CSCC01 - Just on Time Documentation

Arya Sharma, Bhoomi Patel, Sanan Rao, Virthiya Uthayanan, Yuto Omachi

Table of Contents

client/src/services/admin/verifyEventService.js
client/src/services/admin/verifyOrganizerService.js
client/src/services/auth/authService.js
client/src/services/auth/authSlice.js
client/src/services/emailVerification/customer/resendCode.js
client/src/services/emailVerification/customer/verifyEmail.js

Backend

server/routes/userRoutes.js

This file combines all the customer routes together, now it is just handling the post and get requests for the customer i.e registering and logging in using a customer account.

Request

Mimetype	application/json
Method	POST
URL	/api/user/register
Description	It takes in a req and res parameter, where req holds the data of the user trying to sign up. Signup then ensures that the user does not already exist and all the parameters are of proper format. If the data is of the proper format defined and the user does not already exist, the data will then be sent to MongoDB. If that succeeds, res returns 200 as its status, user data as its data. Upon failure, res returns 400 as its status, and the corresponding error as its message.

Body Example	{
	"email": "email1223@gmail.com",
	"password": "my password",
	"firstName": "Yuto",
	"lastName": "Omachi",
	"suitNo": "123",
	"street": "someStreet",
	"city": "Toronto",
	"country": "Canada",
	"postalCode": "A1A2B2"
	}

Mimetype	application/json
Method	POST
URL	/api/user/login
Description	It takes in a req and res parameter, where req holds the data of the user

	trying to log in. Login then ensures that the user all the parameters are of proper format and if the data is of the proper format defined and the user exists, the user password is compared . If that succeeds, res returns 200 as its status, user data as its data. Upon failure, res returns 400 as its status, and the corresponding error as its message.

Body Example	{
	"email": "email1223@gmail.com",
	"password": "my password",
	}

Mimetype	application/json
Method	POST
URL	/api/user/verifyemail/:email/:token
Description	It takes in a req and res parameter, where req holds the email and the token for the user needed to be verified. Verification then checks if the token is valid and checks if the user is already verified else sends an email for verification. If that succeeds, res returns 200 as its status, appropriate message. Upon failure, res returns 400 as its status, and the corresponding error as its message
Body Example	No body

Mimetype	application/json
Method	POST
URL	/api/user/resendCode
Description	It takes in a req and res parameter, where req holds the email of the user needed to be verified. Then checks if the user exists, and sends verification email. If that succeeds, res returns 200 as its status, appropriate message. Upon failure, res returns 400 as its status, and the corresponding error as its message
Body Example	{

	email: “acd55@gmail.com” }

Mimetype	application/json
Method	POST
URL	/api/user/personal-Info
Description	It takes in a req and res parameter, where req holds the id of the user whose info is to be updated. Then checks if the user exists and is allowed to send the request using auth middleware, and then updates user info. If that succeeds, res returns 200 as its status, appropriate message. Upon failure, res returns 400 as its status, and the corresponding error as its message

Body Example	{
	"update": {userInfo.field: newValue} ,
	“id”: id
	}

Mimetype	application/json
Method	POST
URL	/api/user/registerOrganizer
Description	It takes in a req and res parameter, where req holds the id of the user whose info is to be updated to an organizer and info to be updated. Then checks if the user exists and is not already an organizer and is allowed to send the request using auth middleware. Then user info is updated and if that succeeds, res returns 200 as its status, appropriate message. Upon failure, res returns 400 as its status, and the corresponding error as its message

Body Example	{
	“phoneNumber”: “5493295032”,
	….
	}

server/routes/adminRoutes.js

This file combines all the organizer routes together, now it is just handling the post and get requests for the organizer i.e registering and logging in using an organizer account.

Request

Mimetype	application/json
Method	POST
URL	/api/admin/getUnverifiedOrganizers
Description	Checks the request is being sent by the appropriate user i.e admin user with the help of authentication middleware. Returns a list of unverified organizers.
Body Example	No Body

Mimetype	application/json
Method	POST
URL	/api/admin/getUnverifiedEvents
Description	Checks the request is being sent by the appropriate user i.e admin user with the help of authentication middleware. Returns a list of unverified events.
Body Example	No Body

Mimetype	application/json
Method	POST
URL	/api/admin/updateOrganizerStatus

Description	It takes in a req and res parameter, where req holds the email of the organizer whose status is to be updated. Checks the request is being sent by the appropriate user i.e admin user with the help of authentication middleware. Finds the user and updates the user info. If that succeeds, res returns 200 as its status, appropriate success message . Upon failure, res returns 400 as its status, and the corresponding error as its message.

