System Design Document

CaRnS

Group Members:

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

Table of Contents

CRC Cards - Frontend

Pages

BookDetailsRentPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays the checkout page for rent listings.	**Collaborators:** ● BookDetailsRent

BuyCheckoutPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays the checkout page for buy listings.	**Collaborators:** ● BuyCheckout

BuyDetailsPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays the specific details of a vehicle listing.	**Collaborators:** ● BuyDetail

BuyerHistoryPage
Parent Class (if any): None **Subclasses (if any):** None

Responsibilities: ● Displays the history of all the vehicles the customer has bought in the past.	**Collaborators:** ● ProfileSideBar

BuyListingPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays all the active buy listings in descending creation order.	**Collaborators:** ● BuyListing

CreateListingPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays the form where the vendor can create a listing to sell their vehicle.	**Collaborators:** ● ListingForm

CreateRentListingPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays the form where the vendor can create a listing to rent their vehicle.	**Collaborators:** ● RentListingForm

EditProfilePage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays the form where the user can edit their own profile information.	**Collaborators:** ● EditProfileForm

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

ProfilePage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays the user's profile. ● Allows the user to access their profile sidebar, which can allow them to access their history, profile, and listings (if they are a vendor).	**Collaborators:** ● Profile ● ProfileSideBar

RentListingPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays all the active rent listings in descending creation order.	**Collaborators:** ● RentListing

SignInPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays the input fields for the user to login to their account. ● Direct the user to the signup page if they do not have an account.	**Collaborators:** ● SignIn

SignUpPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays the input fields that allow the user to create an account given their input credentials.	**Collaborators:** ● SignUp

VendorHistoryPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays the vendor's history for their buy listings. ● Allows the user to access their	**Collaborators:** ● PastSellList ● ProfileSideBar

profile sidebar, which can allow them to access their history, profile, and listings.	

VendorListingsPage	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays all of the vendors active buy listings. ● Allows the user to access their profile sidebar, which can allow them to access their history, profile, and listings (if they are a vendor).	**Collaborators:** ● ActiveSellList ● ProfileSideBar

VendorListingsPageRent	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Displays all of the vendors active rent listings. ● Allows the user to access their profile sidebar, which can allow them to access their history, profile, and listings (if they are a vendor).	**Collaborators:** ● ActiveRentList ● ProfileSideBar

Components

App
Parent Class (if any): None **Subclasses (if any):** None

Responsibilities: ● Main React component. ● Contains all the mainline logic of the frontend.	**Collaborators:** ● All Components ● All Pages

ActiveHistorySingle	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Contains the information about a singular card in the active sell list.	**Collaborators:** ● None

ActiveRentList	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Subpage to display vendor's active rent listings.	**Collaborators:** ● None

ActiveSellList	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Subpage to display vendor's active sell listings (buy listings on customer side).	**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

ContactInfo	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Sub-component card that displays the user's contact information (name and phone number) inside of the listing detail page.	**Collaborators:** ● None

DateRangePicker	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Create a calendar so the user can rent out a rent listing for a given amount of time. ● Display dates which are unavailable for the user to pick.	**Collaborators:** ● None

EditProfileForm	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Form that allows users to input and edit their name, email and phone number.	**Collaborators:** ● None

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

ListingForm	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Creates a form that takes user input to create a buy or rent listing.	**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

Profile	
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	**Collaborators:** ● None

listings.	

ProfileSideBar	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Customer - account and history buttons to display their respective information. ● Vendor - account, history and listing buttons to display their respective information.	**Collaborators:** ● None

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

RentListingForm	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Input form which allows customers to create rent listings.	**Collaborators:** ● None

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

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

TransactionRoutes
Parent Class (if any): None **Subclasses (if any):** None

Responsibilities: ● Pair transaction/purchase history functions that correspond to their respective HTTP methods and their relative route paths. ● Get controller functions from transactionController. ● Exports the router for Server.	**Collaborators:** ● transactionController ● 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. ● Exports listing schema for listingController.	**Collaborators:** ● listingController

TransactionModel	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Defines transaction schema to be inputted into the database. ● Validates required listing information and returns a transaction if it is a valid input. ● Exports listing schema for transactionController.	**Collaborators:** ● transactionController ● User ● Listing

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 ● MongoDB

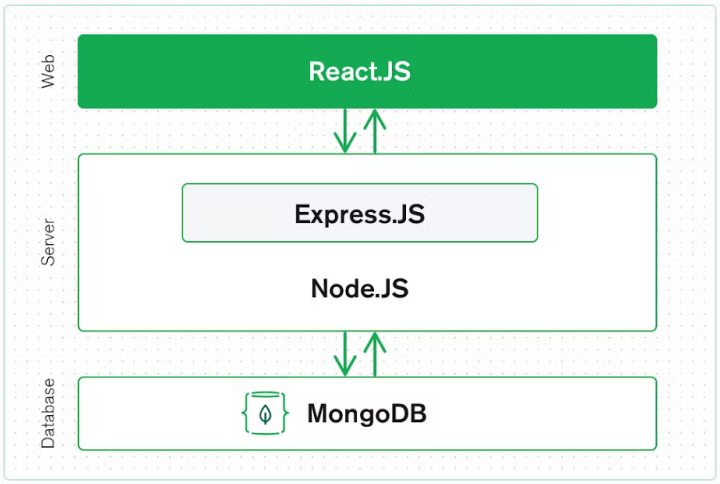
ListingController	
Parent Class (if any): None **Subclasses (if any):** None	
Responsibilities: ● Defines listing CRUD functions for listingRoutes. ● Performs CRUD operations in the database. ● Returns respective HTTP status code if a function works properly or throws an error.	**Collaborators:** ● listingModel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

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

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.