NAP QUEENS ASSIGNMENT

Event Booking System Assessment

Instructions for Setting Up and Using the API

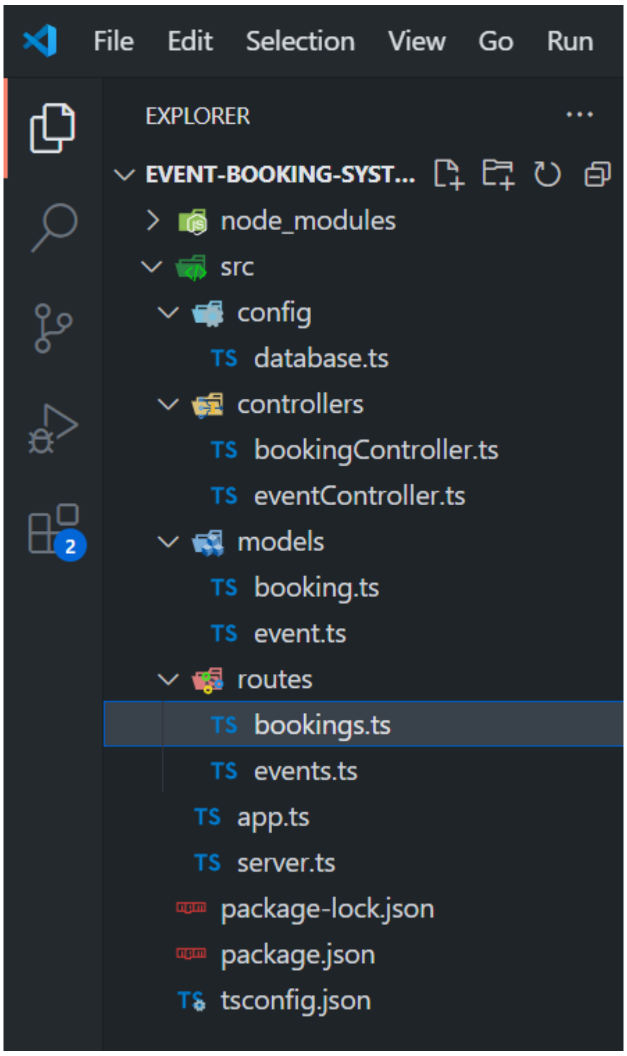
1. **Clone the Repository**
   - Clone the project repository from GitHub to your local machine using:
   - Navigate into the project directory:
2. **Install Dependencies**
   - Ensure Node.js and npm are installed on your machine.
   - Install the necessary npm packages by running:
3. **Set Up MongoDB**
   - Ensure MongoDB is installed and running on your local machine or use a cloud MongoDB service.
   - Update the MongoDB connection string in the src/config/config.ts file to match your MongoDB setup.
4. **Run the Application**
   - Start the server with the following command:
   - The server will run on the configured port (e.g., 3000 or 5000).

Details of Implementation

- Designed the database schema for events and bookings.
- Created collections and set up indexes.
- Created the main server file (server.ts) for Express initialization.
- Configured middleware (e.g., body-parser, CORS).
- Designed event and booking schemas with appropriate fields.
- Implemented controller functions for creating events, booking tickets, cancelling bookings, retrieving events, and printing tickets.
- Set up routes in bookings.ts and events.ts and integrated them with the Express application.
- Implemented error handling for various scenarios.
- Tested API endpoints using Postman, validating functionalities and ensuring proper error messages and status codes.

- Debugged and resolved issues with endpoints and database connectivity.
- Documented API endpoints, request/response formats, and usage instructions, including sample requests and responses.
- Compiled a project report detailing setup, development, testing, and deployment.
- Prepared the application for deployment by configuring environment variables and ensuring code quality.
- Deployed the application to a hosting service or cloud platform if required.
- Reviewed the project for completeness and accuracy, added final touches to documentation and code, and submitted the project with documentation and report.

