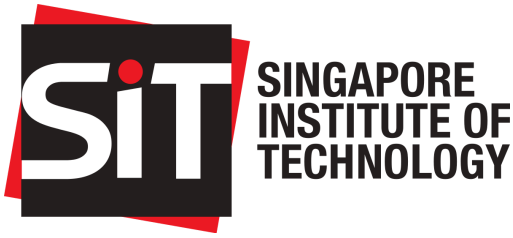


Deliverables 2 - Secure Software Requirement Analysis and Design

A report submitted to the

Singapore Institute of Technology in partial fulfillment of the requirements for the BACHELOR OF INFORMATION AND COMMUNICATIONS TECHNOLOGY (SOFTWARE ENGINEERING / INFORMATION SECURITY)

Team Name: Team 18

Submitted on: 08/07/24

GitHub handle: ICT2101-P11-6

Name	Student Number
Chiu Zheng Hao Jonathan	**2203187**
Goh Yue Jun	**2201471**
Muhammad Fiqri Adam	**2202878**
Nicholas Sng Ray Shiang	**2203197**
Denzel Low	**2202347**
Kee Han Xiang	**2201423**
Jeremy Lim Min En	**2203516**

Table of Contents

1 Design Review

1.1 Non Functional Requirements changed

- Updated NFR10: The system shall send a OTP to verify the user's phone number upon sign in.
- Added NFR 11: The system shall allow users the option to toggle if they want OTP on or off (One-time password was changed to be an option for the user to toggle)

1.2 Authorization Design changes:

- Role based access control (RBAC) changes:

The team has streamlined the RBAC to remove redundant roles and reduce the attack surface. The number of user roles have been reduced from 7 down to 5 after the removal of the Database Admin and Software Engineer roles. The following data have also been removed from the RBAC due to their redundancy: Log Data, Codebase Data, Database Schema.

1.3 Database Schema changes:

- Events table altered to accommodate the removal of different categories of tickets. Price_vip and price_cat1 to price_cat5, have been replaced with ticket_price.
- Creation of Otps table to store OTP and expiry time.
- User table altered to accommodate reset tokens, reset_token_expiry, failed_logins, lock_until, failed_pin_attempts, login_attempts, session_token, session_expiry to accommodate for brute-force attempts and session management.

2 Secure Software Implementation

2.1 CI/CD

The CI/CD process starts with an event (Push or Pull request) which triggers a webhook to start Jenkins to pull the codes from the repository. There are eleven major stages in the Jenkins pipeline: Checking the environment, Checkout, Clean Dependencies, Install Dependencies, OWASP Dependency-Check Vulnerabilities, Run Unit Tests, Archive Test Results, Archive Artifacts, List and Archive Dependencies, Build Frontend, Deploy to Web Server. Using GitHub and Jenkins, the team managed testing, building, and deployment processes, which improved the speed, efficiency, and reliability of releases.

Figure 1. Github Webhook Deliveries *(View in Appendix A1)*

In the CI/CD pipeline, external tools were used. OWASP Dependency Check was used to check the dependency of the packages used inside the web application. A node agent was also deployed to separate the Jenkins master agent from the actual building to lower the risk of potential attacks against Jenkins.

To address discrepancy during the QA check, the node agent had started running out of storage thus multiple branches in the Jenkins pipeline had been deleted. During the development phase as well, the github webhook was working, however due to a configuration change in the pipeline, the past history data of the pipeline has been wiped. However, there is evidence to prove that the pipeline had succeeded in checking out from the github repository shown above in the delivery history.

2.1.1 Jenkinsfile

This section provides details on the Jenkinsfile that does testing, building, and deploying for the web application. A webhook triggers the Jenkins job on changes to the main branch in the GitHub repository. The Docker daemon facilitates Continuous Integration by launching Docker containers upon detecting changes.

Figure 2. Jenkins Pipeline

The stages of the jenkinsfile are defined as follows:

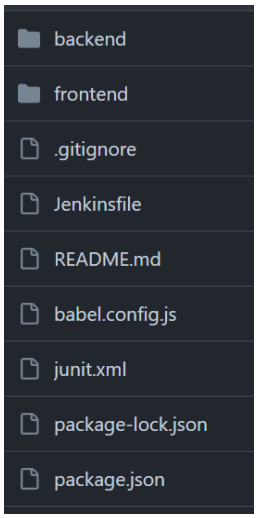
1) The github webhook triggers Jenkins to checkout the repository and to start the build process.
2) Jenkins sets the environment to be the branch name. This sets it so that the correct branch will be used when checking out from the repository. The checkout is done towards the node agent so that everything will be built in it.
3) All dependencies will be cleaned next, to ensure that there are no overlapping dependencies or outdated dependencies remaining.
4) All dependencies in the root folder, frontend and backend are installed in parallel to save time during the installation process.
5) OWASP Dependency Check is done here to ensure that we are notified of all vulnerable modules. The results are stored in a report for future reference and remediation.
6) Unit tests are then run to ensure individual functionality is working as intended.
7) A dependency list is then generated and stored. This helps to keep track of what dependencies have been installed at this version of the build.
8) The frontend folder is then built to optimize it for production. This allows the javascript to be more optimized.
9) Afterwards, it is deployed into the web server through rsync. This allows the files to be synchronized and the web server is then restarted as well.
10) For post actions, Github will then be notified that the build is successful.

2.2 GitHub

This section of the report will dive into the GitHub repository used in the project to store and collaborate on the application

2.2.1 Directories and files

The section will discuss the directories and files within the repository with a brief description to facilitate and assist in source code review.



1. backend

Project backend server functionalities

2. frontend

Project's presentation layer for users to interact with the services

3. Jenkinsfile

Text file that contains the definition of the team's Jenkins Pipeline

4. Package.json and package-lock.json

json file listing all the required library

Figure 3. Root Directory

2.2.1.1 Backend Folder

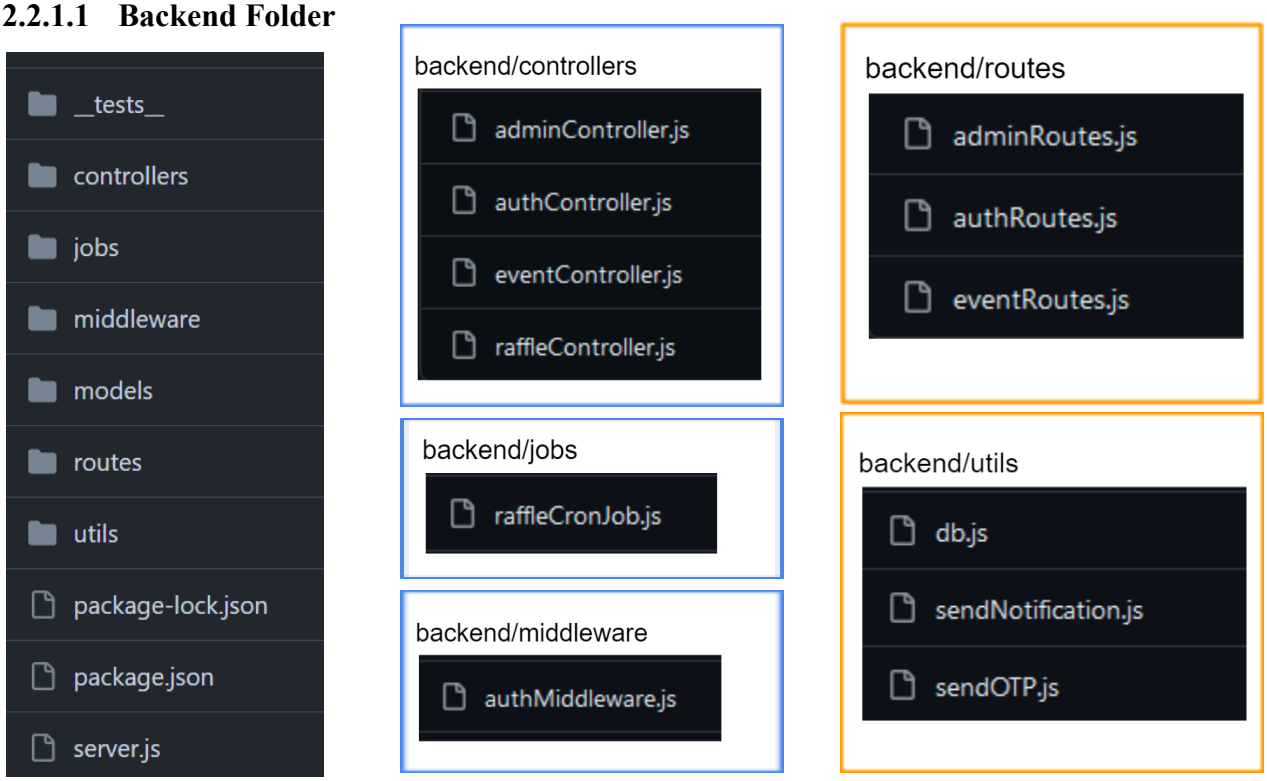


Figure 4. Backend files and subdirectories

Directory/File	Description
backend	
controllers	Directory containing the backend logic for each feature in the project
jobs	Directory containing code for cron jobs.
middleware	Directory containing code for to handle middleware functions like authorization
routes	Directory containing the api application created such as application's URL mapping to allow frontend to call different functions
utils	Directory containing helper functions, constants, and reusable code snippets that can be used across various parts of an application.
	This code defines middleware functions for authenticating a JSON Web Token (JWT) and authorizing access to the admin dashboard based on user roles, specifically allowing only users with roles 'admin', 'event', or 'cus_support'.
server.js	Javascript file that initializes and configures the server, setting up routes, middleware, and handling incoming requests and responses for a web application.
controllers	
adminController.js	Javascript file that defines function that will be used by admin features
authController.js	Javascript file that defines function that will be used by authentications features
eventController.js	Javascript file that defines function that will be used by event features
raffleController.js	Javascript file that defines function that will be used by raffle features
ticketController.js	Javascript file that defines function that will be used by ticket features
jobs	
raffleCronJob.js	Javascript file that defines function to schedule raffle completion and picking winners
middleware	
authMiddleware.js	Javascript file that defines function that will be used to authorize tokens
routes	
adminRoutes.js	Javascript file that maps admins functions to its respective URL
authRoutes.js	Javascript file that maps authentication functions to its respective URL

eventRoutes.js	Javascript file that maps events functions to its respective URL
ticketRoutes.js	Javascript file that maps ticket functions to its respective URL
utils	
db.js	Javascript file that sets up connection between the backend and the database
sendNotification.js	Javascript file that defines function to send notification to a user on winning the raffle
sendOTP.js	Javascript file that defines function to send one time password to a user on login

Table 1. Backend file descriptions

2.2.1.2 Frontend Folder

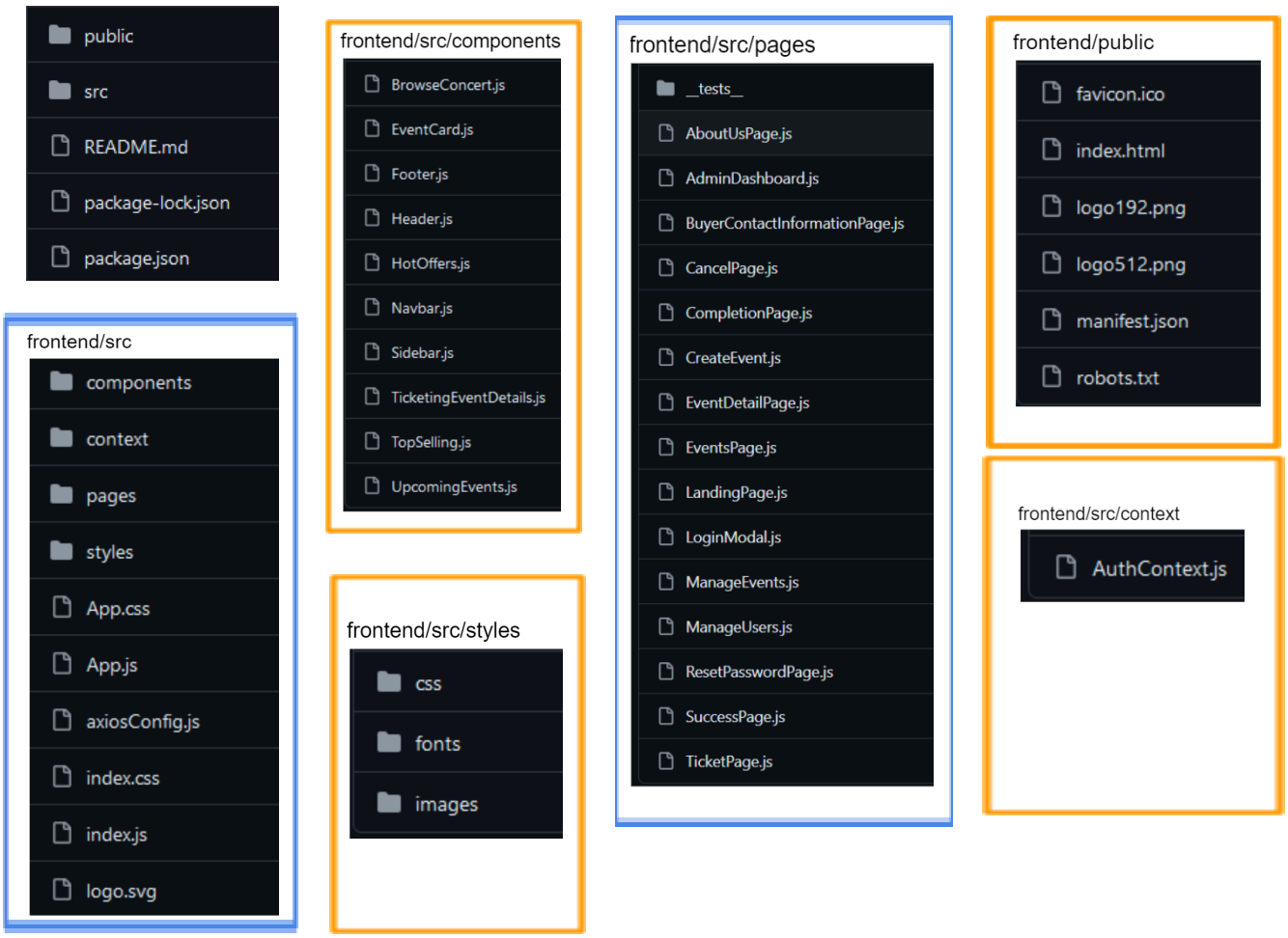


Figure 5. Frontend files and subdirectories

Directory/File	Description
frontend	
public	Directory containing the public assets that is globally accessible
src	Directory containing files for the frontend pages.
src	
components	Directory containing files for reusable components.
context	Directory containing files for context.
pages	Directory containing files for pages.
styles	Directory containing files for stylesheets and assets.
App.js	This code sets up a React application with routing for various pages, including landing, event details, ticket, completion, admin dashboard, about us, events, login modal, and password reset, while integrating authentication context and Bootstrap for styling.
axiosConfig.js	This code creates an Axios instance for making API requests with a base URL, includes cookies with requests, and sets up interceptors to handle request configurations and global error responses, particularly handling 403 errors by clearing cookies, updating login status, and redirecting to the login or landing page.
index.css	Global Css file that contains fonts used across the application
index.js	Code to initialize the React application
components	
BrowseConcert.js	Javascript file containing a reusable component to display browse concert feature
EventCard.js	Javascript file containing a reusable component to display event cards
Footer.js	Javascript file containing a reusable component to display footer
Header.js	Javascript file containing a reusable component to display header
HotOffers.js	Javascript file containing a reusable component to display hot offers feature
Navbar.js	Javascript file containing a reusable component to display navbar
Sidebar.js	Javascript file containing a reusable component to display sidebar
TicketingEventDetails.js	Javascript file containing a reusable component to display event details

