# Capstone Project - Curtain Sales Website - Yi Yang 1995452

## Technology Stack

Frontend: React + ReactHook + ReactRouter + Redux + Axios + Less + Bootstrap + JWT (JSON Web Tokens) + ...

Backend: Node.js (express) + Promise + cors + ...

Database: MySQL

## Plan completion function

1. Home page display

2. City management

3. Search function

4. Pull-up loading

5. Details page

6. Register and login function

7. Favorite function

8. Ratings and reviews

9. Order function

10. Final project structure (Program file description)

## Initialize environment build

1. Project environment: create-react-app scaffolding builds the project environment

2. Support Less syntax

3. Integrate network request Axios

### Configuration supported by Less

In the environment built by React scaffolding, CSS and Sass/Scss are supported by default, so the configuration of Less need to be done.

1. Execute the command: npm run eject (do not do anything after creating the project, execute the command directly), if the file is modified, open the root directory of the file, display hidden files, delete the .git folder, and execute the command again

2. Execute the command: npm install less, npm install less-loader, to install Less dependencies

3. Modify the configuration file

```js

const *lessRegex* =/*\.less$*/;

const *lessModuleRegex* =/*\.module\.less$*/;

{

    test: lessRegex,

    exclude: lessModuleRegex,

    use: *getStyleLoaders*(

            {

                importLoaders: 3,

                sourceMap: isEnvProduction

                    ? shouldUseSourceMap

                    : isEnvDevelopment,

            },

            'less-loader'

        ),

    sideEffects: true,

},

{

    test: lessModuleRegex,

    use: *getStyleLoaders*(

        {

            importLoaders: 3,

            sourceMap: isEnvProduction

                ? shouldUseSourceMap

                : isEnvDevelopment,

            modules: {

                getLocalIdent: getCSSModuleLocalIdent,

            },

        },

        'less-loader'

    ),

},

```

### Configure network requests (Axios)

1. Execute the command: npm install axios, to install Axios dependencies

2. Configure related files - utils, api folder

### Configure initialization style

1. Initialize the css file

2. Introduce icon library

## Realize home page display

1. Create page: Home/Shop/LifeService/User

2. Create Route: 1.Install route dependencies: npm install react-router-dom 2.Configure Router

3. Create footer navigation

4. Create header navigation - REM configuration: Add REM's Computing Scheme to public/index.html

5. Swipeable-View:

   1. Reference: https://react-swipeable-views.com/

   2. Install dependencies:

   ```

   yarn add react-swipeable-views

   yarn add react-swipeable-views-utils

   ```

   3. Pagination need to be implemented independently

   ```

   yarn add classnames

   ```

6. Build a server to provide data

   1. Install dependencies:

   ```

   yarn add express

   yarn add cors

   ```

   2. Cross-domain: Use cors in server

   3. The data comes from /data/home/homehot.js

7. Home list data display

   HomeHotList: Get data, process data, filter data

   HomeHotView: View adaptation, rendering

   Use React useEffect to do business separation

## Realize city management

1. Create city management page and achieve routing jumps

2. Realize components in city page: PubHeader, CurrentCity, CityList

3. Integrate Redux: Use it to store city page data, and the UI will render different results depending on the cities

   - Install redux dependencies

   ```

   yarn add redux

   yarn add react-redux

   yarn add redux-devtools-extension

   ```

   - Create Redux: Store, Reducers, Actions

4. Associate redux to store city data

5. The page data needs to be switched according to the city

6. ABC form of city list

   - Reference: https://github.com/w3cay/react-city-select

   - City list data: src/data/city.js (Lite version to show their function)

   - Install dependencies

   ```

   yarn add react-city-select

   ```

## Realize search function

1. Create a search page and configure routers

   - Extract the input component of the search

   - Configure routers

   - Monitor keycode for 'Enter' jump (keyCode === 13)

   - Routing jump with parameters: Use Hook, providing useParams by React-Router

2. Implements the interface for searching network requests

   - The server returns the same test data for each search due to data limitation

3. Frontend access interface and get data

4. List data rendering: use 'Item' to render each view to replace the previous direct rendering

   - Render html structure:

   ```html

   <p *dangerouslySetInnerHTML*="{{\_\_html:data.description}}"></p>

   ```

5. Implement search header

## Implement pull-up loading

1. Pull-up loading package components

2. Implementation process

   - Listen for scrolling events: Scroll height + view height >= container list height (don't use)

3. getBoundingClientRect: Dynamically get the distance from the top of the element (use)

   - Dynamically get the distance from the top of the element < view height => data loading

4. Debounce and throttle

   - Debounce: Within a time limit (time interval), if multiple requests are initiated, the last one shall prevail

   - Throttle: Within a deadline (time interval), only one request is initiated

5. Pull-up loading - Load more data

6. Mock.js

   - Simulated data, fully randomized

   - Reference (chinese): http://mockjs.com/

   - Install: `yarn add mockjs`

7. Image preprocessing

   - If the image request is successful, load the image, otherwise load the (Placeholder) default image

8. Use the simulated data made by mock to replace the original data to test the search function:

   - More random data for test

   - Network request

   - Pull-up loading

   - Image preprocessing

## Details page

1. Create page, configure router

2. Memory leak:

   - When the scroll event has not ended, jumping to another page will cause a memory leak problem:

     Warning: Can't perform a React state update on an unmounted component. This is a no-op, but it indicates a memory leak in your application. To fix, cancel all subscriptions and asynchronous tasks in a useEffect cleanup function.

