Travel Tracker

Olivia Tarsillo, Michael Hood, Wilfred Naraga, Nur Yavuz

Deliverable Version 1.0

Chapter 1

Purpose:

Travel Tracker is evolving from its previous semester version into a production-ready, cloud-native web application. Our goal is to take the existing vacation planning platform, designed to streamline budgeting, itineraries, and travel logistics, and re-architect it for scalability, resilience, and real-world deployment. This semester's focus is on transitioning the application from a containerized environment using Docker Compose to a fully orchestrated environment using Kubernetes.

Travel planning can be complex, as it involves coordinating various aspects, including budgeting, flights, lodging, and activities. Travel Tracker already centralizes these features in one place. Our next step is to ensure that the platform can scale to production workloads while maintaining security, performance, and availability.

In addition to this Kubernetes transition, we envision Travel Tracker evolving into a richer, more intelligent platform for travelers. Planned enhancements include:

- **AI Recommendations:** Suggest hotels, activities, or destinations tailored to a user's budget and preferences.
- **Improved UI/UX:** A polished, user-friendly design to make the application visually appealing and easy to navigate.
- **Automated Testing:** Improve quality assurance and streamline deployments.
- **Trip Sharing and Group Planning:** Let users collaborate with friends or family on shared itineraries.
- **Advanced Uploads:** Enable users to upload travel documents such as tickets or confirmations, creating a central hub for all their travel information.

High-Level Architecture:

At a high level, our new architecture consists of multiple independently deployed services running on a Kubernetes cluster:

- **Frontend Pod (React.js):** Hosts the user-facing interface of Travel Tracker, exposing it via a Kubernetes Service for stable access and easy scaling.
- **Backend Pod (Flask/Python):** Handles API requests, authentication, and business logic, communicating securely with the database, AI services, and external APIs.
- **Database Pod (MongoDB):** Stores user data, trip information, and expenses. This will be provisioned as a StatefulSet with persistent volumes to ensure data durability.

- **External Services Integration (ExchangeRate API):** Provides real-time currency conversion as a third-party service integrated into the backend.

- **AI Recommendation Service:** A planned microservice running in its own pod, designed to interface with large language model APIs to generate travel suggestions and personalized recommendations.

- **ConfigMaps and Secrets:** Manage application configuration and credentials securely within the cluster.

- **Persistent Volumes:** Ensure data durability for MongoDB and other stateful components.

- **Horizontal Pod Autoscaling:** Automatically adjusts the number of pods in response to load, ensuring reliability under varying traffic.

This architecture transforms Travel Tracker from a simple three-container setup into a modern, cloud-native deployment. The use of Kubernetes allows for rolling updates, zero-downtime deployments, self-healing, and easier scaling, features that are essential for a production-ready application. By also incorporating AI recommendations, improved UI/UX, automated testing, trip sharing, and advanced uploads, Travel Tracker aims to go beyond its original scope and deliver a powerful, intelligent, and collaborative travel planning platform for users.

Key Features:

- **User Authentication (Sign-up/Login)**
    - Users can create an account and log in securely using JWT authentication.
    - Ensures data privacy and secure access.
- **Vacation Budget Management**
    - Users can create a trip budget and track flight, hotel, food, and activity expenses.
    - The budgeting tool ensures expenses stay within limits.
- **Expense & Banking Tracker**
    - Users can manually enter their income, expenses, and recurring payments. It helps in tracking finances efficiently.
- **Real-Time Currency Conversion**
    - Integrated with an *ExchangeRate API* to provide up-to-date currency conversions.
    - Users can convert expenses and budgets into different currencies.
- **Trip Itinerary & Activity Planner**
    - Users can add and organize their travel itineraries, including flights, hotel stays, and activities.
    - It helps users plan each day efficiently.
- **Notes & Additional Travel Info**
    - Users can add custom notes related to their trip.
    - A personalized section for important travel details.
- **Backend Data Storage & Management**
    - All user data, trip details, and transactions are securely stored in *MongoDB*.
    - Provides fast data retrieval and storage.
- **User Interface**
    - Elegant and user-friendly UI designed with *React.js*.
    - Simple navigation between budgeting, itinerary, and expenses.

- **Secure API Communication**
  - *Flask* handles all API requests, authentication, and interactions with MongoDB.
  - Ensures smooth data flow between the front end and the back end.
- **AI Recommendations (Planned):**
  - A dedicated AI Recommendation Service will suggest hotels, activities, or destinations based on a user's budget and preferences.
- **Trip Sharing & Group Planning (Planned):**
  - Allows users to share trips with friends or plan group vacations collaboratively.
- **Automated Testing (Planned):**
  - Full unit and integration test coverage to ensure smooth deployment, testing, and quality assurance.
- **Advanced Uploads (Planned):**
  - Users will be able to upload images or tickets for flights, enabling advanced trip planning and creating a hub for all travel-related information.

