System Design Document

CaRnS

Group Members:

Youssef Iskander, Chenbo Wang, Yinglong Huang, Lion Su, Emin Guliyev, Zexi Zhang

Table of Contents

CRC Cards - Frontend

LandingPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Give the users a general overview of the website, use cases and company goals. ● Direct the user to the login/sign-up pages.	**Collaborators:** ● Hero

SignIn	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Allow a previously registered user to login using their username and password. ● Prevent non-users from accessing the rest of the website.	**Collaborators:** ● None

SignUp	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Allow new users to create a username and password to use for the login page. ● Prevent users from signing up if they have overlapping login information with another user in the database.	**Collaborators:** ● None

UserProfile	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Display user profile information. ● Allow the user to edit their profile information. ● Allow the user to access other information they should have access such as their history and listings.	**Collaborators:** ● None

BuyListing	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Display general listing information of all the vendor listings created for buyers.	**Collaborators:** ● None

RentListing	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Display general listing information of all the vendor listings created for renters.	**Collaborators:** ● None

CreateListing	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Allow the vendor to create a listing. ● Send a listing JSON to the backend so it can be added to the vendor's listings in the database.	**Collaborators:** ● None

NavBar	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Allows users to navigate to different parts of the website.	**Collaborators:** ● None

Hero	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Graphic for the LandingPage.	**Collaborators:** ● None

App	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Main React component. ● Contains all the mainline logic of the frontend.	**Collaborators:** ● LandingPage ● SignIn ● SignUp

	● UserProfile ● BuyListing ● RentListing ● CreateListing ● NavBar ● Hero

CRC Cards - Backend

Server	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Runs the Express.js server on a given port. ● Connects to the MongoDB database with the .env file. ● Defines what routes the API uses.	**Collaborators:** ● authenticationRoutes ● listingRoutes ● MongoDB

AuthenticationRoutes	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Pair login and sign-up functions that correspond to their respective HTTP methods and their relative route paths. ● Get controller functions from authenticationController. ● Exports the router for Server.	**Collaborators:** ● authenticationController ● Server

ListingRoutes	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Pair listing functions that correspond to their respective HTTP methods and their relative route paths. ● Get controller functions from listing Controller. ● Exports the router for Server.	**Collaborators:** ● listingController ● Server

AuthenticationModel	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Defines user schema to be inputted into the database. ● Validates user signup input and returns a user if it is a valid input. ● Validates and authenticates username/password for login. ● Exports user schema for authenticationController.	**Collaborators:** ● authenticationController

ListingModel	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Defines listing schema to be inputted into the database. ● Validates required listing information and returns a listing if it is a valid input.	**Collaborators:** ● listingController

● Exports listing schema for listingController.	

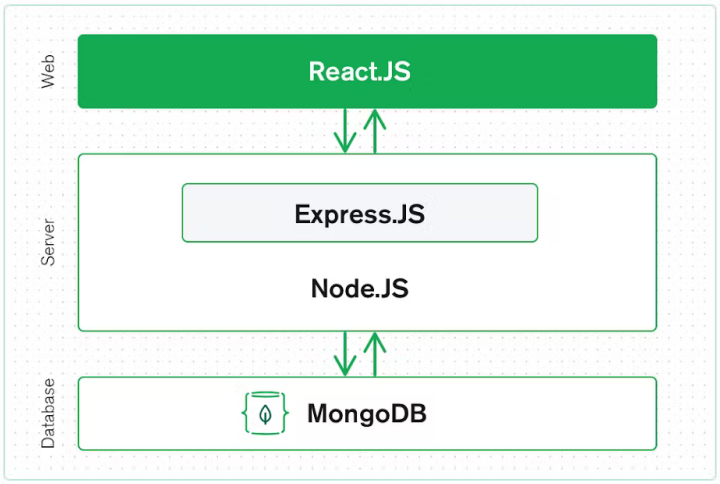
AuthenticationController	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Defines authentication functions for authenticationRoutes. ● Performs CRUD operations in the database. ● Returns respective HTTP status code if a function works properly or throws an error.	**Collaborators:** ● AuthenticationModel

ListingController	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Defines listing CRUD functions for listingRoutes. ● Performs CRUD operations in the database. ● Returns respective HTTP status code if a function works properly or throws an error.	**Collaborators:** ● listingModel

System Architecture

Software Architecture Diagram

Three-tiered architecture



References:

By: IBM Cloud Education. "What Is Three-Tier Architecture." IBM, *https://www.ibm.com/cloud/learn/three-tier-architecture*.

"What Is The Mern Stack? Introduction & Examples." MongoDB, *https://www.mongodb.com/mern-stack*.

Architecture Description

The MERN stack follows the traditional three-tiered architecture (shown above). Our frontend, the presentation tier, uses React JavaScript, HTML and CSS to display and collect information for the backend to handle. The application tier, written in Node.js accompanied by an Express.js server, communicates with the frontend to process the user's API calls using HTTP request methods. Depending on the call, the backend will store, access or update information in the MongoDB database, the data tier, and return some feedback to the frontend. This works conveniently due to the frontend, backend and database all using the JSON data format to send data between tiers.

System Decomposition

As shown through the CRC cards, the frontend is divided between the different pages the various users interact with. The backend is mostly organized between models, controllers and routes for each type of interaction with the database. For example, users need listings to have CRUD operations. Each listing operation has routes for the controller to perform said operations to the database. The controller uses the model to form proper JSONs, check for valid fields and return HTTP statuses.

Invalid input is handled in the backend and returns HTTP statuses to ensure that data has either been handled properly or that the frontend needs to display an error message for the user. If the backend server fails, the user will still be able to interact with the frontend application and use the NavBar to travel between pages, but cannot interact with the database in any way, such as creating a listing or logging in.