CodeandVerse

Tech Stack & Report

Saeed Jamali

Contents

Overview

The **CodeandVerse web app** is a project in which I tried to implement best practices for a modern, robust, and scalable web application. I built it using a comprehensive stack of technologies to enable efficient content handling, smooth user experiences, and reliable performance. Currently, the platform serves as a personal portfolio and a blog where I can write notes and updates about the projects that I am working on. For the future steps, I intend to convert CodeandVerse to a community platform for developers in which they can communicate, share ideas, and build connections which is essential for a successful progress in any project. CodeandVerse is still under development. Here, I will try to outline and briefly explain the technologies used until now.

Backend

- **Code standard:** Both Backend and Frontend are developed with a well-organized hierarchy of directories to ensure modularity and clarity. Key directories are:

  - **routes/:** Contains route definitions for different endpoints.
  - **apis/:** Encapsulates external API integrations for third-party services.
  - **config/:** Stores configuration files, such as database connection file db.ts However, I used prisma and stored procedures to handle database interactions.
  - **middleware/:** Contains middleware logic for error handling, request handling, and validation.
  - **Services/:** Contains the service that handles the SMTP mail service using nodemailer.
  - **prisma/:** Manages database schema and interactions using Prisma ORM.
  - **sockets/:** Handles WebSocket connections for real-time features.
  - **controllers/:** Contains the logic function of the models and interacts with services.

  Environment Variables: Sensitive information and environment-specific configurations, such as API keys, database credentials, and server ports, SMTP, JWT key, etc are stored securely in my .env files.

  TypeScript: The use of TypeScript enforces static type checking, which helps prevent runtime errors and ensures robust code.

- **Node.js with Express.js**: I created the backend of codeandverse Web App using Node.js, a runtime environment that allows JavaScript to be used for server-side scripting. Also, used CORS to enable secure cross-origin requests and resource sharing.

- **Prisma ORM**: Prisma is used as the Object Relational Mapping (ORM) tool to interact with the MySQL database. Prisma offers an elegant and efficient way to query and manage the database by using an auto-generated client. It supports data modeling, migrations, and querying with a focus on type safety and performance.

Database

I created the database in MySQL. For data interaction I used Prisma Object-Relational Mapping to automatically generates queries. I also, used Stored Procedures to apply that method in my design. Also, indexes have been used to improve speed and performance.

Frontend

- **React.js (with TypeScript)**: The frontend of the application is built using React. I created the frontend in a component-based architecture to ensures modularity and maintainability, readability, and enabling myself to apply interesting concepts for my UI. TypeScript is used alongside React for type safety and better experience to ensure error-free code and better tooling support.
- **Material-UI**: Material-UI is a popular React UI framework that provides a robust set of pre-built, customizable components that follow Google's Material Design guidelines. The documentation of MUI is amazing, enabling me to learn and apply it in my application in a hands-on manner.
- **OOP Concepts in React**: It is interesting to me that the concepts of OOP, such as Abstraction, Encapsulation, Polymorphism, and Inheritance applies to React and MUI consequently.
  - **Inheritance:** I created ThemeProvider and wrapped the whole app inside it. So, the child components inherit styles from their parent which is ThemeProvider.
  - **Polymorphism:** Typography takes various forms and Grid can take various layouts depending on the props or children. That is so interesting to me.
  - **Encapsulation:** A Button component can maintain various methods internally and expose only necessary props when needed. I did not use all of these in codeandverse, however, I indeed used all of these concepts in SupplyBoard project.
  - **Abstraction:** For any components, a simple interface can be defined for abstraction and then encapsulate complex functions.
- **Axios**: Axios is used for making HTTP requests to the backend API. It simplifies the process of fetching and posting data from/to the server, with built-in support for promises and async/await.
- **React Router**: React Router from BrowserRouter is used for handling navigation within the web application.
- **Multilingual:** The app is created in three languages, English, Farsi, and French. I used simple useEffect and useState to design and implement multiple language functionality. Also, the file language.tsx defines the LanguageContext which is a React Context using createContext to enable me to design a centralized way of sharing current language. Also, LanguageProvider is another context that wraps the components and supply them with LanguageContext and translation function. However, I learned a good lesson in working on how to implement multilingual functionality in an app which I will explain later in this report.
- **SEO and Performance:** I tried to implement best practices to improve the performance of responsiveness and discoverability of the project by utilizing the following strategies:
  - Lazy Loading: I implemented lazy loading across the project to load components and assets only when required. As you see the solar system renders smoothly, despite the fact that the 3D files of the planets are huge in size.

