

TMTC Backend Assignment

Objective:

Develop a **Travel Itinerary API** that allows users to create, manage, and share travel itineraries. The API should support **CRUD operations**, authentication, and some level of optimization (caching, indexing, etc.).

Project Overview:

You will build a **REST API** using **Node.js (Express.js)** and **MongoDB (Mongoose ORM)** that enables users to:

- Register/Login with authentication
 - Create, update, and delete travel itineraries
 - Fetch itineraries with optional filters
 - Implement caching and performance optimizations
-

Technical Requirements:

1. Tech Stack:

- **Backend Framework:** Node.js with Express.js
 - **Database:** MongoDB (Mongoose ORM)
 - **Authentication:** JWT-based authentication
 - **Caching:** Use Redis (or an in-memory solution like Node-cache)
 - **API Documentation:** Swagger or Postman Collection
-

Project Requirements:

1. User Authentication

- Implement **JWT-based authentication**.
- Users should be able to **register**, **log in**, and get an access token.
- Implement **password hashing** using **bcrypt**.

Endpoints:

- ✓ POST /api/auth/register → Register a new user
 - ✓ POST /api/auth/login → Login and get a JWT token
-

2. Itinerary Management

Users should be able to create, update, delete, and fetch travel itineraries.

Itinerary Schema (MongoDB):

```
{
  "_id": "ObjectId",
  "userId": "ObjectId",
  "title": "string",
  "destination": "string",
  "startDate": "ISODate",
  "endDate": "ISODate",
  "activities": [
    {
      "time": "string",
      "description": "string",
      "location": "string"
    }
  ],
  "createdAt": "ISODate",
  "updatedAt": "ISODate"
}
```

Endpoints:

- ✓ POST /api/itineraries → Create an itinerary
 - ✓ GET /api/itineraries → Get all itineraries (supports filtering by destination)
 - ✓ GET /api/itineraries/:id → Get a specific itinerary
 - ✓ PUT /api/itineraries/:id → Update an itinerary
 - ✓ DELETE /api/itineraries/:id → Delete an itinerary
-

3. API Query Features

- Implement **pagination**: Allow users to retrieve itineraries in chunks (`?page=1&limit=10`).

- Implement **sorting**: Sort itineraries by `createdAt`, `startDate`, or `title` (`?sort=startDate`).
 - Implement **filtering**: Allow filtering by `destination` (`?destination=Paris`).
-



4. Performance Optimizations

- **Database Indexing**: Ensure common query fields like `userId` and `destination` are indexed.
 - **Redis Caching**:
 - Cache GET requests for individual itineraries (`GET /api/itineraries/:id`).
 - Cached itineraries should expire after **5 minutes**.
-

5. Sharing Feature

- Implement a feature that allows users to generate a **public shareable link** for an itinerary.
 - When a public link is accessed, return itinerary details but **exclude sensitive data** (e.g., `userId`).
 - **Example Shareable URL**:
`GET /api/itineraries/share/:shareableId` (should not require authentication)
-

6. Writing Unit & Integration Tests

- Implement **unit tests** using **Jest & Supertest** for:
 -  Authentication endpoints
 -  Itinerary CRUD operations
-

7. Dockerizing the Application

- Write a **Dockerfile** and **docker-compose.yml** to containerize the API and database.
 - The API should run inside a Docker container with MongoDB as a separate container.
-

Bonus Features (Optional, Extra Points)

- **Email Notifications:** Send an email when a new itinerary is created using **Nodemailer**.
 - **GraphQL API:** Provide an alternative GraphQL endpoint.
 - **Rate Limiting:** Protect against abuse using `express-rate-limit`.
 - **API Documentation:** Document the API using Swagger (OpenAPI) or a Postman Collection.
-

Submission Guidelines

- Push the code to a **public GitHub repository**.
- Include a **README** with:
 - Setup and installation steps
 - How to run the project
 - API documentation link or instructions
- Deploy the API (optional, but extra points for a working demo on **Render/Heroku/Vercel**).