Body Example	{
	"email": "email1223@gmail.com",
	"password": "my password",
	}

Mimetype	application/json
Method	POST
URL	/api/admin/updateEventStatus
Description	It takes in a req and res parameter, where req holds the id of the event whose status is to be updated. Checks the request is being sent by the appropriate user i.e admin user with the help of authentication middleware. Finds the event and updates the event info. If that succeeds, res returns 200 as its status, appropriate success message. Upon failure, res returns 400 as its status, and the corresponding error as its message.

Body Example	{
	"email": "email1223@gmail.com",
	"password": "my password",
	}

server/routes/eventRoutes.js

Mimetype	multipart/form-data

Method	POST
URL	/api/event/
Description	It takes in a req and res parameter, where req holds the data for the event to be created. Checks the request is being sent by the appropriate user i.e organizer with the help of authentication middleware, then creates events with info and image. If that succeeds, res returns 200 as its status, appropriate success message. Upon failure, res returns 400 as its status, and the corresponding error as its message.
Body Example	name: "NETS VS RAPTORS" description: "This is a sports events " time: "8:32 pm" street: "Scarborough" city: "Toronto" country: "Canada" postalCode: "M1H3J2" location: "eventImages" Image: undefined

Mimetype	application/json
Method	GET
URL	/api/event/
Description	This returns all the events in the database, if that succeeds, res returns 200 as its status and events as response. Upon failure, res returns 400 as its status, and the corresponding error as its message.
Body Example	No body

Mimetype	application/json

Method	GET
URL	/api/event/getOrganizerEvents
Description	This returns all the organizer's events in the database, checks if the user is authenticated.If that succeeds, res returns 200 as its status and events as response. Upon failure, res returns 400 as its status, and the corresponding error as its message.
Body Example	No body

Mimetype	multipart/form-data
Method	POST
URL	/api/event/updateEvent
Description	It takes in a req and res parameter, where req holds the data for the event to be created and the id for the event to be updated. Checks the request is being sent by the appropriate user i.e organizer with the help of authentication middleware, then updates the event with updated info. If that succeeds, res returns 200 as its status, appropriate success message. Upon failure, res returns 400 as its status, and the corresponding error as its message.
Body Example	name: "NETS VS RAPTORS" description: "This is a sports events " time: "8:32 pm" street: "Scarborough" city: "Toronto" country: "Canada" postalCode: "M1H3J2" location: "eventImages" Image: undefined

server/auth/strategies/userStrategies.js

1. userRegisterStrategy

This is a passport strategy responsible for authenticating requests, which is accomplished by implementing an authentication mechanism. This strategy validates that the request has all the required fields and has a unique email which is essential for every user. Also this strategy encrypts the password for the user so as to make the account and database info secure.

2. userLoginStrategy.js

   This is a passport strategy responsible for authenticating requests, which is accomplished by implementing an authentication mechanism. This strategy validates that the request has all the required fields and then it finds the user in the database using the email and compares the password and returns success or failure. Also return failure when a user doesn't exist.

server/auth/passportContoller.js

This combines all the passport strategies and the serializing together so that it can be easily accessed in the backend.

server/controllers/userController.js

- registerUser is a function that is responsible for returning relevant error messages, status codes or success messages based on the error code or info returned by userRegisterStratergy.js. For instance we get status code:400 and a relevant error message for errors like 'email is already in use' for non unique email, it returns status code 200 and user object when there are no errors. It is also responsible for validating the request body and returning appropriate messages.

- loginUser is a function that is responsible for returning relevant error messages, status codes or success messages based on the error code or info returned by userLoginStratergy.js. For instance we get status code:400 and a relevant error message, it returns status code 200 and user object when there are no errors. It is also responsible for validating the request body and returning appropriate messages

- verifyEmail is a function that sends verification code for the user, and is responsible for returning relevant error messages, status codes or success messages.

- resendCode is a function that resends verification code for the user, and is responsible for returning relevant error messages, status codes or success messages.

- updateInformation is a function that updates user info, and is responsible for returning relevant error messages, status codes or success messages.

