CAB_FARE WEBSITE REPORT

Problem Statement:

In India, numerous cab companies offer their services, but a prevalent issue persists: users often end up paying exorbitant fares despite multiple cab options available at the same price point. This dilemma arises because users are not inclined to download every cab-hailing app, such as Uber, Ola, Rapido, and others. Consequently, they remain unaware that they could save money by exploring different cab options. Additionally, users encounter challenges when attempting to book a cab, including unavailability or extended wait times for pickups.

Idea:

To address this challenge, we propose the development of a comprehensive cab booking platform. This platform will enable users to input their pick-up and drop-off locations, after which it will display all available cab options in real-time. Users will have the convenience of comparing prices, car models, and reviews across various cab services on a single platform. By streamlining the process of cab selection and facilitating informed decision-making, our platform aims to empower users to choose the most suitable option for their needs efficiently. Additionally, to mitigate issues of unavailability and lengthy wait times, our platform will provide users with a centralized website where they can access information on cab availability and make bookings seamlessly.

Technical Solution

By- Rishi.S and Aswathi Ajith

How can we effectively integrate Ola, Uber, Rapido, and other cab companies' APIs to fetch real-time data and display it on our website? To achieve this, we will follow a systematic process:

Research and Documentation: We will begin by thoroughly researching and understanding the documentation provided by each cab company regarding their API integration procedures.

API Integration: Once we comprehend the requirements outlined in the documentation, we will proceed to integrate the APIs of Ola, Uber, Rapido, and other relevant companies into our website's backend system.

Real-time Data Fetching: Through API calls, we will continuously fetch real-time data from the cab companies' servers, including information on available cabs, their locations, and estimated arrival times.

Data Display: Finally, we will develop front-end components on our website to dynamically display the fetched data, allowing users to view the real-time availability of cabs from multiple companies in one centralized platform.

Let's Understand API Segment Step by Step

Data acquisition and integration

Data acquisition and integration involve the process of gathering relevant data from various sources and integrating it into a unified system or database. In the context of developing an AI-based application for ride-hailing fare comparison, data acquisition entails obtaining fare rates, pricing structures, and real-time information from multiple ride-hailing companies. This may involve establishing partnerships with these companies to access their APIs (Application Programming Interfaces) or utilizing third-party data providers that offer access to such information. Integration, on the other hand, refers to the aggregation and consolidation of this data into a single cohesive platform. This involves developing algorithms and systems to collect, organize, and synchronize data from different sources, ensuring consistency and accuracy. Integration also involves mapping data fields from diverse sources to a standardized format, enabling seamless interoperability within the application. Additionally, data integration may involve cleansing and preprocessing the acquired data to remove duplicates, errors, or inconsistencies, thus ensuring the reliability and usability of the integrated dataset. Overall, data acquisition and integration lay the foundation for building a robust and comprehensive system that provides users with access to accurate and up-to-date fare information from multiple ride-hailing providers.

Data acquisition through APIs

Data acquisition through APIs (Application Programming Interfaces) plays a crucial role in collecting data from ride-hailing companies like Uber, Ola, and Rapido. Each of these companies provides APIs that developers can use to access various data points, including fare rates, ride availability, and real-time updates. Here are examples of how API collection facilitates data acquisition from these platforms:

1, Uber API

2.OLA

3. Rapido

1.Uber API:

Uber's API, specifically focus on ride requests and how developers can interact with it using various programming language. Authentication is typically done using OAuth 2.0, where developers obtain access tokens to make authorized requests on behalf of users. This ensures security and restricts access to authorized applications only. When we create an app with Uber, we will receive several credentials, including a server token, client ID, and client secret. These credentials are essential for authenticating our application and allowing it to interact with Uber's API on behalf of users.

In Uber Ride Request API and the Rides SDK, the SDK serves as a convenient wrapper around the API, providing developers with a higher-level interface for interacting with Uber's ride request functionality. Instead of manually crafting HTTP requests and handling low-level protocol details, developers can use the SDK's functions and methods to perform tasks such as requesting rides, retrieving ride details, ride status updates and handling responses in a more straightforward and intuitive manner. Uber Ride Request API, is the core API provided by Uber that allows developers to programmatically request rides, retrieve ride details, estimate fares, and perform other ride-related operations. The Ride Request API exposes a set of HTTP endpoints that developers can interact with to integrate Uber's ride-hailing services into their applications.

