TED UNIVERSITY

Low Level Design Report

by

Ömer Mekin KARTAL

Burak KATI

Ege TELLİ

Özgür ÖZERTURAL

Table of Contents

1.Introduction

1.1 Purpose

This document is provided to clearly explain the technical details which are used in our website.

Secure Donation Chain is a website to provide people with easier access to donation campaigns and make them able to create donation campaigns whenever they want. SDC will use blockchain to enable users' easy access to their coin wallets and transfer money easily.

1.2 Engineering standards

Our website will be using ISO/IEC/IEEE 23026:2015 as standard. Also, ISO 27001 will be used for cyber security purposes.

ISO/IEC/IEEE 23026:2015 defines system engineering and management requirements for the life cycle of websites, including strategy, design, engineering, testing and validation, and management and sustainment for Intranet and Extranet environments. The goal of ISO/IEC/IEEE 23026:2015 is to improve the usability of informational websites and ease of maintenance of managed Web operations in terms of locating relevant and timely information, applying information security management, facilitating ease of use, and providing for consistent and efficient development and maintenance practices. It applies to those using web technology to present information and communications technology (ICT) information, such as user documentation for systems and software, life-cycle documentation for systems and software engineering projects, and documentation of policies, plans, and procedures for IT service management. ISO/IEC/IEEE 23026:2015 provides requirements for website owners and website providers, managers responsible for establishing guidelines for website development and operations, for software developers and operations and maintenance staff who may be external or internal to the website owner's organization. It applies to websites for public access and for limited access, such as for users, customers, and subscribers seeking information on IT products and services. It includes increased emphasis on the human factors concerns for making information easily retrievable and usable for the intended audience. It focuses on vendor- and product-independent considerations.

ISO 27001 defines the main necessities of any company's cyber security regulations.

1.3 Technologies Used

We are creating our codebase with only JavaScript in both front and backend sides. It gives us a lot of useful packages through NPM.

1. Ganache with hardhat for testing environment on local blockchain

    1.1 Using hardhat, we can deploy our smart contracts written in solidity language to an Ethereum network whether it is a test network or main network.

    1.2 For the deployment of smart contracts we are going to use express.js on top of node js with hardhat.

2. We are going to use Ethers.js for server-side Ethereum blockchain interactions.
3. For user interface we are going to use react framework for creating a simple and intuitive User Interface.
4. MongoDB for storing necessary user data such as:
- usernames
- Wallets of a user
- Wallet addresses of a user
- Campaign picture
- Campaign details
- Campaign comments / ratings
5. We can authenticate users with their choices of wallets using smart contracts written in solidity language alongside JWT.
6. Wallet Connect for integration of multiple wallets
   Example donation scenario:
- When donor logs in, create a new session with the username of a user which is bounded with their wallet address.
- If donor has no wallet integrated bind their username with the default SDC wallet and proceed using the default wallet.
- If donor has their wallet(s) integrated to the system, proceed using their wallet of choice.
- Initiate transactions using Wallet Connect API.
- Use alchemy webhook for notifying the donor and the campaign owner, the transaction status.
7. Monitoring the state of transactions
- Alchemy webhooks for real time notifications (TBD)

Further explanation about packages and interfaces will be made on following pages.

1.4 Definitions and Abbreviations

NPM: Node Package Manager: A store-like platform that enables a user to download and use previously built node.js apps in or out of their projects.

JWT: Json Web Token

2. Packages

Since our codebase consists of JavaScript, all the packages are available on the NPM platform and the ones that are not will be imported manually to the project.

2.1 Front-end Packages

React.js: A powerful frontend JavaScript framework that offers compartmentalized components which makes implementation and the maintenance of the project relatively easy.

JWT: For authentication and authorization on certain actions (I.e. Adding wallets, creating wallets, making a donation), JWT package will be used. For each user a token will be created when they log in or register and a session is started for that user at that time. The token allows them to only operate if they have authorization. For example, if a user started a campaign and another user wants to edit the campaign information. If we did not use the JWT or any similar system, it would create complications since every user was authorized over every other user's resources.

2.2 Back-end Packages

Express.js: A backend node.js framework that is used for building RESTful API's and servers. It has many powerful features such as middleware and routing. Middleware are like normal functions in Express.js but they are executed before the routing functions, so it helps us organize our services and requests made to any other API or service. Routing functions describe when the request comes what should be responded.

We are using several functions within express.js as follows:

Express.Router(): Express router lets us create API endpoints and responses both dynamically and statically. It enables us to create services out of simple requests, which makes our web app more scalable.

Express.json(): Express json lets us send and receive data in json format. This makes sending data through our RESTful APIs more comfortable. For example, we just obtained user data from the input fields, and we are going to send it to the mongodb cluster. If the convert our user details to an object, and then converting it into a json object would form a communication standard between the API's.

Mongoose.js: A mongodb connection driver for NodeJS that allows us to connect to the database and perform crucial operations such as add, delete and update.

Dotenv.js: A helpful tool that allows us to create a .env file that stores and export the necessary public data on developer side such as the port number and the connection url.

