

# Team 5 (FULLSTACK) - TripScribe - Project Document

## INTRODUCTION

Aims and objectives of the project: create a digital travel-diary web application. We wanted to create an app where users could login and upload details of their trips, so that they have an organised record of the places they have visited, they can view them on a map (similar to putting pins in a wall-map) and can search them to recall details and find recommendations they may wish to share with friends and family.

We wanted to create an app that was easy to navigate for all users, where the design was modern, simple & accessible, where the images uploaded would not be detracted from.

We also wanted to learn from one another, implement skills taught on the course and challenge ourselves to create an app we could be proud of!

### Roadmap of the report

- **Background**
- **Specifications & Designs**
- **Implementation & Execution**
- **Testing & Evaluation**
- **Conclusion**

**BACKGROUND:** TripScribe is a platform designed to help users document their travel experiences. It allows users to create, manage, and showcase trips by uploading photos, adding descriptions, and keeping track of the places they've visited. [TripScribe/README](#)

## SPECIFICATIONS AND DESIGN

### Overview

The TripScribe application is designed to provide users with a personalised and organised way to document and visualise their travel experiences. To access the app's features, users must first register and log in. The application includes several key features that allow users to manage their travel records effectively.

### Accessibility

We prioritised creating an inclusive app by following best practices for a user-friendly interface, guided by Nielsen's design principles. Key accessibility features include descriptive alt-text for images and strong colour contrast for readability, especially for users with visual impairments. After a 92% Lighthouse accessibility audit, we made improvements like adding descriptive page titles and a keyboard-accessible navigation bar to enhance accessibility.

**Colour Palette:** We chose calming shades of green, ideal for a travel diary app. Greens evoke nature and exploration, while blue-greens are widely liked, gender-neutral, and easy on the eyes. The dark green (#476a6f) contrasts well with white for readability, and the predominantly white background keeps user-uploaded images as the focal point. [Link to Figma UI Designs](#)

