2 Method Specifications

2.1 Authentication Controller

2.1.1 Sign up

Description
This Node.js code exports a function called singup_post which is an asynchronous function. This function is responsible for creating a new user account by storing the user's information in the database. The function also sends an email to the user to confirm their account.

Route
POST/signup

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The email and password fields must be provided in the HTTP request body.
The email must not already exist in the database.

Returns
This function returns a JSON object that contains the following properties:
user: A string indicating the ID of the newly created user account.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that the user account could not be created.

Code Explanation
The singup_post function first extracts the user's information from the HTTP request body.
It then checks whether the user's email already exists in the database. If it does, the function sends an HTTP response with a status code of 418 and a message indicating that the user already exists.
If the user's email is not in the database, the function creates a new user account by calling the User.create function with the user's information. The function then generates an email

verification link for the user and sends an email to the user's email address. Finally, the function sends an HTTP response with a status code of 201 and a JSON object containing the ID of the newly created user account. If an error occurs during the creation of the user account, the function catches the error, generates an error message using the handleErrors function, and sends an HTTP response with a status code of 400 and a JSON object containing the error message.

2.1.2 Login

Description
This Node.js code exports a function called login_post which is an asynchronous function. This function is responsible for authenticating a user and creating an access token for them if the authentication is successful. The function first extracts the email and password from the HTTP request body. If the user is a traveler, the function checks whether their account has been approved by the admin, and whether their account has been revoked. If the user is an admin, the function checks whether the email and password are valid. If the user is a registered user but has not yet confirmed their email, the function returns an HTTP response with a status code of 406 and a message indicating that the user should confirm their email. If the email and password are valid and the user has been authenticated, the function creates an access token and a refresh token for the user, saves the tokens to the database, and returns an HTTP response with a status code of 200 and the user's ID, type, and access token in JSON format.

Route
POST/login

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
User already has an account

Returns
This function returns a JSON object that contains the following properties:
user: The user's ID.
type: The user's type.
token: The authentication token.

Throws

If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that the authentication was unsuccessful.

Code Explanation
The login_post function first extracts the email and password from the HTTP request body. If the user is a traveler and their account has not been approved, the function returns an HTTP response with a status code of 400 and a message indicating that the account has not been approved. If the user is a traveler and their account has been revoked, the function returns an HTTP response with a status code of 400 and a message indicating that the account has been revoked. If the user is a registered user but has not yet confirmed their email, the function returns an HTTP response with a status code of 406 and a message indicating that the user should confirm their email. If the email and password are valid and the user has been authenticated, the function creates an access token and a refresh token for the user, saves the tokens to the database, and returns an HTTP response with a status code of 200 and the user's ID, type, and access token in JSON format. If an error occurs during the execution of the function, the function catches the error, handles it, and returns an HTTP response with a status code of 400 and the error message in JSON format.

2.1.3 Confirm Email

Description
This Node.js code exports a function called confirmEmail_get which is an asynchronous function. This function is responsible for confirming a user's email address by updating the user's valid_e field to true in the database.

Route
GET/confirm-email/:id/:token

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must have clicked on the email verification link and provided a valid id and token.
The id and token must be valid and exist in the database.

Returns
This function returns a string indicating whether the email has been successfully confirmed or not.

Throws
If an error occurs during the execution of this function, it will send an HTTP response with a message indicating that an error has occurred.

Code Explanation
The confirmEmail_get function first extracts the id and token from the HTTP request parameters.
It then verifies the token using the secret email key.
If the token is valid, the function retrieves the user object from the database and updates the user's valid_e field to true. The function then sends an HTTP response with a message indicating that the email has been successfully confirmed.
If the token is invalid, the function sends an HTTP response with a message indicating that an error has occurred.

2.1.4 Logout

Description
This Node.js code exports a function called logout_post which is an asynchronous function. This function is responsible for logging out a user by deleting their access token from the database.

Route
POST/logout

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid access token to access the route.

Returns
This function returns a JSON object that contains the following properties:
message: A string indicating whether the user has been successfully logged out or not.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 500 and a message indicating that there was an error logging out.

Code Explanation
The logout_post function first extracts the access token from the HTTP request header.

It then deletes the access token from the database using the Token model's deleteOne method.
If the deletion is successful, the function sends an HTTP response with a status code of 200 and a message indicating that the user has been successfully logged out. If an error occurs during the execution of the function, the function catches the error, logs it to the console, and sends an HTTP response with a status code of 500 and a message indicating that there was an error logging out.

2.2 Client Controller

2.2.1 Confirm Order

Description
This code exports a function that confirms an order and updates its status and client confirmation in the database. The function receives a JWT token and an order ID as parameters and uses them to verify if the order is valid and if the client has confirmed it. If the order is valid and the client has confirmed it, the function updates the order's status to 2 and sets its client_confirmed field to true. If the order has not been approved by an admin or the confirmation date has expired, the function returns an error message.

Route
GET/confirmorder/:token/:orderid

Parameters
req - the request object from the client
res - the response object to send back to the client
req.params.token - a JWT token used to verify the order's confirmation
req.params.orderid - the ID of the order to confirm

Preconditions
The Order model must be defined and imported.
The jwt package must be installed and imported.
The SECRET_CONFIRM_ORDER environment variable must be defined.

Returns
If the order is confirmed successfully, the function returns a JSON object with a message indicating that the order has been confirmed.

