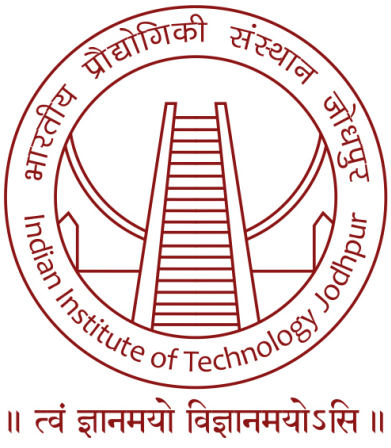
Coding Documentation Topic: IITJ Travel Sathi - A Platform to Find Companions for Shared Rides

A Coding Documentation Submitted by

Akaash Chatterjee - M24CSE002

Aman Saini - M24CSE003

Bera Swaminath Ansuman Sabita - M24CSE007



Indian Institute of Technology Jodhpur

Department of Computer Science and Engineering

September, 2024

Contents

1 Objective

The objective of this code documentation is to provide a comprehensive and clear overview of the TravelSathi application, which facilitates collaborative travel among students.

2 File Descriptions

1. **app.py**: The `app.py` file is where all of the configuration and initialization of the Flask web framework happens in our TravelSathi application. To define the program's behavior, handle user authentication, establish a connection to the database, and set up the application environment, this file is essential.

   The first step is to import the required libraries, which include Python, SQLAlchemy, Bcrypt, LoginManager, and CORS, among others. The primary application object is created by instantiating the `Flask` class, and `CORS` allows our application to interact with other domains if necessary by enabling Cross-Origin Resource Sharing. To keep private details like database URLs and secret keys organized and safe, the application configuration is loaded from a separate `Config` class.

   The application's data models are easily managed with SQLAlchemy, which is used to initialize the database. We further strengthened security by establishing Bcrypt for password hashing, which guarantees the secure storage of user passwords. For the purpose of controlling application-wide authentication and user sessions, the `LoginManager` is initialized.

   Flask-Login retrieves a user from the database using the session's user ID and a user loader function that has been configured. Features like user authentication and access control are made easier using this function, which enables easy retrieval of user data during a session.

   We utilize a `before_request` function to construct the required database tables every time the application starts, which helps to preserve the schema. Without having to manually set everything up every time, this guarantees that our models are accurately mirrored in the database.

   Finally, we import the application's routes, which let us specify different endpoints that users may interact with. For testing and development purposes, the app is configured to run on port 5000 on host `0.0.0.0`. We can easily build and test the application in real time with this arrangement.

   As a whole, the `app.py` file integrates critical features and components that power the application, making it the backbone of the TravelSathi platform. Our well-structured startup and configuration procedures provide a strong basis for building a powerful and intuitive shared transportation platform.

2. **routes.py**: The `routes.py` file manages the web routes and business logic of the TravelSathi application, making it an essential component of the whole. This section provides a synopsis of its main features and functions:

Overview

Modules are imported and the Flask application, database, and authentication manager are initialized at the beginning of the program. Additionally, it enables user session management by defining the user loader for Flask-Login. Several route handlers are defined to make it easier to register users, login, create rides, and manage them.

Important Elements

(a) The home route (`'/'` and `'/home'`) verifies the user's authentication status. If authenticated, it sorts through all of the rides and displays only the ones that are appropriate according to their gender preference.

(b) **User Registration**: Users can create an account through the `'/register'` route. A confirmation message is sent when the user's password is hashed for security after registration is successful.

(c) **User Login**: The user login process takes care of user authentication through the `'/login'` route. It checks the user's credentials and controls the page redirection after login. An error message is shown if the login attempt fails.

(d) The `'/ride/new'` route enables authenticated users to generate new ride offers. It takes note of specifics about the ride and then automatically adds the user to their own creation, changing the number of available seats in the process.

(e) **Join a Ride**: Users can join available rides through the `'/ride/join/<int:ride_id>'` route. Prior to the user being added to the ride, it verifies their current ride status and the ride's gender choice. It also handles the availability of seats.

(f) **Rides Joined**: Journeys that the user has either created or joined are retrieved and shown by the `'/joined_rides'` route. If the user does not have any rides available, a notification will be shown to them.

(g) Use the `'/logout'` route to log the user out of the application and return them to the home page.