Overall, the combination of the Uber Ride Request API and the Rides SDK empowers developers to seamlessly incorporate Uber's ride-hailing functionality into their own applications, providing users with the convenience of booking rides directly from within the app. This integration enhances the user experience and extends the reach of Uber's services to a broader audience through third-party applications.

I. Creating an Uber session with a server token

The server token specifically allows our application to authenticate itself with Uber's servers when making API requests. Unlike user-specific tokens, such as access tokens obtained through OAuth authentication, the server token is used to authenticate requests made from our application's backend or server-side code. This means that endpoints that don't require user-specific context can be accessed directly from our application server using the server token.

i. GET /products:

This endpoint returns a list of available Uber products in each location, when users need to know what types of Uber services are available to them in their current location. By accessing this endpoint, developers can retrieve information about the various Uber products available, such as UberX, Uber Pool, Uber Black, and their respective features, vehicle types, capacity, and pricing details. This

allows users to choose the most suitable service for their needs and preferences. It also provides information about each product, such as its display name, description, capacity, and pricing details.

```
from uber_rides.session import Session
from uber_rides.client import UberRidesClient

session = Session(server_token=<TOKEN>)
client = UberRidesClient(session)
```

Get a list of available products

```
response = client.get_products(37.77, -122.41)
products = response.json.get('products')
```

ii. GET /estimates/time:

This endpoint provides estimates for the time it will take for an Uber vehicle to arrive at a given location when users want to know how long they'll have to wait for an Uber ride to arrive at their pickup location. This endpoint helps users plan their trips by providing estimates of arrival times based on factors such as current traffic conditions, driver availability, and distance to the pickup location. It allows users to make informed decisions about when to request a ride and helps them avoid unnecessary wait times. It considers factors such as current traffic conditions, driver availability, and distance to the pickup location.

Get price and time estimates

```
response = client.get_products(37.77, -122.41)
products = response.json.get('products')
```

```
response = client.get_price_estimates(
    start_latitude=37.770,
    start_longitude=-122.411,
    end_latitude=37.791,
    end_longitude=-122.405,
    seat_count=2
)

estimate = response.json.get('prices')
```

iii. GET /estimates/price:

This endpoint provides estimates for the price of an Uber ride between two locations. When users need to know how much an Uber ride will cost before they book it. This endpoint returns estimated fares for each available Uber product based on factors such as distance traveled, time taken, and any applicable surge pricing. It helps users' budget for their trips and compare prices between different Uber products, enabling them to choose the most cost-effective option. Additionally, knowing the estimated price in advance can help users avoid surprises and plan their transportation expenses accordingly.

To use these endpoints, our application's backend would make HTTP GET requests to the corresponding URLs, including the necessary parameters such as location coordinates or ride details. These requests would include the server token in the request headers to authenticate the application with Uber's servers. Once authenticated, Uber's servers would process the requests and return the relevant data, which our application can then use to provide users with information about available Uber products, estimated time to arrival, and estimated ride prices.

iv <u>Authorize a rider for our application:</u>

```
from uber_rides.auth import AuthorizationCodeGrant
auth_flow = AuthorizationCodeGrant(
    <CLIENT_ID>,
    <SCOPES>,
    <CLIENT_SECRET>,
    <REDIRECT_URI>
)
auth_url = auth_flow.get_authorization_url()
```

i. <u>Obtaining OAuth 2.0 Credentials:</u>

Developers need to obtain OAuth 2.0 credentials (client ID and client secret) from the Uber Developer Dashboard. These credentials are used to authenticate the application and authorize it to access user data on behalf of the rider.

ii. <u>Authorization Request:</u>

When a rider wants to use the application's features that require access to their Uber account (such as requesting a ride or viewing trip history), the application initiates the OAuth 2.0 Authorization Code flow. The application constructs an authorization URL (auth_url) with the appropriate parameters, including the client ID, redirect URL, and requested scopes (permissions). The rider is redirected to the authorization URL, where they are prompted to sign in to their Uber account and grant the requested permissions to the application.

iii. <u>Authorization Grant:</u>

After signing in and granting access, the rider's browser is redirected back to the specified redirect URI (redirect_url) with an authorization code appended to the URL. The application extracts the authorization code from the URL and uses it to obtain an access token from the Uber API.

iv. <u>Token Exchange</u>:

The application exchanges the authorization code for an access token by making a token request to the Uber API's token endpoint. Along with the authorization code, the token request includes the client ID, client secret, redirect URI, and grant type. If the token request is successful, the Uber API responds with an access token that the application can use to make authorized API requests on behalf of the rider.

v. Token Management:

The application securely stores the access token and associated credentials in a secure data store. The SDK automatically handles token refreshing when making API requests using the UberRidesClient, ensuring that the access token remains valid and up-to-date. Storing credentials securely helps prevent riders from needing to repeat the authorization process each time they use the application.