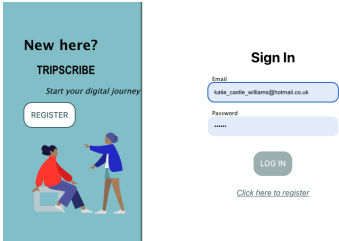
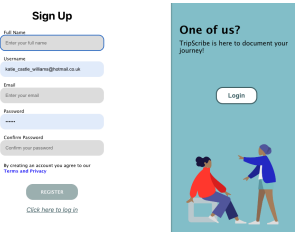
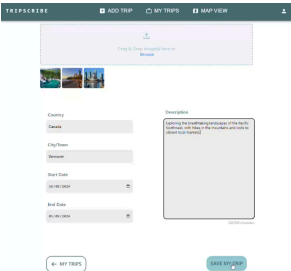
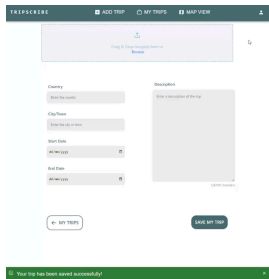
Feature- Page	MVP Business Requirements User workflow	Technical Requirements
<p><i>Fig.1 Login page where users can either register or login</i></p>  <p><i>Fig 2. Registering page</i></p> 	<p><b>User Registration and Login</b></p> <p>Users create accounts and log in to access app features, ensuring secure data storage and access. Users can register their details and then log in. Passwords will be encrypted before storing in the database.</p> <p>User registration Acceptance Criteria- FE's <a href="#">user story</a> and BE's <a href="#">user story</a></p> <ul style="list-style-type: none"> <li>• Required Fields validation for: email, password, and confirm password.</li> <li>• Receive and Validate Registration Data</li> <li>• Hash User Password</li> <li>• Store User Data</li> <li>• Return Success Response</li> </ul> <p>Login Authenticate Acceptance Criteria- BE's <a href="#">user story</a> and FE's <a href="#">user story</a></p> <ul style="list-style-type: none"> <li>• Receive and Validate Login Credentials: the system validates the credentials</li> <li>• Verify Hashed Password</li> <li>• Generate JWT or Session Token</li> <li>• Return Token and User Details (decoded)</li> <li>• Secure Token Storage and Transmission</li> <li>• The user enters a valid email and password.</li> <li>• The system verifies the credentials.</li> <li>• The user is redirected to the dashboard upon successful login.</li> </ul>	<ul style="list-style-type: none"> <li>• Node.js</li> <li>• Express</li> <li>• Middleware</li> <li>• Form Validation</li> <li>• Bcrypt</li> <li>• JWT - env variables</li> <li>• Axios</li> <li>• Context management</li> <li>• Session storage- token</li> <li>• Redux implementation</li> <li>• Reusable components (inputs &amp; buttons)</li> </ul> <p><b>API Endpoints (Axios)</b></p> <ul style="list-style-type: none"> <li>• <a href="http://localhost:8000/api/login">http://localhost:8000/api/login</a></li> <li>• <a href="http://localhost:8000/api/register">http://localhost:8000/api/register</a></li> </ul> <p><b>Page routes</b></p> <ul style="list-style-type: none"> <li>• /</li> <li>• /register</li> </ul>
<p><i>Fig. 3 Addtrip page with 3 photos uploaded</i></p> 	<p><b>Trip Management- adding a new trip</b></p> <p>Users register trip details such as country, city, start-date, end-date and description. And they can upload photos</p> <p>Images cloud storage Acceptance Criteria- BE' s <a href="#">user story</a> and FE's <a href="#">user story</a></p> <ul style="list-style-type: none"> <li>• Image Upload by Drag and Drop via drag and drop or by browsing files.</li> <li>• Displays a preview of the uploaded image</li> <li>• Delete images in the stage area</li> <li>• Image Format and Size Validation (#4 items, JPEG, PNG, maximum 10MB)</li> <li>• Receive Image File: system can accept image files via an API endpoint             <ul style="list-style-type: none"> <li>◦ Frontend data conversion to a base64 string to the backend</li> <li>◦ Validate Image File: image file format and size</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Node.js</li> <li>• Express</li> <li>• Middleware</li> <li>• Form Validation</li> <li>• File validation format -size</li> <li>• Convert img Base64</li> <li>• JWT - env variables</li> <li>• Context management</li> <li>• Axios</li> <li>• Session storage- token</li> <li>• Reusable components (inputs &amp; buttons)</li> </ul>

Fig. 4 Add trip page after 'save my trip' button clicked.



- Store Image in designated cloud location (Cloudinary) and generate Unique Identifier or URL is used to access the image later

- Retrieve Stored Image and display in the frontend
- Error Handling and Logging during the process
- Access Controls and Authentication mechanisms are in place to protect image data. Only authorised users can access the stored image

Form inputs - Acceptance Criteria BE' s [user story](#) and FE's [user story](#)

- Mandatory Fields Validation: country, city, dates and description
- Date Range Validation
- Display Success Message Upon Successful Trip Save
- API endpoint to save trip information
- JSON object with the following fields: image URL(s), country, city, start date, and end date
- Save trip information in the DB, associating it with the user who created the trip

- Cloudinary API integration
- Store metadata
- Auth access control

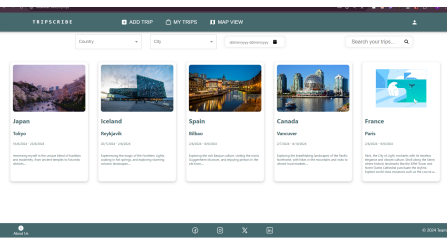
#### API Endpoints (Axios)

- <http://localhost:8000/api/uploadImages>
- <http://localhost:8000/api/addtrip>

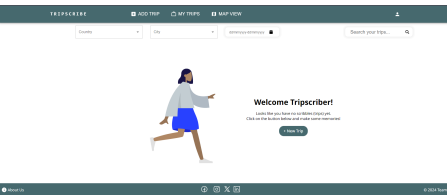
#### Page routes

- /addtrip

Fig. 5 My Trips page, larger screen



#### 'No trips' page



#### Trip Management- Dashboard

Users can see cards for all trips they have uploaded, with a single photo, country, city, dates & a short description (250 char). Users can filter and search cards. User can click on a specific trip-card to be redirected to the trip details