Project Structure



```
EVENT-BOOKING-SYST...
  node_modules
  src
    config
      database.ts
    controllers
      bookingController.ts
      eventController.ts
    models
      booking.ts
      event.ts
    routes
      bookings.ts
      events.ts
    app.ts
    server.ts
  package-lock.json
  package.json
  tsconfig.json
```

Database.ts

```typescript
import mongoose from 'mongoose';

const connectDB = async () => {
  try {
    await mongoose.connect('mongodb://localhost:27017/event-booking');
    console.log('MongoDB connected');
  } catch (err) {
    console.error('MongoDB connection error:', err);
    process.exit(1);
  }
};

export default connectDB;
```

bookingController.ts

```typescript
import { Request, Response } from 'express';
import Event from '../models/event';
import Booking from '../models/booking';

export const createBooking = async (req: Request, res: Response) => {
  try {
    const { eventId, quantity } = req.body;
    const event = await Event.findById(eventId);
    if (!event) {
      return res.status(404).json({ error: 'Event not found' });
    }

    if (event.totalTickets - event.bookedTickets < quantity) {
      return res.status(400).json({ error: 'Not enough tickets available' });
    }

    const booking = new Booking({
      eventId,
      quantity,
    });
    await booking.save();

    event.bookedTickets += quantity;
    await event.save();

    res.status(201).json(booking);
  } catch (err) {
    res.status(500).json({ error: 'Error creating booking' });
  }
};

export const cancelBooking = async (req: Request, res: Response) => {
  try {
    const bookingId = req.params.id;

    const booking = await Booking.findById(bookingId);
    if (!booking) {
      return res.status(404).json({ error: 'Booking not found' });
    }

    const event = await Event.findById(booking.eventId);
    if (event) {
      event.bookedTickets -= booking.quantity;
      await event.save();
    }

    await Booking.deleteOne({ _id: bookingId });

    res.json({ message: 'Booking cancelled' });
  } catch (err) {
    res.status(500).json({ error: 'Error cancelling booking' });
  }
};
```

eventController.ts

```
import { Request, Response } from 'express';
import Event from '../models/event';

export const createEvent = async (req: Request, res: Response) => {
  try {
    const { name, date, totalTickets } = req.body;
    const event = new Event({ name, date, totalTickets });
    await event.save();
    res.status(201).json(event);
  } catch (err) {
    res.status(500).json({ error: 'Error creating event' });
  }
};

export const getEvents = async (req: Request, res: Response) => {
  try {
    const events = await Event.find();
    res.json(events);
  } catch (err) {
    res.status(500).json({ error: 'Error fetching events' });
  }
};

export const getEventById = async (req: Request, res: Response) => {
  try {
    const event = await Event.findById(req.params.id);
    if (!event) {
      return res.status(404).json({ error: 'Event not found' });
    }
    res.json(event);
  } catch (err) {
    res.status(500).json({ error: 'Error fetching event' });
  }
};
```

Routes

Bookings.ts

```
import express from 'express';
import { createBooking, cancelBooking } from
'../controllers/bookingController';

const router = express.Router();

router.post('/', createBooking);
router.delete('/:id', cancelBooking);

export default router;
```

events.ts

```
import express from 'express';
import { createEvent, getEvents, getEventById } from
'../controllers/eventController';

const router = express.Router();

router.post('/', createEvent);
router.get('/', getEvents);
router.get('/:id', getEventById);

export default router;
```

models

booking.ts

```
import mongoose, { Schema, Document } from 'mongoose';

interface IBooking extends Document {
  eventId: mongoose.Types.ObjectId;
  quantity: number;
  timestamp: Date;
}

const bookingSchema = new Schema<IBooking>({
  eventId: { type: Schema.Types.ObjectId, ref: 'Event', required: true },
  quantity: { type: Number, required: true },
  timestamp: { type: Date, default: Date.now },
});

const Booking = mongoose.model<IBooking>('Booking', bookingSchema);

export default Booking;
```