TopSelling.js	Javascript file containing a reusable component to display top selling feature
UpcomingEvents.js	Javascript file containing a reusable component to display upcoming events feature
context	
AuthContext.js	Javascript file that provides authentication context, managing login, logout, and user state, and checking user authentication status using an API client.
pages	
AboutUsPage.js	Javascript file containing a code to display the about us page
AdminDashboard.js	Javascript file containing a code to display the admin dashboard page
CompletionPage.js	Javascript file containing a code to display the completion of joining raffle page
CreateEvent.js	Javascript file containing a code to display the create event page for admins
EventDetailPage.js	Javascript file containing a code to display the event details page
EventsPage.js	Javascript file containing a code to display the events page
LandingPage.js	Javascript file containing a code to display the landing page
LoginModal.js	Javascript file containing a code to display the login and sign up popup
ManageEvents.js	Javascript file containing a code to display the manage events page for admins
ManageUsers.js	Javascript file containing a code to display the manage users page for admins
PaymentPage.js	Javascript file containing a code to display the payment on winning raffle page
ResetPasswordPage.js	Javascript file containing a code to display the reset password page
MyTickets.js	Javascript file containing a code to display the possessed tickets and trader page
TicketCard.js	Javascript file containing a code to display the tickets in card form
TicketPage.js	Javascript file containing a code to display the ticketing page

Table 2. Frontend file descriptions

2.2.2 Class diagram mapping of github repository

The diagram below shows an overview of the directory mapped into a class diagram of the respective files and folders in the GitHub Repository. View in Appendix A1 or here

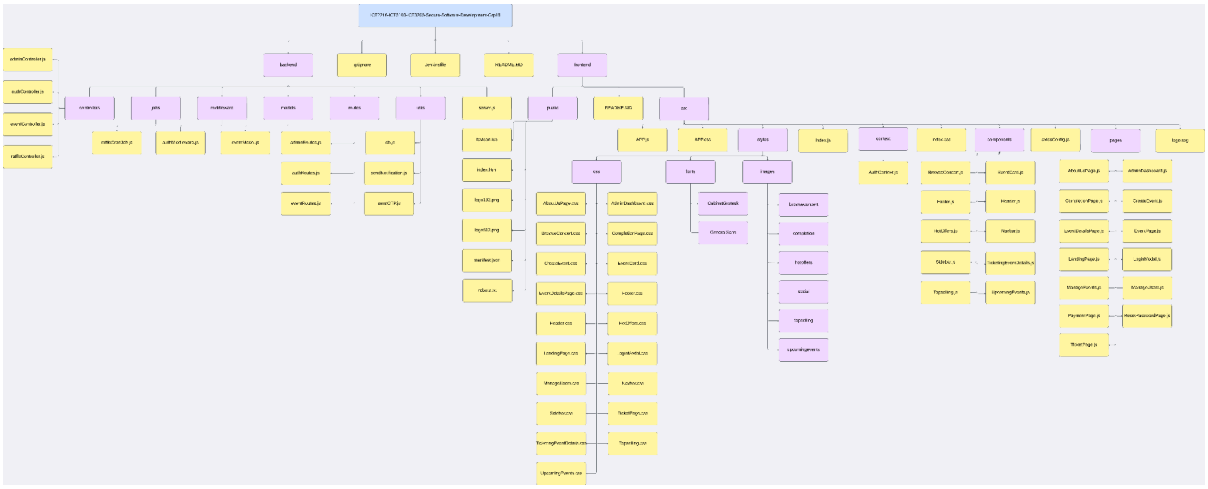


Figure 6. Class diagram mapping of github repository (View in Appendix A2)

2.2.3 Github Contribution

The table below shows the contributions of each member. As shown in figure 7 below, every member contributed to the team GitHub repository during the implementation phase.

No.	Student Name	Contributions
1	Chiu Zheng Hao Jonathan	CI/CD, Docker and Deployment on EC2, Report, HTTPS, OWASP Dependency Check, NGINX
2	Goh Yue Jun	Payment, Report, Unit Testing
3	Muhammad Fiqri Adam	Admin, Login/Register, Bugs, Report, Nodemailer, Raffle
4	Nicholas Sng Ray Shiang	Session, Events, Bugs, Report, Admin, Login/Register
5	Denzel Low	Payment, Report, Unit Testing
6	Kee Han Xiang	RBAC, Ticket Transfer, Admin, Report
7	Jeremy Lim Min En	Admin, CSS, Events, MFA , Report

Table 3. Contributions from every member

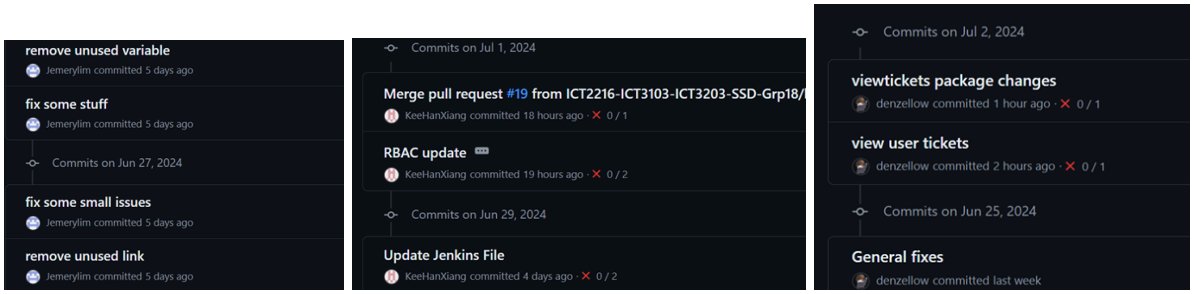


Figure 7. Commits to team GitHub repository from every member

2.2.4 Github DigitalOcean VM

Branch protection rules were set up such that code reviews were required along with passing code scans before changes are merged into the main branch. A GitHub webhook triggers a deployment script on our VM whenever new changes are pushed to the main branch. This script pulls the latest changes from GitHub and restarts the application, with deployment logs confirming the updates. However, in order for the rulesets to be enforced on our repository, payment to upgrade to GitHub Team is required, as shown in Figure 8 below.

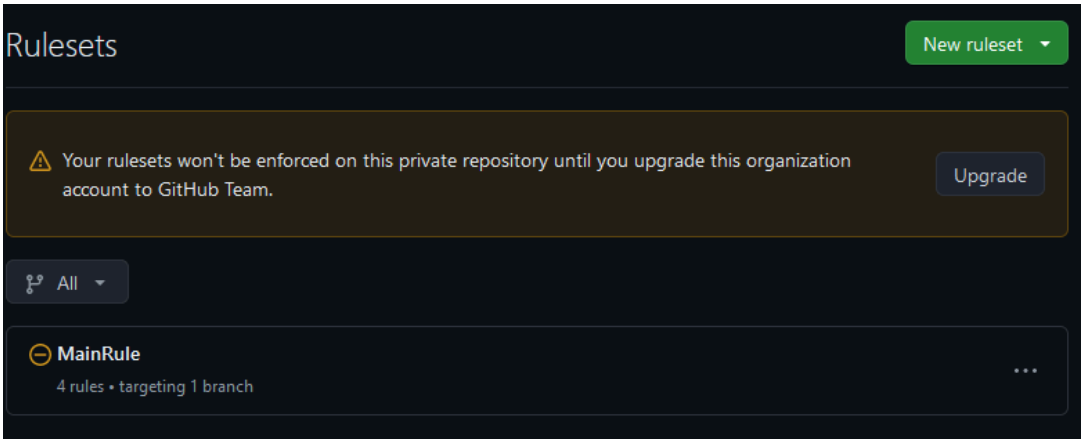


Figure 8. Github Rulesets requiring upgrade

Figure 9. Main branch JSON configuration

2.3 Secure Implementation of Login

This section describes the implementation of the secure login and the steps that ensures the login is secure using sequence diagrams

2.3.1 Sequence Diagram for Login

The sequence diagram is split into 2 parts to make it easier to digest.

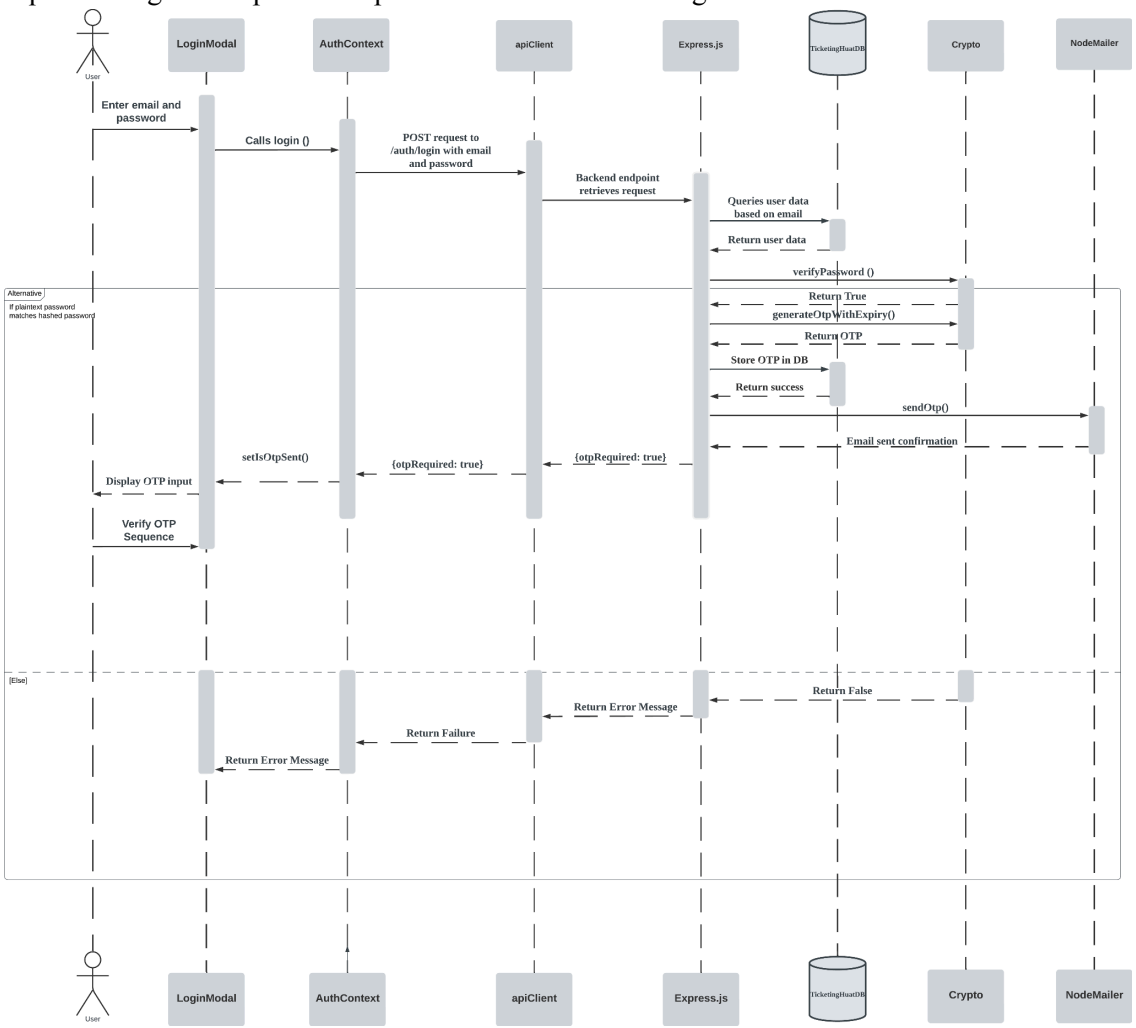


Figure 10. Sequence Diagram for login to get OTP (View in Appendix A3)

Figure 10 above illustrates the sequence of events involved in the multi-factor authentication (MFA) process for TicketingHuat. The process begins when a user enters their email and password in **"LoginModal"**, which calls **"login"** function in **"AuthContext"**. The 'AuthContext' sends a POST request to the **"/auth/login"** endpoint via **"apiClient".** The "Express.js" server handles the request and querying **"TicketingHuatDB"** for user data and verifying the password using the **"verifyPassword"** function. If the password is correct, a 6-digit OTP is generated using 'crypto' library and stored in database to be used later on, before it is sent to the user's email via **"NodeMailer"**. The server responds with **"otpRequired: true"**. The **"AuthContext"** then sets the **"isotpSent"** State to true and will prompt the "LoginModal" to display the OTP input field. The user submits the OTP, which is sent to the **"/auth/verify-otp"** endpoint. The server validates the OTP and, if the OTP is valid the server will establish a session and generate a JWT.