Trips Dashboard - Acceptance Criteria BE' s [user story](#) and FE's [user story](#)

- The backend provides an API endpoint to retrieve trip information.
- The trip cards are displayed in a grid layout; API endpoint returns a list of trips with details including image URL, country, city, and date range.
- The endpoint accepts parameters to filter trips by user, date range, country, and city.
- Fetch List of Cities/Towns Based on Selected Country
- Filter Trips by Date Range
- Search Trips by Text (available on the description field)

- Node.js
- Express
- Middleware
- JWT - env variables
- MUI Grid
- Axios
- Session storage- token
- Reusable components (inputs & buttons)
- Redux implementation

#### API Endpoints (Axios)

- <http://localhost:8000/api/trips>

#### Page routes

- /trips

#### Trip Management- view & edit (delete) trip details

Users can see the trips details: photo carousel (max 4 photos) country, city and dates. User can see the details of trips which were previously uploaded. (via trip-details).

- Node.js
- Express
- Middleware
- Form Validation
- File validation format -size
- Convert img Base64

Fig. 6 Trip Details page

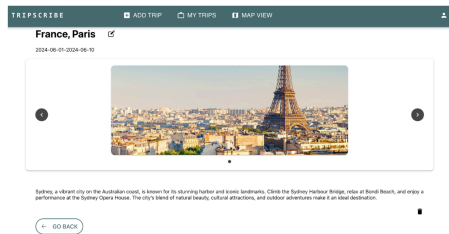


Fig. 7 EditTrip

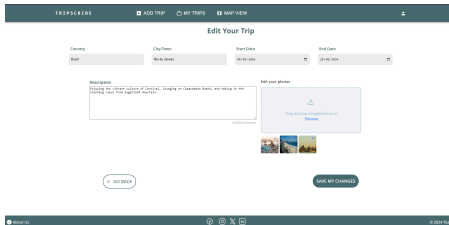
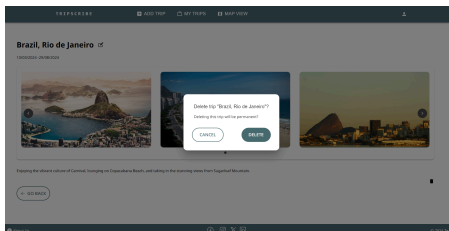


Fig. 8 DeleteTrip



## Trips Details - View &amp; Edit Acceptance Criteria

BE' s [user story](#) / [user story](#) and FE's [user story](#) and [user story](#)

- The backend provides an API endpoint to update trip details
  - The system ensures that only authorised users can update trip details
  - The updated information is validated and saved to the database..
  - The system queries the database for images related to the specified trip.an retrieved images are sent to the frontend for display in the carousel.
- The backend provides an API endpoint to receive delete trip requests..
- The endpoint accepts the trip ID as a parameter..
- Validate Delete Request: the system verifies that the user requesting the deletion is authorised to delete the trip
- Associated data (e.g., images) is also removed or disassociated..
- Return Success or Error Response

- JWT - env variables
- Context management
- Axios
- Session storage- token
- Reusable components (inputs & buttons)
- Cloudinary API integration
- Store metadata
- Auth access control

## API Endpoints (Axios)

- [http://localhost:8000/api/trips/\\${tripID}](http://localhost:8000/api/trips/${tripID})
- [http://localhost:8000/api/edittrip/\\${tripID}](http://localhost:8000/api/edittrip/${tripID})

## Page routes

- /trips/:tripID
- /edittrip/:tripID

Fig 9 - Map View

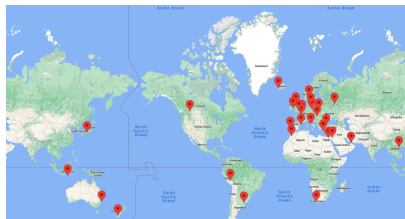
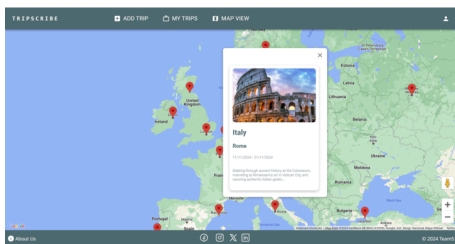


Fig. 10 Map view card trip



### Map View

Visual representation of visited locations on an interactive map, enhancing the travel experience. Users can zoom in/out and move around the world-map.

