Package: Idea

- Shipment broker.
- Travelers from A to B and have extra space on them can utilize that to ship other user's (documents, shipments) who also want to send those from A to B.

Application Terminology

- Journey: A journey form A to B that belongs to a user (who has published said journey) which also has a date that it should occur on.
- Origin and Destination: A and B in journey (locations).
- User: a user of the app and could be a traveler, someone who wants to ship something or both.

Context

- Location (If user permits the location (credit to Dominik Florencki)) :
  - Trips will be shown in order relevant to the distance between the user’s position and the traveler who created the trip if location allowed. **Otherwise, the trips are sorted relevant to creation date**.
- Network Connection:
  - Used to determine whether user is in online/offline mode.
- End-user filter rules:
  - Filter trips data according to trip attributes

Filters
Application Data adaptation mechanism

- Filters are applied for journey queries upon end-user input context detection and parameters. Journeys that do not match parameters are filtered out. However, if that results in too few journeys (<=10) additional journeys are kept as suggestions and they’re determined based and relaxing the parameters
- Filter parameters:
  - Date: Filter trips according specific trip date +- tolerance period to add suggestions.
  - Origin – Destination: Return journeys going from A to B only (journeys going from nearby to A or ending close to B are candidates for suggestions)
  - Available weight: Return journeys that can accommodate my large shipment.

Sorting journey queries (Application Data adaption mechanism):

- After journeys are filtered, we have two sets of journeys; matching and suggestions. Those are returned in order of relevance according to:
  - Matching journeys are always before suggestions.
  - Date (duration between journey’s date and required date) <ascending>
  - Origin and destination distance of journey to the required origin and destination <ascending>
  - Distance between the journey user and the requesting user (usually relevant when there isn’t an origin parameters) <ascending>
- Any of the previous rules is skipped if the corresponding parameter was not supplied.

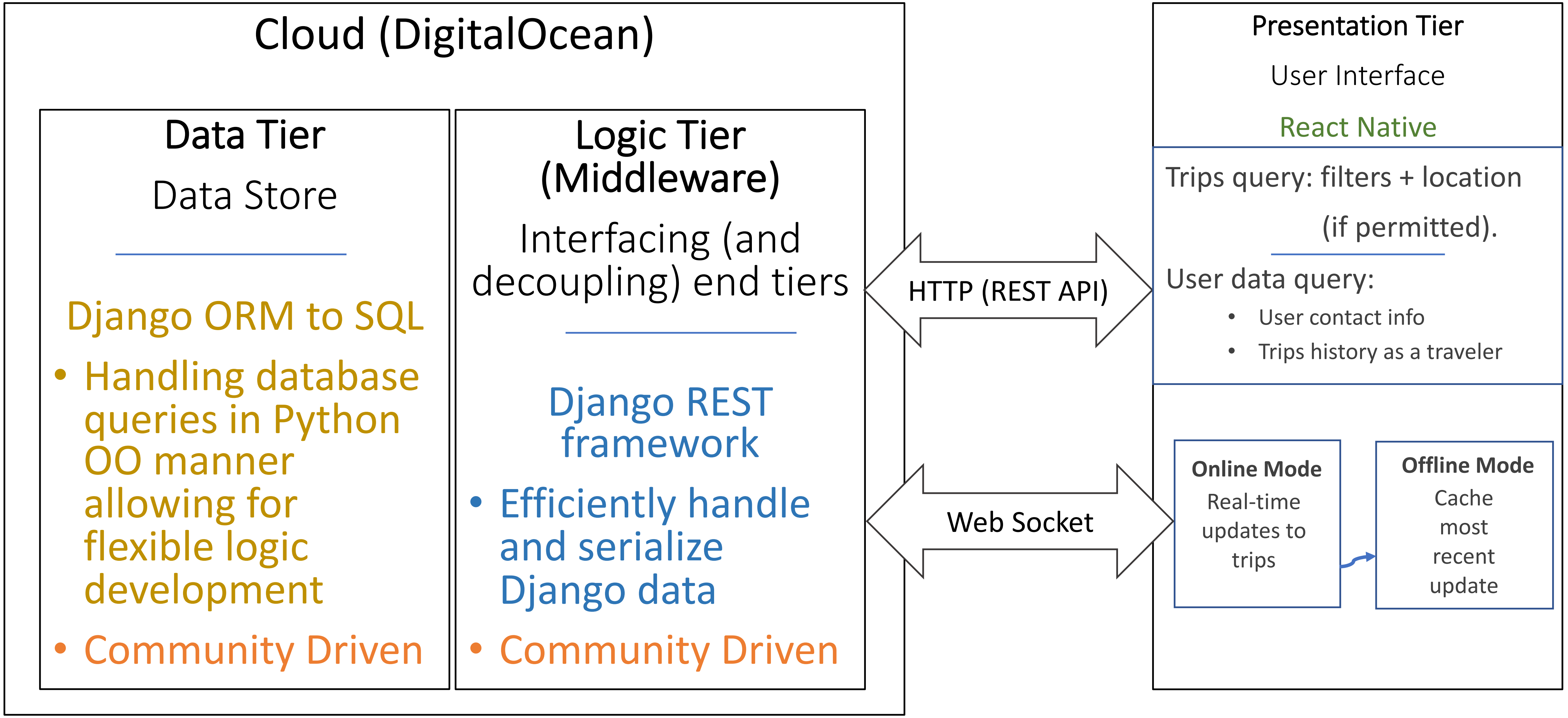
Offline usage
Data Transmission adaption mechanism

- Online mode:
  - User gets real time updates (list is updated once a new trip is created)
  - User can refresh list with list refresh gesture
- Offline mode:
  - When user is offline, offline cache is used where it contains the result of the the original query (but shown according to currently applied filters), which can be further filtered offline.
  - Publish queue so user can publish a journey offline (queue publish requests), obviously it would only be available to others when user is online again.
  - A snack-bar shown to the user to indicate connection status.

Limiting power consumption

- GPS location is the only actual threat to battery drain. So to limit battery usage, location is only computed (if permitted) at key points:
  - When user first starts the app
  - After each time the app is minimized and returned to again.

Architecture (Tiers) and Technologies


Cloud (DigitalOcean)
Data Tier
Data Store
Django ORM to SQL
• Handling database queries in Python OO manner allowing for flexible logic development
• Community Driven
Logic Tier (Middleware)
Interfacing (and decoupling) end tiers
Django REST framework
• Efficiently handle and serialize Django data
• Community Driven
HTTP (REST API)
Web Socket
Presentation Tier
User Interface
React Native
Trips query: filters + location (if permitted).
User data query:
• User contact info
• Trips history as a traveler
Online Mode
Real-time updates to trips
Offline Mode
Cache most recent update