If the order has not been approved by an admin, the function returns a JSON object with a message indicating that the order has not yet been approved.
If the confirmation date has expired, the function returns a JSON object with a message indicating that the confirmation date has expired.
Throws
If there is an error finding or updating the order, the function logs the error to the console.

Code Explanation
The function receives the JWT token and order ID from the request parameters.
The function retrieves the SECRET_CONFIRM_ORDER environment variable, which is used to verify the token.
The function verifies the token using the jwt.verify() method, which decodes the token and returns its payload. If the token is invalid, the function returns an error message.
If the token is valid, the function tries to find the order in the database using the Order.findById() method.
If the order is found and its status is 1 (meaning it has been accepted by an admin) and its traveler field is not null, the function updates the client_confirmed field to true, sets the status to 2, and saves the order to the database. The function then sends a response to the client with a message indicating that the order has been confirmed.
If the order has not been approved by an admin, the function sends a response to the client with a message indicating that the order has not yet been approved.
If there is an error finding or updating the order, the function logs the error to the console.
If the confirmation date has expired, the function sends a response to the client with a message indicating that the confirmation date has expired.

2.2.2 Get List of Pending Orders

Description
This code exports a function that retrieves a list of pending orders for a client, reads the order and product data, and sends it as a JSON response. The function receives a client ID and a NAT token as parameters, and uses them to find the client and authenticate the request. The function retrieves the list of pending orders from the client's pending_orders field, retrieves the order and product data using their respective IDs, and creates a new object containing both pieces of data. The function then adds the new object to a list and sends it as a JSON response to the client.

Route
GET/client/home/pendingorders

Parameters

req - the request object from the client
res - the response object to send back to the client
req.userId - the ID of the authenticated client
req.nat - the NAT token used to authenticate the request
Preconditions
The User, Order, and Product models must be defined and imported.

Returns
If the client is found and has pending orders, the function returns a JSON object containing a list of objects, each containing an order and a product, and the NAT token used to authenticate the request.
If the client is not found, the function returns a JSON object with a message indicating that the client was not found.
Throws
If there is an error finding the client, orders, or products, the function logs the error to the console.

Code Explanation
The function receives the authenticated client ID and NAT token from the request parameters.
The function finds the client in the database using the User.findById() method.
If the client is found, the function retrieves the list of pending orders from the client's pending_orders field.
The function initializes an empty array list1 to hold the order and product objects.
The function uses a for loop to iterate over each order ID in the list of pending orders.
For each order ID, the function retrieves the order object from the database using the Order.findById() method.
The function retrieves the product object associated with the order using the Product.findById() method.
The function creates a new object obj containing both the order and product objects.
The function adds the new object obj to the list1 array.
After all order and product objects have been retrieved, the function sends a JSON response to the client containing the list1 array and the NAT token used to authenticate the request.
If the client is not found, the function sends a JSON response to the client with a message indicating that the client was not found.
If there is an error finding the client, orders, or products, the function logs the error to the console.

2.2.3 Get list of Active Orders

Description
This function gets the list of active orders for a client, reads the order and product data, and sends it as JSON.

Route
GET /client/home/activeorders

Parameters
req - the request object from the client
res - the response object to send back to the client
req.userId - the ID of the authenticated client
req.nat - the NAT token used to authenticate the request

Preconditions
The User, Order, and Product models must be defined and imported.

Returns
If successful, the function returns an HTTP status code of 200 and a JSON object containing an array of active orders and a token.
If unsuccessful, the function returns an HTTP status code of 404 and a JSON object containing an error message and a token.
Throws
If there is an error while reading the active orders or product data, the function logs the error to the console.

Code Explanation
The function starts by extracting the userId and nat parameters from the request.
It then tries to find the client in the database using the clientId parameter.
If the client is found, the function gets the list of active orders for the client and reads the order and product data.
It then constructs an array of objects, with each object containing an order and its associated product.
Finally, the function sends a JSON response containing the array of orders and a token. If the client is not found, the function sends an error message and a token.

2.2.4 Mark Order as Complete

Description

This Node.js code exports a function called complete_order_post which is an asynchronous function. This function is responsible for completing an order by updating the order and client objects in the database. The function also sends an email to the traveler informing them that their order has been completed.

Route
POST/client/home/activeorder/:orderid/markascomplete

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.
The clientId and orderId must be valid and exist in the database.
The order status must be 6.

Returns
This function returns a JSON object that contains the following properties:
message: A string indicating whether the order has been successfully completed or not.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 404 and a message indicating that the client or order could not be found.

Code Explanation
The complete_order_post function first extracts the token, clientId, and orderId from the HTTP request.
It then uses the clientId and orderId to retrieve the client and order objects from the database.
If the client and order exist in the database and the order status is 6, the function updates the order status to 7 and updates the client and traveler objects in the database by adding the completed order to their respective completed_orders array and removing it from their active_orders array.
If the traveler has no new orders or assigned orders, the function sets the traveler's active status to false and removes the pickup location.
The function then sends an email to the traveler informing them that their order has been completed and returns a JSON object with a status code of 200 and a message indicating that the order has been successfully completed.

If the client or order cannot be found, the function returns a JSON object with a status code of 404 and a message indicating that the client or order could not be found.

2.2.5 Get Profile

Description
This Node.js code exports a function called getProfile which is an asynchronous function. This function retrieves the profile of a client by using the client's userId and returns it in a JSON object along with an authentication token.

Route
GET/client/home/profile

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.

Returns
This function returns a JSON object that contains the following properties:
client: An object containing the client's profile information.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that something went wrong.