- registerOrganizer is a function that registers an already registered user as organizer, and is responsible for returning relevant error messages, status codes or success messages.

server/controllers/adminController.js

- getUnverifiedOrganizer is a function that is responsible for returning a list of unverified organizers in the database and is responsible for returning relevant error messages, status codes or success messages.

- updateOrganizerStatus is a function that is responsible for changing the organizer status from unverified to a relevant one and is responsible for returning relevant error messages, status codes or success messages.

- getUnverifiedEvents is a function that is responsible for returning a list of unverified events in the database and is responsible for returning relevant error messages, status codes or success messages.

- updateEventStatus is a function that is responsible for changing the event status from unverified to a relevant one and is responsible for returning relevant error messages, status codes or success messages.

server/controllers/eventController.js

- addEvents is a function that is responsible for creating an event in the database and is responsible for returning relevant error messages, status codes or success messages.

- getEvents is a function that is responsible for returning a list of all the events in the database and is responsible for returning relevant error messages, status codes or success messages.

- getOrganizerEvents is a function that is responsible for returning a list of all the organizer's events in the database and is responsible for returning relevant error messages, status codes or success messages.

- updateEvents is a function that is responsible for changing the event info and is responsible for returning relevant error messages, status codes or success messages.

Models

server/models/userModel.js

userModel.js defines the customer model with Mongoose Schema to present to the server and MongoDB the format of the user. It imports Mongoose and each customer takes in an email, phoneNumber, firstName, lastName, street, suitNo, country, city, postalCode that are all of type String. An example of a user model follows along with how it would be called and utilized:

```
const user = new User(
        {
            userInfo: {
```

```
            email,
            firstName,
            lastName,
            password
        }
    })
```

server/models/eventModel.js

eventModel.js defines the event model with Mongoose Schema to present to the server and MongoDB the format of the eventModel. It imports Mongoose and each event takes in an eventInfo, tags, bidHistory that are all of type String and an eventOrganizer object with the organizer id and name. An example of a event model follows along with how it would be called and utilized:

```
const event = new Event(
    {
        eventInfo: {
            name,
            description,
            time,
            address:{
                street,
                city,
                country,
                postalCode
            }
        },
        tags,
        bidHistory,
        organizer_id,
        eventImage_path
    }
)
```

server/models/schemas/address.schema.js

address.schema.js defines the address model with Mongoose Schema to present to the server and MongoDB the format of the address. It imports Mongoose and each address takes in a street, suitNo, country, city, postalCode that are all of type String.

server/models/schemas/contact.schema.js

contact.schema.js defines the contact model with Mongoose Schema to present to the server and MongoDB the format of the contact. It imports Mongoose and each address takes in an email and phoneNumber that are all of type String.

server/models/schemas/personalInfo.schema.js

personalnfo.schema.js defines the personalnfo model with Mongoose Schema to present to the server and MongoDB the format of the personal info. It imports Mongoose and each address takes in firstName, lastName that are all of type String and an object of type address.

server/models/schemas/organizer/bankInfo.schema.js

bankInfo.schema.js defines the bankInfo model with Mongoose Schema to present to the server and MongoDB the format of the bankInfo. It imports Mongoose and each bankInfo object takes in a bankName, branchNum, accountNum that are all of type String.

server/models/schemas/event/bidInfo.schema.js

bidInfo.schema.js defines the bidInfo model with Mongoose Schema to present to the server and MongoDB the format of the bid. It imports Mongoose and each bid takes in a bid price and date.

server/models/schemas/event/eventInfo.schema.js

eventInfo.schema.js defines the eventInfo model with Mongoose Schema to present to the server and MongoDB the format of the event. It imports Mongoose and each event takes in a name, description, time, address and status.

Utils

server/util/http/httpResponse.js

httpResponse.js is responsible for handling standard responses to be sent to the frontend after receiving the request.

server/util/passport/authentication.js

Authentication.js is a middleware responsible for checking if the user is authenticated and the correct type of user is accessing the request.

server/util/validation/eventValidationSchema.js

eventValidationSchema.js is responsible for checking the request has the correct format and the event fields are appropriate, else returns appropriate error if the validation fails.

server/util/validation/userValidationSchema.js

userValidationSchema.js is responsible for checking the request has the correct format and the user fields are appropriate, else returns appropriate error if the validation fails.

server/util/ImageService.js

ImageService.js is responsible for all the functions that are needed for uploading images as a single object so that it is easily accessible throughout the backend.

server/util/email/verification/userVerification.js

userVerification.js is responsible for all the functions that are needed for sending verification email as a single object so that it is easily accessible throughout the backend.