(h) The ability to remove rides and opt out is provided by the `'/delete_ride/<int:ride_id>'` route. By using the `'/opt_out/<int:ride_id>'` route, users can update the availability of seats by opting out of rides they have already joined.

(i) **Search Rides**: According to the user's gender preference, the `'/search_rides'` route retrieves rides and shows them.

(j) **Static Pages**: All the routes for static pages, including `'/about'`, `'/contact'`, and `'/drivers'`, are defined to display their respective templates.

(k) **Cache Management**: To make sure that users always have access to the most up-to-date information, a `nocache` decorator is utilized to avoid caching certain views.

(l) **Insertion of Current Year**: The current year can be easily shown in footer sections or other relevant areas by having a context processor inject it into templates.

In Summary

Users will have a smooth experience navigating the site, managing their rides, and registering for trips thanks to the `routes.py` file, which connects user interactions to the backend database. It provides a solid foundation for the application's operations by efficiently managing user sessions, validating user inputs, and enforcing business logic.

3. **forms.py**: The TravelSathi application's `forms.py` file defines multiple forms using Flask-WTF, an extension for Flask that integrates WTForms with Flask applications. Forms for user registration, login, and ride posting are all part of this file, which helps with validation and managing user data.

Overview

Users can enroll in the program using the `RegistrationForm` class. It includes fields for the user's name, gender, email, password, and password confirmation. To prevent fields from being blank, validators like `DataRequired` are used, while `Email` ensures that the email format is correct. The `confirm_password` field contains an `EqualTo` validator to ensure that it matches the password field.

LoginForm Class

The `LoginForm` class serves a similar purpose in terms of user authentication. It includes the required fields for email and password. To ensure the data is valid, this form checks the input against defined validators before submission.

RideForm Class

The process of adding new rides is handled by the `RideForm` class. It asks for details like `start_location`, `end_location`, `meetup_point`, `departure_time`, `available_seats`, `cost_per_head`, and `gender_preference`. The TravelSathi app has a predefined list of valid location options from which the start and end locations are chosen. Validators are applied to ensure all required fields are completed by the user.

Custom Validation Methods

The `RideForm` class includes custom validation methods to enforce specific constraints:

- The `validate_end_location` method ensures that the start and end locations are distinct.
- The `validate_available_seats` method verifies that the number of available seats is greater than zero to avoid posting rides that are effectively unavailable.
- The `validate_cost_per_head` method checks that the cost per head is positive to ensure the ride is feasible.

Summary

Overall, the `forms.py` file effectively manages user input through structured forms, ensuring validation and proper handling of data before it is processed by the application. By making it clear when a user makes a mistake, this structure enhances the application's stability and user experience.

4. **models.py**: The TravelSathi application's `models.py` file uses the Flask-integrated SQLAlchemy to define the database models. In this file, the application's fundamental data structures—including those for users and rides—are defined, and the interactions between them are managed.

Overview

To handle the many-to-many link between users and rides, an association table called `ride_passengers` is constructed at the beginning of the file. The `user_id` and `ride_id` columns in this table are foreign keys that refer to the `User` and `Ride` models, respectively. The program can conveniently keep track of which users have joined which rides by using this table.

User Class

Application users are represented by the `User` class, which derives from the `UserMixin` class and offers methods for user authentication through Flask-Login. A number of fields are included in the model: `id`, `name`, `email`, `password`, and `gender`. To avoid the creation of duplicate accounts, we provide each user a unique identifier (`id`) and ensure that the email field is also unique. You can access the rides that the user has made through the `ride_posts` relationship, and you can view which rides the user has joined using the `rides_joined` relationship.

Ride Class

The `Ride` class represents the specifics of every ride. Details like `id`, `start_location`, `end_location`, `meetup_point`, `departure_time`, `available_seats`, `cost_per_head`, and `gender_preference` are all part of it. Just like in the `User` model, the `id` field is the primary key. To make sure that every ride is well defined

and has all the information that participants need, other fields record important ride facts. Connecting every ride to its creator, the `author_id` field forms a foreign key relationship with the `User` model. This guarantees that each ride may be linked to its original poster.

Summary

In general, the `models.py` file is the most important part of the TravelSathi application's data structure. The application runs smoothly because it maintains user and ride data effectively and facilitates interactions between the two. This organized method of modeling improves the data's readability and integrity, which in turn enables features like ride management and user identification.