Code Explanation
The getProfile function first extracts the clientId and token from the HTTP request.
It then uses the clientId to retrieve the client object from the database.
If the client object exists, the function sends an HTTP response with a status code of 200 and a JSON object containing the client object and the authentication token.
If the client object does not exist, the function sends an HTTP response with a status code of 400 and a message indicating that something went wrong.

2.2.6 Edit Profile

Description
This Node.js code exports a function called editProfile which is an asynchronous function. This function is responsible for editing a user's profile information by updating the user's document in the database.

Route
PUT/client/home/profile/edit

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.

Returns
This function returns a JSON object that contains the following properties:
user: The updated user document.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 500.

Code Explanation
The editProfile function first extracts the clientId and token from the HTTP request. It then uses the clientId to retrieve the user object from the database. If the user object exists, the function attempts to update the user's document with the new profile information contained in the request body. If the update is successful, the function sends a JSON response with the updated user document and the authentication token. If an error occurs during the update process, the function sends a JSON response with a status code of 500 and the error message. If the user object does not exist, the function sends a JSON response with a status code of 500 and an error message.

2.2.7 Change Password

Description

This Node.js code exports a function called changePass_post which is an asynchronous function. This function is responsible for changing the password of a client by hashing and saving the new password to the client object in the database.

Route
POST/client/home/profile/edit/changepassword

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.
The client's old password must be valid and match the password stored in the database.

Returns
This function returns a JSON object that contains the following properties:
message: A string indicating whether the password has been successfully changed or not.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 500 and a message indicating that the password could not be changed.

Code Explanation
The changePass_post function first extracts the token, password, newPassword, and clientId from the HTTP request.
It then uses the clientId to retrieve the client object from the database.
If the client object exists, the function compares the old password with the password stored in the database using bcrypt.compare. If the passwords match, the function generates a salt using bcrypt.genSalt and hashes the new password using bcrypt.hash. The function then saves the hashed password to the client object and returns an HTTP response with a status code of 200 and a message indicating that the password has been successfully changed.
If the passwords do not match, the function returns an HTTP response with a status code of 500 and a message indicating that the old password is incorrect.
If an error occurs during the execution of the function, the function returns an HTTP response with a status code of 500 and a message indicating that the password could not be changed.

2.2.8 Get USD to LBP rate

Description
This Node.js code exports a function called getRate_get which is an asynchronous function. This function is responsible for fetching the latest exchange rate of a specific currency from a website and returning it in a JSON object.

Route
POST /getRate

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
None.

Returns
This function returns a JSON object that contains the following properties:
rate: A string representing the latest exchange rate of a specific currency**.**

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that an error has occurred.

Code Explanation
The getRate_get function first creates an options object containing the URL of the website to fetch the exchange rate and other options. It then uses the request module to send a request to the website and fetches the HTML response. The cheerio module is used to parse the HTML response and extract the latest exchange rate. The extracted rate is then returned in a JSON object in the HTTP response. If an error occurs during the execution of the function, it sends an HTTP response with a status code of 400 and an error message.

2.2.9 Send FeedBack After Completion of Order

Description
This Node.js code exports a function called giveFeedback_post which is an asynchronous function. This function is responsible for adding feedback to an order by creating a Feedback object and updating the order's feedback field. The function then sends an HTTP response indicating whether the feedback was added successfully or not.

Route
POST/client/home/activeorder/:orderid/markascomplete/feedback

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.
orderid (string): A string representing the ID of the order that the feedback is being added to.

Preconditions
The user must be authenticated and have a valid token to access the route.
The order with the given orderid must exist in the database.

Returns
This function returns a JSON object that contains the following properties:
message: A string indicating whether the feedback has been successfully added or not.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 500 and a message indicating that a server error has occurred. If the order with the given orderid is not found in the database, the function will throw an error with a status code of 404 and a message indicating that the order was not found**.**

Code Explanation
The giveFeedback_post function first extracts the token and orderid from the HTTP request. It then uses the orderid to retrieve the order object from the database. If the order object is found, the function creates a Feedback object using the feedback data from the request body. It then updates the order's feedback field with the ID of the newly created Feedback object and saves the order object to the database. Finally, the function sends an HTTP response indicating whether the feedback was added successfully or not. If an error occurs during the execution of this function, the function sends an HTTP response with an appropriate error message and status code.

2.2.10 Submit a Contact Form

Description
This Node.js code exports a function called submitContactForm which is an asynchronous function. This function is responsible for submitting a contact form by creating a new

ContactForm object in the database. The function also returns an HTTP response with a success message and an authentication token.

Route
POST/client/home/contactform

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information**.**

Preconditions
The user must be authenticated and have a valid token to access the route.

Returns
This function returns a JSON object that contains the following properties:
message: A string indicating whether the contact form has been successfully submitted or not.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that something went wrong.

Code Explanation
The submitContactForm function first extracts the clientId and token from the HTTP request.
It then uses the clientId to retrieve the client object from the database.
If the client is found, the function tries to extract the email, team, subject, and message from the HTTP request and creates a new ContactForm object with this data.
If the ContactForm object is successfully created, the function sends an HTTP response with a status code of 200 and a success message containing the authentication token.
If an error occurs during the execution of the function, the function sends an HTTP response with a status code of 400 and a message indicating that something went wrong. If the client is not found, the function sends an HTTP response with a status code of 404 and a message indicating that the client could not be found.

2.2.11 Submit Support Form

Description
This Node.js code exports two functions, submitContactForm and submitSupportForm, both of which are asynchronous. The submitContactForm function is responsible for saving a new

contact form submission to the database and sending a response with a success message and authentication token. The submitSupportForm function is responsible for sending an email to the support team and sending a response with a success message and authentication token.

