To enhance data protection in the project, I will focus on securing the API endpoints, protecting stored data, and ensuring safe data transmission. Here's a summary of the necessary changes implemented in both the front-end and back-end components of the project.

Backend Modifications

1. Use HTTPS for All Communications
   - Ensure the server is set up to serve content over HTTPS. This encrypts data in transit, preventing it from being intercepted.

2.Implement Input Validation and Sanitization
   - Validate and sanitize all inputs to prevent SQL injection and other injection attacks. This can be done using libraries such as `express-validator`.
   - Example of input validation in an Express route

```
const { check, validationResult } = require('express-validator');

app.post('/api/journeys', [
  check('startPoint').isString().trim().escape(),
  check('endPoint').isString().trim().escape()
], (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }
});
```

3. Hash Passwords Before Storing Them
   - Use bcrypt to hash passwords if the system handles user authentication.
   - Example of hashing a password

```
const bcrypt = require('bcrypt');
const saltRounds = 10;

bcrypt.hash('myPassword', saltRounds, function(err, hash) {
  // Store hash in the password DB.
});
```

4. Implement CORS Properly
    - Configure CORS correctly to ensure that the API only accepts requests from allowed origins.
    - Example of setting CORS in Express

      ```
      const cors = require('cors');
      const corsOptions = {
      origin: 'https://trusteddomain.com',
      optionsSuccessStatus: 200
      };
      app.use(cors(corsOptions));
      ```

Front End Modifications

1. Use HTTPS for API Requests
    - Ensure all requests to the backend are made over HTTPS to protect data in transit. This can be configured in the Axios setup.
2. Implement Content Security Policy
    - Set up a Content Security Policy (CSP) to reduce the risk of XSS attacks by specifying trusted sources of content.
3. Sanitize Any Dynamic Content
    - Use libraries that automatically escape HTML if the user needs to output any user-generated content.