5. **config.py:** Our TravelSathi app's `config.py` file handles all of the configuration options that are crucial to the app's functionality. In order to keep the codebase structured and clean, this file serves as a repository for all of the database setups and sensitive information.

   The `Config` class, which holds the configuration variables, is defined in this file. The string value `"SKIBIDI"` is set to one of the most important variables, `SECRET_KEY`. Flask manages sessions and protects against cross-site request forgery (CSRF) with this key. In a production setting, protecting session data from unauthorized access requires keeping this hidden.

   In order to find the application's directory, the `os` module is used to determine the `basedir` variable. To accomplish this, you need to call the function `os.path.abspath(os.path.dirname(__file__))`. This function returns the absolute path to the directory of the current file. With this method, the program may be simply deployed or relocated without affecting the path references, making it more flexible.

   Specifying the database connection string, the `SQLALCHEMY_DATABASE_URI` is the next configuration variable to set. Our database file is called `"database.db"` and it is created using SQLite. By dynamically creating the path to the database file in the same directory as the program, the path to this file is `os.path.join(basedir, 'database.db')`. Since SQLite eliminates the need to set up a separate database server, it is ideal for local programming.

   At last, we convert the variable `SQLALCHEMY_TRACK_MODIFICATIONS` to a false value. This option turns off Flask-SQLAlchemy's signal-emitting and object-tracking features, which aren't needed for most applications and can chew up resources.

   In general, the `config.py` file contains all of the TravelSathi application's configuration settings; this allows for quick access to and alteration of important parameters and promotes organization and security best practices. In addition to laying the groundwork for future application setting management, this approach aids in streamlining the development process.

6. **dockerFile:** A lightweight and efficient containerized environment is created by modifying the Dockerfile for the TravelSathi application. Using Docker, we ensure that the app operates consistently across all environments, simplifying deployment and dependency management.

Base Image

The Dockerfile begins with the instruction `FROM python:3.9-slim`, specifying the base image. This official Python image is based on Debian and has been designed to be compact, thereby reducing the container's overall footprint. It contains everything required to run Python programs, ensuring our code runs smoothly without compatibility issues.

Setting Working Directory

The next step is to use the `WORKDIR /app` directive to set the application's working directory. This command sets `/app` as the default location for all subsequent commands. Setting a working directory simplifies container file and dependency management.

Copying Application Files

After setting the working directory, the line `COPY . /app` transfers all files from the current directory (where the Dockerfile is located) into the container. This includes everything from the application's source code to configuration files.

Handling Dependencies

The Dockerfile then runs the command `RUN pip install -r requirements.txt` to manage dependencies. This command installs all Python packages listed in the `requirements.txt` file, which typically includes Flask, Flask-SQLAlchemy, Flask-Bcrypt, and others. Installing dependencies during build time provides the container with all the resources needed to run the application.

Port Exposure

The command `EXPOSE 5000` is used to expose port 5000 to the outside world. This enables users to access the Flask application from their web browsers.

Environment Variables

The Dockerfile defines an environment variable using `ENV NAME World`. Although this particular variable may not be directly used in the application, Docker environment variables are useful for configuration and can be referenced if necessary.

Application Start

Lastly, the `CMD ["python", "app.py"]` instruction specifies the command to be executed when the container starts. This command runs the `app.py` file, launching the Flask application.

Summary

The Dockerfile provides a containerized environment for the TravelSathi app, ensuring consistent performance across different platforms and enabling rapid deployment. By adhering to this approach, the application will function properly, regardless of the underlying infrastructure.

7. **about.html:** The TravelSathi project's "About Us" page template builds upon the `base.html` base HTML template to guarantee uniform styling and layout on all pages. To improve navigation clarity, the page has a block for defining the title (<`title`>`About Us`</`title`>), which will appear in the browser tab. The primary content block ({`% block body %`}) provides details on TravelSathi, including its goals and the people working on it. The Bootstrap container (<`div class="container mt-4"`>) that houses this content guarantees a responsive layout with uniform padding and alignment.