Route
POST/support

Parameters
For both functions:
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
For submitContactForm:
The user must be authenticated and have a valid token to access the route.
The clientId must be valid and exist in the database.
For submitSupportForm:
The user must be authenticated and have a valid token to access the route.
The clientId must be valid and exist in the database.
The client or traveler must be identified in the request body.
The email configuration and transporter object must be properly set up.

Returns
For both functions:
This function returns a JSON object that contains the following properties:
message: A string indicating whether the operation has been successfully completed or not.
token: The authentication token.
Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that the operation could not be completed.

Code Explanation
For submitContactForm:
The function first extracts the clientId and token from the HTTP request.
It then retrieves the client object from the database using the clientId.
If the client exists, the function extracts the contact form details from the request body and saves them to the database.
If the save operation is successful, the function sends an HTTP response with a status code of 200 and a success message and token.

If the save operation fails, the function sends an HTTP response with a status code of 400 and an error message and token.
If the client does not exist, the function sends an HTTP response with a status code of 404 and an error message and token.
For submitSupportForm:
The function first extracts the clientId and token from the HTTP request.
It then retrieves the client or traveler object from the database using the clientId.
If the object is a traveler, the function constructs an email message and sends it to the support team.
If the object is a client, the function constructs an email message and sends it to the support team.
If the email is sent successfully, the function sends an HTTP response with a status code of 200 and a success message and token.
If the email fails to send, the function sends an HTTP response with a status code of 400 and an error message and token.

2.3 Product Controller

2.3.1 Scrape Amazon Product Details

Description
This Node.js code exports a function called productsearch_post which is responsible for processing a product link received through an HTTP POST request and returning details about the product. The function calls the getData function to retrieve the necessary data from the product link and formats it into a JSON object with the required details.

Route
POST/client/home/searchproduct

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
User should logged in

Returns
This function returns a JSON object that contains the following properties:

title: The title of the product.
price: The price of the product.
asin: The Amazon Standard Identification Number of the product.
imageSource: The URL of the product image.
dimensions: The dimensions of the product in inches, formatted as a string. If the dimensions or weight are unavailable, they are set to -1.
weight: The weight of the product in pounds. If the weight is unavailable, it is set to -1.
inStock: A boolean value indicating whether the product is currently in stock.
url: The URL of the product page
.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that the link is invalid or empty.

Code Explanation
The productsearch_post function first extracts the product link from the HTTP request.
It then calls the getData function to retrieve the necessary details about the product from the link.
If the dimensions or weight are unavailable, they are set to -1.
The function then formats the details into a JSON object and sends it in the HTTP response.

2.3.2 Request An Amazon Product

Description
This Node.js code exports a function called productrequest_post which is an asynchronous function. This function is responsible for creating a new order for a client. If the requested product already exists, the function creates a new order with the cost of the product updated for the requested quantity, and if it does not exist, it creates a new product with the requested details and creates a new order for the client. The function returns a JSON object with a message and a token.

Route
POST/client/home/searchproduct/:asin/:quantity

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.

Returns
This function returns a JSON object that contains the following properties:
message: A string indicating whether the order has been successfully created or not.
accessToken: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 406 and a message indicating that the product is out of stock.

Code Explanation
The productrequest_post function first extracts the newNAT, data, and quantity from the HTTP request.
It then uses the data to retrieve the product object from the database with the matching asin.
If the product exists in the database, the function checks if there is already a cost associated with the product. If so, it creates a new order with the updated cost for the requested quantity. If not, it creates a new order without a cost for the requested quantity.
If the product does not exist in the database and has no price, the function throws an error with a status code of 406 and a message indicating that the product is out of stock.
If the product does not exist in the database and has a price, the function creates a new product with the requested details and creates a new order for the client with the requested quantity. The function then returns a JSON object with a status code of 200 and a message indicating that the order has been successfully created, along with the new authentication token.

2.4 Password Reset

2.4.1 Forgot Password Email Submission

Description
This Node.js code exports an asynchronous function called fp_post. This function is responsible for generating and sending a password reset link to a user's email. The function first extracts the email from the HTTP request body. It then uses this email to search for the user in the User and Traveler collections in the database. If the user is found, the function generates a JSON Web Token (JWT) using the user's information and a secret key. The function then creates a password reset link using the user's ID and the JWT and sends this link to the user's email address. Finally, the function sends an HTTP response to the client indicating that the password reset link has been sent.

Route
POST/forgot-password

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The email parameter must be present in the Database as well as the HTTP request.
The user with the specified email must exist in either the User or Traveler collection in the database.

Returns
This function returns an HTTP response with a message indicating that the password reset link has been sent.

Throws
N/A

Code Explanation
The fp_post function first extracts the email from the HTTP request body. It then searches for the user with this email in the User and Traveler collections in the database using the findOne method of the Mongoose library. If the user is found, the function creates a payload object containing the user's name, last name, email, ID, and user type. It then generates a JWT using this payload and a secret key. The function then creates a password reset link using the user's ID and the JWT and sends this link to the user's email address using the sendEmail function. Finally, the function sends an HTTP response to the client indicating that the password reset link has been sent.

2.4.2 Reset Password

Description
This Node.js code exports a function called rp_post which is an asynchronous function. This function is responsible for changing the password of a user or a traveler by hashing and updating their new password in the database. The function takes in the user/traveler ID, token, and new password as parameters from an HTTP POST request. The function verifies the token, checks the type of user (User or Traveler), finds the user/traveler object in the database using their ID,

hashes and updates the new password, and returns a JSON object containing a message indicating whether the password was successfully changed or not.