Map view - Acceptance Criteria - [user story](#)

- Display Map on the Maps view
  - The map should display the user's current location (if permission is granted).
  - The map should allow the user to zoom in/out and pan around.
- Responsive Map Experience
- Integrate Google Maps API
  - The API key should be securely stored and managed.
  - The backend should handle API requests and responses efficiently.
- Provide Location Data to Frontend

- Node.js
- Express
- Middleware
- JWT - env variables
- Context management
- Axios
- @react-google-maps/api library

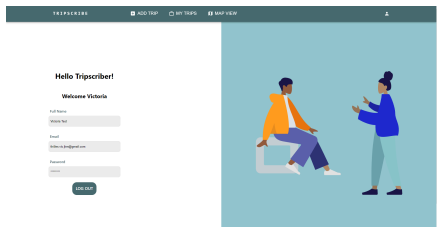
### API Endpoints (Axios)

- <http://localhost:8000/api/trips>

### Page routes

- /maps
- /google-maps-key

Fig. 11 User Profile page, larger screen.



### User Profile Page

Displays user profile details (name, email) with logout option.

User Profile - Acceptance Criteria - [user story](#)

- Display User Profile Information: user's full name, username, email, and password (masked).
- The backend provides an API endpoint to fetch user profile information.
- Logout functionality

- Node.js
- Express
- Middleware
- JWT - env variables
- Context management
- Axios
- Redux implementation

### API Endpoints (Axios)

- [http://localhost:8000/api/user/\\${userID}](http://localhost:8000/api/user/${userID})

### Page routes

- /userprofile

Fig. 12 AboutUs page

### About Us Page

Contact information about the app's developers, including GitHub and LinkedIn links, and an overview of app features.

About us - Acceptance Criteria - [user story](#)

- MUI's Grid system and Box components.

### Page routes

- /aboutus

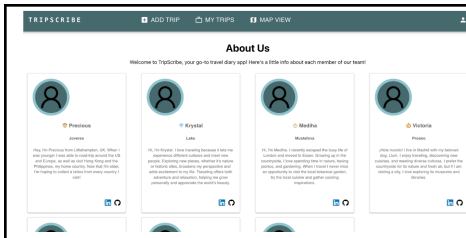


Fig. 13 Responsive Navbar

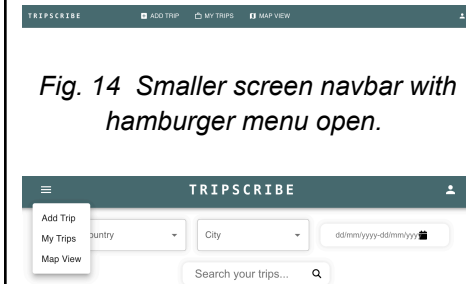


Fig. 14 Smaller screen navbar with hamburger menu open.

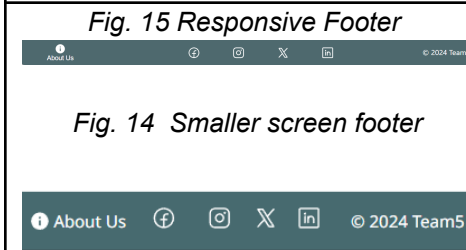
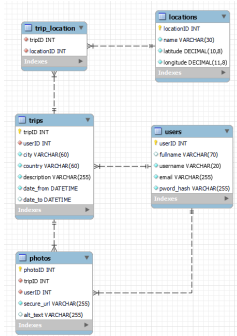


Fig. 14 Smaller screen footer

Fig. 15 ERD for Tripscribe DB



- Display Team Members' Information - static page
- Each card includes icons/links to the team member's social media profiles
- Responsive Design

## Navbar

The navbar bar contains clear links and icons to 'My Trips', 'Add Trip', 'Map View' and 'User Profile' (user-icon).

As the navbar is responsive, when resized for small screens, the navbar adapts to contain a hamburger menu with a dropdown containing the links to "/mytrips", "/addtrip" and "/map".