In summary, the OAuth 2.0 Authorization Code flow enables applications to authenticate with the Uber API and obtain access tokens to access user data with the rider's consent.

II. Fetch a user's profile and activity

We can also fetch the profile of a user and the user's activity using the set of codes as,

Fetch a user's profile

```
response = client.get_user_profile()
profile = response.json

first_name = profile.get('first_name')
last_name = profile.get('last_name')
email = profile.get('email')
```

Fetch a user's activity

```
response = client.get_user_activity()
history = response.json
```

Ride Requests

To request an Uber ride for a rider through the Uber API, developers follow a sequential process. Initially, they retrieve available Uber products in the rider's location using the GET /products endpoint, ensuring that the upfront_fare_enabled field is set to true to enable upfront fare estimation. Upon selecting a desired Uber product, developers utilize the POST /requests/estimate endpoint to request a fare estimate for the trip, specifying the corresponding product_id. The response includes a fare_id, which is utilized in the subsequent step to set an upfront fare and arrival time for the trip. With the obtained fare_id, developers proceed to request a ride for the rider using the POST /requests endpoint, including the fare_id in the request payload to establish the upfront fare. The Uber API processes the request, assigns a driver to the trip, and facilitates the ride booking. In case of an expired or invalid fare_id, the API returns a 422 error, ensuring proper handling of such scenarios within the application's logic. Through this API-driven approach, developers seamlessly integrate ride-booking functionality into their applications, enhancing user convenience and experience.

III. The Sandbox

Request a ride

The Uber API offers a sandbox environment for testing ride request endpoints without affecting drivers and riders on the live platform. Requests made to the sandbox simulate ride requests, allowing developers to programmatically update them. To set up testing environments for making ride requests, developers can refer to the sandbox guide provided by Uber. During various stages of a trip, it's essential to provide riders with clear trip details to enhance their experience. For instance, when a request is in the processing state, indicating that Uber is attempting to find a driver, displaying a spinner or loading indicator effectively communicates this to the rider. Once a driver has accepted the request, informing the rider that a vehicle is en route is beneficial. Additionally, providing reminders of the pickup location, displaying driver details, or showing the driver's location on a map can be helpful. As the request status changes on arrival, it's important to notify riders in a noticeable manner. Once the trip is in progress, offering useful information for riders while they're in the vehicle enhances their journey. If a driver cancels a request, informing the rider promptly allows them to request another driver efficiently. Furthermore, enabling easy communication between riders and drivers within the app, such as providing options to call or send SMS messages, ensures smooth coordination and a seamless experience for all parties involved.

The possible status values are:

Status	Description
`processing`	Uber is matching the rider and driver.
`no_drivers_available`	The request was unfulfilled because no drivers were available.
`accepted`	The driver accepted the request and is enroute to the start location.
`arriving`	The driver arrived or will shortly.
`in_progress`	The trip is in progress from the start location to the end location.
`driver_canceled`	The driver cancelled the request.
`rider_canceled`	The rider cancelled the request.
`completed`	The trip is completed by the driver.

For reference, here is the link for the uber- **click here**

2. Ola API:

I. Ride Availability/Estimate

Like Uber, the Ola API enables developers to fetch fare estimates for various ride options offered by Ola, such as Ola Micro, Ola Mini, and Ola Prime. The documentation gives an overview of the ride estimation feature, explaining its importance in providing users with upfront information about the expected cost of a ride before booking. Parameters such as pickup location, drop-off location, ride type, vehicle category, are included in the request to customize the ride estimation process. With the Ola API, developers can query the availability of rides within a specific area or time frame, helping users find the nearest available ride. The Ola API allows developers to initiate ride bookings and handle ride cancellations programmatically, streamlining the booking process for users.