   The page presents the "Meet Our Team" section, which uses Bootstrap classes like `row` and `col-md-4` to display team member profiles in a grid layout. Each member of the team is represented by a card (<`div class="card mb-4"`>) that includes an image, name, and roll number taken from the static files directory. To keep the cards responsive and ensure they appear side by side on larger screens, they are grouped in columns (`col-md-4`). The page's visual appeal is enhanced by the use of the Bootstrap classes `card` and `mb-4`, which provide the card-style design and suitable margin spacing. This methodical approach facilitates easy maintenance of consistency across the website in addition to offering a clear and intuitive user interface.

8. **base.html:** Your TravelSathi project's fundamental layout is provided by this HTML template, which guarantees uniformity on all pages. The usual HTML5 doctype declaration comes first, followed by necessary meta tags for viewport configuration, character encoding, and browser compatibility. Along with custom CSS styles, the `<head>` section contains links to Google Fonts for a custom font (`Fredoka`), Font Awesome for icons, and Bootstrap for styling. The page has a distinct appearance thanks to the custom styles, which also include a sticky footer that remains at the bottom of the page and a darker, rounded-edge navbar.

   Using Bootstrap classes, the navigation bar (`<nav>`) in the `<body>` section allows for simple navigation throughout the TravelSathi site. Links to important sections such as Home, Create a Ride, My Rides, Find Rides, and Driver Contacts can be found in the navbar. Using Jinja templating, a conditional block ({`% if current_user.is_authenticated %`}) decides whether to show a dropdown menu with the user's email address and a logout link for authenticated users, or login/register options.

   Each page's primary content is entered using {`% block body %`}, which allows each page to specify its own content while still utilizing this template. Users may navigate with ease thanks to the footer (`<footer>`), which has a dark background that matches the navbar and contains links to the "About Us" and "Contact

Us" pages, as well as the current year. Lastly, JavaScript libraries such as jQuery, Popper.js, and Bootstrap JS are linked at the end to enable interactive features for elements such as the dropdown menu. The TravelSathi platform offers a unified, user-friendly experience thanks to this architecture.

Sure! Here's the provided text formatted similarly to the example you gave:

9. **contact.html:** To keep a uniform structure across the platform, we extended our standard HTML layout (`base.html`) to develop a "Contact Us" page template for TravelSathi. We have designated this page with a specific title, `<title>Contact Us</title>`, to make it easier for visitors to understand its function when navigating the site.

   To guarantee appropriate spacing and alignment, a container (`<div class="container mt-4">`) appears before the main content block, `{% block body %}`. To provide users with a clear introduction to the contact form, we incorporated a header (`<h2>Contact Us</h2>`) within the container. The "Name" (`<input type="text">`), "Email" (`<input type="email">`), and "Message" (`<textarea>`) fields make up the contact form. They are all marked as required to ensure complete submissions. Using Bootstrap's styling, each form element is wrapped in a `<div class="form-group">` to maintain the form's user-friendliness and organization.

   At the end of the form is a submit button (`<button type="submit" class="btn btn-primary">Submit</button>`), which is visually consistent thanks to Bootstrap classes. This button gives people an easy way to send in their questions. By keeping the design similar to other pages in the TravelSathi project, the `container`, `form-group`, and `btn btn-primary` classes contribute to a visually appealing, clean layout that complements the overall aesthetic of our platform, improving the user experience.

10. **create_ride.html:** In order to make sure that the whole TravelSathi platform has the same look and feel, we extended our base HTML layout (`base.html`) to create the "Create a Ride" page template for this project. We made sure that users could easily understand what they were doing by giving this page a specific title (`<title>Create a Ride</title>`).

    Start the main content block (`{% block body %}`) with a Bootstrap container (`<div class="container mt-3">`) that has a header (`<h1 class="text-center">Create a New Ride</h1>`) explaining the page's objective. The ride creation form is designed to securely transmit data to our server using a method set to "POST" (`<form method="POST" class="row g-3">`). The form contains fields like `start_location`, `end_location`, `meetup_point`, `departure_time`, `available_seats`, and `cost_per_head`. The form structure utilizes Flask-WTF, and using Jinja templating, each field is dynamically rendered with a class of `form-control form-control-sm` to keep everything neat and organized with Bootstrap's design.

11. **drivers.html:** For this project, we extended our base layout (`base.html`) to create the "Rickshaw Drivers" page template for TravelSathi. This ensures that the platform's visual style is consistent throughout. To make its function apparent to users, we gave the page the explicit title of "Rickshaw Drivers."