## Footer

All pages use a responsive footer component, which makes use of *BottomNavigation* from MUI.

The footer becomes visible at the bottom of the screen at the end of the content and there are links to [/aboutus](#)


The footer component is imported into the app.js file and used inside the <Router/>, to ensure that it is visible on all pages.

## Data Storage

Secure storage of user data (personal information and trip details) in a database for reliable access and management.

Our database, tripscribedb, stores information related to users, their trips, locations visited, and photos uploaded in relation to those trips. The tables within the database are as follows:

- Users: userID, fullname, username, email, pword\_hash
- Trips: tripID, userID(FK [User.userID](#)), city, country, description, date\_from, date\_to
- Locations: locationID, name, latitude, longitude
- Photos: photoID, tripsID(FK [Trips.tripID](#)), userID(FK [Users.userID](#)), secure\_url, alt\_text
- Trip\_Location (junction table): tripID(FK [Trips.tripID](#)), locationID([Locations.locationID](#))

	<p>Most tables have many-to-one relationships, except for the many-to-many connection between trips and locations, managed by a junction table called <code>trip_location</code>. Currently, each trip is linked to only one location, but the structure is designed to allow multiple locations per trip in the future, providing scalability and flexibility.</p>
<p><i>Fig. 16 Project structure</i>  <a href="#">see detail here</a></p> 	<p><b>Backend</b></p> <p>the Model-View-Controller (MVC) framework- the Backend is built using, which divides the application into three core components: the Model, handling data and business logic; the View, managing the user interface; and the Controller, acting as an intermediary between the Model and View. This separation enhances maintainability and scalability.</p> <p><b>Frontend</b></p> <p>This React project is organised with a <code>src</code> directory containing key folders such as components for reusable UI elements, context for state management, features for application-specific logic, hoc for higher-order components, and pages for different view templates. The project also includes essential setup files like <code>App.js</code>, <code>index.js</code>, and testing files to ensure component functionality.</p>

## IMPLEMENTATION AND EXECUTION:

### Development approach and team member roles

We decided on Agile methodology for our project but adapted it to fit the unique needs and limitations of the team (full time jobs, family commitments, etc). Our approach included essential elements like sprint planning, status stand-up (these were held twice a week on Slack) and sprint reviews.

As the project evolved, we remained flexible, adjusting assigned roles throughout the sprints to best meet the project's current demands. Our excellent communication played a crucial role in ensuring we delivered the agreed-upon requirements, allowing us to collaborate effectively and respond to challenges as they arose.

See the [Project Assignment.md](#) for further details on roles and task distribution (tables below).

Project-Product management Cycle  
Github- roles & tasks

Project-Product management Cycle- roles & tasks							
	Katie Williams	Krystal Lake	Lily Wright	Marta Walters	Mediha Mustafaova	Precious Joveres	Victoria Proaño
Project Definition	✓	✓	✓	✓	✓	✓	✓
Taking Minutes	✓	✓	✓	✓	✓	✓	✓
Team Facilitators	✓	-	-	✓	-	-	-
Trello's Facilitators	-	-	-	-	✓	-	✓
Sprint Planning & Review	✓	✓	✓	✓	✓	✓	✓
Figma Wireframe	✓	-	-	-	✓	-	✓
User stories- Backlog	-	-	-	-	-	-	✓
Compilation of Proj. Assignment	-	✓	-	-	-	-	✓
Compilation of Proj. Documentation	✓	✓	✓	✓	✓	✓	✓
Compilation of Proj. Presentation	✓	-	-	-	-	-	-
Github- roles & tasks							
	Katie Williams	Krystal Lake	Lily Wright	Marta Walters	Mediha Mustafaova	Precious Joveres	Victoria Proaño
GitHub Repo Creation & Organisation	✓	-	✓	-	-	-	-
PR Reviewers	✓	✓	✓	✓	✓	✓	✓
Compilation of README file	-	-	-	-	✓	-	✓
Development Task Commitment UPDATED							

Development tasks UPDATED (TBC)

