the Data Access Layer.

Security - Within the database we will implement firewalls, making sure the one requesting queries have the correct roles and permissions for whatever data it is trying to access by the Data Access Layer, and data encryption on any data that is trying to be retrieved by the Data Access Layer.

6. Data Model



This diagram illustrates the architecture of the application and paths taken in the normal user experience. Login entry is presented to the user with a graphical user

interface, which is then validated and encrypted through a security layer. Once a user is authenticated, they are given user-specific data through GET requests, along with the ability to edit and overwrite the data through PUT and POST. With multiple of the main functionalities of the app being queries to the database, all search functions will start in the frontend and will path through the authentication layer before reaching the database. All outgoing data from the database will pass through a processing layer to ensure validated data. Lastly, API use in our application will pass through a similar data formatting layer to adhere to each API's respective documentation.

7. References

"Boksi", B. P. B. aka. (2021, May 21). How to implement error handling in SQL server. SQL Shack - articles about database auditing, server performance, data recovery, and more. Retrieved October 4, 2022, from https://www.sqlshack.com/how-to-implement-error-handling-in-sql-server/

VanMSFT. (n.d.). Securing SQL server - SQL server. SQL Server | Microsoft Learn. Retrieved October 4, 2022, from https://learn.microsoft.com/en-us/sql/relational-databases/security/securing-sql-server?view=sql-server-ver16