Route
POST/reset-password/:id/:token

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.
Preconditions
The user/traveler must be authenticated and have a valid token to access the route.
The user/traveler ID and token must be valid and exist in the database.
The new password and confirm password fields in the request body must match.

Returns
This function returns a JSON object that contains the following properties:
message: A string indicating whether the password was successfully changed or not.

Throws

If an error occurs during the execution of this function, it will throw an error with a status code of 500 and a message indicating that a server error occurred, or a status code of 400 and a message indicating that the token is invalid.

Code Explanation

The rp_post function first extracts the user/traveler ID, token, and new password fields from the HTTP request. It then verifies the token using the secret key, checks the type of user, and finds the user/traveler object in the database using their ID. If the user/traveler object exists and the new password and confirm password fields in the request body match, the function generates a salt and hashes the new password using bcrypt. It then updates the user/traveler object in the database with the new hashed password and returns a JSON object with a message indicating that the password was successfully changed. If the token is invalid, the function returns a JSON object with a status code of 400 and a message indicating that the token is invalid. If a server error occurs, the function returns a JSON object with a status code of 500 and a message indicating that a server error occurred.

2.5 Traveler Controller

2.5.1 Accept Order

Description
This Node.js code exports a function called accept_order which is an asynchronous function. This function is responsible for assigning an order to a traveler by updating the traveler's assigned orders array and the order's waiting_resp and ticket fields. The function also updates the client's active orders array and sends an email to the client informing them that their order has been assigned to a traveler.

Route
POST/traveler/home/neworders/:orderid/accept

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.
Preconditions
The user must be authenticated and have a valid token to access the route.
The travelerId and orderId must be valid and exist in the database.
The order must not have been previously confirmed by the client.
Return
This function returns a JSON object that contains the following properties:
message: A string indicating whether the order has been successfully assigned or not.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that the order could not be assigned.

Code Explanation
The accept_order function first extracts the travelerId, orderId, and token from the HTTP request.
It then uses the travelerId and orderId to retrieve the traveler and order objects from the database.
If the order has not been confirmed by the client yet, the function sends an HTTP response with a status code of 400 and a message indicating that the traveler should wait for the client to confirm the order.
If the order has been confirmed by the client, the function checks whether the traveler has not previously accepted the order, the order status is 2, and the traveler has a valid ticket. If these conditions are met, the function assigns the order to the traveler by updating the assigned_orders,

waiting_resp, and ticket fields of the order, and the active_orders and pending_orders fields of the client. The function then sends an email to the client informing them that their order has been assigned to a traveler. Finally, the function saves the updated traveler and order objects to the database and sends an HTTP response with a status code of 200 and a message indicating that the order has been successfully assigned.

2.5.2 Reject Order

Description
The `reject_order` function is an async function that allows a traveler to reject an order. This function updates the status of the order to "0" which means the order is available for other travelers to accept.

Route
POST/traveler/home/neworders/:orderid/reject

Parameters
- `req`: The request object containing the user ID, order ID, and token.
- `res`: The response object that will contain the result of the function.

Preconditions
- The traveler must be logged in to the system and authorized to reject orders.
- The order must exist in the database and be waiting for a traveler to accept.
- The order must not have been assigned to another traveler.
- The traveler must have received the order as a new order.

Returns
The function returns a JSON object with a message indicating whether the order was successfully rejected or not, along with a token.

Throws
The function may throw an error if there is an issue with the database or if the traveler is not authorized to reject orders.

Code Explanation:
The `reject_order` function begins by getting the `travelerId`, `orderId`, and `token` from the request object. It then tries to find the traveler and order objects in the database using the IDs provided.

If the order meets the conditions to be rejected, the function updates the order's status, pickup location, estimated arrival, waiting_resp, and traveler properties to their initial values. It then saves the updated order and traveler objects to the database.
Finally, the function sends a JSON response with a message indicating that the order was rejected and a token. If there is an error during this process, the function sends a JSON response indicating that the order could not be rejected and a token.

2.5.3 Cancel Flight

Description
This Node.js code exports a function called cancel_flight which is an asynchronous function. This function is responsible for canceling all orders assigned to a traveler by setting their respective fields to default values, updating the traveler's order arrays, and sending an email to each client informing them that their order has been canceled.

Route
POST/traveler/home/uploadTicket

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.
The travelerId must be valid and exist in the database.

Returns
This function returns a JSON object that contains the following properties:
message: A string indicating whether the flight has been successfully canceled or not.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that the flight could not be canceled.

Code Explanation
The cancel_flight function first extracts the token and travelerId from the HTTP request and retrieves the corresponding traveler object from the database.

The function then loops through the traveler's new_orders and assigned_orders arrays, canceling each order by setting their respective fields to default values and sending an email to each client to inform them of the cancellation.
After canceling all the orders, the function updates the traveler's order arrays, sets their active status to false, and sets their provided_pickup field to an empty string before saving the updated traveler object to the database.
Finally, the function sends an HTTP response with a status code of 200 and a message indicating that the flight has been successfully canceled, along with the authentication token. If an error occurs during the execution of this function, it sends an HTTP response with a status code of 400 and a message indicating that the flight could not be canceled.