Development Task Commitment UPDATED							
	Katie Williams	Krystal Lake	Lily Wright	Marta Walters	Mediha Mustafaova	Precious Joveres	Victoria Proaño
Database Design & Creation	-	-	-	-	-	✓	-
Authentication: Login & Register Page FE-BE	-	-	-	-	✓	✓	✓
Register FE	-	✓	-	-	-	✓	-
Add My Trip: Form & Image Upload	✓	-	-	-	-	-	✓
Add My Trip: Cloudinary API Integration	-	-	-	-	-	-	✓
My Trip Page: Data Retrieval for Grid View Cards	-	-	-	-	✓	-	-
My Trip: Grid View, Filters & Search Bar	-	-	✓	-	✓	-	-
Trip Details: View	-	-	-	-	✓	-	-
Trip Details: Edit FE	-	-	-	✓	✓	-	-
Trip Details: Edit BE	-	-	-	-	✓	-	-
Map View: GoogleMaps API	-	-	✓	-	-	-	-
User Profile	-	-	-	-	✓	-	✓
About Us	✓	-	-	-	-	-	-
Redux Implementation	✓	-	-	✓	-	-	✓
Unit tests	✓	-	-	-	✓	-	✓
Functional testing	✓	✓	✓	✓	✓	✓	✓
Reusable Components: Navigation Bar	✓	-	-	-	-	-	-
Reusable Components: Buttons, Input Fields	✓	-	-	-	-	-	✓
Reusable Components: Uploader img	-	-	-	-	-	-	✓
Reusable Components: Card trips, Filters, Search Bar	-	-	✓	-	✓	-	-
Reusable Components: Carousel img, Modal Dialog	-	-	-	-	✓	-	-
Reusable Components: Footer	✓	-	-	✓	-	-	-

We utilised Agile methodology to develop our Tripscribe app, emphasising an iterative approach and continuous improvement. Our workflow was organised using a [Trello board](#) to track user stories for each sprint, ensuring clear task prioritisation. Code reviews were vital; any pull requests (PRs) to merge into the 'dev' branch required two reviews for approval, fostering collaboration and quality assurance. We held regular stand-ups via Slack on Mondays and Wednesdays at 9 AM, where we shared updates on our progress—what we had completed (DONE), what we were working on (DOING), and any obstacles (BLOCKERS). Our full team meetings on Thursdays allowed us to conduct Sprint Reviews and plan for the next iteration. This approach helped us adapt quickly and deliver a digital travel diary that meets user needs.

Tools and libraries

We used a range of tools and technologies to build the Tripscribe web app. **VSCode** served as our code editor with built-in GitHub integration for seamless collaboration. **GitHub** hosted our project repository, supporting version control and pull requests. **React** was the core framework for creating dynamic, component-based UIs, while **MySQL** handled backend data storage. We ensured a consistent look with **Material UI** and **Phosphor Icons**. **Lighthouse in Chrome Dev-Tools** was used for performance auditing, and **React Testing Library** ensured component reliability. **React Helmet** managed dynamic page titles, improving accessibility. We used **Postman** for API testing, **Cloudinary** for image uploads, and **Axios** for HTTP requests between the frontend and backend. For more details, see the dependencies listed in our package.json

Implementation process

TripScribe was shaped by team achievements, challenges, and key decisions, resulting from a collaborative and adaptive development process.