Frontend

Components

client/src/components/admin/node/verifyRejectNode.jsx

verifyRejectNode.jsx is a frontend component which creates a node that stores information about events or organizers and buttons that allow admins to reject or verify them. The properties that can be set for the component are as follows: key, object, firstField, secondField, onClickVerify, onClickReject. Key takes the object id from the database. Object takes the whole object from the database. firstField and secondField are fields that can be used to display relevant information to the user. OnClickVerify and onClickReject take functions that set the status of the object as verified or rejected in the database based on the button that is pressed. The only method in the file is: VerifyRejectNode.

- VerifyRejecNode
    - Returns the verify reject node component

client/src/components/admin/verifyEvent/VerifyEventForm.jsx

VerifyEventForm.jsx is a frontend component that loads all the unverified events into nodes and displays them. There is only one property that can be set for this component and it is: onDeleteNode. OnDeleteNode takes a function that tells the component what to do if an event is deleted (i.e a node is rejected or verified). Methods in this file are as follows: onClickVerify, onClickReject, organizersList, VerifyEventForm.

- onClickVerify
    - Verifies the event in the database when the verify button is clicked
- onClickReject

    - Rejects the event in the database when the reject button is clicked
- organizersList
    - Puts all the relevant information about the unverified events into a list of verifyRejectNodes then returns the list
- VerifyEventForm
    - Styles the value returned by organizersList then returns it

client/src/components/admin/verifyEvent/VerifyOrganizerForm.jsx

VerifyOrganizerForm.jsx is a frontend component that loads all the unverified events into nodes and displays them. There is only one property that can be set for this component and it is: onDeleteNode. OnDeleteNode takes a function that tells the component what to do if an organizer is deleted (i.e a node is rejected or verified). Methods in this file are as follows: onClickVerify, onClickReject, organizersList, VerifyEventForm.

- onClickVerify
    - Verifies the organizer in the database when the verify button is clicked
- onClickReject
    - Rejects the organizer in the database when the reject button is clicked
- organizersList
    - Puts all the relevant information about the unverified organizers into a list of verifyRejectNodes then returns the list
- VerifyOrganizerForm
    - Styles the value returned by organizersList then returns it

client/src/components/forms/input/InputField.jsx

InputField.jsx is a frontend component which creates an input box with error handling and an icon next to it. The properties that can be set for this component are as follows: error, name, onChange, className, icon, type, placeholder. Error is a function or variable that sets a string as a string to display if an error occurs else leaves it as the empty string. Name is like an id for a specific object. onChange takes a function that tells the input field what to do when it detects a change. ClassName is used to style the component with bootstrap. Icon takes anything that it can display as an icon to the left of the input box. Type and placeholder are the same as the respective properties for <input> in html. The only method in this file is: InputField.

- InputField
    - Returns the input field component

client/src/components/forms/login/LoginForm.jsx

LoginForm.jsx is a frontend component which creates a form that can be used to login at Justontime. At the top of the form is the justontime logo. Then the form has two fields: email and password. There is a link to the registration page for unregistered users. It also has a (non-functional) button that allows the user to login with google instead. The properties of the form that can be set are as follows: loading, onSubmit, onChange, error. Loading takes a boolean that tells the component whether to render, a loading a symbol when set to true it will render the symbol else it will not. OnSubmit takes a function that tells the form what to do when the submit button for the form is clicked. OnChange takes a function that tells the form what to do when there is a change in the input fields of the form. Error is a function that is used to store all the errors of the form if there are any. The only method in this file is: LoginForm.

- LoginForm
    - Returns the login form component

client/src/components/forms/signup/SignUpForm.jsx

SignUpForm.jsx is a frontend component which creates a form that can be used to signup a user to Justontime. It has two parts, one side to fill in information and the other side with an image and an animation. The information side has five fields: first name, last name, email, password and confirm password.There is a link to the login page for registered users. It also has a (non-functional) button that allows the user to sign up with google instead. The properties of the form that can be set are as follows: loading, onSubmit, onChange, error. Loading takes a boolean that tells the component whether to render, a loading a symbol when set to true it will render the symbol else it will not. OnSubmit takes a function that tells the form what to do when the submit button for the form is clicked. OnChange takes a function that tells the form what to do when there is a change in the input fields of the form. Error is a function that is used to store all the errors of the form if there are any. The only method in this file is: SignUpForm.

