High Level Design

Team Gen ChimpanZ’s

October 5, 2022

Team Leader

Mark Fastner

Team Members

Liam Joseph Abalos

Anh Huynh

Justin Le

Aster Lee

Brendan Paing

Github: https://github.com/markfastner/Nuclei

Table of Contents

Introduction

Overview

This document will outline the architecture of Nuclei, our proposed sports training management system. Modeling the architecture at a high level will showcase how each component of our system will interact with each other, allowing our team to identify and revise any design-level issues before the development phase. As a high-level overview, this document also serves as a reference guide for overarching milestones to anticipate in the development timeline.

Scope

This document will detail the structure of each architectural component and its relation to other components in our system. Additionally, standards will be defined to ensure that all developed components follow the same format for easier future maintenance. Architecture will be described at a high level, meaning that no specific implementation details will be described.

Design Details

Main Design Features

Our product will consist of a single-page web application. A single-page application attempts to minimize the size of data requests and responses after initial loading of the application. From a client perspective, this would reduce the amount of perceived delay and stutter encountered in the user interface. The following components will comprise our application architecture: user interface, connection between front-end and back-end, application models, application logic, data access, and data store.

Microservices

Each microservice is a containerized runtime environment that is specialized for certain tasks. Because each task is self-contained within its respective microservice, if more throughput is needed from the task or if there is a higher influx of traffic, its parent microservice can be scaled by duplicating it and propagating it throughout cloud servers. On a related note, if a microservice is scaled horizontally, it should have its own environment instance, versus if a microservice was scaled vertically (the optimal choice), it would take advantage of most cloud hosting service's "pay-as-you-go" feature by simply allocating resources to match the demands–reducing costs.

Microservice A:

Features as the primary environment where end-users will interact with.

Microservice B:

The environment which handles computation of core business and application logic.

Microservice C:

An environment that handles persistent data storage, data accessing, and data mutation.

Technology Architecture

Our architecture will follow the Clean Architecture for building an enterprise level and scalable application. Also, a programming paradigm we will adhere to greatly is that of a Domain Driven Design. One of the main benefits of Clean Architecture is to regulate all the dependencies between each layer of architecture. This modular approach will allow our team to modify a component without conflicting with the rest of the layers in the application, reducing the amount of time spent on code maintenance when implementing new features.

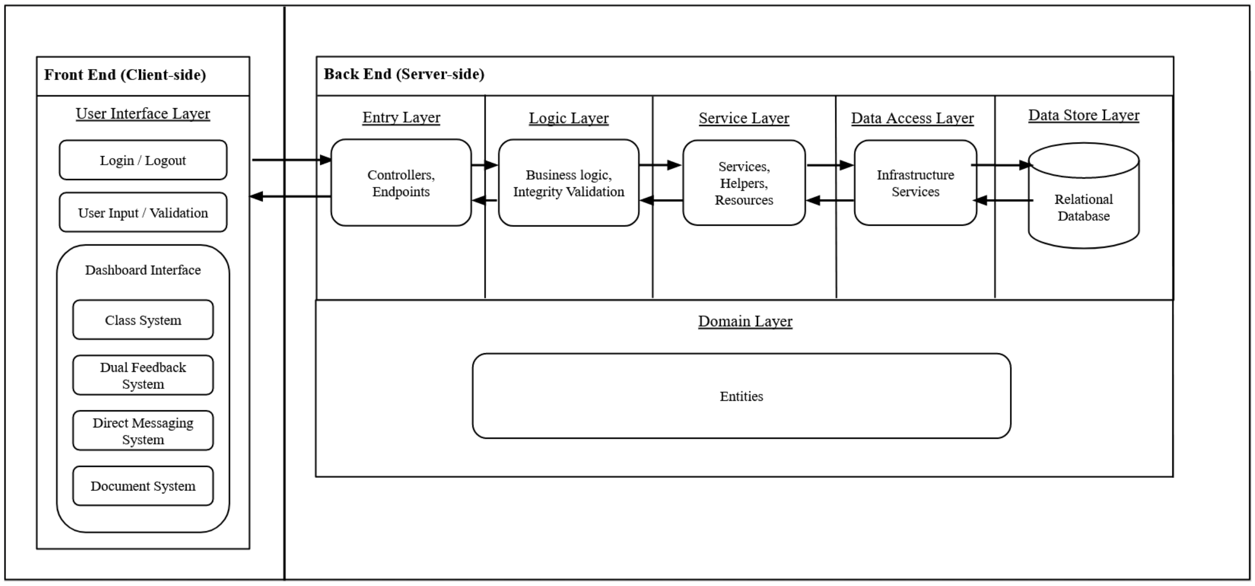

Fig. 1 Architecture and Data Flow

Front-End

User Interface Layer

The user interface layer is the main layer the client browser will use to interface with the Nuclei application. After successfully authenticating, the user will gain access to Nuclei and its application features. Nuclei's features in this current iteration include a dashboard interface, a class system, a notification system, a dual feedback system, a direct messaging system, and a document system. The user interface layer will display all of these features for the user to interact with except for the notification system which has functionality outside of the application. To ensure the integrity of our application, input data will be validated before it is sent to the back-end.