Team Formation and Roles:

Our team consists of four members with defined responsibilities to ensure efficient development and deployment:

Role	Name	Responsibilities
Project Lead / PM	Olivia Tarsillo	Oversees project direction, deliverables, and documentation; ensures alignment with course milestones.
DevOps Engineer	Michael Hood	Manages Kubernetes setup, deployments, scaling, and automation (CI/CD).
Backend Engineer (Flask / Inference)	Wilfred Naraga	Develops Flask backend, handles API integration, authentication, and AI microservice communication.
Frontend Developer / QA	Nur Yavuz	Implements React frontend, handles UI/UX testing, and ensures end-to-end functionality.

Chapter 2

Implementation:

React: The client side of our application will be developed using JavaScript and React. This allows for functionality for user experience for easy logins, a clean user interface, and robust performance. Users can access and manage their trip details, such as budget, excursion plans, and notes. After logging in, users can input trip details stored in their user profiles. This data is synchronized with the Python-based backend through API calls. Changes in currency exchange rates will be up-to-date and displayed to the user through the ExchangeRate API. Users can monitor, edit, and save their trip plans and updates in real time.

Key Frontend Features:

- *Login/Signup Pages:* Secure authentication interfaces for user registration and login.
- *Tracker Page:* Users can log expenses, track budgets, and manage allowances. A drop-down menu allows users to convert finances into various currencies.
- *Travel Planner Page*: Users can create, manage, and edit vacation plans. The budgeting feature automatically compares expenses against the allocated budget and integrates with the currency conversion tool for international trips.
- *Currency Conversion Tool*: Real-time currency conversion using the ExchangeRate API for accurate budgeting and financial tracking.
- *Trip Sharing:* Planned addition of a feature allowing users to share trips with friends or plan group vacations.
- *Advanced Uploads:* Planned ability for users to upload travel documents, tickets, and confirmations to build a complete travel hub.
- *AI Recommendations*: Planned integration of an AI-powered recommendation system to suggest hotels, activities, and destinations based on user budget and preferences.

MongoDB: MongoDB will handle user data. Each user profile will be stored, including username and password, and user inputs for budget, flight information, stay information, itinerary planning, personal notes, and interests.

Python: The backend will be developed using Python to manage server-side logic. Python will talk to the client about user data and communicate with the ExchangeRate API. This will use Flask. The back end will interact with MongoDB for data storage and retrieval.

API: To provide real-time currency conversion, we will integrate the ExchangeRate API. This API supplies up-to-date exchange rates for various global currencies, ensuring that users can accurately budget for international trips.

Key API Functions:

- *Currency Conversion*: Fetch current exchange rates based on user-selected currencies.
- *Automated Updates*: Regularly update exchange rates to reflect the latest market changes.
- *Integration with Expense Tracker*: Convert user-entered expenses into different currencies using API data.
- *Chat AI Recommendations (OpenAI):* Generate personalized travel suggestions based on user input and trip details.

Kubernetes Deployment: This semester, we will transition our Docker Compose setup into a Kubernetes cluster. Each service, frontend, backend, database, and planned AI recommendation service, will run as a separate Kubernetes Deployment with Horizontal Pod Autoscaling for scalability. The React frontend is exposed via a Service of type LoadBalancer (or NodePort) for external access. Backend and database remain internal (ClusterIP). Persistent storage will be provisioned for MongoDB to ensure data durability. This migration will make Travel Tracker production-ready with rolling updates, self-healing, and fault tolerance.

Automated Testing: We will begin implementing a test suite of unit and integration tests across the backend and frontend to ensure stable deployments and catch regressions early. This will lay the foundation for a continuous integration/continuous deployment (CI/CD) pipeline tied to Kubernetes.

AI: As part of this semester's expansion, we plan to integrate a Chat AI Recommendation Service into Travel Tracker to enhance personalization and streamline trip planning. This service will run as a dedicated microservice within our Kubernetes cluster and will communicate securely with the backend. We intend to use OpenAI's API to power this feature, leveraging its large language models to generate tailored suggestions for hotels, activities, and destinations based on a user's budget, preferences, and trip details. By integrating OpenAI into our architecture, we will be able to provide conversational, context-aware recommendations directly within the Travel Tracker interface, turning it into an interactive planning assistant rather than just a static tool.