- SignUpForm
    - Returns the sign up form component

client/src/components/header/userbutton/UserButton.jsx

Header.jsx is a frontend component which creates a button which can be used to navigate between login, signup and/or account pages. Methods in this file are as follows: CustomToggle, UserButton.

- CustomToggle
    - Allows the button to open and close the menu of links when clicked
- UserButton
    - Returns the user button and shows the appropriate links depending on if the user is logged in or not

client/src/components/header/Header.jsx

Header.jsx is a frontend component which creates a header with the justontime logo and a button which can be used to navigate between the user pages.
The only method in this file is: Header.

- Header
    - Returns the navbar with the image and the user button on the header.

client/src/components/setting-sidebar/settingSidebar.jsx

settingSidebar.jsx is a frontend component which creates a display that allows a logged in user to navigate between the account settings pages. The sidebar holds all the links of pages relevant to the user's account such as the: personal info page. The only method in this file is: SettingSidebar.

- SettingSidebar
    - Returns the sidebar with the links it holds

client/src/components/spinner/Spinner.jsx

Spinner.jsx is a frontend component that renders a loading animation whenever the pages are loading. The properties that can be set are as follows: color, loading, and size. Color and size can be used to set the color and size of the loading symbol. Loading takes a boolean that tells the component whether to render, when set to true it will render else it will not. The only method in this file is: Spinner.

- Spinner
    - Determines if a page is loading then returns a loading animation if so, else returns null

Pages

client/src/pages/admin/dashboard.jsx

dashboard.jsx is the frontend page that renders the unverified events and organizers for the admin to verify. The admin user can verify the event or organizer and it will be reflected in the database as this happens. Methods in this file are as follows: onDeleteNode, AdminDashboard.

- onDeleteNode

    - Sends a popup alert to the user when an organizer or event is deleted
- AdminDashboard
    - Returns the display for the admin dashboard

client/src/pages/user/customer/customerInfo/customerInfo.jsx

customerInfo.jsx is the frontend page that allows a registered user of Justontime to view and update their personal information. This page loads a display of the information they've inputted. This page loads input fields with buttons that toggle editing mode. Once edit mode is toggled there are 2 buttons that either save the changes or allow the user to cancel their changes and toggle the edit mode as well. All changes to the information are validated and can only be saved if the changes are valid and were able to be updated in the backend. Errors are sent if the input is not valid, for example if the name entered is too short it will send an error telling the user the length of the input is invalid. Methods in this file are as follows: onSave, validate, organizer, displaytype, CustomerInfo.

- onSave
    - Saves the user's changes when the save buttons is clicked
- validate
    - Validates the user's inputted changes to the users information
- organizer
    - Determines if the user is an organizer or not then returns the corresponding information based on type of user
- displaytype
    - Displays all the information about the user
- CustomerInfo
    - Returns that same as displaytype if the user is logged in else returns an error message and links that redirect to the login or registration page

client/src/pages/user/customer/home/CustomerHome.jsx

CustomerHome.jsx is the landing page for users when they visit our website. It displays events that users might be interested in seeing based on their location, and price. It displays some events on a large carousel near the top of the page then the events are grouped based on categories and displayed in horizontal sliders. The only method in this file is: CustomerHome.

- CustomerHome
    - Returns the display for the homepage

client/src/pages/user/verification/verificationRequired/verificationRequired.jsx

verificationRequired.jsx is the frontend page that the user is redirected to after signing up. It informs the user to check their email to verify it so they can use the service and it has a link to

resend the verification link if needed. Methods in this file are as follows: handleResendCode, CustomerVerifyEmail.

- handleResendCode
    - Sends an email when the resend button is clicked and return an error if one occurs
- CustomerVerifyEmail
    - Returns the messages to the reader about verifying their email and returns and error if one occurring during the email resend

client/src/pages/user/verification/verifyEmail/verifyEmail.jsx

verifyEmail.jsx is the frontend page that the user gets redirected to after the verificationRequired page. It is blank if everything goes smoothly otherwise if some error occurred when sending the verification email to the user it will show an error message. The only method in this file is: CustomerVerifyEmail.