Back-End

Entry Layer

The entry layer serves as a connection layer between the front-end and back-end sections of our overall application. Business logic will not be defined in this layer, but instead will be sent to the Business Logic layer for handling. This layer requires that each of its entry endpoints be authenticated and secured to prevent malicious usage. When necessary, the entry layer will also map user input parameters into a valid format to ensure correct functionality of the other back-end layers.

Business Logic Layer

The Business Logic layer is where the core business logic will be defined. This layer takes the models defined in the Domain layer and builds business rules off of them. These business rules may use services stored in the Service layer to assist with any computations or be provided additional resources, then will return a response back to the Entry layer.

Domain Layer

The Domain layer is where the application models will be stored. Custom error types based on each application model are also defined here. According to the Clean Architecture model, the Domain Layer should not contain any dependencies to other layers. Therefore, business logic pertaining to these entities will not be defined here. Data requests and responses will also not flow through this layer. Placing our application models in a separate abstraction layer allows us to easily define new feature models for future development without affecting the functionalities of current features.

Service Layer

The Service Layer will store our services for use with our application. These services will be invoked by the Business Logic Layer whenever a business rule is triggered. Additionally, this layer will store utility functions outside of our services for the Business Logic layer to use.

Data Access Layer

The Data Access layer serves as the primary data access layer thus establishing a connection between the Nuclei application and the Nuclei data store. This layer will contain logic to map data store responses into a format the Business Logic layer can use. There will also be logic within this layer to update the data store from the application side whenever there is a request to create, read, update, or delete data. To ensure that data requested and received between the application and data store are the same, the data access layer will validate data store modification requests and will only execute these actions if they are valid.

Data Store Layer

The data store layer will act as persistent storage to contain the information inputted and created by the users of the Nuclei application. A relational database will be used as our product feature entails multiple user types and interactions between different user types. The data store layer encompasses the entire enterprise schema of Nuclei, such as account information and other information related to our features. All audit logging used for development purposes will be stored here as well.

Tools Used

Developer Environments

- Visual Studio
- Visual Studio Code
- SQL Server Management Studio

Front End

- React.js (Javascript)

Back End

- .NET 6.0.X (C#)
- SQLServer

Web Server

- Azure

Standards

Security

The User Interface layer will focus its security through its user-registration system. The application's features can only be accessed by valid users and only valid users of the frontend application are authorized to make requests to the backend.

The entire backend application will emphasize its security over two services: the Entry layer and Data Store layer. Since the only way to access subsequent layers of the Entry layer besides the Data Store layer, is through the Entry layer itself, a bottleneck is formed for the flow of data.

Therefore, by securing the Entry layer with authentication and authorization, the entire backend is safe. Furthermore, it is mandatory for persistent data to be secure and authorized to entities with the correct credentials, thus the database requires authentication.

Error Handling

Since the User Interface layer parses little business logic, the primary source of errors will originate from the backend. So then the frontend is responsible for interpreting those errors and displaying them to the user interface so it is understandable and can be corrected.

Typically in the backend, errors and exceptions are expected to be thrown in the Logic layer and Data Access layer; those errors are routed to the Entry layer to be returned to the requester as a response body with a proper status code and context details.

Data Handling

Our API will establish RESTful endpoints at the Entry layer. The purpose of this implementation is so the User Interface layer can interact with said layer via the Hypertext Transfer Protocol Secure (HTTPS). Formally, the User Interface layer makes a request to the Entry layer and is returned a response.

Logging

The logging behavior will log each individual action performed within the application. More specifically, the actions performed in the lifecycle and data of an invoked endpoint, making the Entry layer the target for this behavior.

To ensure our product functions as intended during development and release, data will be logged to maintain a tangible list for future development. Any feature that changes the database such as creating data, modifying data, or deleting data will additionally create a new log, detailing timestamp and the changes made to the database based on the feature’s function. Logs will be stored in the database under its own data category.

Glossary

Term	Definition
Back-end	The part of an application client can not access, usually containing application logic and database information that will be sent back to the front-end.
Clean Architecture	A software design philosophy that separates the elements of a design into ring levels to organize code such that it encapsulates the business logic but separates it from the delivery method.
Domain Driven Design	A software design philosophy that is centered on the sphere of knowledge of those that utilize it and enables the development of software focusing effort on the high priority complex requirements
Endpoint	A particular routing point of an API which is responsible for relaying data from the API and is accessed by an internet protocol.
Front-end	The part of an application a client can view and interact with.
Microservice	A contained computing environment capable of being scaled in size and or replicated.
Relational Database	The organization of data in predefined relationships in one or more tables (relations) of columns and rows with the intent of easily visualizing the relationships between data structures
Schema	The organization of a database

References

1. https://cutewallpaper.org/24/sports-png/view-page-24.html

Version Changelog

Version	**Submission Date**	**Changelog**
1	09/28/22	Initial Draft Version
2	10/03/22	Second Draft Version ● Add new sections ○ Microservices under Design Details ○ Data Handling under Standards ○ Logging under Standards ● Combine Entry into Application layer ● Rename Infrastructure to Data Access
3	10/05/22	Initial Submission ● Move Entry back into its own layer ● Rename Application to Business Logic ● Add Service Layer ● Finish Standards section ● Edit all layer content to be more “high-level” ● Update Glossary section