The endpoint, GET /v1/products, allows developers to access real-time information about Ola rides available at a given user location. By providing the latitude and longitude coordinates of the user's location, developers can retrieve the following information, such as;

- Obtain details about the different ride categories available nearby, such as Ola Prime, Ola Micro, Ola Mini, etc.
- Estimated Time of Arrival (ETA): To get an estimate of how long it will take for a ride to arrive at the user's location based on current traffic conditions and driver availability.
- Ride Cancellation Policy: To access information about the policy regarding ride cancellations, including any associated charges or penalties.
- Rate Card and Fare Details: To retrieve details about the fare structure for each ride category, including base fare, per kilometer charge, and per minute charge.
- Peak/Lean Pricing: To determine whether peak or lean pricing is currently applicable, which may affect the fare rates during certain times or in specific areas.
- Exact Locations of Nearby Rides: To receive information about the precise locations of nearby rides available for booking.
- Previous Ride Cancellation Charges: To check for any previous ride cancellation charges that may need to be paid before booking the current ride.
- Hotspot Zones and Designated Pick-Up Points: To determine if the pickup location lies within a hotspot zone and access information about designated pick-up points applicable to each ride category.

Additionally, if the desired drop location of the user is provided, developers can obtain the following additional information:

- **Ride Estimate:** To receive an estimated fare range for the ride from the user's current location to the desired drop location.
- **Upfront Price:** In select cases with specific partners, developers can access upfront pricing information for the ride, providing users with transparency regarding the cost before booking.

II. Cab Ride Booking

This gives an overview of the cab booking feature provided by Ola Cabs API, outlining its importance in facilitating seamless ride booking for users. Parameters such as pickup location, drop-off location, ride type, vehicle category, payment details, and any other relevant details are included in the booking request to customize the ride booking process. Details about the format of the response returned by the API endpoint(s) after a booking request is made typically includes information about the structure of the response data, such as the fields and values returned, and how to interpret them. Information about potential errors that may occur when making booking requests to the cab booking endpoint(s) includes details about error codes, error messages, and recommended troubleshooting steps.

Once the user selects their preferred ride category and confirms the booking, the API for booking creation is called. Upon receiving the correct request, the API generates a booking ID and sends it in the response to confirm that the booking has been initiated. After the booking is created, the system starts searching for nearby cabs to assign to the booking. There are two possible outcomes:

1) **Successful Allotment:** If nearby cabs and drivers are available, one of them is allotted to the booking, and the user is notified accordingly.
2) **Stockout:** If there are no cabs available due to non-availability, the system notifies the user about the stockout situation, indicating that a cab cannot be allocated now.

After receiving a successful response from the booking creation API, the next step is to keep polling the Track Ride API to obtain updates on the status of cab allotment. The response from the Track Ride API will contain a booking_status field indicating one of the following statuses:

ALLOTMENT_PENDING: This status indicates that the system is still searching for cabs to be allotted to the booking. While the status is ALLOTMENT_PENDING, users on the app can be shown a loading screen, indicating that the system is still searching for a cab.

CALL_DRIVER: When the status is CALL_DRIVER, it means that the system has successfully allotted a cab, and the response will also contain cab/driver details. Upon receiving this status, the cab/driver/OTP details can be displayed on the user's screen, indicating that a cab has been successfully allocated.

ALLOTMENT_FAILED: If the status is ALLOTMENT_FAILED, it indicates that the system could not locate any cab nearby for allotment. In this case, users can be notified that the booking could not be completed due to non-availability of cabs, and they may be prompted to try booking again later.

By continuously polling the Track Ride API and handling the different booking statuses appropriately, developers can provide users with real-time updates on the progress of their booking and ensure a smooth booking experience within the app.

III. Cab Ride Tracking

This gives an overview of the ride tracking feature offered by the Ola Cabs API, explaining its significance in providing users with real-time updates on the status of their booked rides. Details about the specific API endpoint(s) related to ride tracking, including the endpoint URL(s), HTTP method(s) (e.g., POST), and any required parameters or headers are needed to track a ride. Parameters such as the booking ID or ride ID can be included in the tracking request to specify the ride to be tracked. Details about the format of the response returned by the API endpoint(s) after a tracking request is made typically includes information about the structure of the response data, such as the fields and values returned, and how to interpret them. Explanation of the different status values that may be returned in the response to indicate the current status of the tracked ride include statuses such as ALLOTMENT_PENDING, CALL_DRIVER, ALLOTMENT_FAILED, etc. Overall, the ride tracking documentation provides developers with all the necessary information and resources to implement ride tracking functionality into their applications using the Ola Cabs API. It covers everything from making tracking requests to handling responses and dealing with different ride status scenarios, ensuring a seamless and reliable ride tracking experience for both developers and end users. Once a booking is created, user can get to know status of the ride. The API will give different responses based on the state of the booking.