2.3 Blockchain Interaction Packages

Ganache: A JavaScript package that creates a local blockchain node for testing purposes. We are going to deploy and test our smart contracts in a ganache node.

Infura: An Ethereum Blockchain provider that enables us to create our own node in the Ethereum Blockchain. Once we complete our tests with our smart contracts, we are going to deploy them to an Infura node and later interact with it.

Ethers.js: Ethereum development packages for node.js. It has some features that allows us to interact with the Infura Ethereum Node on the blockchain or any other node.

WalletConnect.js: A node.js package that has support for over 100 wallets on the Ethereum blockchain. It has some key features such as, sending transactions, signing transactions or send custom requests. It was built on top of Ethers.js and it will be a powerful tool alongside native Ethers.js.

Alchemy.js: Alchemy has an API that allows us to notify our users when a certain event occurs such as the donation quota met, or a new donation has been received. Instead of pinging the server periodically we are going to use Alchemy's API for creating dynamic notifications.

3. Class Interfaces

In our project, we added user.js file to controller, models, routes and services components for our user object to be connected to our database and user class codes will be work in harmony with our Service Oriented Architecture(SOA).

Also we created a mongodb cluster for our project database and in config folder we defined our mongodb database, which appears in mongodbconnection.js

Src:

```
│  index.js      # Entry point for application
└──config        # Application environment variables and secrets
└──controllers   # Express controllers for routes, respond to client requests, call services
└──loaders       # Handles all startup processes
└──middleware    # Operations that check or manipulate request prior to controller utilizing
└──models        # Database models
└──routes        # Express routes that define API structure
└──services      # Encapsulates all business logic
└──test          # Tests go here
```

1-Controllers take inbound client requests and use services

2-Services include all business logic and can invoke the data access layer.

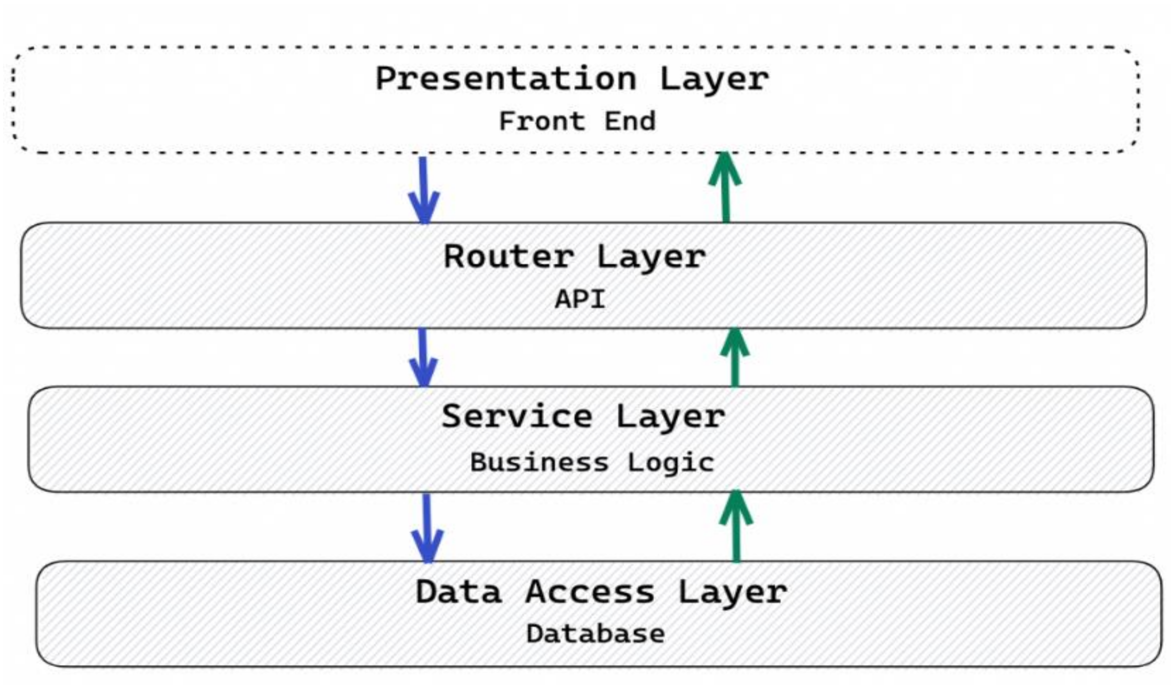
3-The querying process is how the data access layer communicates with the database.

4-The service layer receives the results and passes them up.

5-At that point, the controller may get everything from the service layer.

6-Responses from the controller to clients are then possible!

7-Router.js file Routing is handled by creating a standard express router with router middleware functions. The middleware functions are returned from module functions, to allow injecting services into the route handling middleware.


Presentation Layer
Front End
Router Layer
API
Service Layer
Business Logic
Data Access Layer
Database

4. Glossary

All the technical terms used in this document have been explained in related sections.

5. References

https://www.iso.org/standard/62440.html

https://medium.com/@SoftwareDevelopmentCommunity/what-is-service-oriented-architecture-fa894d11a7ec

https://docs.walletconnect.com/

https://jwt.io/