2.5.4 Provide Pickup Location

Description
This Node.js code exports a function called providePickup_post which is an asynchronous function. This function is responsible for updating the provided_pickup field of a traveler object in the database. It receives the pickup location from the request body and updates the traveler object with this information.

Route
POST /providePickup

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.
The travelerId must be valid and exist in the database.

Returns
This function returns a JSON object that contains the following properties:
message: A string indicating whether the pickup location has been successfully provided or not.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that the pickup location could not be provided.

Code Explanation
The providePickup_post function first extracts the travelerId and token from the HTTP request. It then uses the travelerId to retrieve the traveler object from the database.
The function then updates the provided_pickup field of the traveler object with the pickup location received in the request body. Finally, the function saves the updated traveler object to the database and sends an HTTP response with a status code of 200 and a message indicating that the pickup location has been successfully provided.

2.5.5 Upload Receipt

Description
This Node.js code exports a function called uploadReceipt which is an asynchronous function. This function is responsible for uploading a receipt file for an order and updating the corresponding order object in the database.

Route
POST/traveler/home/uploadReceipt

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.
The orderId must be valid and exist in the database.

Returns
This function does not return a JSON object. Instead, it sends an HTTP response with a message indicating that the upload is complete.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that the upload failed.

Code Explanation
The uploadReceipt function first uses the upload1 middleware to upload a single file from the HTTP request. If an error occurs during the upload process, the function sends an HTTP response with a status code of 400 and a message indicating that the upload failed.

The function then extracts the orderId from the HTTP request and retrieves the corresponding order object from the database. It updates the order's receipt field with the filename of the uploaded file and saves the updated order object to the database. Finally, the function sends an HTTP response with a message indicating that the upload is complete.

2.5.6 Upload Proof

Description
This Node.js code exports a function called uploadProof_post which is an asynchronous function. This function is responsible for uploading a file and updating the proof field of an order in the database.

Route
POST/traveler/home/activeorders/:orderid/uploadproof

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.
The orderId must be valid and exist in the database.

Return
This function returns a JSON object that contains the following properties:
- message: A string indicating that the upload is done.
- token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that the file could not be uploaded.

Code Explanation
The uploadProof_post function first extracts the token from the HTTP request and uses it for authentication.
It then uses the orderId to retrieve the order object from the database.
If there is an error during file upload, the function sends an HTTP response with a status code of 400 and an error message.

Otherwise, the function updates the proof field of the order with the filename of the uploaded file and saves it to the database. Finally, the function sends an HTTP response with a status code of 200 and a message indicating that the upload is done. If an error occurs during the execution of the function, it sends an HTTP response with a status code of 400 and an error message.

2.5.7 Mark Order as Shipped

Description
This Node.js code exports a function called markshipped which is an asynchronous function. This function is responsible for updating the status of an order to 'shipped' and sending an email to the client informing them that their order is on the way.

Route
POST/traveler/home/activeorders/:orderid/markassent

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.
The orderId must be valid and exist in the database.

Returns
This function returns a JSON object that contains the following properties:
- message: A string indicating whether the order has been successfully marked as shipped or not.
- token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 500 and a message indicating that a server error occurred. If the order is not found, it will throw an error with a status code of 404 and a message indicating that the order was not found.

Code Explanation
The markshipped function first extracts the token and orderId from the HTTP request.
It then uses the orderId to retrieve the order object from the database.
If the order exists, the function checks whether the order status is 3 (i.e., 'ready to ship').
If the order status is 3, the function updates the order status to 4 (i.e., 'shipped') and saves the changes to the database. The function then retrieves the client and product objects related to the

order and sends an email to the client using the sendOnTheWayEmail function, informing them that their order is on the way. Finally, the function sends an HTTP response with a status code of 200 and a message indicating that the order has been successfully marked as shipped. If the order does not exist, the function sends an HTTP response with a status code of 404 and a message indicating that the order was not found. If an error occurs during the execution of the function, it sends an HTTP response with a status code of 500 and a message indicating that a server error occurred.

2.5.8 Mark Order as Arrived

Description
This Node.js code exports a function called markarrived which is an asynchronous function. This function is responsible for updating the status of an order to "arrived" by setting its status field to 5. It also sends an email to the client informing them that their order has arrived.

Route
POST/traveler/home/activeorders/:orderid/markarrived

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.
The orderId must be valid and exist in the database.

Returns
This function returns a JSON object that contains the following properties:
message: A string indicating whether the order has been marked as arrived or not.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 500 and a message indicating that a server error occurred. If the order is not found, it will throw an error with a status code of 404 and a message indicating that the order was not found.

Code Explanation
The markarrived function first extracts the orderId and token from the HTTP request.
It then uses the orderId to retrieve the order object from the database.

If the order exists, the function checks whether the order status is 4 (shipped). If it is, the function updates the order status to 5 (arrived) and saves it to the database. It then retrieves the client and product objects related to the order and sends an email to the client informing them that their order has arrived.
Finally, the function sends an HTTP response with a status code of 200 and a message indicating that the order has been successfully marked as arrived. If an error occurs during the execution of the function, it sends an HTTP response with a status code of 500 and a message indicating that a server error occurred. If the order is not found, it sends an HTTP response with a status code of 404 and a message indicating that the order was not found.

2.5.9. Get All Pending Orders

Description
This Node.js code exports a function called getPendingTrav_get which is an asynchronous function. This function is responsible for retrieving a list of pending orders for a specific traveler from the database. The function then sends an HTTP response with a JSON object containing the list of pending orders and the authentication token.

Route
GET/traveler/home/pendingorders

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.
The travelerId must be valid and exist in the database.