SOFT ALLOCATED

This is an optional service that can be given to the user, and this booking_status will be available only for specific type of booking locations(like airport) where a specific cab won't be allotted instead an OTP will be provided in response and the user need to go the Ola Zone and he/she can onboard any available cab of the booked car category.

Few of the example responses are;

- ALLOTMENT PENDING (Driver is yet to be allotted)
- CALL DRIVER (Driver is on the way to pick-up location)
- CLIENT LOCATED (Driver reached the pickup location)
- IN PROGRESS (Ride started and on-going)
- COMPLETED (Ride ended and bill generated)
- CANCELLED (Ride cancelled)
- ALLOTMENT FAILED (Driver could not be allotted to the booking)

IV. Ride cancellation

Request Parameters

Name	Data Type	Description	Type	Remark
booking_id	string	Reference ID for the ride sent in booking create response	Payload	Mandatory
reason	string	Reason selected by user	Payload	Mandatory
Authorization	string	Authorization Token identifies the user	Header	Mandatory
X-APP-TOKEN	string	Key which identifies the partner	Header	Mandatory

This gives an overview of the ride cancellation feature provided by the Ola Cabs API, outlining its importance in allowing users to cancel booked rides when necessary. Details about the specific API endpoint(s) related to ride cancellation, including the endpoint URL(s), HTTP method(s) (e.g., POST), and any required parameters or headers needed to cancel a ride. Information about the parameters that can be included in the cancellation request to specify the ride to be canceled. This may include parameters such as the booking ID or ride ID. Details about the format of the response returned by the API endpoint(s) after a cancellation request is made. This typically includes information about the structure of the response data, such as the fields and values returned, and how to interpret them. Explanation of the cancellation policy enforced by Ola Cabs, including any associated charges or penalties that may apply when canceling a ride. This information helps users understand the implications of canceling a ride. Information about potential errors that may occur when canceling rides and how to handle them gracefully. This includes details about error codes, error messages, and recommended troubleshooting steps. Sample code snippets demonstrating how to make a cancellation request to the ride cancellation endpoint using various programming languages (e.g., URL, Python, Java) and how to handle the response data.

Ride Cancellation Reasons

Name	Data Type	Description	Type
category	string	ex: mini (valid are micro/mini/sedan/prime/lux/suv/share/rental)	Payload
pickup_lat	float	The latitude part of the pickup location.	Payload
pickup_lng	float	The longitude part of the pickup location.	Payload
Authorization	string	Authorization Token identifies the user	Header
X-APP-TOKEN	string	Key which identifies the partner	Header

The API endpoint, GET /v1/bookings/cancel/reasons, allows users to retrieve a list of acceptable reasons for canceling a ride. Using this API, it provides users with a comprehensive list of acceptable reasons for canceling a ride. This ensures that users can choose an appropriate reason when canceling their ride, providing valuable feedback to the service provider. The API may accept parameters such as the pickup location (latitude and longitude) to tailor the list of cancel reasons based on the user's location. Additionally, for rental categories, the sub-category (e.g., mini) may also be provided to further refine the list of reasons. The API returns a response containing the list of acceptable cancel reasons. Each reason may be accompanied by additional metadata, such as a unique identifier or a description, to help users understand the nature of each reason. The API dynamically generates the list of cancel reasons based on various factors such as the ride category, pickup location, and sub-category (for rental categories). This ensures that users are presented with relevant reasons that align with their specific context. Users can utilize this API to populate a dropdown menu or a list within their application's user interface, allowing users to select a reason from the provided list when canceling their ride. This enhances the user experience by streamlining the cancelation process and providing users with clear options. Overall, the GET /v1/bookings/cancel/reasons API plays a crucial role in facilitating the ride cancelation process by empowering users to provide meaningful feedback through a structured and predefined set of cancel reasons.

For further reference, here is the link- Click here

3. Rapido API:

The Rapido API also provides endpoints to calculate fares for bike rides based on factors such as distance traveled, time taken, and surge pricing conditions. Developers can leverage the Rapido API to track the status of ongoing rides, including the current location of the rider and the estimated time of arrival. Through the Rapido API, developers can retrieve feedback and ratings provided by users for completed rides, allowing users to make informed decisions based on the experiences of past customers.