  - SEO Optimization with React Helmet: I utilized React Helmet to manage dynamic meta tags, titles, and descriptions to enhanced SEO performance by optimizing pages for better search engine visibility.
  - Performance Optimization: I optimized the app for both desktop and mobile devices, ensuring a smooth experience for users across different platforms.
  - Improved User Experience: As a proactive task, I studied about colors, icons, buttons, etc and their sizes, shapes, and edges to learn more about UX design and user experience and tried some of those in my UX design.

- **Three.js library and the Solar System:** Just for fun, I designed a 3D model of solar system for the background of the homepage. I have countless hours of experience of working on 3D models in python and React in SupplyBoard. I downloaded the 3D model of the planets and their texture files from free online sources, and designed a scene, camera, rotation, inclination, tilt, etc so that they simulate a working solar system in small scale. The tilt and the inclination, rotation direction of planets are similar to their real parameter. For sure it can be improved if one spends more time on its user-interactivity, Realism, etc.

Containerization

Every time a project is created, there is a huge overhead in build process, from dependencies, to variables, and more. To address this, I containerized both the frontend and backend into separate Docker images, and they work smoothly, consistently, and reliably regardless of the environment in which they operate.

- **Dockerfile:** Each project, have a Dockerfile in which I defined the procedure of building the projects, frontend and backend.
- **Nginx:** For frontend containerization, Nginx is used to serve the static files generated by the React app. And finally, the image is exposed the port defined in the Dockerfile.
- **Prisma Schema:** This schema is copied and then the prisma client is generated based on the binaryTargets defined in the schema.prisma file. In this case, it is ubuntu, Debian-openssl-3.0.x.

Cloud Services & Deployment

The deployment of the application involved containerizing the backend and frontend services on Docker and deploying them on AWS using Elastic Container Service (ECS) with Fargate as the launch type. The process ensured scalability, reliability, and seamless integration of cloud services.

Each project is available in a GitHub repository on my GitHub account. I pushed the projects their corresponding repository. And this push, triggers the procedures defined in the ci-cd.yml of each project for CI/CD workflow and pipelining.

- **Elastic Container Repository (ECR):** The Docker images of the backend and frontend project, each containing the application, was pushed to AWS ECR, serving as the centralized repository for managing container images.
- **Elastic Container Service (ECS):** For both backend and frontend, I created two ECS clusters. And defined a service for each of them, and assigned a task to each as well. Environment variables such

as RDS credentials, URLs, SMTP credentials, etc are defined here. However, Amazon recommends the use of AWS Secret Manager.

- **Fargate:** Fargate is a serverless compute engine for Amazon Elastic Container Service (ECS) service that I used and helped me focus solely on deploying and running containers, without the need to manage server modifications. Fargate benefits are isolation, scalability, being serverless, and in term of service cost Fargate is Pay-As-You-Go based on CPU and memory resources used during execution.
- **Task Definition:** A Task Definition is a blueprint in JSON format that describes how the image should run in ECS. I created Task Definitions and defined the port and protocol, and allocated CPU and memory resources. And the networking for them, are handled using the Security Group, subnet, and VPC that I defined globally on this project.
- **Load Balancer configuration:** First I created two Target Groups in EC2 and created an Application Load Balancer for each. For both front and back, an application load balancer was configured to route traffic on their corresponding ports, 80 for frontend and 5000 for backend accordingly.
- **RDS:** I created a MySQL, instance Class db.t3.micro since my project is small, and allocated storage to it. I defined policies and configured internal accessibility accordingly to allow ECS backend cluster to connect to the database in the RDS instance.