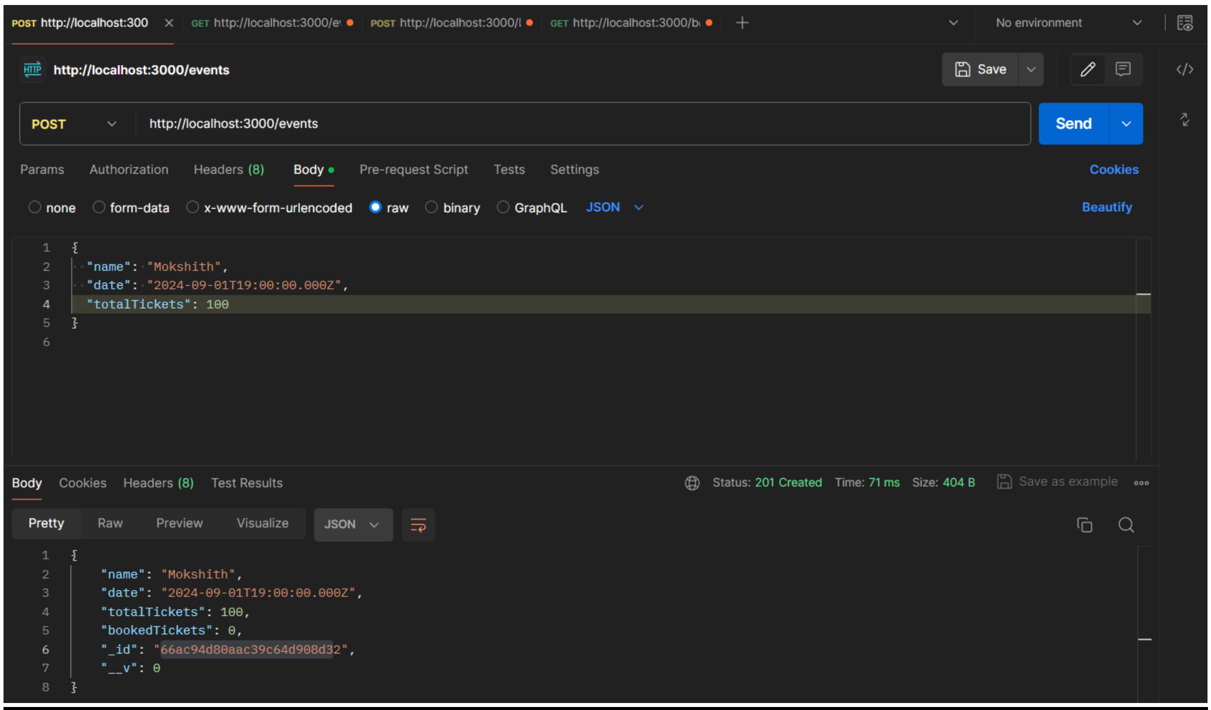
event.ts

```
import mongoose, { Schema, Document } from 'mongoose';

interface IEvent extends Document {
  name: string;
  date: Date;
  totalTickets: number;
  bookedTickets: number;
}

const eventSchema = new Schema<IEvent>({
  name: { type: String, required: true },
  date: { type: Date, required: true },
  totalTickets: { type: Number, required: true },
  bookedTickets: { type: Number, default: 0 },
});

const Event = mongoose.model<IEvent>('Event', eventSchema);

export default Event;
```

app.ts

```
import express from 'express';
import mongoose from 'mongoose';
import eventRoutes from './routes/events';
import bookingRoutes from './routes/bookings';

const app = express();
app.use(express.json());

app.use('/events', eventRoutes);
app.use('/bookings', bookingRoutes);

const startServer = async () => {
  try {
    await mongoose.connect('mongodb://localhost:27017/event-booking-system');
    console.log('Connected to MongoDB');
  } catch (err) {
    console.error('Failed to connect to MongoDB', err);
  }
};

startServer();

app.listen(3000, () => {
  console.log('Server is running on port 3000');
});

export default app;
```