    A Bootstrap container (`<div class="container mt-4">`) introduces the list of dependable rickshaw drivers that users can contact in the content block `{% block body %}` with a header (`<h2>Rickshaw Drivers</h2>`).

In order to give readers more background and help them understand the page's goal, we included a descriptive paragraph.

Bootstrap provides a responsive grid framework that is used to display the list of drivers. A card (`<div class="card mb-4">`) is used to represent each driver's information, with the driver's name as the card title (`<h5 class="card-title">`). Each driver's specifics are documented in the card's body (`<div class="card-body">`), which includes their vehicle type, contact number, availability hours, and languages spoken (`<p class="card-text">`). Users will have no trouble locating the specific information they need due to this well-organized and aesthetically pleasing layout.

Authentic rickshaw drivers who work inside the college or are authorized to work on the campus—such as Rajesh Kumar, Suraj Singh, Anil Sharma, and Suresh Verma—have their contact details displayed on driver cards that we included. We optimized space and improved the user experience by displaying the drivers' cards in two columns using Bootstrap's grid system (`<div class="col-md-6">`). With this configuration, users can easily see all of the drivers and pick the one that works best for them.

The "Rickshaw Drivers" page is well-designed and organized, helping us achieve our goal of improving customer convenience. It gives users easy access to reliable and safe transportation options.

12. **index.html:** For this project, we extended the primary framework from `base.html` to build the `index.html` template for TravelSathi's home page, which ensures that the platform's layout and styling are consistent. With the title "Travel Sathi - Shared Ride Platform - Home," the page aims to guide people towards their desired destination.

    Using inline CSS within the `<style>` tags in the {`% block head %`} section, we altered the look of the navigation bar and buttons. The `.navbar` navigation bar has a black color scheme and white text links that turn yellow when you hover over them. Headers (`h2`) made using the `Comic Sans MS` typeface offer a distinct and approachable appearance, adding to the user-friendly interface.

    We gave the platform's navigation buttons unique styles by changing their font size, padding, and border-radius to make them stand out and be easy to click and drag. The buttons were given custom colors that were chosen to coordinate with the site's color scheme. These colors range from dark tones for the default state to slightly lighter shades for hover effects, creating a cohesive visual concept.

    Starting with the {`% block body %`} section, we made sure that users would receive instant feedback on their activities by displaying any flash notifications using Bootstrap alerts. In order to grab the user's attention, these messages are positioned in the middle.

    Greeting IIT Jodhpurians and highlighting that TravelSathi is a platform made by and for them is the main content's greeting header. Content changes dynamically based on the user's authentication status. Once the user has successfully authenticated, a dashboard will be displayed. It will provide options such as "Show My Rides," "Search Rides," and "Create a Ride." Styled buttons set centrally in a group (`.btn-group d-flex justify-content-center`) make each option accessible. A message prompting the user to either log in or register is displayed if they are not authenticated, along with links to the relevant pages.

By making sure that important tasks are readily available in an organized and aesthetically pleasing way, this home page design strives to offer a user-friendly experience. With an emphasis on students at IIT Jodhpur, our goal was to build an intuitive and inviting experience for all users.

% Add some vertical space between sections

joined_rides.html: This project involved developing the `joined_rides.html` template for the TravelSathi platform, which displays a user's joined rides. Consistent formatting throughout the program is achieved by expanding the base layout from `base.html` to structure the page.

Users are able to immediately discern the page's objective because of the descriptive page title, "Your Joined Rides," provided by the {`% block head %`} section.

Starting with a container (`.container`) that holds the page's primary content, we entered the {`% block body %`} section. The user is greeted with a header that clearly states what they will find on this page ("Your Joined Rides").

To keep the user informed of any changes or actions they have made, we integrated flash messaging with Bootstrap alerts. The {`% with messages = get_flashed_messages(with_categories=true) %`} block is used to process the messages in a way that guarantees feedback is given in a straightforward and brief manner.

A table detailing all of a user's rides is the page's primary feature. The table has a clean and organized look thanks to the Bootstrap classes (`table table-striped`). The table headers consist of "Start Location," "End Location," "Departure Time," "Available Seats," and "Action."