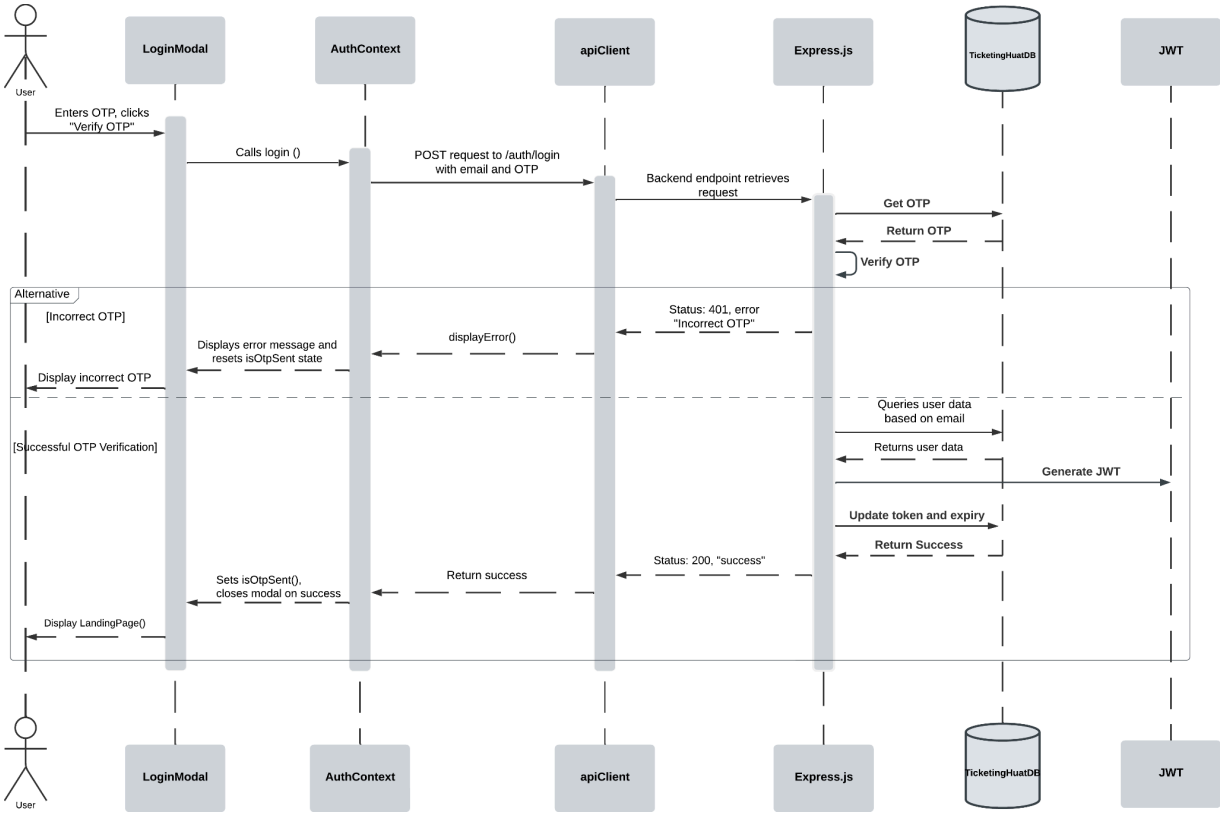


Figure 11. Sequence Diagram for Verify OTP (View in Appendix A4)

Figure 11 illustrates the OTP verification process in the TicketingHuat authentication flow. The user enters the OTP received via email and clicks **“Verify OTP”** in the **“LoginModal”**. This action triggers the login function in the AuthContext, which sends a POST request to the **/auth/login** endpoint through apiClient, including the email and OTP. The request will then be processed by the **“Express.js”** backend, which will then retrieve OTP from database and validate it. If the OTP is correct, the backend will retrieve the corresponding user data from the **“TicketingHuatDB”**, and generate a JWT token and return the user data to **“apiClient”**. **”apiClient”** will then communicate this success back to the **”AuthContext”**, which will then update the application state accordingly.
If the OTP is incorrect, the backend responds with a status **401** and an error message. This error is propagated back through the apiClient to the AuthContext, which then updates the LoginModal to display an "**Incorrect OTP"** message to the user.

2.3.2 Server Authentication

The server authentication is achieved through the implementation of a digital certificate which is issued by name.com. Name.com then generated an SSL certificate for the web server using the domain name issued by name.com as well. (ticketinghuat.ninja)

2.3.3 HTTPS Cipher Suites Enabled

To secure communications between clients and servers, we implemented TLS 1.2 and TLS 1.3 cipher suites with the latest cryptographic standards. To enhance security, older versions such as TLS 1.0 and TLS 1.1 have been disabled on the server. These older versions utilize outdated cryptographic algorithms which are vulnerable to known attacks like the Browser Exploit Against SSL/TLS (Beast). Furthermore, TLS 1.2 and TLS 1.3 provide perfect forward secrecy. This will help ensure that even if the long-term

private keys are compromised in the future, the confidentiality of the previous communications sessions will remain intact.

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_prefer_server_ciphers off;

ssl_ciphers 'ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-C
```

Figure 12. HTTP cipher suites enabled in config file

The SSL certs are applied on NGINX as both the web server and Jenkins are using NGINX as a reverse proxy. As both of them are running off the same domain name, the SSL certificates secure both services.

We did an SSL test using Qualys which gave us a rating of A, which shows that cipher suites that have been implemented and configured on the server are secure and up to date.

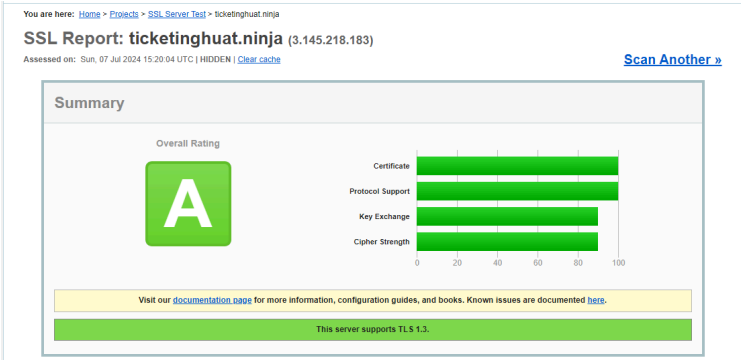


Figure 13. Qualys SSL test results

2.3.4 HTTPS Server Private Key Storage

SSL certificates which are generated by name.com also create two pivotal files “fullchain.pem” and “privkey.pem”. The server private key is stored in the “privkey.pem” file, which is essential for establishing secure web communications. This file is highly sensitive because it contains the private key used for decrypting data and signing session keys during SSL/TLS handshakes. To ensure its security, the “privkey.pem” file is assigned permissions set to 600 which restrict read and write access exclusively to the root user.

```
-rw-r--r-- 1 root root 7888 Jul  4 11:08 fullchain.pem
-rw------- 1 root root 1704 Jul  4 08:47 priv.key
```

Figure 14 . Permissions for fullchain.pem and privkey.pem

2.3.5 How key establishment is implemented, and how secret keys are derived

Key establishment is implemented using TLS protocols 1.2 and 1.3 which have been mentioned in 2.3.3, which mainly utilize a combination of key agreement and key transport mechanism to ensure the key exchange is secured. During SSL/TLS handshake, the client and server will agree on a cipher suite and generate a shared secret using Diffie-Hellman key exchange. This process provides a perfect secrecy to ensure the session keys cannot be derived from the compromised server key. Once the shared secret is established, it will be fed into a Key Derivation Function (KDF) along with nonces and other session data to generate the session keys.

2.3.6 Authentication Implementation

For TicketingHuat, we have implemented MFA to authenticate users, staff and admins using two factors of proof, which are required. One of the factor proofs we are using is “Something you know” which refers to the password that the user used to log in. The second factor of the proof is “Something that you have” which refers to the OTP sent to the user’s email that they have used when registering with TicketingHuat. On the frontend, users will enter their email and password. Upon submission, the login function checks the credentials in the backend. If the OTP is correct, an OTP is generated and sent to the user’s email using the **“nodemailer”**. The user must then enter this OTP to complete the login process.

The backend will then validate the input data using “**express-validator**”, verify passwords, and generate a 6-digit OTP using “**crypto.randomInt**”. This OTP, along with its expiration time, is stored temporarily for verification.

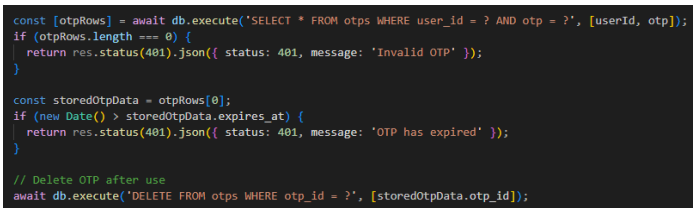


Figure 15. Verifying otp entered by user (View in Appendix A5)

Figure 16. Generate OTP with expiry

Figure 17. Send OTP to user email

2.3.7 User Password Hashing

In TicketingHuat, user credentials are stored securely in the database. Before storing, the passwords undergo hashing using the PBKDF2 algorithm with HMAC-SHA512. Which each password is combined with a unique 32-byte salt generated using **‘crypto.randomBytes’**. The password is then hashed with the salt using PBKDF2, performing 100,000 iterations and producing a 64-byte key. This method significantly increases the difficulty of successful attacks which also incur a robust security for the user password. The hashed password is stored in the format of **‘salt:hash’**.

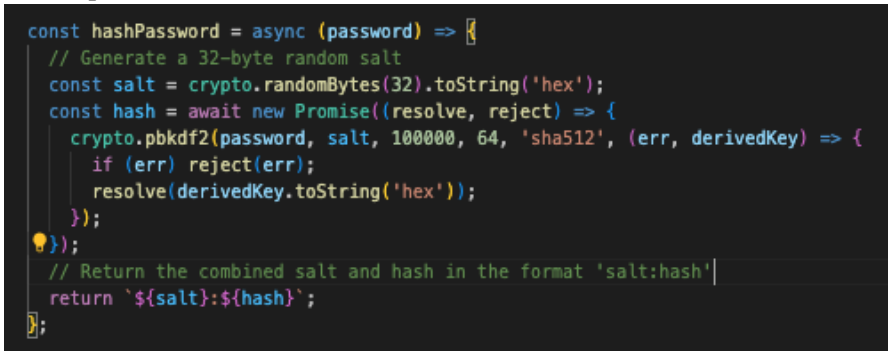


Figure 18. Password Hashing using PDKDF2_sha512

2.3.8 Prevention of Unauthorized access to Stored Passwords

name	phone_number	email	password	user_role
adam tan	84181159	adamtan@gmail.com	$2b$10$flRgKb5GfLZTz2P9upl7EuLbZNSXsqJCbZnVrqr.noNy7VQZqg7tK	admin
Fiqri Adam	84181159	fiqriadam2@gmail.com	$2b$10$.EbtQaLmjPZGIk8T7OQViuinfxdTPTIF1Uh39Zz9edEc3McgQfl32	admin
fiqri adam	84181159	fiqri_adam@rocketmail.com	$2b$10$dUIsVl5l89zrFohNd7BysehGlsP7IFWF8XXhltG55ml9/a3jcr1nG	user
fiqri adam	84181159	fiqri_adam@outlook.com	bb2d8e3dec4448e63bc18227f0b403bf14ff2839c9d3e19cbd80f8cddd3f258…	user
NULL	NULL	NULL	NULL	NULL

Figure 19. User data in database

The SQL database is hosted on an Amazon EC2 instance to prevent unauthorized access to stored passwords and sensitive data. The database connections are restricted to local connection only which means that the database can only be accessed by applications running on the same EC2 instance. This is enforced by configuring the database to bind to localhost. Furthermore, the database can only be accessed through an admin account with CRUD permission.

2.4 Secure implementation of Session Management

TicketingHuat uses the "express-session" middleware to handle session management. Sessions are created with various configurations to ensure security and proper handling of session data.

2.4.1 Session management configuration

When a user is successfully authenticated after logging in and verifying their OTP, the user will be assigned a new JWT token. The token will be used to authenticate user when they perform actions. User Id, role and email will be stored in session. A valid session will be checked when their session is about to expire. Users will be prompted to extend their session 3 minutes before session expiry.

```
// Regenerate the session
req.session.regenerate(async (err) => {
  if (err) {
    return res.status(500).json({ status: 500, message: 'Failed to regenerate session' });
  }

  // Store user information in the session
  req.session.userId = user.user_id;
  req.session.email = user.email;
  req.session.role = user.user_role

  // Store the session ID and expiry in the database
  const sessionToken = req.sessionID;
  const sessionExpiry = new Date(Date.now() + 2 * 60 * 1000); // 30 minutes

  await db.execute('UPDATE user SET session_token = ?, session_expiry = ? WHERE email = ?', [sessionToken,

  // Generate JWT
  const token = jwt.sign({ id: user.user_id, email: user.email, role: user.user_role, sessionToken }, jwtS

  res.cookie('token', token, {
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'Strict',
    maxAge: 2 * 60 * 1000, // 30 minutes
  });
```

Figure 20. Session assignment upon successful authentication *(View in Appendix A6)*

The session is configured as followed:

```
34  app.use(session({
35    secret: process.env.JWT_SECRET,
36    resave: false,
37    saveUninitialized: false,
38    cookie: {
39      httpOnly: true,
40      secure: process.env.NODE_ENV === 'production',
41      sameSite: 'Strict',
42      maxAge: 30 * 60 * 1000 // 30 minutes
43    }
44  }));
```

Figure 21. Session Code

Secret: A secret key is used to sign the session ID cookie, ensuring its integrity.
Resave: Set to false to prevent sessions from being saved back to the session store if they haven't been modified.
Save Uninitialized: Set to false to prevent storing sessions that are uninitialized.
Cookie: Configured with several security attributes:

httpOnly: Prevents JavaScript access to the session cookie.
secure: Ensures cookies are sent only over HTTPS when in production.
sameSite: Controls cross-site request behavior, set to Strict to prevent CSRF.
maxAge: Sets the expiration time for the session cookie to 30 minutes.

2.4.2 Transmission and modification prevention of Session Tokens and Cookies

From Server to Client:
When a session is created, the server sends a Set-Cookie header in the HTTP response, instructing the browser to store the session ID in a cookie. The session ID is signed with a secret key, preventing tampering. Any modification of the cookie invalidates the session. To further improve security, the session cookie is set with the httpOnly flag, making it inaccessible to JavaScript running in the browser. Additionally, The secure flag ensures that cookies are only sent over HTTPS, preventing interception during transmission.

From Client to Server:
For subsequent requests, the browser includes the session cookie in the Cookie header, allowing the server to identify the session. The signed session ID ensures that tampering of session ID will result in invalidation of the session.

2.4.3 Preventing Potential Session Attacks

Session Hijacking:

- SameSite Cookies: Setting the sameSite attribute to “Strict” helps prevent cookies from being sent along with cross-site requests, mitigating CSRF attacks.
- HttpOnly and Secure Flags: By setting httpOnly and secure flags, cookies are protected from client-side scripts and are only transmitted over HTTPS, reducing the risk of XSS and interception attacks.

Other Security Measures:

- Helmet Middleware: Utilized to set various HTTP headers for securing the application against common web vulnerabilities.
- Rate Limiting: Implemented using the express-rate-limit middleware to limit the number of requests an IP can make within a given timeframe, protecting against brute-force attacks.

Session Replay:

- Session Extension: Session has been configured with an expiration time of 30 minutes. In the event where users want to extend their session, a valid session will be checked before regenerating a new token and session. The new session_token and session_expiry will then be added to the database.

2.4.4 Session Invalidation

When a user chooses to logout, the logout function will clear JWT token and connect.sid from cookies, and set session_token and session_expiry to NULL in user table.