<b>Achievements</b>	<ul style="list-style-type: none"> <li>● <b>Fullstack development in practice:</b> Successfully connecting the frontend with the backend enabled core features like user authentication and real-time data.</li> <li>● <b>API Integration:</b> The integration with Cloudinary API and Google Maps API was a highlight. It helped us understand how to consume third-party services and enhance user functionality in our application.</li> <li>● <b>Teamwork and Collaboration:</b> Strong teamwork, daily standups, and weekly meetings facilitated problem-solving and idea-sharing. Regular Sprint Reviews boosted morale and helped track progress</li> <li>● <b>Continuous Learning:</b> The team expanded their skills through tackling newly learned tools &amp; technologies like React, Figma, and SQL.</li> </ul>
<b>Challenges</b>	<ul style="list-style-type: none"> <li>● <b>Balancing Schedules:</b> Members juggled project work with personal responsibilities, requiring careful planning and communication.</li> <li>● <b>Diverse Experience Levels:</b> Varied expertise presented both learning opportunities and task distribution challenges.</li> <li>● <b>Backend -Frontend:</b> Smooth integration required effective coordination, pair programming, and group meetings.</li> <li>● <b>Git Merging Conflicts:</b> Merging code changes required careful conflict resolution and strong version control practices.</li> <li>● Security and Performance:</li> </ul>
<b>Changes and adaptations</b>	<ul style="list-style-type: none"> <li>● <b>MUI Design Adjustments:</b> Material UI was used to improve visual consistency with the footer and navbar.</li> <li>● <b>Figma Design Simplification:</b> Simplified designs prioritised core functionality while maintaining aesthetic appeal.</li> <li>● <b>Accessibility Enhancements with Helmet:</b> Implementing Helmet improved the app's accessibility and inclusivity.</li> <li>● <b>Agile Methodology Flexibility:</b> Sprint plans were adjusted to support integration and reassess goals.</li> <li>● <b>Third-party Component Removal:</b> The Cloudinary uploader component was removed to simplify the project. We decided to create a simple in-house component, focusing on essential features and maintaining a consistent UI design.</li> <li>● <b>Auth &amp; Token storage:</b> forcing a user to log out and removing their authentication token to prevent unauthorised users from accessing the system or service.</li> </ul>

## TESTING AND EVALUATION:

### Testing strategy & Unit Tests

We have implemented snapshot and unit testing for frontend components, along with mock testing for backend logic, ensuring comprehensive coverage of both success and error scenarios. Our testing framework is designed to catch potential issues early, contributing to a more robust and reliable application. In addition, we place a strong emphasis on performance and security in our testing and development processes, ensuring that our code not only works as expected but also adheres to industry best practices for speed and data protection.

### Quality Assurance and Functional user testing

We conducted thorough user flow tests to ensure all features functioned as expected. Starting with registration and logging in as a new user, we added, edited, and deleted a trip, reviewed the user profile, and completed the process by logging out.

Next, we tested the app as an existing user, specifically using the profile of Lily's account, to verify the functionality of the map and pin features. During this process, we identified a few design and responsiveness issues that needed adjustment. After implementing these changes, we conducted another round of user flow tests to confirm everything worked correctly.

### Effective use of Redux

We implemented Redux across the application using `redux`, `react-redux`, and `@reduxjs/toolkit`, with reducers defined in `userRedux.js` and the store set up in `index.js`. Redux manages state across pages, during our user testing we checked that Redux updated the name field based on user actions like registration, login, and logout.

### System limitations & Next Steps

Current Limitation	Next Steps
App is currently only accessible on our local machines.	Deploy the app using a platform such as Vercel or Netfly (FE) and Heroku or Render (BE)
How could we monetise this app?	Subscription-fee options for more photos, ability to 'friend' users. Advertisements for travel companies or countries' tourist information boards.
Mobile application	Develop a react native application so the user can add trips in real time on mobile devices
SSO- Register and Login	Allows users to authenticate once and gain access to multiple applications or services without needing to log in again. User Gmail or AppleID, social media accounts, etc
Number of photos	Look into paid cloud storage plan that allows more cloud space
User can only see their own trips	Enable users to 'friend' others, allowing them to view each other's trips, and share trip links via messaging apps and social networks like WhatsApp, Telegram, Instagram, and X.
Database management improvements	Allow users to include trips with multiple locations (or regions) that will improve the DB scalability and flexibility.
Currently we only have the latitude, longitude of limited places (currently hard-coded) in Maps view.	Improve Integration of an API such as Google Places API. This API allows you to query for place information, including the latitude and longitude of specific locations. The user could use this to facilitate uploading a trip whilst on holiday.

## CONCLUSION

Our TripScribe project aimed to create a user-friendly travel diary, and we achieved this by building an app that lets users document, manage, and visualise their trips. The development process taught us valuable lessons in teamwork, adaptability, and time management. Overall, this project strengthened our technical skills and highlighted the importance of clear communication and goals through collaboration. In the end, we made an app that we're proud of and for some members it was their first project ever, making this even more exciting! We hope you love our app as much as we do.