CI/CD Pipeline

I love learning more and more about CI/CD because it eliminates huge amount of work that is required normally in deployment and integration of any project. I designed the deployment so that it is automatic using Docker and GitHub actions to manage CI/CD pipeline.

- **Dockerfile:** In each project, the Dockerfile defines the chain of commands to containerize the application and install required dependencies, generate prisma client for the development environment (Ubuntu in my case), expose the corresponding ports, and starting the application.
- **CI/CD Configuration:** The configurations are defined in .yml files to automate the build and deployment process whenever changes are pushed to the corresponding branch on GitHub. The deployment process involves getting the latest source code, setting up the build using Docker Buildx, then logging, tagging and pushing the image to the ECR repositiory on AWS, and finally updating the ECS cluster with the latest image.
- **GitHub Actions**: This is a CI/CD tool used in my project to automate the whole process. This whole workflow process is triggered using simple git commands. As mentioned, the workflow pipeline is defined in .yml files, outlining the chain of processes required to push changes to the ECS endpoint service cluster.
- **Automated Deployment**: Code push, Image build, image push to ECR, and service update on ECS clusters are all done automatically ensuring consistency, speed, scalability, while simplifying the development process.

The Solar System

The Solar System on the homepage is designed to simulate a 3D model of the solar system using Three.js library. The main objective of the component is to visually represent the planets orbiting the sun, their rotational behaviors, and the tilts of the planets, closely mirroring their actual characteristics in the real world except the rotational speed.

Libraries used include, Three.js which is a 3D model and 3D animation library of JavaScript. OBJLoader and MTLLoader are libraries to render 3D models of the planets and the sun. And TextureLoader which is a texture loader to give solar system objects a real look.

Using Three.js I implemented the ambient and point light sources to illuminate the scene and utilized a perspective camera to view that scene. The planet's motion and orbit are defined and implemented so that their tilt and inclination simulate their real-life counterpart. This was achieved by creating rotation.x, rotation.y, rotation.z, and initialTilt variables. Math.cos and Math.sin are used to define the current position, while dynamically updating them to mimic their circular motion.

It may be interesting to know that, there are two exceptions in the rotations of the planets in the solar system. Everything is supposed to rotate counterclockwise (eastward or Prograde rotation). However, there are two interesting rotations. One is the Uranus, which rotates around itself with an inclination of 98 degrees. This sideways rotation may result from a collision with an earth-sized object in the early stages of Solar system formation. The other one is Venus which rotates around itself in a clockwise direction. This can be interpreted as a tilt of approximately 177 degrees which is the opposite direction of all the rotations in the Solar system. This could also be a result of a collision or the planet's dense atmosphere.

Conclusion

The **CodeandVerse Web App** demonstrates a range of modern technologies that help me to create a highly interactive, secure, and scalable web application. I tried to practice technologies such as MySQL and Prisma and ORM techniques, database indexing, stored procedures, Express.js, Node.js, email services, error handling, React, MUI, SEO and performance optimization techniques, containerization, many cloud services on AWS, and CI/CD pipelining practices. I tried to ensure a smooth and efficient experience for both users and myself as the developer.

Currently, this project serves as my personal blog, where I showcase the technologies I am using and the projects I am working on (mainly SupplyBoard). I also share some notes about mathematics, AI, and summaries of books on history, poetry, art, etc that I read in my free time.

My ultimate goal in developing CodeandVerse is to turn this application to a community platform for developers including, enabling them and myself to communicate, share ideas, and build connections. Honestly, my main goal in developing such a platform is to enable myself to know more people who are interested in tech world and collaborate with them.