Conclusion

By integrating these APIs into the AI-based application, developers can aggregate fare rates, ride availability, and other relevant data from Uber, Ola, Rapido, and other ride-hailing companies. This integration enables users to compare fare options across multiple platforms, access real-time updates, and make informed decisions when booking rides.

Requirements

By-Shambhavi.B

To develop this website, we need various resources and tools at our disposal. Firstly, we require a skilled development team proficient in web development languages such as HTML, CSS, JavaScript, and frameworks like React or Angular. Additionally, we need access to server infrastructure and databases to store and retrieve real-time data efficiently. In terms of devices, developers will need access to computers or laptops with internet connectivity for coding and testing purposes. Furthermore, a budget allocation for hosting services, domain registration, and any potential subscription fees for accessing cab companies' APIs is necessary. With these resources in place, we can proceed with the development of the cab booking platform.

Let Us Understand This Step by Step: -

1. Tech Team:

- **Roles and Responsibilities:**
  - **Mobile App Developers:** Responsible for building the front-end interface and implementing app functionalities.
  - **UI/UX Designers:** Design visually appealing and user-friendly interfaces for the app.
  - **Backend Developers:** Develop the server-side logic, database management, and integration with external APIs.
  - **Skills Required:** Proficiency in programming languages (e.g., Swift, Kotlin, Java), mobile app development frameworks, UI/UX design principles, RESTful APIs, database management systems (e.g., Firebase, MongoDB).
  - **Team Size:** A team of [insert number] developers, designers, and backend engineers.

2. Website Features:

- **Price Comparison:** Provide users with real-time fare comparisons from multiple cab service providers.
- **Booking Functionality:** Enable users to book rides directly from the app, with options to select preferred providers, vehicle types, and pickup/drop-off locations.
- **User Profiles:** Allow users to create profiles, save favorite destinations, view ride history, and manage payment methods.
- **Payment Integration:** Integrate secure payment gateways (e.g., Stripe, PayPal) for seamless and secure transactions.
- **Location Services:** Utilize GPS functionality for accurate location tracking, route optimization, and navigation.

3. Data Sources:

- **API Integration:** Integrate APIs from cab service providers (e.g., Uber, Lyft, Ola) to fetch real-time pricing data, availability, and ride options.
- **Database Management:** Implement a scalable and secure database system to store user profiles, ride history, and app data.

4. User Interface:

- **Intuitive Design:** Design a visually appealing and intuitive interface that prioritizes ease of use and seamless navigation.
- **Responsive Layout:** Ensure the app is compatible with various screen sizes and devices, including smartphones and tablets.

5.Advertising:

- **Platforms:** Utilize digital marketing channels such as social media advertising, Google Ads, and app store optimization (ASO) to increase app visibility.
- **Target Audience:** Tailor advertising campaigns to reach potential users based on demographics, location, and user behavior.
- **Budget Allocation:** Allocate a portion of the budget for advertising expenses, with flexibility for adjustments based on campaign performance and ROI.

6. Rights:

- **Data Usage:** Ensure compliance with data privacy regulations (e.g., GDPR, CCPA) and obtain user consent for data collection and usage.
- **Licensing:** Obtain necessary licenses or agreements for using third-party data sources, APIs, and content.

7. Government Permission:

- **Regulatory Compliance:** Ensure compliance with local transportation regulations, licensing requirements, and data privacy laws.
- **Legal Documentation:** Obtain permits, licenses, or approvals required for operating the app in specific regions or jurisdictions.

8. Expenses:

- **Development Costs:** Budget for salaries, contractor fees, and development resources required for app development.
- **Infrastructure:** Allocate funds for server hosting, cloud services, and API usage fees.
- **Marketing Expenses:** Plan for advertising costs, app store optimization services, and promotional activities to attract users.

- **Risks and Mitigation Strategies:**
    - Identify potential risks such as technical challenges, market competition, regulatory hurdles, and budget constraints.
    - Develop mitigation strategies to address each risk, including contingency plans, alternative solutions, and proactive risk monitoring.

Conclusion:

The CabPriceCompare Website represents an innovative solution for users seeking affordable and convenient transportation options. By leveraging cutting-edge technologies and a user-centric design approach, the website aims to revolutionize the way people book and compare cab rides. With careful planning, execution, and ongoing optimization, the project is poised for success in meeting user needs and achieving business objectives.