```
const logout = async (req, res) => {
  try {
    console.log(req.session.userId)
    await db.execute('UPDATE user SET session_token = NULL, session_expiry = NULL WHERE user_id = ?', [req.session.userId]);
    req.session.destroy((err) => {
      if (err) {
        return res.status(500).json({ message: 'Failed to log out' });
      }
      res.clearCookie('connect.sid');
      res.clearCookie('token');
      res.status(200).json({ message: 'Logout successful' });
    });
```

Figure 22. Logout Backend Code

```
mysql> SELECT user_id, name, phone_number, email, session_token, session_expiry
    -> FROM user;
+---------+------------+--------------+-------------------------+----------------------------------+---------------------+
| user_id | name       | phone_number | email                   | session_token                    | session_expiry      |
+---------+------------+--------------+-------------------------+----------------------------------+---------------------+
|       2 | adam tan   | 84181159     | adamtan@gmail.com       | NULL                             | NULL                |
|       3 | Fiqri Adam | 84181159     | fiqriadam2@gmail.com    | NULL                             | NULL                |
|       4 | fiqri adam | 84181159     | fiqri_adam@rocketmail.com | NULL                           | NULL                |
|       5 | fiqri adam | 84181159     | fiqri_adam@outlook.com  | ldfFeCDvP_gUKFomE67DacrtSA40as6B | 2024-07-07 23:45:14 |
|       6 | Yue Jun Goh | 86660831    | gohyuejun95@gmail.com   | gHytMRgzuKTxJZkhWzSLelydHElrn8Wf | 2024-07-08 14:16:48 |
+---------+------------+--------------+-------------------------+----------------------------------+---------------------+
5 rows in set (0.00 sec)
```

Figure 23. Session token and expiry in database

2.5 Secure implementation of Access Control

Our team implemented RBAC into TicketingHuat, and users are split into 5 different roles as shown in table 4. Customers will only have access to basic functionalities, whereas staff and admin will be able to manage events and users as shown in Figure 24.

User Roles	Role Description
Admin	Authenticated user with permission to mainly manage customer data and event data.
Event Staff	Authenticated user with permission to manage event data.
Customer Support Staff	Authenticated staff user with permission to read customer data in order to support Customer's permission.
Customer	Authenticated user with permission to mainly create an account, join raffle, and transfer and purchase tickets.
Guest	Unauthenticated user with permission to only search and view events

Table 4. RBAC Roles

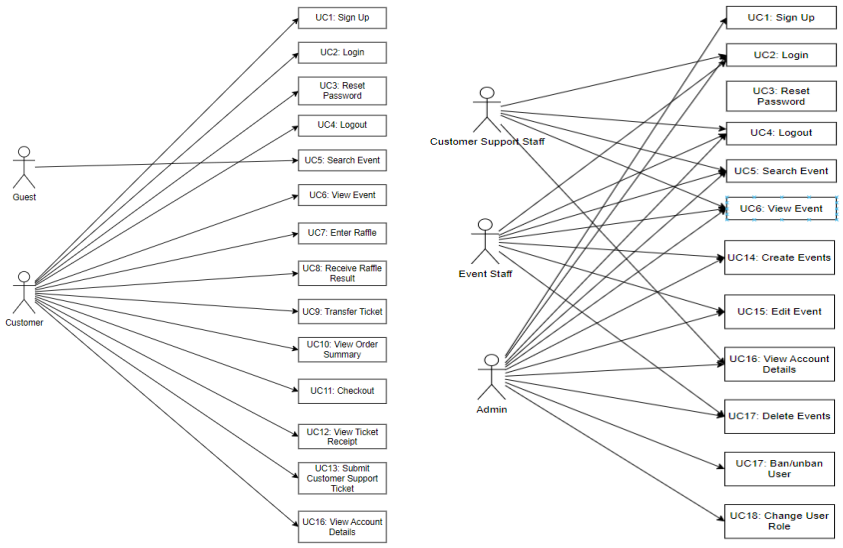


Figure 24. Staff, Admin, User Access Control *(View in Appendix A6)*

To implement authentication check for normal users, a valid JWT token is checked to ensure that a customer is logged in when performing any customer functionalities. As shown in Figure 25, checkAuth will be called to confirm if a user is authorized to perform any customer functionalities.

```
const checkAuth = (req, res) => {
  const token = req.cookies.token;

  if (!token) {
    return res.status(200).json({ message: 'No token, user is not logged in' });
  }

  jwt.verify(token, jwtSecret, (err, user) => {
    if (err) {
      return res.status(403).json({ message: 'Token is not valid' });
    }

    res.status(200).json({ message: 'Authorized' });
  });
};
```

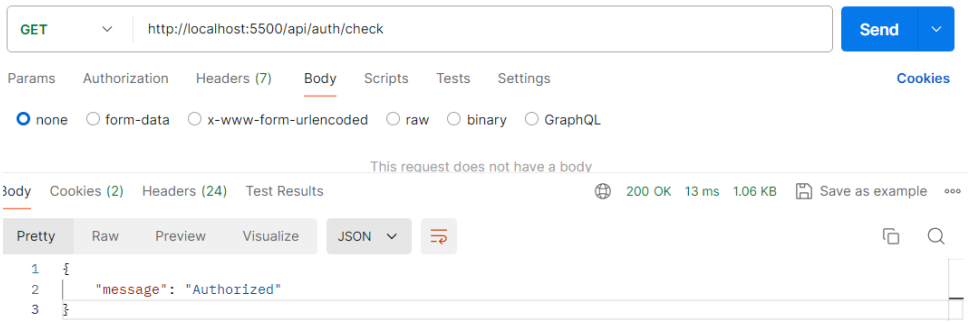


Figure 25. JWT token check

For administrator and staff functionalities, ‘isAdminDashboardUser’ function is called to check if the user has either admin or staff user role as shown in Figure 26. An 403 error code will be sent to the user if they attempt to access forbidden pages and will be brought back to the home page.

```
const isAdminDashboardUser = (req, res, next) => {
  if (req.user && ['admin', 'event', 'cus_support'].includes(req.user.role)) {
    next();
  } else {
    res.sendStatus(403);
  }
};
```

```
// Admin Dashboard Routes
router.get('/metrics', authenticateToken, isAdminDashboardUser, getMetrics);

// Admin Routes
router.post('/users', authenticateToken, isAdminDashboardUser, register);

// Event Staff Routes
router.post('/events', authenticateToken, isAdminDashboardUser, upload.single('image'), createEven
router.put('/events/:id', authenticateToken, isAdminDashboardUser, upload.single('image'), updateE
router.delete('/events/:id', authenticateToken, isAdminDashboardUser, deleteEvent);

// Customer Support Routes
router.get('/users', authenticateToken, isAdminDashboardUser, getUsers);
router.get('/users/search', authenticateToken, isAdminDashboardUser, searchUsers);

// Common Routes
router.get('/events', authenticateToken, getEvents);
router.get('/events/search', authenticateToken, searchEvents);
```

Figure 26. Admin and staff role check

2.6 Secure Coding

2.6.1 Define Security Requirements

Security requirements were outlined in deliverable 1 report.These requirements provide a foundation for the security functionality of the application. A more in-depth requirement can be found in section 1.3 of Deliverable 1 report.

1.3 Security Requirements

1.3.1 Secure Functional Requirements

FR	FR Description	SFR	SFR Description
FR1	The system shall allow authenticated users to log in to TicketingHuat.	SFR1	The system shall not allow brute force, dictionary attack, MiTM, session hijacking, credential stuffing and SQLi attacks.
FR2	The system shall allow unauthenticated users to sign up for an account.	SFR2	The system shall not allow account registration using duplicate email addresses.
			The system shall not allow the creation of an user account not following the strong password policy.
			The system shall not allow SQLi attacks.

Figure 27. Security Requirements

2.6.2 Leverage Security Frameworks and Libraries

To Speed up the process, we integrated several third-party libraries and frameworks during the development of the web application. To maintain the safety and verification of the tools, our team uses libraries and frameworks from reputable sources. And uses OWASP Dependency Check to scan for publicly disclosed vulnerabilities in all the third-party libraries and frameworks.

2.6.3 Secure Database Access

To prevent SQL Injection attacks, parameterized queries were used for our SQL queries, which separates SQL code from user input. An example is that of Figure 28.

```
const [rows] = await db.execute('SELECT * FROM user WHERE email = ?', [email]);
```

Figure 28. Parameterized query code snippet

```
+----------+-----------+-------------+-------------+-------------+-------------+-------------+-----------+
| user     | host      | Select_priv | Insert_priv | Update_priv | Delete_priv | Create_priv | Drop_priv |
+----------+-----------+-------------+-------------+-------------+-------------+-------------+-----------+
| Student3 | localhost | N           | N           | N           | N           | N           | N         |
+----------+-----------+-------------+-------------+-------------+-------------+-------------+-----------+
```

Figure 29. SQL User table

To follow the principle of least_privilege, we have a Student3 account with only “SELECT” privileges.

2.6.4 Encode and Escape Data

In the frontend, the team uses ‘DOMPurify’ library to sanitize harmful user’s input. ‘he’ library is used to encode special characters, ensuring that characters such as ‘&’ are properly represented in HTML. While in the backend, the team uses ‘sanitize-html’ and ‘he’ library to sanitize and encode inputs. As shown in Figure 30, an error message is returned when attempting a XSS attack on the backend.

```
const sanitizeInput = (input) => {
  const sanitized = DOMPurify.sanitize(input.trim());
  return he.encode(sanitized);
};
```

```
const sanitizeInput = (input) => {
  if (typeof input === 'string') {
    const sanitized = sanitizeHtml(input.trim(), {
      allowedTags: [],
      allowedAttributes: {}
    });
    return he.encode(sanitized);
  }
  return input;
};
```

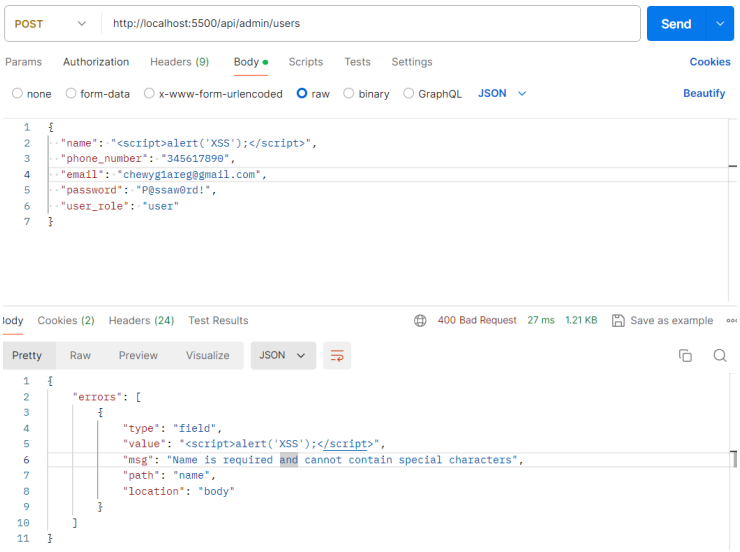


Figure 30. Input Sanitization for frontend and backend

2.6.5 Validate all Inputs

To prevent injection attacks, inputs are sanitized before they are validated both on client and server side to ensure that user inputs are properly formatted.

On the frontend, form validation attributes are used as shown in Figure 31. The ‘required’ attribute is added to the input fields to ensure that fields have to be filled in before submitting the form. ‘Type’ attribute is also used to specify the expected data types for each field. Express validator and regex is used to ensure that inputs are valid before submitting to the backend.

```
<label>
  Email Address:
  <input
    type="email"
    name="email"
    value={email}
    onChange={(e) => setEmail(e.target.value)}
    required
  />
  {errors.email && <p className="error">{errors.email}</p>}
</label>
<label>
  Password:
  <input
    className="auth-form-input"
    type="password"
    name="password"
    value={password}
    onChange={(e) => setPassword(e.target.value)}
    required
  />
  {errors.password && <p className="error">{errors.password}</p>}
</label>
```

```
const validateEmail = (email) => {
  return validator.isEmail(email);
};

const validatePhoneNumber = (phoneNumber) => {
  return validator.isMobilePhone(phoneNumber, 'en-SG');
};

const validatePassword = (password) => {
  const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,12}$/;
  return passwordRegex.test(password);
};

const validateName = (name) => {
  const nameRegex = /^[a-zA-Z\s]+$/;
  return nameRegex.test(name);
};
```

Figure 31. Frontend Input Validation

In the backend, input validation is also important for ensuring data integrity and security. The register function shown in Figure 32 shows how the backend validates input using the express-validator. It checks the request data from frontend to ensure that it meets the criteria. If any of these validations fail, an error message is returned to the client.

```
const register = [
  body('name')
    .isLength({ min: 1 })
    .matches(/^[a-zA-Z\s]+$/)
    .withMessage('Name is required and cannot contain special characters')
    .customSanitizer(sanitizeInput),
  body('phone_number')
    .isMobilePhone()
    .withMessage('Invalid phone number')
    .customSanitizer(sanitizeInput),
  body('email')
    .isEmail()
    .withMessage('Invalid email')
    .customSanitizer(sanitizeInput),
  body('password')
    .isLength({ min: 8, max: 12 })
    .matches(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])/)
    .withMessage('Password must be 8-12 characters long and include a mix of uppercase letters, lowercase letters, numbers
    .customSanitizer(sanitizeInput),

  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
```

Figure 32. Backend Input Validation

2.6.6 Implement Digital Identity

2.6.6.1 Password Requirements

Our team adhered guidelines by OWASP and NIST and the requirements are as follows;

- Password minimum of 8 characters, and max of 12 characters.
- Passwords have to have a mix of uppercase, lowercase, numbers and special characters.
- Passwords cannot have all numerical characters.

The passwords are validated using regex.

```
const validatePassword = (password) => {
  const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,12}$/;
  return passwordRegex.test(password);
};
```

Figure 33. Validate password using regex

2.6.6.2 Multi-Factor Authentication

All accounts are to use MFA by default. A 6 digit OTP code will be sent to the registered email address every time a user logs into the website. For more details on OTP, please refer to Section 2.3.6 of the report.

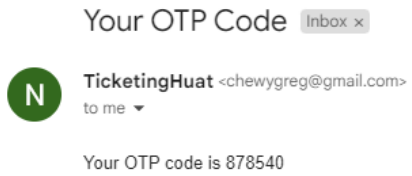


Figure 34. OTP received in email inbox

2.6.6.3 Session Management

For more details on Session Management, please refer to Section 2.4 of the report.

2.6.7 Enforce Access Controls

To enforce access control, RBAC is implemented for TicketingHuat. Authentication checks are enforced when a user is logging in and when performing a customer, staff or administrator functionality. Table 5 shows the different permissions assigned to the different roles allowed in TicketingHuat. For more information on user roles, refer to Section 2.5 of the report.

Data	User Roles				
	Admin	Event Staff	Customer Support Staff	Customer	Guest
Customer Data	C, R, U, D	R	R	C, R, U	-
Event Data	C, R, U, D	C, R, U, D	R	R	R
Ticket Data	-	-	R	R	-
Order Data	-	-	R	R	-

Table 5. User Role Permissions

2.6.8 Protect Data Everywhere

2.6.8.1 Data in Transit

To protect sensitive data during transmission, a TLS certificate was created. It ensures the security and integrity of web application data by safeguarding it against unauthorized access and tampering while in transit.

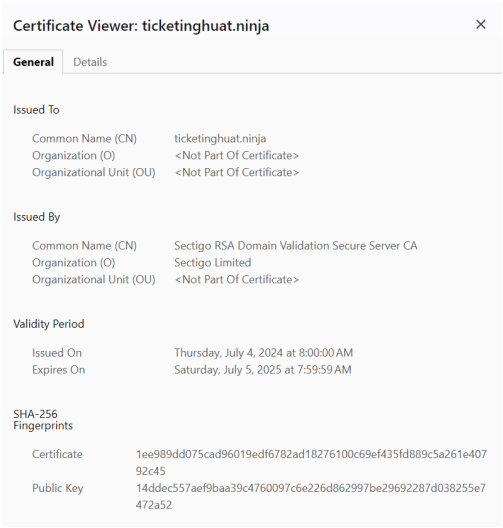


Figure 35. TLS certificate

2.6.8.2 Data at REST

For password storage, a unique 32-bit salt is combined with a user's password and hashed using PBKDF2 algorithm using SHA512 hashing function with 100,000 iterations. The salt and hashed password are concatenated before storing into the database.