   - Prevent memory leak problem:

     Cancel network request, close event, clear timer

3. Favorite function (Show later)

## Register and login function

1. Routing configuration, interface drawing (register: bootstrap, login: self-written), collecting form data, configuring redux

2. Frontend and backend data communication

3. Frontend and backend data validate and frontend error prompt

   - Frontend:

     Validation: import validation file: /utils/validator.js

     - Install dependencies

     ```js

     yarn add lodash

     yarn add validator

     yarn add classnames

     ```

   - Backend: /schema (Joi package)

4. Store registration information in database

5. Backend generates token (jwt.sign)

6. The frontend synchronizes the login status and user information to redux and persists them (local store)

7. Display user data after successful login (only username)

8. The backend parses the request header token (jwt.verify)

9. Logout (Clear data in local and redux)

10. Route interception: If you are not logged in, you cannot access the user page

## Favorite Function

1. It can be used after login, and jumps to the login page when not logged in

2. Real-time update favorite status

3. There is a button state change when favorite/unfavorite, and save/delete the product id when unfavorite/favorited (Redux)

## Ratings and reviews

1. Tabs view: props.Children.map()

   - Reference: https://legacy.reactjs.org/docs/react-api.html#reactchildrenmap

   - Structure:

     ```js

     <Tabs>

       <Tab label="Details">

         <div>

           <h3>Details information</h3>

         </div>

       </Tab>

       <Tab label="Review">

         <div>Review information</div>

       </Tab>

     </Tabs>

     ```

   - Rendering structure:

   ```js

   <ul>

      <li>Details</li>

      <li>Review</li>

   </ul>

   <div>Details information</div>

   <div>Review information</div>

   ```

2. You can view the ratings and comments, and support pull-up loading

## Order function

1. The page of the order, including the jump function of the route

2. If you are not logged in, go to the login page, and you can view the order information after successful login

3. Order information needs to include user information

4. You can fill in the evaluation, transmit and store it to the server

5. Update the evaluation status in real time, and the evaluation cannot be re-evaluated

## Final project structure

Considering the technical framework used and the components required by the application, the detailed project structure can be obtained as follows. If you need to add new interfaces, functions, and introduce new components, you can also easily add new content to the project.

- documents // User\_Manual and Software\_Design\_Document (SDD)

- public

  - index.html // Quote bootstrap

- server (Backend and database)

  - config // Rules for generating tokens

  - data // Details page/Home page/Orders/Reviews/Search result data

  - database // Database connection/sql language configuration

  - middleware // Middleware that establishes rules to associate with input information

  - router // Post router: login/register/user Get router: the rest routers

  - routesHandle

    - login // Verify login information and generate token

    - register // According to the existing user to judge the registration is successful

  - schema // Validation rules using Joi plugin

  - index.js // Main file: routing, cross-domain, error handling, etc.

- src (Frontend)

  - api // Interface for exchanging data with the server

  - assets // Store frontend static resources: pictures, icons and fonts, etc.

  - components

    - FooterNav // Footer navigation and style

    - HeaderNav // Header navigation and style

    - PubHeader // Title and back navigation and style

    - PullupLoading // Pull up function (self-written)

    - SearchInput // Input box (search function)

    - StarRead // Star rating (antd UI library, read only)

    - Swiper // Carousel on home page (self-written)

    - Tabs // Display title corresponding information (self-written)

  - data // Using ABC form to display city

  - pages

    - City

      - CityHotList // Hot city list display

      - CityList // Choosen city information display

      - CurrentCity // Current city display

      - index.jsx // City page display and interact with redux

    - Details

      - BuyAndFav // Favorite function and interact with redux

      - BuyAndFavView // Favorite View and data processing

      - DetailsList // Datails page and interact with server

      - DetailsListView // Details View

      - Reviews // Reviews function and interact with server

      - ReviewsView // Reviews View

      - index.jsx // Accept parameters and pass to the view (product's id)

    - Login

      - LoginView // Login view (self-written), data processing and interact with server

      - index.jsx // Interact with redux and local store

    - Main

      - Home

        - HomeProList // Interact with server

        - HomeProView // Home products View (new and hot)

        - index.jsx // Interact with redux

      - LifeService // Life service page (only route, no data)

      - Shop // Shop page (only route, no data)

      - User // User page (only display user name) and route interception

    - Order

      - OrderList // Interact with server

      - OrderListView // Order list view

      - Userinfo // Display user information (name and city)

      - index.jsx // Interact with redux

    - Register

      - RegisterView // Enter and submit information, interact with server

      - index.jsx // Interact with redux

    - Search

      - SearchHeader // Call the SearchInput component

      - SearchList // Interact with server, call pull up function

      - SearchListView // Search result view

      - index.jsx // Accept parameters and pass to the view (keywords)

  - redux

    - actions // city/favorite/login/register/search actions

    - constants // Constantization of type names

    - reducers // All reducers, index is for combination

    - store // Create store with all reducers

  - router // Routing table

  - utils

    - init // Token initialization processing

    - loadImg // Image loading processing, when the image is not loaded, it will be displayed with the default image

    - request // Network request configuration

    - validator // Check the login input information on the frontend

  - App.jsx // Route display

  - index.js // Entry file

- README.md // Software Design Document (SDD)