APPROX BUDGET ANALYSIS

By-Shambhavi.B

Creating an accurate budget for a website involves considering various factors such as the complexity of features, team size, development duration, and other associated costs. Here's an approximate breakdown of budget allocation for developing the CabPriceCompare Website:

1. Development Costs:

- **Team Salaries:** Assuming a team of developers, designers, and backend engineers, with salaries ranging from INR 30,000 to 1,00,000 per month per team member, for a period of 6 to 12 months. This could range from INR 18,00,000 to 36,00,000.
- **Development Resources:** Budget for tools, software licenses, and development kits, ranging from INR 2,00,000 to 7,00,000.
- **Contingency:** Allocate around 10-20% of the total development costs for unforeseen expenses, ranging from INR 2,00,000 to 7,00,000.

2. Infrastructure:

- **Server Costs:** Budget for server hosting fees and cloud services, ranging from INR 2,00,000 to 10,00,000.
- **API Usage Fees:** Allocate funds for API usage fees, varying based on the number of API calls and pricing models. This might range from INR 1,00,000 to 5,00,000.

3. Marketing Expenses:

- **Advertising:** Allocate a portion of the budget for digital marketing campaigns, ranging from INR 5,00,000 to 25,00,000.
- **Promotional Activities:** Budget for promotional materials, events, and partnerships, ranging from INR 2,00,000 to 10,00,000.

4. Miscellaneous Expenses:

- **Legal Fees:** Budget for legal consultation, documentation, and compliance, ranging from INR 1,00,000 to 5,00,000.
- **Office Costs:** If renting office space, consider expenses such as rent, utilities, and supplies. Depending on the location and space requirements, this could range from INR 2,00,000 to 10,00,000 per year.

Considering the above breakdown, the total budget for developing and launching the CabPriceCompare Website, including office space rent, could range from approximately INR 35,00,000 to 1,00,00,000 or more, within lakhs.

Please note that these are rough estimates, and actual costs may vary based on specific project requirements, team composition, market conditions, and location-specific factors.

BUSINESS MODEL

By – Rishi.S and Kartik.S

Business Model Monetization Approach:

Our revenue strategy involves options of subscription plans and commission-based earnings from cab bookings. Subscriptions offer users perks like priority booking, while commissions are earned from each successful booking. We also explore partnerships with other transportation providers to diversify income streams and enhance user experience.

1. BUSSINESS IDEAS

To sustain a business model, it is necessary to have monetization on the website or application, so some of ideas for monetization in cab fare compare business are:

1.1 ADS CHARGING MODEL EXPLANATION:

Implementing an ads charging model involves offering advertising space within the application to third-party businesses. This can include display ads, sponsored listings, or targeted promotions tailored to user preferences and behaviour. By partnering with relevant advertisers, such as local businesses, restaurants, or travel agencies, we can generate revenue through ad placements. Advertisers pay for exposure to our user base, while users benefit from discovering relevant services or deals within the app.

1.2 COMMISION FROM CAB BOOKINGS:

A key revenue stream for our platform involves charging a commission fee for every cab booking made through our application. Upon successful completion of a ride booked via our platform, we can levy a commission ranging from 0.5% to 1.5% of the total fare. This commission model incentivizes cab companies to collaborate with us, as they gain access to a larger customer base and increased bookings. Additionally, after establishing our platform's credibility and stability over the first year, we can negotiate higher commission rates with cab companies, ranging from 1.5% to 3% of the total fare amount, thus enhancing our revenue potential.

1.3 SUBSCRIPTION MODEL:

We can have a subscription model where the user of over application will pay in advance for a few months, or for a year to enjoy some beneficial premium services like:

- **USER INPUT AND SCHEDULING**

  Users can set up recurring trips by specifying Pick-up and Drop-of locations (e.g., home to office), preferred travel days and times, car type preferences (e.g., sedan, hatchback).

- **SEAMLESS PAYMENT**

  Payments would be automatically deducted from the user's linked payment method within the app, eliminating the need for manual transactions during each ride.