```
const hashPassword = async (password) => {
  const salt = crypto.randomBytes(32).toString('hex');
  const hash = await new Promise((resolve, reject) => {
    crypto.pbkdf2(password, salt, 100000, 64, 'sha512', (err, derivedKey) => {
      if (err) reject(err);
      resolve(derivedKey.toString('hex'));
    });
  });
  return `${salt}:${hash}`;
};
```

Figure 36. hashPassword Function

2.6.9 Implement Security Logging and Monitoring

2.6.10 NGINX logs

NGINX docker provides access logs including IP addresses, user-agent and requests to various APIs that the user has made. These logs can be used for tracking user activity, monitoring traffic and identifying potential security threats. By analyzing this logs, insights can be gain by reading into the usage patterns, anomalies and optimize server performance.

```
116.87.28.113 - - [08/Jul/2024:21:08:39 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/logText/pr
ogressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/conso
le" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36"
116.87.28.113 - - [08/Jul/2024:21:08:40 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/logText/pr
ogressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/conso
le" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36"
116.87.28.113 - - [08/Jul/2024:21:08:40 +0000] "GET /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/statusIcon
HTTP/1.1" 200 349 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/console" "Mozilla
/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36"
116.87.28.113 - - [08/Jul/2024:21:08:41 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/logText/pr
ogressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/conso
le" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36"
116.87.28.113 - - [08/Jul/2024:21:08:43 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/logText/pr
ogressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/conso
le" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36"
116.87.28.113 - - [08/Jul/2024:21:08:44 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/logText/pr
ogressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/conso
le" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36"
116.87.28.113 - - [08/Jul/2024:21:08:45 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/logText/pr
ogressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/conso
le" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36"
116.87.28.113 - - [08/Jul/2024:21:08:45 +0000] "GET /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/statusIcon
HTTP/1.1" 200 354 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/console" "Mozilla
/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.0.0 Safari/537.36"
116.87.28.113 - - [08/Jul/2024:21:08:46 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/logText/pr
ogressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Test/24/conso
```

Figure 37. NGINX Logs

2.6.11 PM2 Logs

PM2 logs can be accessed inside the docker of the web server at “/home/jenkins/.pm2/logs/app-out.log” These logs can be use to see detailed information about the web application’s runtime behavior, including HTTP requests, response statuses, headers cookies, session data, and client information. These logs can be used to identify issues, and debug errors effectively.

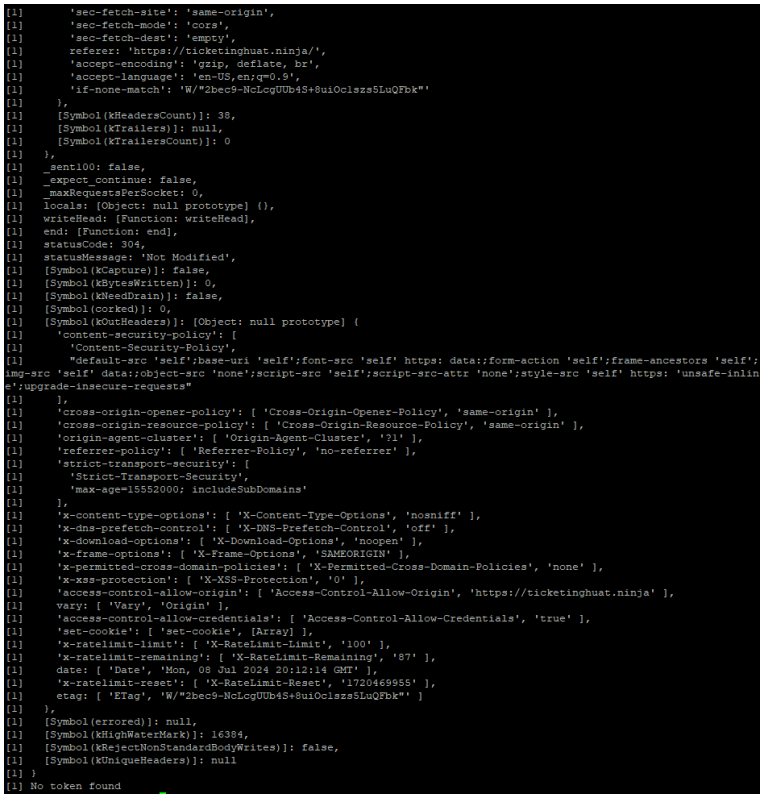


Figure 38. NGINX Logs

2.6.12 Docker-Compose Logs

Using Docker-Compose logs, a compiled list of all the dockers in the docker-compose can be seen at a glance.

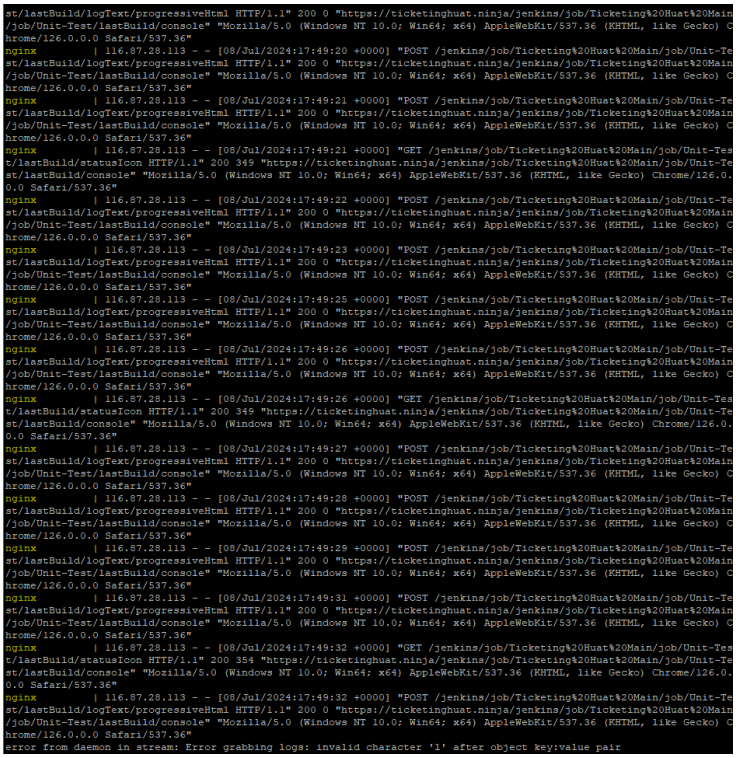


Figure 39. NGINX Logs(View in Appendix A8)

2.6.13 Handle all Errors and Exceptions

Error and exception handling are implemented on both frontend and backend. For the frontend, our team implemented try catch blocks to handle errors and exceptions without revealing technical details.

On our backend, our team also implemented try catch blocks and return status code indicating different errors. Figure 40 below shows the backend returning 400-403 status code for different errors and exceptions.

```
const login = async ({ email, password, otp }) => {
  try {
    let response;
    if (otp) {
      response = await apiClient.post('/auth/verify-otp', { email, otp }, {
        withCredentials: true,
        onSessionInvalidated: handleSessionInvalidation
      });
    } else {
      response = await apiClient.post('/auth/login', { email, password }, {
        withCredentials: true,
        onSessionInvalidated: handleSessionInvalidation
      });
      if (response.data.otpRequired) {
        return { otpRequired: true };
      }
    }
    if (response && response.status === 200) {
      setIsLoggedIn(true);
      const userResponse = await apiClient.get('/auth/getUser', { withCredentials: true });
      setUser(userResponse.data);
      return { status: 200 };
    } else {
      throw new Error('Login failed');
    }
  } catch (error) {
    throw new Error('Login failed');
  }
};
```

```
try {
  if (!email || !password) {
    return res.status(400).json({ message: 'Email and password are required' });
  }

  const [rows] = await db.execute('SELECT * FROM user WHERE email = ?', [email]);

  if (rows.length === 0) {
    return res.status(401).json({ message: 'Invalid email or password' });
  }

  const user = rows[0];

  if (user.lock_until && new Date(user.lock_until) > new Date()) {
    return res.status(403).json({ message: 'Account locked. Try again later.' });
  }

  const isMatch = await verifyPassword(password, user.password);
```

Figure 40. Snippet of code for Error handling on frontend and backend(View in Appendix A9)

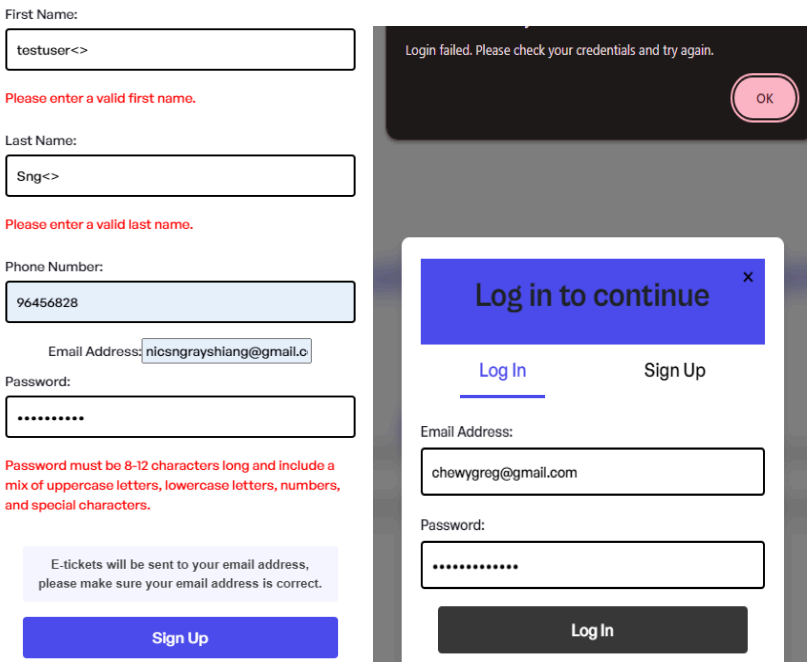


Figure 41. Snippet of Error messages

3 Test Automation

3.1.1 Dependencies Inventory

Root	Frontend	Backend
babel-jest:^27.4.7 concurrently:^8.2.2 jest:^27.4.7 jest-junit:^16.0.0	@emotion/react: ^11.11.4 @emotion/styled": ^11.11.5 @fortawesome/fontawesome-free: ^6.5.2 @mui/material: ^5.15.21 axios: ^1.7.2 bootstrap:^5.3.3 dompurify:^3.1.5 eslint-config-react-app:^7.0.1 he:^1.2.0 react: ^18.3.1 react-bootstrap: ^2.10.4 react-dom: ^18.3.1 react-router-dom: ^6.23.1 react-scripts:^5.0.1 validator:^13.12.0	body-parser:^1.20.2 cookie-parser^1.4.6 cors:^2.8.5 csurf:^1.11.0 dotenv^16.4.5 express:^4.19.2 express-rate-limit:^7.3.1 express-session: ^1.18.0 express-validator:^7.1.0 he: ^1.2.0 helmet:^7.1.0 iconv-lite:^0.6.3 jsonwebtoken:^9.0.2 multer:^1.4.5-lts.1 mysql2:^3.10.1 node-cron:^3.0.3 nodemailer:^6.9.14 sanitize-html:^2.13.0 stripe:^16.2.0 @babel/core:^7.24.7 @babel/preset-env:^7.24.7 babel-jest: ^29.7.0 jest:^29.7.0

		supertest:^7.0.0

Table 6. Dependencies Inventory

3.1.2 OWASP Dependency Checker

OWASP Dependency Checker identifies any vulnerable version, and flags the vulnerabilities. Most vulnerabilities listed in Figure 42 are newly found in dependencies latest versions. There are currently no fix and updates available as listed in NVD and online sources. Patches will be done as soon as the fixes and updates are out.

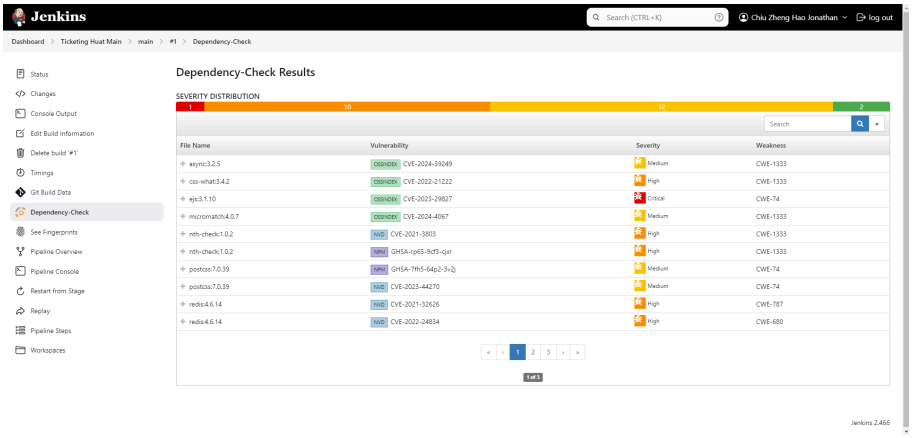


Figure 42. OWASP Dependency Checker

3.1.3 Automated Unit Testing

In order to check if the application is working properly before pushing it into production, the team utilized Jest, a testing framework, integrated into the continuous integration pipeline managed by Jenkins. Additionally, we employed React Testing Library for frontend component testing.