With the help of the `location_labels` dictionary, the template shows the following for every ride:

- The beginning and ending points, with human-readable names.
- The time of departure and the quantity of seats that can be reserved for the transportation.
- The user's options include "Opt Out" and "Delete Ride." Bootstrap classes (`btn btn-danger btn-sm`) style buttons on forms indicating these actions. To indicate that these are controls that will end the ride or the user's involvement, the buttons are small and red.

For users who haven't joined any rides yet, an informative alert (`alert alert-info`) is shown to let them know.

The goal in creating this template was to provide users with an easy way to manage joined rides. We attempted to make this platform portion user-friendly and efficient by offering clear visual feedback, easy-to-use options, and well-organized information.

13. **login.html:** The login page that users can utilize to safely access their accounts was built as part of our TravelSathi project. A basic HTML form is used for user authentication on this Flask-built page. In addition to a submit button, the form asks for an email address and a password. By making the layout adaptable and labeling each input field clearly, we made care to build an interface that is easy for users to navigate.

We included flash messages that let users know if there were any login issues or confirmations to enhance the user experience. To ensure these messages are seen by the user, they are displayed above the login form. Additionally, we have included a link that will take users to the signup page in case they do not currently have an account.

We put a lot of effort into the design since we wanted the login card to look good and be easy to find. For the purpose of element spacing and alignment, we made use of Bootstrap classes. The login form looks sleek and contemporary with the card arrangement, which makes it easy to navigate. Achieving a professional appearance that matches our platform's general design ethos was done by grouping the form elements within a card and centering it on the page.

Additionally, we integrated CSRF protection using Flask-WTF to guarantee the form's security, as this is critical in avoiding cross-site request forgery attacks. With `POST` as the form submission method, sensitive data is not exposed in the URL, adding an extra degree of security. This login page is an important first step for users to access the TravelSathi platform, and it prioritizes both ease of use and safety.

14. **register.html:** Our TravelSathi project's registration page is an integral part of the onboarding process for new users. Built with Flask, this website showcases an easy-to-use HTML form that requests basic information from users in order to create an account. The form is designed to be as secure as possible by including fields for the user's name, email, gender, password, and a confirmation password. The registration procedure is designed to be user-friendly with clearly defined entry fields.

    We integrated flash notifications that tell users of the registration status—whether successful or if there are errors—because we placed a strong emphasis on user feedback. With these messages prominently displayed at the top of the registration form, any concerns that may develop during the process may be communicated immediately and clearly. Also, we made it easy for users who already have an account to go to the login page by including a link for them.

    Using Bootstrap components for layout and decoration, the registration form is made to look visually appealing. Not only does the card format seem sleek and contemporary, but it also makes the form easy to use and fill out. We made it easy for customers to submit their details by positioning the card in the middle and keeping the spacing between elements consistent.

    In addition, we made sure security was our top priority by using Flask-WTF CSRF protection, which is essential for preventing cross-site request forgery attacks. Passwords and other sensitive information cannot be exposed in the URL since the form submission method is set to `POST`. In sum, this registration page exemplifies our dedication to form and function while serving as a crucial component of the TravelSathi platform, allowing for a safe and easy onboarding process for users.

15. **search_rides.html:** The search rides page is an essential part of our TravelSathi platform where users may see all the available rides. Guests may quickly choose transportation that suits their travel requirements thanks to this page's user-friendly interface. A structured table layout makes it easy to peruse the available rides according to important parameters like gender, cost per person, number of available seats, start and end locations, departure times, and more.

Using the Flask templating engine, we iteratively retrieve a list of rides from the server and populate the table dynamically. You can see the specifics of each ride laid out in a distinct row. By adding a localization function to the location labels, we were able to make the start and end locations more readable, which greatly improves the user experience. Users are never left in the dark when there is a clear notice that lets them know that there are no rides available.

The page has a conditional statement that verifies if there are seats available for every ride in order to make it easier to join them. Users can easily opt into the journey by clicking the "Join Ride" link that is displayed when seats are available. When a ride is fully booked, the interface will politely let you know that it is unavailable. Managing expectations is made easier with this feature, which also improves user interaction.

Designed to be both functional and aesthetically pleasing, the search rides page is an integral aspect of the TravelSathi experience. We work hard to make sure that riders can uncover and hop on routes that work for them by arranging information in a logical way and giving connections that people can actually use. Our dedication to developing a platform that puts users first and streamlines traveler connections is reflected on this website.