Kubernetes Deployment Plan:

Objectives: Deploy Travel Tracker on a Rancher-managed Kubernetes cluster with reliable scaling, secure configuration, and clear promotion from development to staging. The plan covers build, configuration, deployment, validation, rollback, and data durability.

Environments: Development runs locally with Docker for quick iteration. Staging runs on CloudLab via Rancher to validate Kubernetes behavior, scaling, and external API integrations before class demos.

Configuration and Secrets: Non-sensitive settings are stored in ConfigMaps. Sensitive values (JWT secret, MongoDB URI, ExchangeRate API key, OpenAI API key) are stored in Kubernetes Secrets. Workloads consume these via environment variables and mounted files without hardcoding credentials.

Service types: The frontend is exposed externally via LoadBalancer or NodePort. The backend and MongoDB are internal and exposed via ClusterIP.

Scaling target: The backend is the primary scaling target. If the AI function is split into its own service in a future phase, it will become the secondary scaling target. HPA will scale these Deployments based on CPU or memory utilization.

Kubernetes Resources: Frontend runs as a Deployment with a Service. Backend runs as a Deployment with a Service. MongoDB runs as a StatefulSet with a PersistentVolumeClaim for durable storage. The AI recommendation logic is invoked by the backend and, can be deployed as a separate Deployment with an internal Service. Horizontal Pod Autoscaling is enabled for frontend and backend based on CPU or memory utilization. Liveness and readiness probes are configured on all pods to support updates and self-healing.

Network Exposure: The React frontend is exposed externally through a Service of type LoadBalancer or NodePort. The backend and database remain internal and are exposed only via ClusterIP. The frontend communicates with the

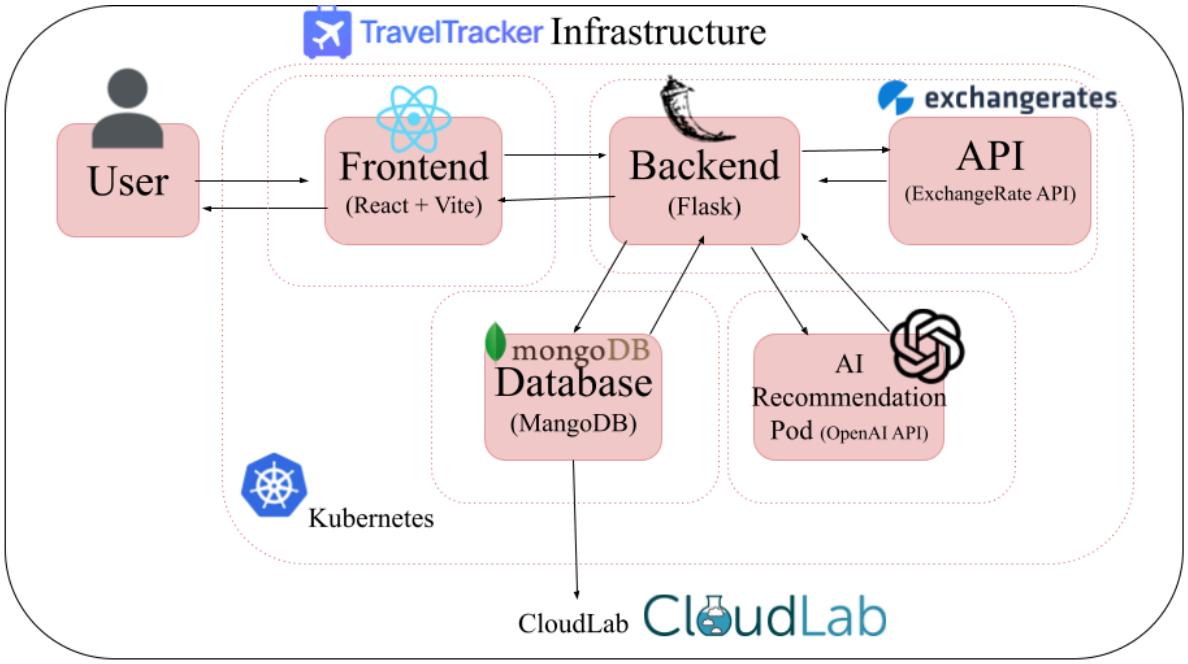
backend over the cluster network, the backend communicates with MongoDB, and calls external APIs for currency exchange and AI.

Persistence and Data Initialization: MongoDB uses a PVC so trip data and user accounts persist across pod restarts. Backup procedures capture snapshots of the MongoDB volume for disaster recovery.