Returns
This function returns a JSON object that contains the following properties:
porders: An array of objects containing the traveler's pending orders and their corresponding products.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will log the error to the console.

Code Explanation

The getPendingTrav_get function first extracts the travelerId and token from the HTTP request. It then retrieves the traveler object from the database using the travelerId.
If the traveler object is found, the function retrieves the list of pending orders for the traveler by iterating through the traveler's new_orders array and retrieving the corresponding order and product objects from the database. The function then creates an **array of objects, each** containing an order and its corresponding product, and sends the array as a JSON object in the HTTP response along with the authentication token.
If the traveler object is not found in the database, the function sends an HTTP response with a status code of 404 and a message indicating that the traveler was not found along with the authentication token.

2.5.10 Get All Active Orders

Description
This Node.js code exports a function called getActiveTrav_get which is an asynchronous function. This function is responsible for retrieving the list of assigned orders for a traveler, along with their corresponding product information. The function returns a JSON object containing the list of assigned orders and the authentication token.

Route
POST/traveler/home/activeorders

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.

Returns
This function returns a JSON object that contains the following properties:
aorders: An array of objects containing the assigned order and its corresponding product information.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will log the error to the console.

Code Explanation

The getActiveTrav_get function first extracts the traveler's ID and token from the HTTP request. It then uses the traveler's ID to retrieve the traveler object from the database.
If the traveler object exists, the function retrieves the list of assigned orders for the traveler and iterates over them to retrieve the product information for each order. The function creates an object containing the order and product information and pushes it to a new array.
Finally, the function sends an HTTP response with a status code of 200 and a JSON object containing the list of assigned orders and the authentication token. If the traveler object does not exist, the function sends an HTTP response with a status code of 404 and a message indicating that the client was not found. If an error occurs during the execution of the function, it logs the error to the console.

2.5.11 Splash Screen Loader

Description
This Node.js code exports a function called splashScreen_get which is an asynchronous function. This function is responsible for displaying a splash screen for the user based on their user type. If the user is a User, the function returns a JSON object with a status code of 691 and the user's authentication token. If the user is a Traveler, the function returns a JSON object with a status code of 690 and the traveler's authentication token. If the user is neither a User nor a Traveler, the function returns a JSON object with a status code of 692 and an empty token.

Route
GET/checktokenmobile

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.

Returns
This function returns a JSON object that contains the following properties:
status: A number indicating the status of the splash screen.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that a server error has occurred.

Code Explanation
The splashScreen_get function first extracts the token and user type from the HTTP request.
It then checks the user type to determine which splash screen to display. If the user is a User, the function returns a JSON object with a status code of 691 and the user's authentication token. If the user is a Traveler, the function returns a JSON object with a status code of 690 and the traveler's authentication token. If the user is neither a User nor a Traveler, the function returns a JSON object with a status code of 692 and an empty token.
If an error occurs during the execution of this function, the catch block logs the error and sends an HTTP response with a status code of 400 and a message indicating that a server error has occurred.

2.5.12 Get Profile

Description
This Node.js code exports a function called getProfile which is an asynchronous function. This function is responsible for retrieving a traveler's profile information by using the traveler's ID from the HTTP request. The function then sends an HTTP response with the retrieved traveler object and the authentication token.

Route
GET/profile

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.
The traveler's ID must be valid and exist in the database.

Returns
This function returns a JSON object that contains the following properties:
trav: An object containing the traveler's profile information.
token: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that something went wrong.

Code Explanation
The getProfile function first extracts the traveler's ID and token from the HTTP request.
It then uses the traveler's ID to retrieve the traveler object from the database.
If the traveler object exists, the function sends an HTTP response with a status code of 200 and the retrieved traveler object and the authentication token.
If the traveler object does not exist, the function sends an HTTP response with a status code of 400 and a message indicating that something went wrong.

2.5.13 Edit Profile

Description
This Node.js code exports a function called `editProfile` which is an asynchronous function. This function is responsible for updating a traveler's profile in the database.

Route

POST/profile/edit

Parameters
- `req` (object): An object containing the HTTP request information.
- `res` (object): An object containing the HTTP response information.

Preconditions
- The user must be authenticated and have a valid token to access the route.
- The travelerId must be valid and exist in the database.

Returns
This function returns a JSON object that contains the following properties:
- `trav`: An object representing the updated traveler profile.
- `token`: The authentication token.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 500.

Code Explanation
The `editProfile` function first extracts the `travId` and `token` from the HTTP request. It then uses the `travId` to retrieve the traveler object from the database.

If the traveler exists in the database, the function attempts to update the traveler's profile with the data in the HTTP request body using the `findByIdAndUpdate` method. If the update is successful, the function sends a response with the updated traveler object and the authentication token.
If an error occurs during the update process, the function sends a response with a status code of 500 and the error message.
If the traveler does not exist in the database, the function sends a response with a status code of 500 and an error message.

2.5.14 Check if Traveler is Active

Description
This Node.js code exports a function called hasTicket which is an asynchronous function. This function is responsible for checking whether a traveler has a valid ticket or not by retrieving the traveler object from the database and checking the active field. If the active field is true, it retrieves the ticket object from the database and sends a JSON response indicating that the traveler has a valid ticket. If the active field is false, it sends a JSON response indicating that the traveler does not have a valid ticket.

Route
GET/hasTicket

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.
The travelerId must be valid and exist in the database.