```
stage('Run Unit Tests') {
    steps {
        dir('backend') {
            withCredentials([
                usernamePassword(credentialsId: 'db_credentials', usernameVariable: 'DB_USER', passwordVariable: 'DB
                string(credentialsId: 'db_name', variable: 'DB_NAME'),
                usernamePassword(credentialsId: 'smtp_credentials', usernameVariable: 'EMAIL_USER', passwordVariable
                string(credentialsId: 'jwt_secret', variable: 'JWT_SECRET')
            ]) {
                sh 'npx jest --detectOpenHandles --forceExit __tests__'
            }
        }
    }
}
stage('Archive Test Results') {
    steps {
        junit 'junit.xml'
        archiveArtifacts artifacts: 'junit.xml', fingerprint: true
    }
}
```

Figure 43. Jenkins Unit Test Stage

Frontend

Test Case	Expected Outcome	Actual Outcome	Status
Login submission with valid data	No Login failed. Please check your credentials and try again	No Login failed. Please check your credentials and try again	Pass
Sign up submission with valid data	No Registration failed. Please try again.	No Registration failed. Please try again.	Pass
Passwords do not match	Passwords do not match	Passwords do not match	Pass
Successful password reset	Password reset successfully	Password reset successfully	Pass
Failed password reset	Displays error message	Displays error message	Pass
Sanitize input to prevent XSS attack during login	Login failed. Please check your credentials and try again.	Login failed. Please check your credentials and try again.	Pass
OTP login flow correctly	OTP:	OTP:	Pass

Table 7. Frontend Test Cases

Backend

Test Case	Expected Outcome	Actual Outcome	Status
Empty Name when registering	400	400	Pass
Invalid Phone Number on registering	400	400	Pass
Invalid Email Address on registering	400	400	Pass
Weak Password when registering	400	400	Pass
Successful registering with valid input	201	201	Pass
Invalid email when log in	400	400	Pass
Weak password on POST /login	400	400	Pass
Invalid email on POST /login	400	400	Pass
Invalid email or password on POST /login	400	400	Pass
Successfully login with valid credentials	200	200	Pass

Send password reset email on POST /login	200	200	Pass
Success reset password with valid input	200	200	Pass

Table 7. Backend Test Cases

Unit Testing for Utility Functions

Utility functions were tested to ensure their reliable operation in sending emails, the figure below shows a successful test case.

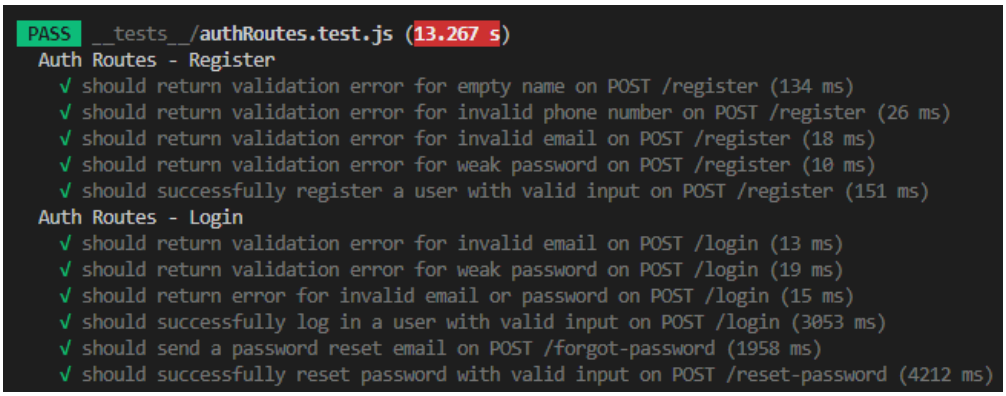


Figure 44. Unit Testing

```
Test Suites: 2 passed, 2 total
Tests:       16 passed, 16 total
Snapshots:   0 total
Time:        7.349 s, estimated 11 s
Ran all test suites matching /__tests__/i.
```

Figure 45. Jenkins Testing output

For utility functions, the objective of the tests were to verify the functionality of sending OTP emails and sending password reset emails. An example can be seen in Figure 43, where it simulates a user login to the website and receive an OTP. A HTTP request posts a valid user's credentials to the backend verify-otp API, and it expects the body to have certain properties.

```
  expect(response.body).toHaveProperty('message', 'OTP sent to your email');
  expect(response.body).toHaveProperty('otpRequired', true);

  // Fetch the OTP from the database
  const otp = await getOtpForUser('chewygreg@gmail.com');
  expect(otp).not.toBeNull(); // Ensure the OTP was fetched

  // Verify the OTP
  const otpResponse = await request(app)
    .post('/api/auth/verify-otp')
    .send({
      email: 'chewygreg@gmail.com',
      otp: otp,
    });