- **AUTO-BOOKING**

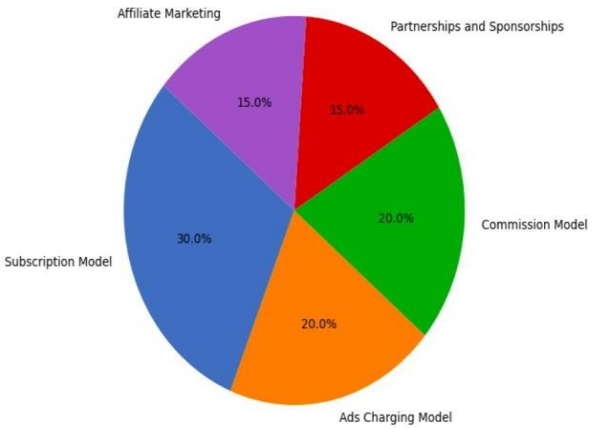
  The platform would monitor cab fares from various providers in real-time based on the user's preferences. When a ride falls within the user's scheduled timeframe and the fare from the cheapest provider meets their budget (which can also be set by the user), the platform automatically books the cab. Users would receive notifications about confirmed bookings with details like driver information, estimated arrival time, and fare.

- **GUARANTEED AVAILABILITY**

  For an additional fee within the subscription, users could opt for guaranteed cab availability during their scheduled trips. This would be useful in high-demand periods.

1.4 PARTNERSHIPS AND SPONSERSHIPS:

Forge strategic partnerships with brands, events, or local businesses to sponsor rides or offer co-branded promotional campaigns. Partnering with events, festivals, or conferences to provide sponsored rides to attendees can enhance brand visibility and generate sponsorship revenue. Additionally, collaborate with corporate clients or travel agencies to offer specialized ride-hailing services tailored to their needs, such as employee transportation solutions or travel packages.



DATA COLLECTION

For conducting market segmentation analysis in the cab industry to understand the importance of cab fare compare in future, a dataset was collected from Kaggle. The dataset contains information on 154235 cab bookings and includes the following 16 features:

- **ID** - Unique Identifier
- **vendor_id** - Taxi data providing vendor; 1 = TaxiTech Inc. 2 = DataCollectors Inc.
- **pickup_loc** - Location ID from where passenger was picked up
- **drop_loc** - Location ID where passenger was dropped
- **driver_tip** - Tip given to driver
- **mta_tax** - Automatically triggered tax amount
- **distance** - Distance covered in the trip
- **pickup_time** - Date/Time when meter started
- **drop_time** - Date/Time when meter stopped
- **num_passengers** - Cab passenger count
- **toll_Amt** - Toll paid in the booths
- **payment_Method** - Method of payment symbolised by a numeric code (1 = Credit Card, 2 = Cash, 3 = Free ride, 4 = Disputed, 5 = Unknown, 6 = Void trip)

- **rate_code -** Rate code for the trip (1 = Standard, 2 = Airport, 3 = Connaught Place, 4 = Noida, 5 = Negotiated Fare, 6 = Pooled ride)
- **stored_flag** - Flag which signifies whether trip data was immediately sent to Chh-OLA's database or not (Y=Yes, N=No, because of connection error)
- **extra_charges -** Miscellaneous charges
- **improvement_charge -** Charge levied for improvement in infrastructure

The dataset provides a comprehensive overview of customer preferences on cabs, and their needs to book a cab as well.

EDA PROCESS and Analytical Insights

By- Rahul Moolchandani

Through comprehensive market research, we have gathered valuable insights into user preferences, cab availability, and pricing dynamics. These insights have been meticulously analyzed and visualized through graphs and charts, providing a clear understanding of market trends and customer behavior. Through continuous monitoring and analysis of market trends, we remain agile and responsive to evolving customer needs, ensuring our platform remains competitive and relevant in the dynamic cab services industry.

Here is the complete EDA process, you can check out: - **Click here**

Coding on Market Segmentation

By- Kartik.S and Rahul. M

In coding for market segmentation analysis, various techniques and algorithms are employed to classify and group customers or market segments based on specific characteristics or behaviors. This involves preprocessing and cleaning of data, followed by the application of clustering algorithms such as K-means, hierarchical clustering, or density-based clustering to identify distinct groups within the dataset. Additionally, classification algorithms like decision trees or logistic regression may be used to predict segment membership based on input features. The chosen approach depends on the nature of the data and the objectives of the analysis. Once segments are identified, further analysis, such as profiling and interpretation of segment characteristics, is conducted to understand the unique needs and preferences of each group, enabling targeted marketing strategies and personalized product offerings.

Here is the Coding you can check- Click Here