Validation and Smoke Tests: Login and signup confirm authentication works. Trip creation verifies CRUD paths to MongoDB. Currency conversion validates ExchangeRate API calls and unit conversions in the UI. AI recommendations verify OpenAI integration and response handling. A short load test confirms that autoscaling reacts to concurrent traffic and maintains acceptable latency.

Monitoring and Logs: Application logs are available via 'kubectl logs' and Rancher's UI. Basic resource metrics validate HPA behavior. Probe failures and restarts are tracked through Kubernetes events to catch regressions early.

Architecture:



User opens the Travel Tracker frontend and authenticates. The frontend sends API requests to the backend to create trips, add itinerary items, and record expenses. For each request, the backend validates input and reads or writes data in MongoDB. When currency conversion is needed, the backend requests current exchange rates from the ExchangeRate API and returns values normalized to the user's base currency. When AI recommendations are requested, the backend assembles trip context and budget constraints, calls the AI recommendation component, and returns ranked suggestions. The frontend renders updated trip data, converted amounts, and recommendations to the user.

User Stories:

User Story 1: Trip Planning and Management

As a user, I want to create and manage detailed trip plans so I can keep my itineraries, budgets, and travel information organized in one place. This includes the ability to input travel details such as flights, hotels, activities, and personal notes. Each trip will have its own budget, allowing users to allocate funds across different expense categories such as lodging, food, transportation, and entertainment. Users will be able to view and edit their plans at any time, helping them stay organized throughout the entire travel process from planning to completion. The trip dashboard serves as a central hub, displaying all relevant travel information, schedules, and associated costs in an intuitive and user-friendly layout.

User Story 2: Financial Tracking and Currency Management

As a user, I want to track my income, expenses, and recurring payments to stay within my budget and accurately monitor my financial progress before and during travel, even when using different currencies. The system will enable users to log transactions and categorize them as either income or expense. Integrated with the ExchangeRate API, the platform automatically converts all recorded transactions into the user's base currency, allowing for seamless financial tracking across international trips. Users can visualize their spending breakdowns through real-time charts and see how currency fluctuations impact their available budget. This ensures financial clarity and helps prevent overspending while traveling abroad.

User Story 3: AI-Powered Travel Recommendations

As a user, I want to receive AI-generated travel recommendations such as hotels, restaurants, or activities that align with my preferences, destination, and budget. Using the OpenAI-powered recommendation service, the backend will process user trip data, including destination, spending habits, and available budget, to generate tailored suggestions. The AI analyzes user preferences such as previous destinations, saved interests, and trip categories to offer relevant and personalized advice. For international trips, the AI will also consider currency exchange data from the ExchangeRate API, ensuring all recommended options are realistically priced within the user's budget range. This transforms Travel Tracker into a smart, conversational assistant that enhances the trip planning experience.

Risk & Test Plan:

Top Risks:

1. API latency or failure from ExchangeRate or OpenAI integration.
2. Authentication bugs (token expiration or invalid JWTs).
3. Model timeout during AI recommendation calls.
4. MongoDB data inconsistency under concurrent writes.
5. Misconfigured Kubernetes scaling or resource limits.

Testing Strategy:

Login/Signup: Register and log in using valid credentials to confirm that the authentication process works correctly and grants secure access to the dashboard.

Invalid Credentials: Attempt to log in with an incorrect password to ensure that the system handles authentication errors gracefully and prevents unauthorized access.

Trip Creation: Add a trip, itinerary, and expenses to verify that all user data is successfully stored and retrievable from MongoDB without loss or corruption.

Currency Conversion: Convert a trip budget or expense using the ExchangeRate API to confirm that the system displays accurate and real-time currency conversion values.

AI Recommendations: Request travel suggestions for activities, destinations, or hotels to ensure that the AI returns valid, context-aware, and personalized recommendations based on user input.

Load Test (Autoscaling): Simulate multiple concurrent users accessing the application to confirm that the Horizontal Pod Autoscaler scales pods automatically under increased traffic.

Fault Injection: Simulate an API outage or network disruption to verify that the system responds with clear, user-friendly error messages and maintains stability without crashing.

Team Charter:

- *Meeting Cadence:* Twice weekly (Tuesday/Friday) for sprint updates and task alignment.
- *Collaboration Tools*: GitHub for version control, Discord for communication, and CloudLab for deployment.
- *Definition of Done (DoD):*
    - Code reviewed and merged to the main branch
    - Tests pass (unit + integration)
    - Functionality demonstrated on staging cluster
    - Documentation updated in README and report