Frontend:

Vue.Js (MIT): Vue.js provide functionalities to our html web-file, from the vuejs, we are able to provide RWD, and a lot more functionality, see below:
Vue Router: the vue router creates a front-end web router from pages to pages, it allows us to display different pages in different paths.

> https://github.com/vuejs/vue-router/blob/dev/src/index.js .
> - vue router class provides functionalities such as letting us define a mode, in the code the mode is defined as key:value pair where key is mode and value is a string, which the source code will then check by a switch to determine the mode that web-app is running on (line 61)
> -On line 94, the vue router explained that it will push the parameter “app” (a list of routing paths, with name, path, and components) into a const variable “apps”, which every path will be listened on line 141 by using a foreach statement
> -On line 201 in the same file, the vue router provides how it binds the components into the vue router, it decodes the route from line 141 and maps the component file with that path location.

Vuex : vuex provides a central data storage system for us to track some global variables which exist in different paths/files , for example a logged in status should be tracked everywhere in the site to prevent users from getting locked out or having to log in on every path.

> The index.js https://github.com/vuejs/vuex/blob/dev/src/index.js will be importing all its functionality (“mapState” is what we need) from other component files and export it as “Store” (what we need)
> https://github.com/vuejs/vuex/blob/dev/src/store.js and this is the file we want to dig.
> In the project,we are going to use the “state’, and “commit”
> -State : state is relatively simply, it simply sets a variable to a certain value which we can run get and set on it, the get function, on line 73 , will simply return the value of the values which we want. Then we have the replaceState on line 197 , which will take in a variable and set the old variable with the new value.
> - Commit: I mentioned on how replaceState work in “State”, but in reality, for the state we do not want to make any changes for the variable in the

state, in fact for any changes that want to be made to the state, we will call Commit on that variable in the state. Starting on line 449 in https://github.com/vuejs/vuex/blob/96a265a345c76ec7d0f81a115aef74b7eda89452/dist/vuex.common.js#L449 , we have the prototype function commit, which takes in a type (“state”) and a payload (“some value to pass in”).

Axios(MIT)
From axios, we used two methods for posting and getting the jsons to/from server.
Get:
In the case of the get request, what axios will do is take the information from the client, then read whether if the method requested is in fact an item in the method that it allowed, in fact this is a part of `function forEachMethodNoData`: see https://github.com/axios/axios/blob/f2057f77b231d88ea94ad88e84e4fd9f99506880/lib/core/Axios.js#L114
After the recognition of the correct method, by looking at the souce code from https://github.com/axios/axios/blob/f2057f77b231d88ea94ad88e84e4fd9f99506880/dist/axios.js#L1537
It was recognized that the requested method sent to the destination url, and asked for the information which server should provide, then came back and started packing the response to send back to the server, in this packet, 3 information were sent back: 1. The methods, 2. The url of the destination, 3. The data achieved from the destination url. Note that 1 and 2 are provided at the initial axios get request, axios module simply pick they up by specific index.
https://github.com/axios/axios/blob/f2057f77b231d88ea94ad88e84e4fd9f99506880/lib/core/Axios.js#L27

Post:
https://github.com/axios/axios/blob/f2057f77b231d88ea94ad88e84e4fd9f99506880/dist/axios.js#L583
The post request are somewhat identical to the get request, the only big difference is how one is method with no data and the other one is `function forEachMethodWithData`.Once again, axios will decode the method and url from

https://github.com/axios/axios/blob/f2057f77b231d88ea94ad88e84e4fd9f99506880/lib/core/Axios.js#L27, However One big difference is that for the post method is does contain some data, so when it come to post request, note that the data is no longer an empty json, instead it will use the data from the client side and pass that json into the data (line 589) so that post will able to send the data to the server and wait for an response.

Backend: Java + Spring Framework

Yuan Meng:

In our backend part, we are designed in DAO (Data Access Object) pattern which isolates and separates layers by business and object. And Spring framework will support annotation and maven dependency for us to quickly build service, controller and configuration etc.

Interface/Model: During the project, we start with an interface about user operators by annotation Repository on it in order to mark this class as a Repository class that tries to connect with service annotation. It is the base interface with User since it has all the functionality with user and easy to add or delete in the future.

Business Service/Function call: Set annotation “service” for the Userservice class which relates to the “User” Repository, then it is able to automatically pair with User operator class’s information without creating an object. The purpose of this class is to help connect the Data, Model with Restful API, so, we don’t need directly use the model or implement from original data, we use service class to call the method or data from the original, it will help us to change the requirements and maintain the structure in the future.

HTTP Request/Response: Last step we set up the annotation “Restcontroller” then in the class we define the User Service which I mention the before it will automatically pair the instance User service in which we could call the User Service class’ function to access the data from User or temp database.

Generally this design pattern and framework help us to avoid complex and repetitive creating objects for each class, and help to automatically pair with each other by annotation (name).

In the RESTful API: we are using annotation “@RequestMapping” by (“/”) to receive the root path and annotation “@CrossOrigin” to allow Vue’s localhost to send requests to our server. And then in the server we will use annotation PostMapping/GetMapping (path) to receive the data, by the time in the PostMapping due to we have Header and Body in the request, then we use annotation @Requestbody to receive the key-value from the post body, and based on the

Service class we are able to call service class method to access the data in the temp database do put or delete.

PostMapping:
https://github.com/spring-projects/spring-framework/blob/master/spring-web/src/main/java/org/springframework/web/bind/annotation/PostMapping.java

RequestBody:
https://github.com/spring-projects/spring-framework/blob/master/spring-web/src/main/java/org/springframework/web/bind/annotation/RequestBody.java

In our sending message part, we use websocket API for sending message:

Websocket(Frontend):

1. In the WebSocket API, the browser and the server only need to do a handshake action, and then a fast channel is formed between the browser and the server. Data can be transferred directly between the two.
2. The browser sends a request to establish a WebSocket connection to the server through JavaScript. After the connection is established, the client and server can directly exchange data through the TCP connection.
3. After the Web Socket connection, the user can send data to the server through the send() method, and receive the data returned by the server through the onmessage event.

Websocket(Backend):

1. Create a WebSocket server through the annotation **@ServerEndpoint**. (ServerEndpointExporter is a standard implementation officially provided by Spring for scanning ServerEndpointConfig configuration classes and **@ServerEndpoint** annotation instances.)
2. Annotate **@ServerEndpoint** to declare the instantiation of the WebSocket server.
3. Declare callback functions through annotations **@OnOpen**(triggered when the connection is opened), **@OnMessage**(triggered when receiving client message), **@OnClose**(Triggered when the connection is closed), and **@OnError**(triggered when connection have error).
4. When the client sends a message, get its username and message and send the message to everyone. When the connection is closed, the session object is removed.

Spring-boot-websocket:
https://github.com/spring-projects/spring-framework/tree/master/spring-websocket

ServerEndpointExporter:
https://github.com/spring-projects/spring-framework/blob/master/spring-websocket/src/main/java/org/springframework/web/socket/server/standard/ServerEndpointExporter.java