  expect(otpResponse.status).toBe(200);  // Expect a successful OTP verification
  expect(otpResponse.body).toHaveProperty('message', 'Login successful');
  expect(otpResponse.body).toHaveProperty('user');
  expect(otpResponse.body.user).toHaveProperty('id');
  expect(otpResponse.body.user).toHaveProperty('email', 'chewygreg@gmail.com');
  expect(otpResponse.body.user).toHaveProperty('role');
});
```

Figure 46. Test case

References

[1] Jenkins user documentation. (n.d.). https://www.jenkins.io/doc/

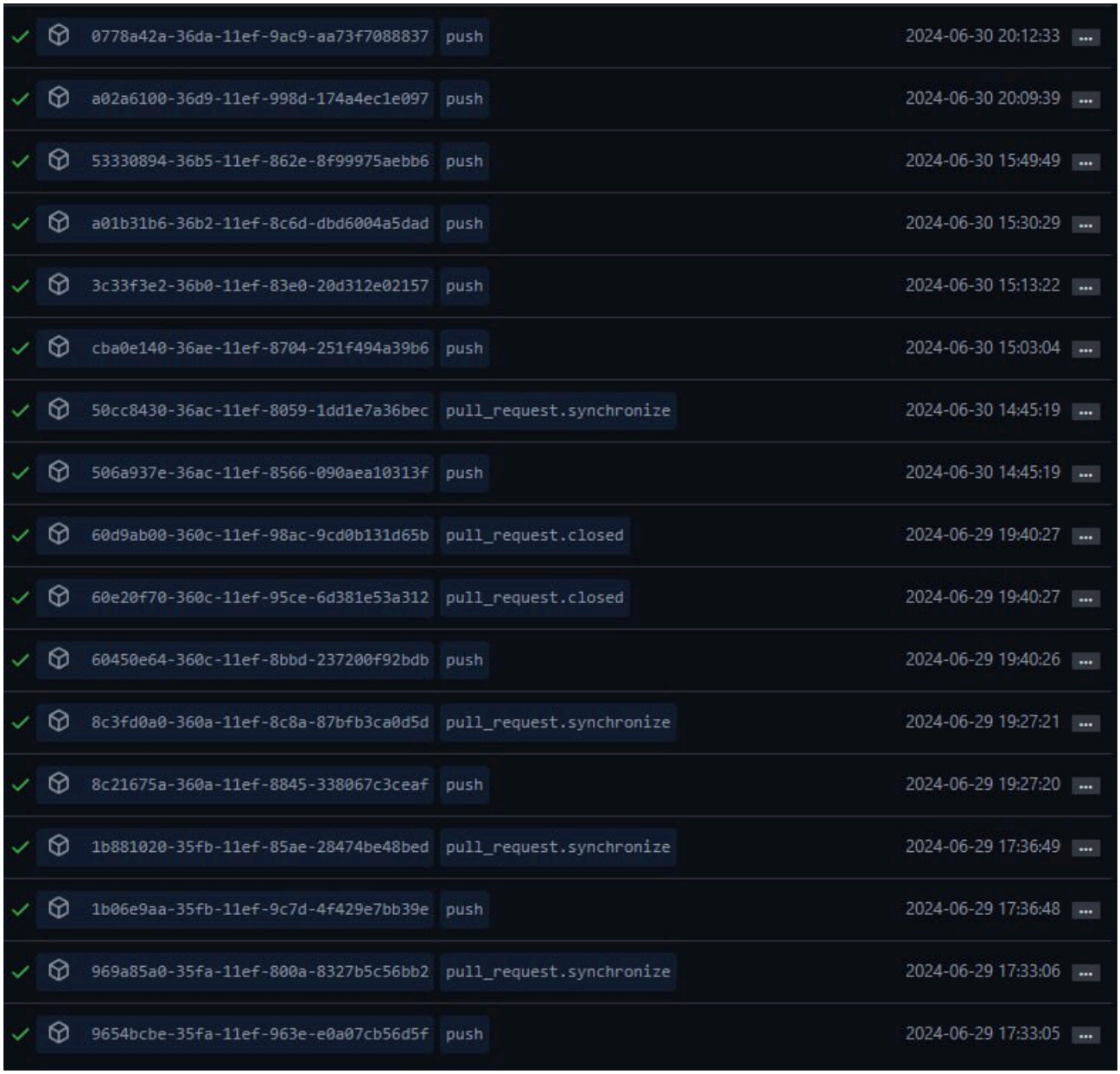
Appendix

Words	Meaning
MiTM Attack	Man in the middle attack
SQLi Attack	SQL injection attack
DoS Attack	Denial-of-service attack
RBAC	Role-based access control
WAF	Web Application Firewall
XSS	Cross-Site
CSRF	Cross-Site Request Forgery
CDN	Content Delivery Network
MFA	Multi-Factor Authentication
DNS	Domain Name Server
RBAC	Role-Based Access Control
WFA	Web Application Firewall
CRUD	Create, Read, Update, Delete
OTP	One Time Password
CSA	Cyber Security Agency of Singapore’s
PBKDF2	Password-Based Key Derivation Function 2
TLS	Transport Layer Security
PDPA	Personal Data Protection Act
PIN	Personal Identification Number
API	Application Programming Interface
OWASP	Open Worldwide Application Security Project
SQL	Structured Query Language
HTTPS	Hypertext Transfer Protocol Secure
JWT	JSON Web Token

CSP	Content Security Policy
RCE	Remote Code Execution

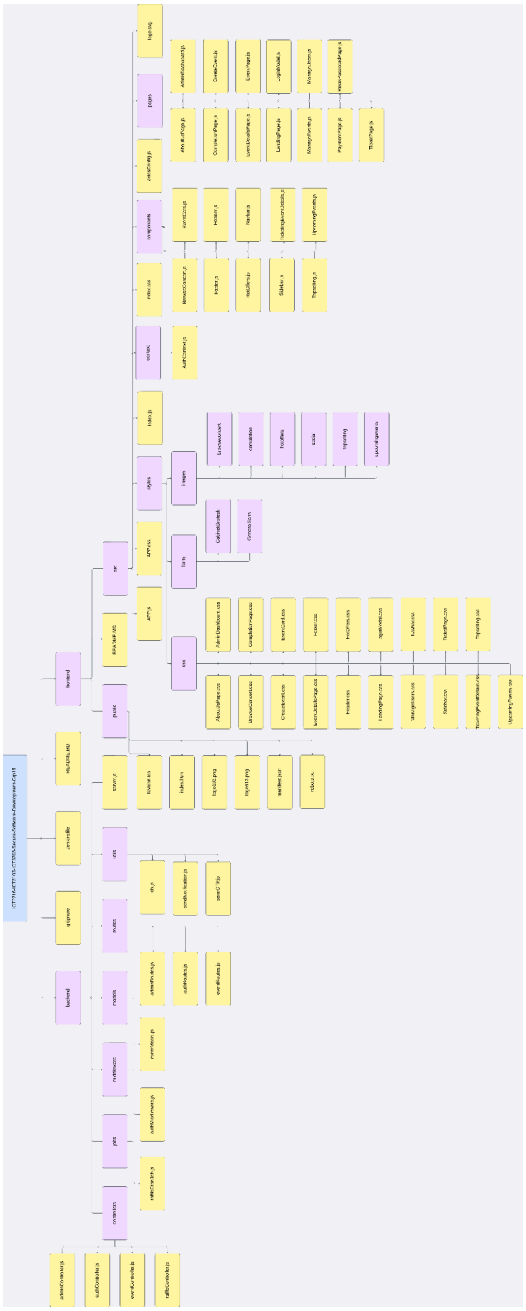
Table 8. Data Dictionary

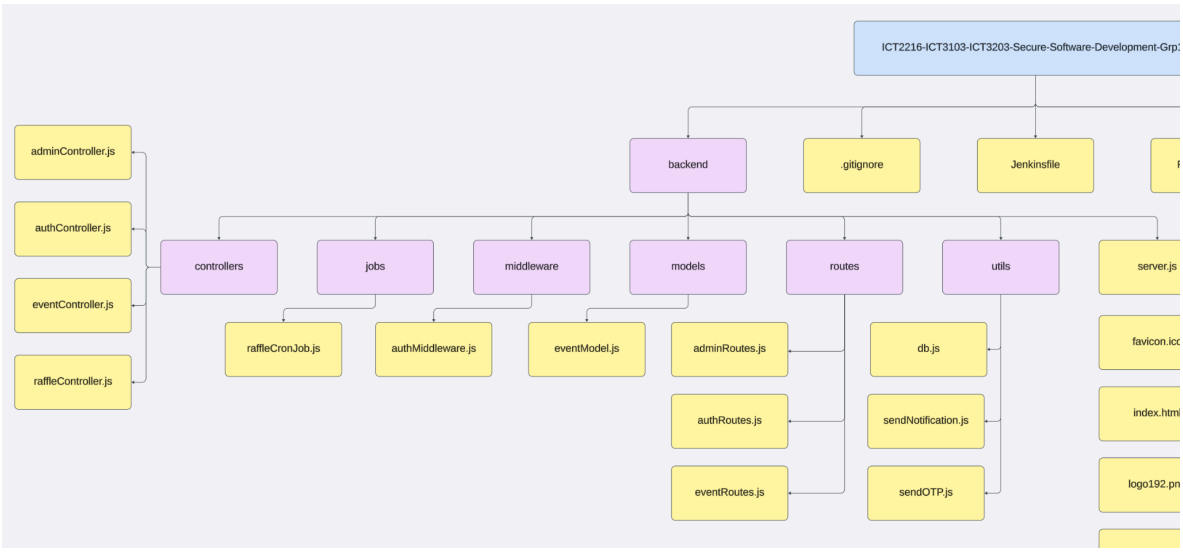
Github Webhook Deliveries

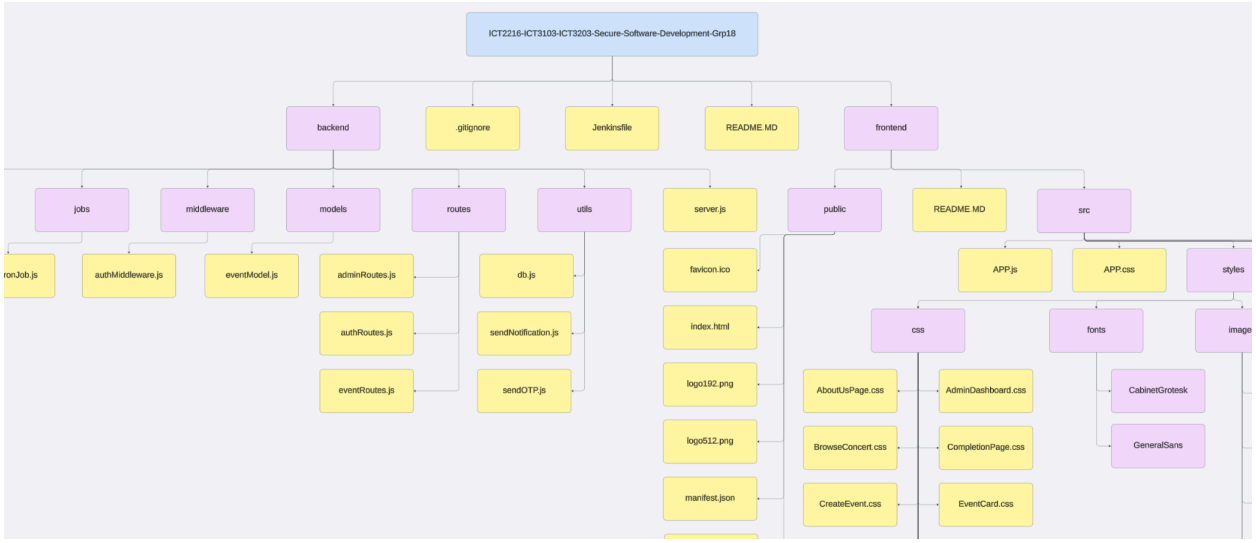


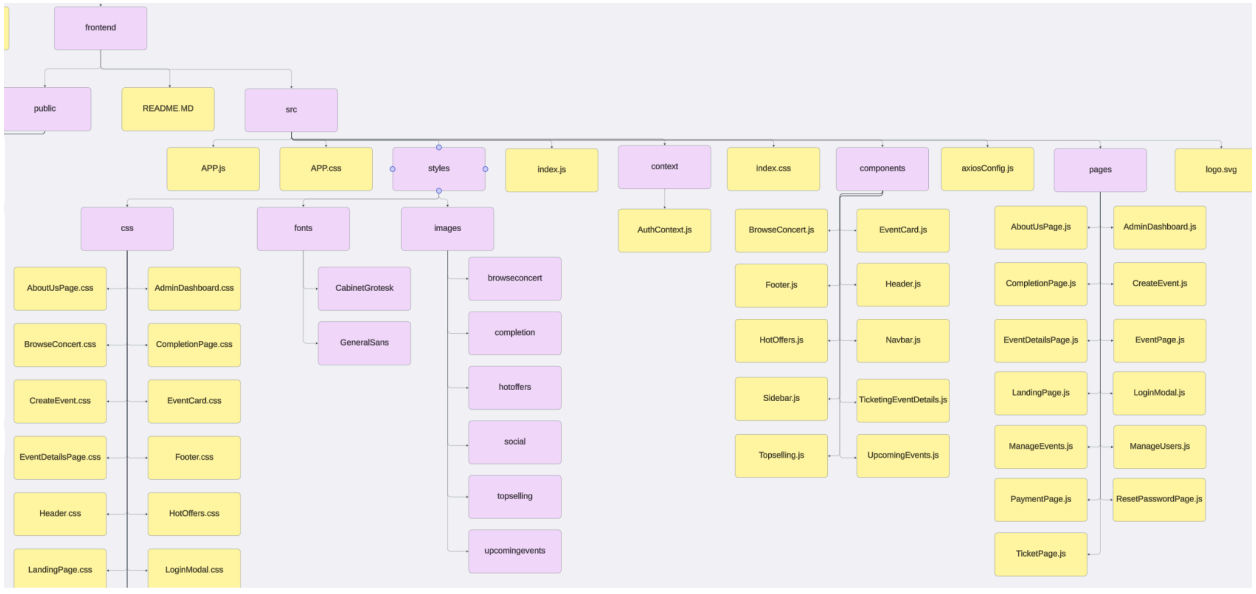
Class diagram mapping of github repository

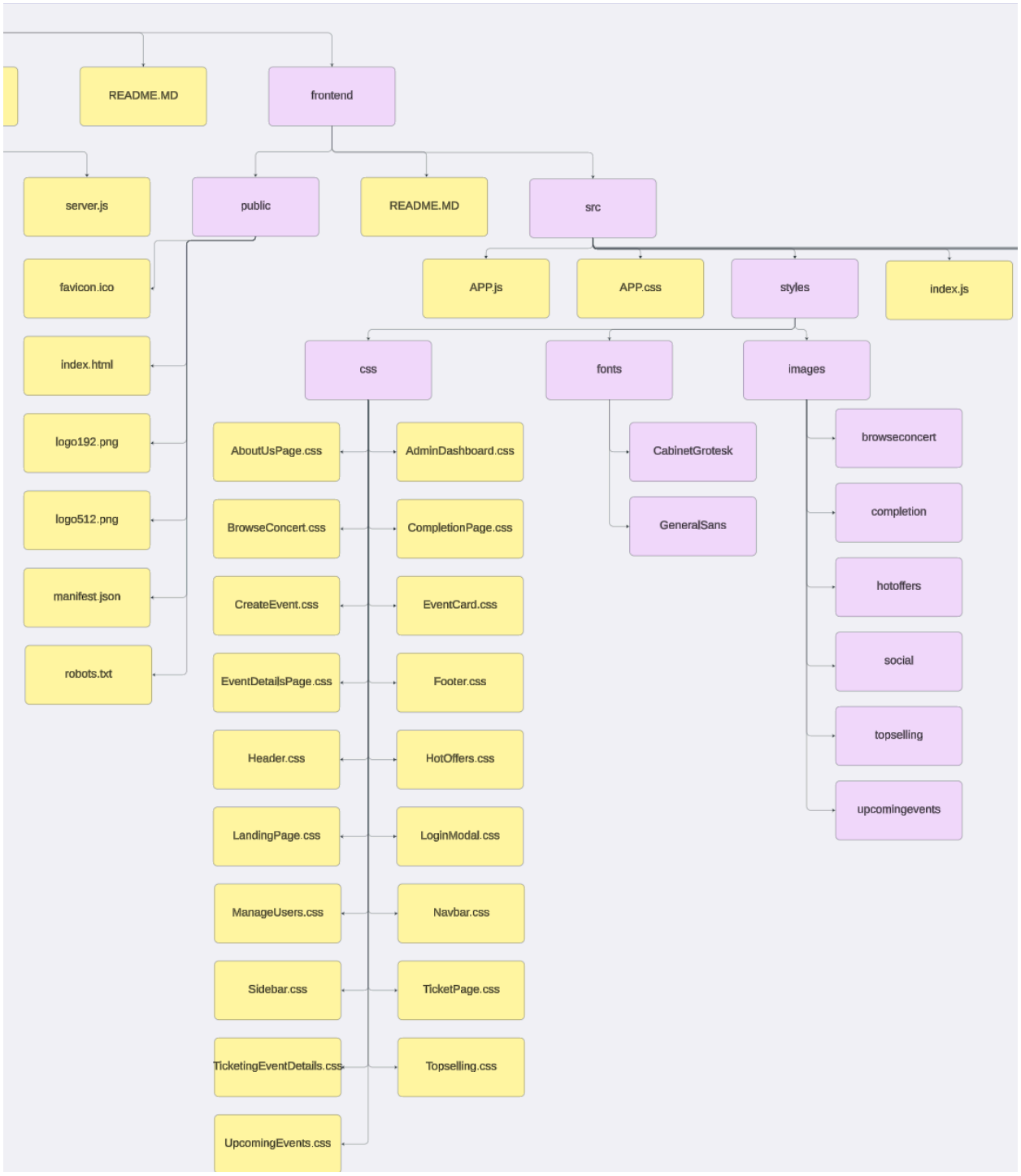
Larger Version available below this image



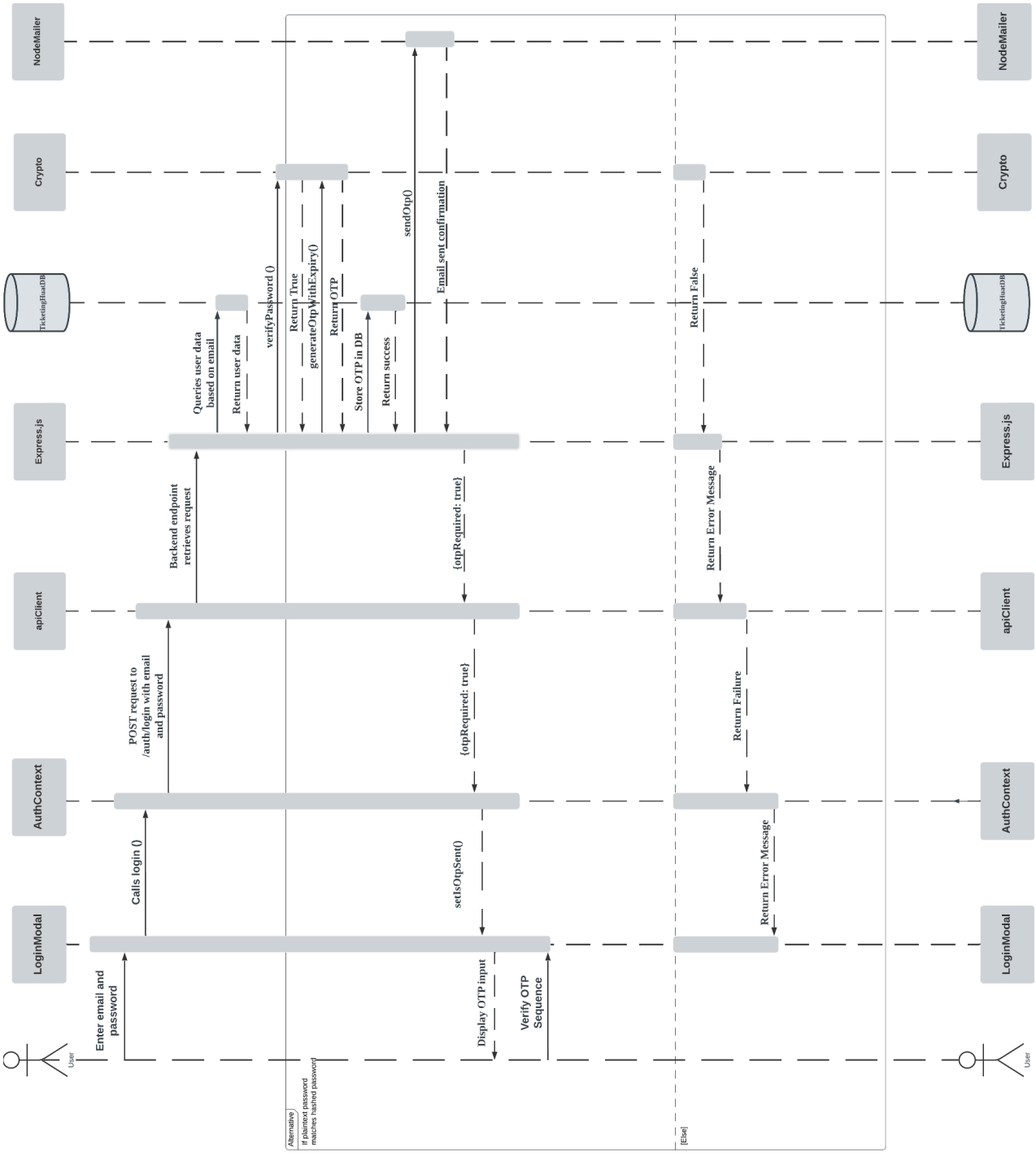

ICT2216-ICT3103-ICT3203-Secure-Software-Development-Grp
backend
.gitignore
Jenkinsfile
adminController.js
authController.js
eventController.js
raffleController.js
controllers
jobs
middleware
models
routes
utils
server.js
raffleCronJob.js
authMiddleware.js
eventModel.js
adminRoutes.js
authRoutes.js
eventRoutes.js
db.js
sendNotification.js
sendOTP.js
favicon.ico
index.html
logo192.png


ICT2216-ICT3103-ICT3203-Secure-Software-Development-Grp18
backend
.gitignore
Jenkinsfile
README.MD
frontend
jobs
middleware
models
routes
utils
server.js
public
README.MD
src
authMiddleware.js
eventModel.js
adminRoutes.js
authRoutes.js
eventRoutes.js
db.js
sendNotification.js
sendOTP.js
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
APP.js
APP.css
styles
css
fonts
AboutUsPage.css
BrowseConcert.css
CreateEvent.css
AdminDashboard.css
CompletionPage.css
EventCard.css
CabinetGrotesk
GeneralSans

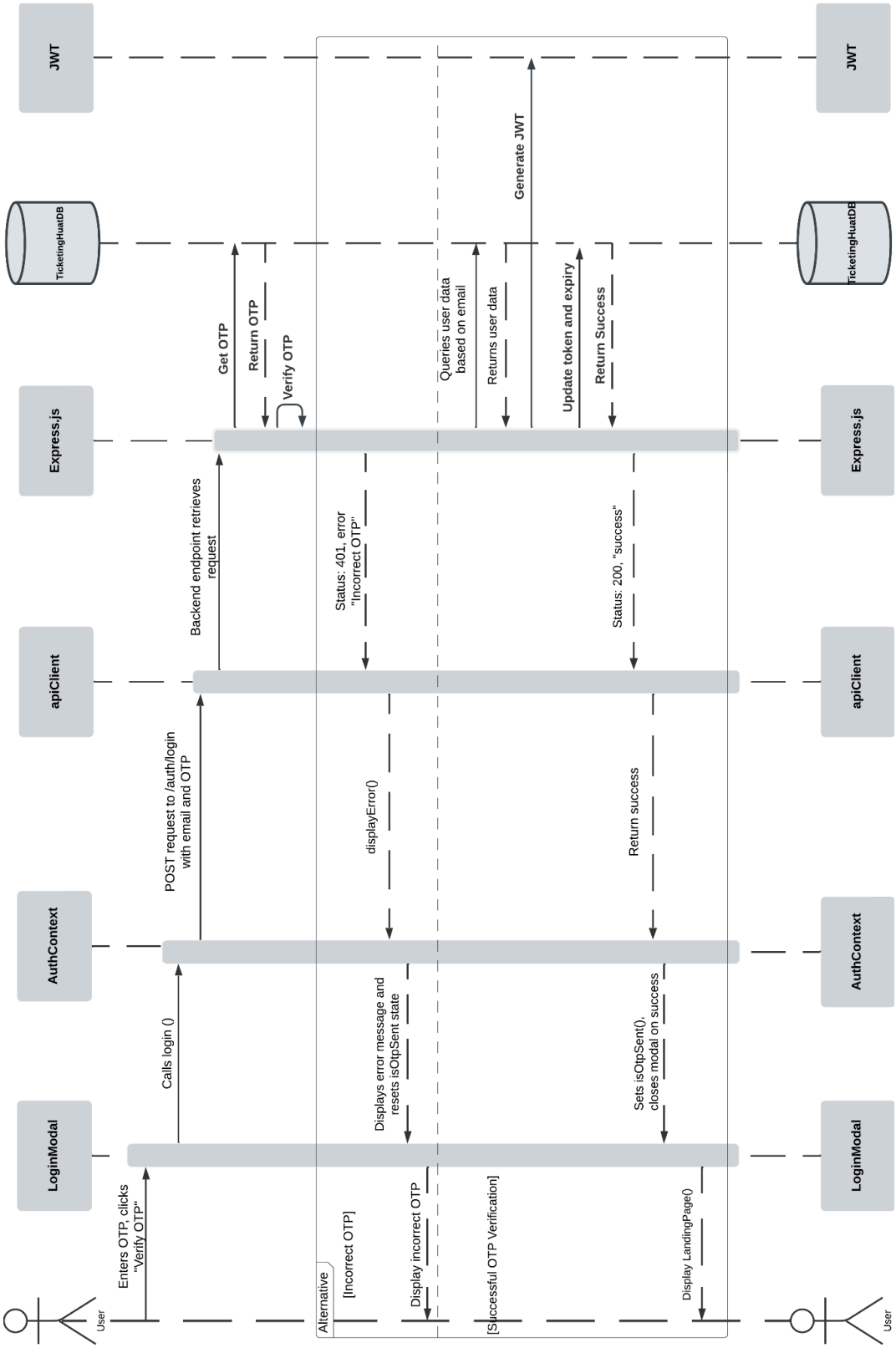

frontend
public
README.MD
src
APP.js
APP.css
styles
index.js
context
index.css
components
axiosConfig.js
pages
logo.svg
css
fonts
images
AuthContext.js
AboutUsPage.css
BrowseConcert.css
CreateEvent.css
EventDetailsPage.css
Header.css
LandingPage.css
AdminDashboard.css
CompletionPage.css
EventCard.css
Footer.css
HotOffers.css
LoginModal.css
CabinetGrotesk
GeneralSans
browseconcert
completion
hotoffers
social
topselling
upcomingevents
BrowseConcert.js
Footer.js
HotOffers.js
Sidebar.js
Topselling.js
EventCard.js
Header.js
Navbar.js
TicketingEventDetails.js
UpcomingEvents.js
AboutUsPage.js
CompletionPage.js
EventDetailsPage.js
LandingPage.js
ManageEvents.js
PaymentPage.js
TicketPage.js
AdminDashboard.js
CreateEvent.js
EventPage.js
LoginModal.js
ManageUsers.js
ResetPasswordPage.js


README.MD
frontend
server.js
public
README.MD
src
favicon.ico
APP.js
APP.css
styles
index.js
index.html
css
fonts
images
logo192.png
AboutUsPage.css
AdminDashboard.css
CabinetGrotesk
browseconcert
logo512.png
BrowseConcert.css
CompletionPage.css
GeneralSans
completion
manifest.json
CreateEvent.css
EventCard.css
hotoffers
robots.txt
EventDetailsPage.css
Footer.css
social
Header.css
HotOffers.css
topselling
LandingPage.css
LoginModal.css
upcomingevents
ManageUsers.css
Navbar.css
Sidebar.css
TicketPage.css
TicketingEventDetails.css
Topselling.css
UpcomingEvents.css

Sequence Diagram for login to get OTP



Sequence Diagram for Verify OTP



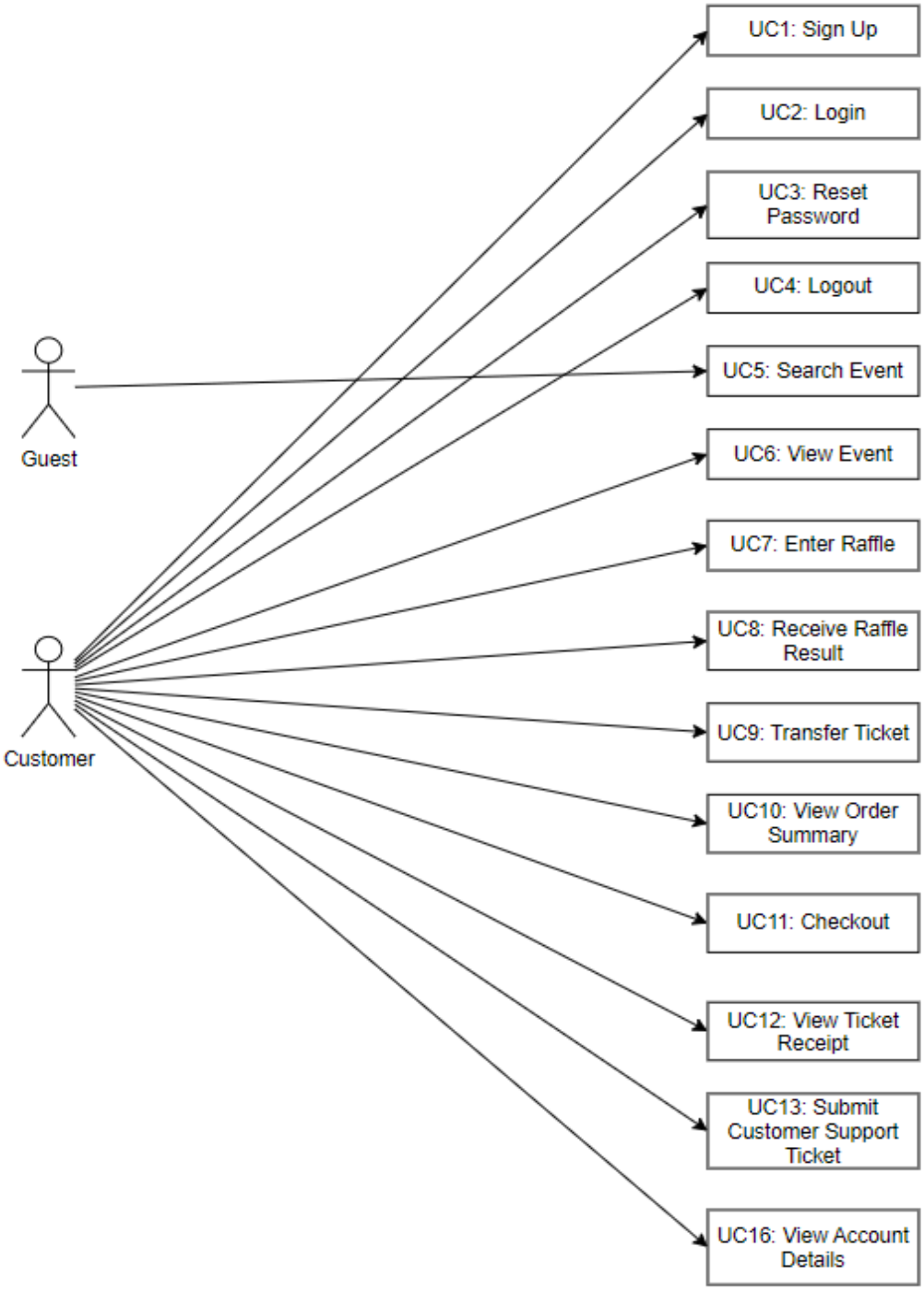
Verifying otp entered by user

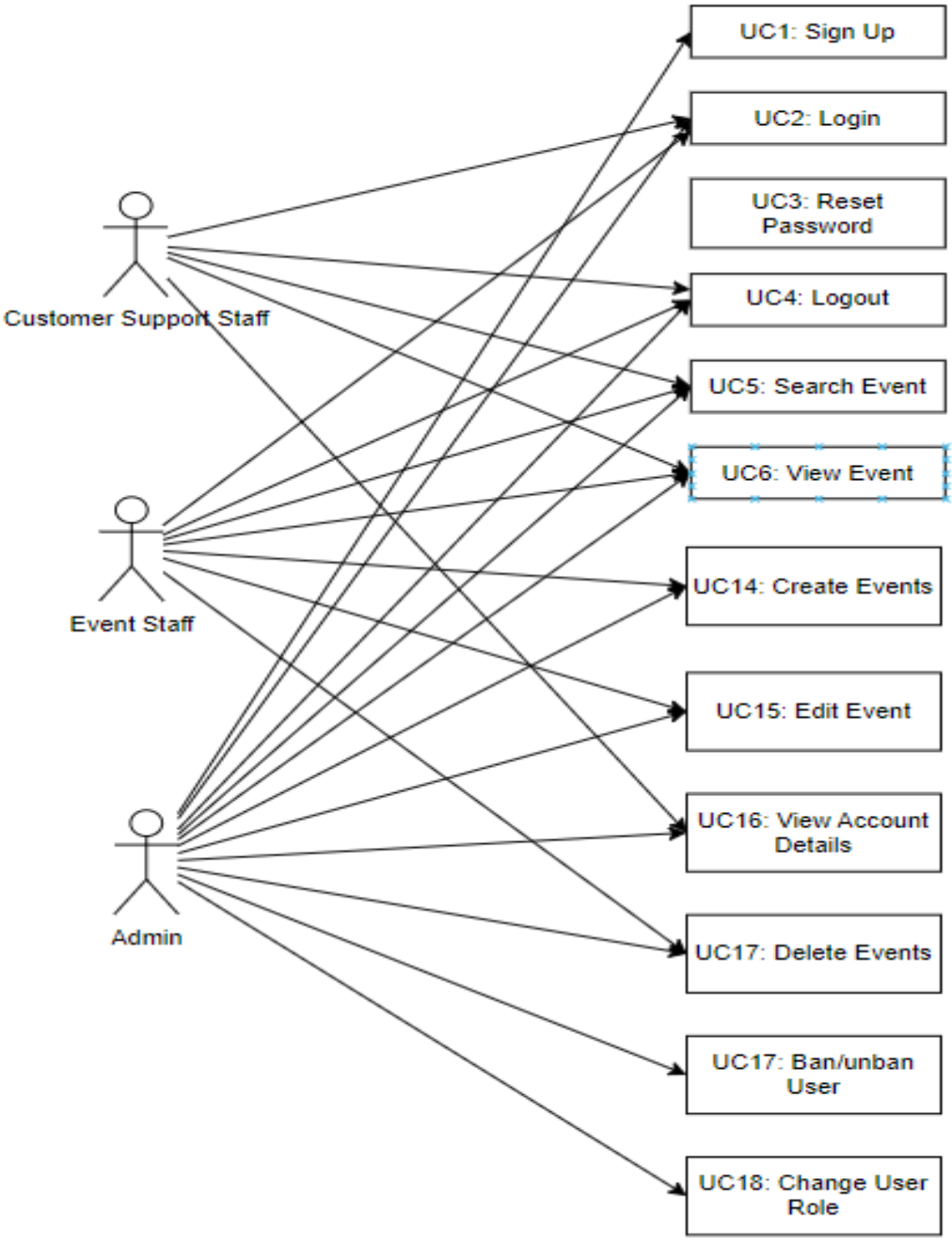
```
const verifyOtp = async (req, res) => {
  const { email, otp } = req.body;
  try {
    if (!email || !otp) {
      return res.status(400).json({ status: 400, message: 'Email and OTP are required' });
    }