- CustomerVerifyEmail
    - Returns an error if one has occurred else displays a blank page

client/src/pages/user/login.jsx

login.js is the frontend page that allows a registered user of Justontime to use our website. This page loads a display of the login page. This page loads a form that takes 2 inputs: Username and password. There is a ‘login’ button that allows the user to submit the data and a link to the sign page if the user is not already registered. All input on the page is required as such errors are sent to the user if any of the text fields are left empty. Errors are also sent to the login page from the backend if the information provided is not accurate. For example if the provided username does not exist. Methods in this file are as follows: onSubmit, onChange, Login.

- onSubmit
    - Ensures all the data fields are not empty and returns an error if there are any invalid input fields
- onChange
    - Updates form as changes are made
- Login
    - Returns the display for the login page

client/src/pages/user/signup.jsx

signup.jsx is the frontend page that allows a potential customer to become a registered user of Justontime. This page loads a display of the signup page. This page loads the SignUpForm component only. Methods in this file are as follows: onChange, onSubmit, Signup.

- onChange
    - Updates form as changes are made
- onSubmit
    - Ensures all the data fields are not empty and returns an error if there are any invalid input fields
- Signup
    - Returns the display for the signup page

Services

client/src/services/admin/verifyEventService.js

verifyEventService.js connects the frontend with the backend using axios. It stores all api calls needed to change the status of an organizer's verification. Methods in this file are as follows: loadEvents, verifyEvent, rejectEvent.

- loadEvents
    - Makes a request to the endpoint ‘/api/admin/getUnverifiedEvents’ then returns the response which is all the unverified events from the backend
- verifyEvent
    - Makes a request to the endpoint ‘/api/admin/updateEventStatus’ with a body that is meant to verify the event using the eventId
- rejectEvent
    - Makes a request to the endpoint ‘/api/admin/updateEventStatus’ with a body that is meant to reject the event using the eventId

client/src/services/admin/verifyOrganizerService.js

verifyOrganizerService.js connects the frontend with the backend using axios. It stores all api calls needed to change the status of an organizer's verification. Methods in this file are as follows: loadOrganizers, verifyOrganizer, rejectOrganizer.

- loadOrganizers
    - Makes a request to the endpoint ‘/api/admin/getUnverifiedOrganizers’ then returns the response which is all the unverified organizers from the backend
- verifyOrganizer

    - Makes a request to the endpoint '/api/admin/updateOrganizerStatus' with a body that is meant to verify the organizer using the organizers email
- rejectOrganizer
    - Makes a request to the endpoint '/api/admin/updateOrganizerStatus' with a body that is meant to reject the organizer using the organizers email

client/src/services/auth/authService.js

authService.js connects the frontend with the backend using axios. It stores all api calls needed during a user's logged in session. Methods in this file are as follows: registerUser, loginUser, and updateUser.

- registerUser
    - Makes a request to the endpoint '/api/user/register' to register the user, if successful adds the user to the local storage then returns the response
- loginUser
    - Makes a request to the endpoint '/api/user/login' to login the user, if successful adds the user to the local storage then returns the response
- updateUser
    - Makes a request to the endpoint '/api/user/personal-info' to update the user's information, if successful adds the user to the local storage then returns the response

client/src/services/auth/authSlice.js

authSlice.js handles the user session part of the redux store. It gets the user from local storage and then handles the storing of information that pertains to the user once they are logged in and any errors that relate to registration or logging in. Methods in this file are as follows: registerUser, loginUser, updateUser, authSlice.

- registerUser
    - Calls the corresponding method from authService to register the user then returns the response from the method
- loginUser
    - Calls the corresponding method from authService to login the user then returns the response from the method
- updateUser
    - Calls the corresponding method from authService to update the user information then returns the response from the method
- authSlice
    - Creates a slice of the redux store that store the relevant information needed to register, login and update a user, and updates the store based on the return value of the other methods in the file

client/src/services/emailVerification/customer/resendCode.js

resendCode.js connects the frontend with the backend using axios. It stores the api call needed to resend a verification email. The only method in this file is: resendCode.

- resendCode
    - Makes a request to the endpoint '/api/customer/resendcode' and sends the user to send the email to then returns the response.

client/src/services/emailVerification/customer/verifyEmail.js

verifyEmail.js connects the frontend with the backend using axios. It stores the api call needed to send a verification email. The only method in this file is: verifyEmail.

- verifyEmail
    - Makes a request to the endpoint '/api/customer/verifyemail' + 'email/token' then returns the response.