Returns
This function returns a JSON object that contains the following properties:
hasTicket: A boolean indicating whether the traveler has a valid ticket or not.
ticket: An object representing the ticket if the traveler has a valid ticket. This property is only present if hasTicket is true.
token: The authentication token.

Throws

If an error occurs during the execution of this function, it will throw an error with a status code of 500 and a message indicating that a server error occurred. If the traveler object cannot be retrieved from the database, it will send an HTTP response with a status code of 400 and a message indicating that something went wrong.

Code Explanation
The hasTicket function first extracts the travelerId and token from the HTTP request.
It then uses the travelerId to retrieve the traveler object from the database.
If the traveler object exists and the active field is true, the function retrieves the ticket object from the database and sends an HTTP response with a status code of 200 and a JSON object containing the hasTicket property set to true and the ticket object. If the active field is false, the function sends an HTTP response with a status code of 200 and a JSON object containing the hasTicket property set to false.
If the traveler object cannot be retrieved from the database, the function sends an HTTP response with a status code of 400 and a message indicating that something went wrong. If an error occurs during the execution of this function, it will throw an error with a status code of 500 and a message indicating that a server error occurred.

2.6 Ticket Controller

2.6.1 Upload Ticket

Description
This Node.js code exports a function called `uploadTicket_post` which is an asynchronous function. This function is responsible for uploading a ticket for a traveler by creating a ticket object in the database and populating it with information extracted from the uploaded PDF file. The function also updates the traveler's active status to true and sends an email to the traveler informing them that their ticket has been received.

Route
POST/traveler/home/uploadTicket

Parameters
- `req` (object): An object containing the HTTP request information.
- `res` (object): An object containing the HTTP response information.

Preconditions

- The user must be authenticated and have a valid token to access the route.

Returns
This function does not return anything. If the ticket is successfully uploaded and parsed, an email is sent to the traveler indicating that their ticket has been received. If the ticket cannot be parsed or contains invalid information, an HTTP response with a status code of 400 and an error message is sent.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that the ticket could not be uploaded.

Code Explanation
The `uploadTicket_post` function first extracts the `traveler`, `token` and `fileData` from the HTTP request. It then creates a new `Ticket` object in the database with the `traveler` ID and assigns it to the `ticket` variable.
If the `traveler` object is found in the database and its `active` status is false, the function proceeds with uploading the ticket. It uses the `upload.single` middleware to handle the file upload and extracts information from the uploaded PDF file using regular expressions.
If the information is successfully extracted, the `ticket` object is populated with the information and saved to the database. The `traveler`'s `active` status is updated to true and an email is sent to the `traveler` informing them that their ticket has been received.
If any of the information cannot be extracted or is invalid, an HTTP response with a status code of 400 and an error message is sent, and the `Ticket` object is deleted from the database.

2.7 Traveler Authentication Controller

Description
This Node.js code exports a function called `tsignup_post` which is an asynchronous function. This function is responsible for handling a POST request for traveler registration. The function extracts the user's personal information and uploaded files, creates a new Traveler object, and sends an email to the user acknowledging receipt of their application.

Route

POST/travelersignup

Parameters

- `req` (object): An object containing the HTTP request information.
- `res` (object): An object containing the HTTP response information.

Preconditions
None.

Returns
This function returns a JSON object that contains the newly created `Traveler` object.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 400 and a message indicating that the registration could not be completed.

Code Explanation
The `tsignup_post` function first logs the body of the HTTP request to the console.
It then creates a new instance of the `multer` middleware to handle file uploads, specifying that files should be stored in a specific S3 bucket and assigning them unique filenames based on the current date and original filename.
The function then calls the `upload.fields` method to handle the file uploads, passing in an array of objects containing the names of the expected files. If an error occurs during the upload process, the function sends an HTTP response with a status code of 400 and a message indicating the error.
If the upload is successful, the function extracts the user's personal information from the `otherData` property of the request body and checks whether a user or traveler with the same email address already exists in the database. If so, the function sends an HTTP response with a status code of 406 and a message indicating that the email already exists.
If the email is unique, the function creates a new `Traveler` object with the provided personal information and assigns the uploaded files to the `cv` and `identification` properties of the object. The function then saves the new `Traveler` object to the database.
The function then composes an email message acknowledging receipt of the user's application and sends it using the configured `transporter` object. Finally, the function sends an HTTP response with a status code of 200 and the newly created `Traveler` object. If an error occurs at any point during the execution of the function, the function logs the error to the console and sends an HTTP response with a status code of 400 and a message indicating that the registration could not be completed.

2.8. Landing Page Controller

2.8.1 Get All Testimonials

Description
This Node.js code exports a function called testimonials_get which is an asynchronous function. This function retrieves all the feedbacks marked as testimonials from the database and sends them as a response in JSON format.

Route
N/A

Parameters
req (object): An object containing the HTTP request information.
res (object): An object containing the HTTP response information.

Preconditions
The user must be authenticated and have a valid token to access the route.

Returns
This function returns a JSON object that contains the following properties:
testimonials: An array of feedbacks marked as testimonials.

Throws
If an error occurs during the execution of this function, it will throw an error with a status code of 500 and a message indicating the error.

Code Explanation
The testimonials_get function first retrieves all feedbacks marked as testimonials from the database using the Feedback model.
If there are testimonials, the function sends an HTTP response with a status code of 200 and a JSON object containing the testimonials array.
If there are no testimonials, the function sends an HTTP response with a status code of 404 and a message indicating that no testimonials were found.
If an error occurs during the execution of this function, the function sends an HTTP response with a status code of 500 and a message indicating the error.