    const storedOtpData = otps[email];
    if (storedOtpData && storedOtpData.otp === otp) {
      if (new Date() > storedOtpData.expiresAt) {

        // Invalidate expired OTP
        delete otps[email];
        return res.status(401).json({ status: 401, message: 'OTP has expired' });
      }
      // Invalidate OTP after successful verification
      delete otps[email];

      const [rows] = await db.execute('SELECT * FROM user WHERE email = ?', [email]);
      const user = rows[0];

      // Invalidate any existing sessions
      await db.execute('UPDATE user SET session_token = NULL, session_expiry = NULL WHERE email = ?', [email]);

      // Regenerate the session
      req.session.regenerate(async (err) => {
        if (err) {
          return res.status(500).json({ status: 500, message: 'Failed to regenerate session' });
        }

        // Store user information in the session
        req.session.userId = user.user_id;
        req.session.email = user.email;

        // Store the session ID and expiry in the database
        const sessionToken = req.sessionID;
        const sessionExpiry = new Date(Date.now() + 30 * 60 * 1000); // 30 minutes

        await db.execute('UPDATE user SET session_token = ?, session_expiry = ? WHERE email = ?', [sessionToken, sessionExpiry, email]);

        // Generate JWT
        const token = jwt.sign({ id: user.user_id, email: user.email, role: user.user_role, sessionToken }, jwtSecret, { expiresIn: '30m' });

        res.cookie('token', token, {
          httpOnly: true,
          secure: process.env.NODE_ENV === 'production',
          sameSite: 'Strict',
          maxAge: 30 * 60 * 1000, // 30 minutes
        });

        return res.status(200).json({ status: 200, message: 'Login successful', user: { id: user.user_id, email: user.email, role: user.user_role } });
      });
    } else {
      return res.status(401).json({ status: 401, message: 'Invalid OTP' });
    }
  } catch (error) {
    console.error('Error during OTP verification:', error);
    return res.status(500).json({ status: 500, message: 'Server error', error: error.message });
  }
};
```

Session assignment upon successful authentication

```
// Regenerate the session
req.session.regenerate(async (err) => {
  if (err) {
    return res.status(500).json({ status: 500, message: 'Failed to regenerate session' });
  }

  // Store user information in the session
  req.session.userId = user.user_id;
  req.session.email = user.email;
  req.session.role = user.user_role

  // Store the session ID and expiry in the database
  const sessionToken = req.sessionID;
  const sessionExpiry = new Date(Date.now() + 2 * 60 * 1000); // 30 minutes

  await db.execute('UPDATE user SET session_token = ?, session_expiry = ? WHERE email = ?', [sessionToken,

  // Generate JWT
  const token = jwt.sign({ id: user.user_id, email: user.email, role: user.user_role, sessionToken }, jwtSe

  res.cookie('token', token, {
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'Strict',
    maxAge: 2 * 60 * 1000, // 30 minutes
  });
```

Staff, Admin, User Access Control




UC1: Sign Up
UC2: Login
UC3: Reset Password
UC4: Logout
UC5: Search Event
UC6: View Event
UC14: Create Events
UC15: Edit Event
UC16: View Account Details
UC17: Delete Events
UC17: Ban/unban User
UC18: Change User Role
Customer Support Staff
Event Staff
Admin

NGINX Logs

```
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:20 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:21 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:21 +0000] "GET /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Tes
t/lastBuild/statusIcon HTTP/1.1" 200 349 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.
0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:22 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:23 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:25 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:26 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:26 +0000] "GET /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Tes
t/lastBuild/statusIcon HTTP/1.1" 200 349 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.
0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:27 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:28 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:29 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:31 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:32 +0000] "GET /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Tes
t/lastBuild/statusIcon HTTP/1.1" 200 354 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.
0.0 Safari/537.36"
nginx         | 116.87.28.113 - - [08/Jul/2024:17:49:32 +0000] "POST /jenkins/job/Ticketing%20Huat%20Main/job/Unit-Te
st/lastBuild/logText/progressiveHtml HTTP/1.1" 200 0 "https://ticketinghuat.ninja/jenkins/job/Ticketing%20Huat%20Main
/job/Unit-Test/lastBuild/console" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) C
hrome/126.0.0.0 Safari/537.36"
error from daemon in stream: Error grabbing logs: invalid character 'l' after object key:value pair
```

Snippet of code for Error handling on frontend and backend

```
const login = async ({ email, password, otp }) => {
  try {
    let response;
    if (otp) {
      response = await apiClient.post('/auth/verify-otp', { email, otp }, {
        withCredentials: true,
        onSessionInvalidated: handleSessionInvalidation
      });
    } else {
      response = await apiClient.post('/auth/login', { email, password }, {
        withCredentials: true,
        onSessionInvalidated: handleSessionInvalidation
      });
      if (response.data.otpRequired) {
        return { otpRequired: true };
      }
    }
    if (response && response.status === 200) {
      setIsLoggedIn(true);
      const userResponse = await apiClient.get('/auth/getUser', { withCredentials: true });
      setUser(userResponse.data);
      return { status: 200 };
    } else {
      throw new Error('Login failed');
    }
  } catch (error) {
    throw new Error('Login failed');
  }
};
```

```
try {
  if (!email || !password) {
    return res.status(400).json({ message: 'Email and password are required' });
  }

  const [rows] = await db.execute('SELECT * FROM user WHERE email = ?', [email]);

  if (rows.length === 0) {
    return res.status(401).json({ message: 'Invalid email or password' });
  }

  const user = rows[0];

  if (user.lock_until && new Date(user.lock_until) > new Date()) {
    return res.status(403).json({ message: 'Account locked. Try again later.' });
  }

  const isMatch = await verifyPassword(password, user.password);
```