API endpoints validation using Postman



POST http://localhost:300
GET http://localhost:3000/e
POST http://localhost:3000/l
GET http://localhost:3000/b
No environment
Save
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Beautify

http://localhost:3000/events

POST http://localhost:3000/events

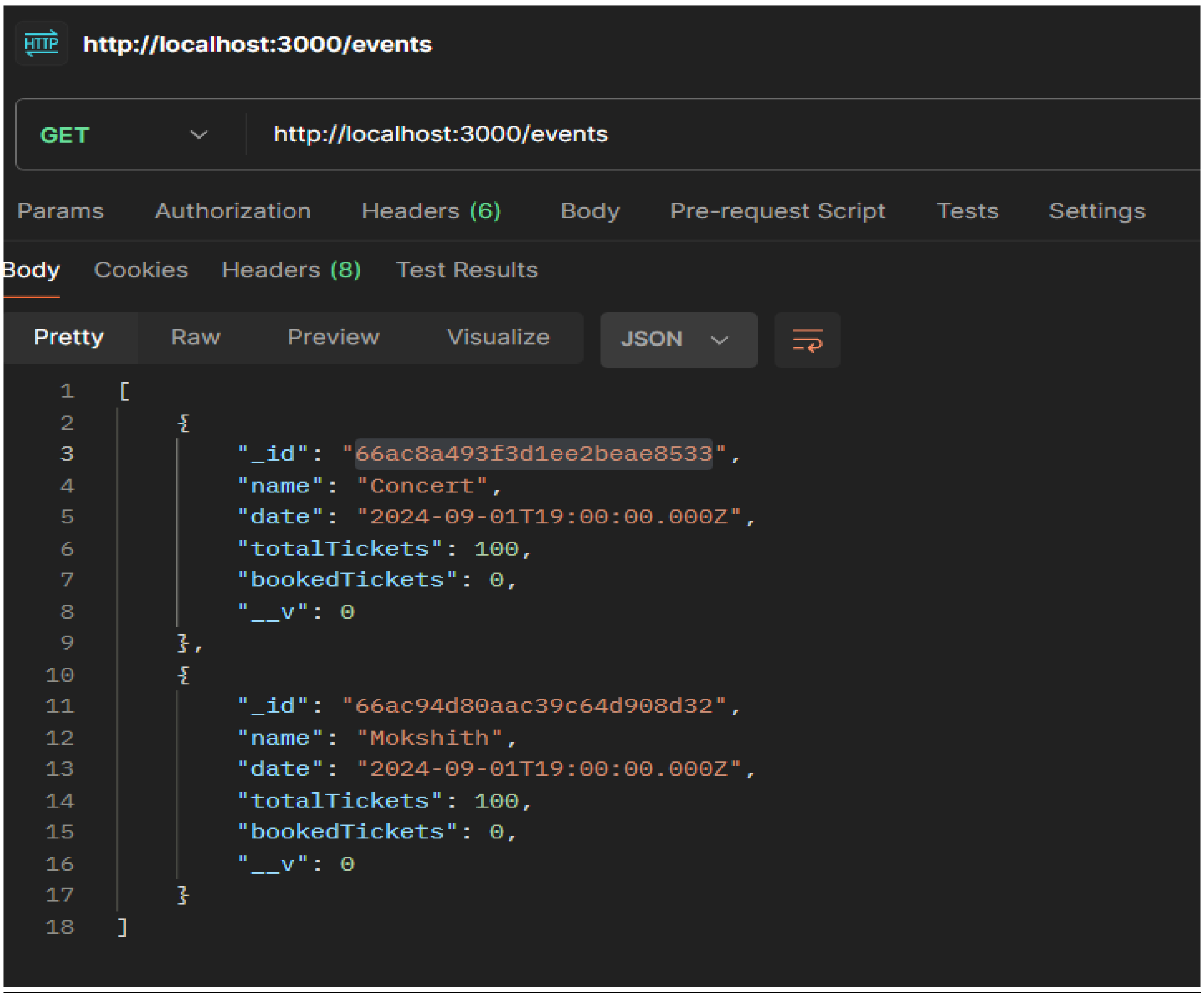
Body: none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{
  "name": "Mokshith",
  "date": "2024-09-01T19:00:00.000Z",
  "totalTickets": 100
}
```

Body Cookies Headers (8) Test Results — Status: 201 Created Time: 71 ms Size: 404 B Save as example

Pretty Raw Preview Visualize JSON

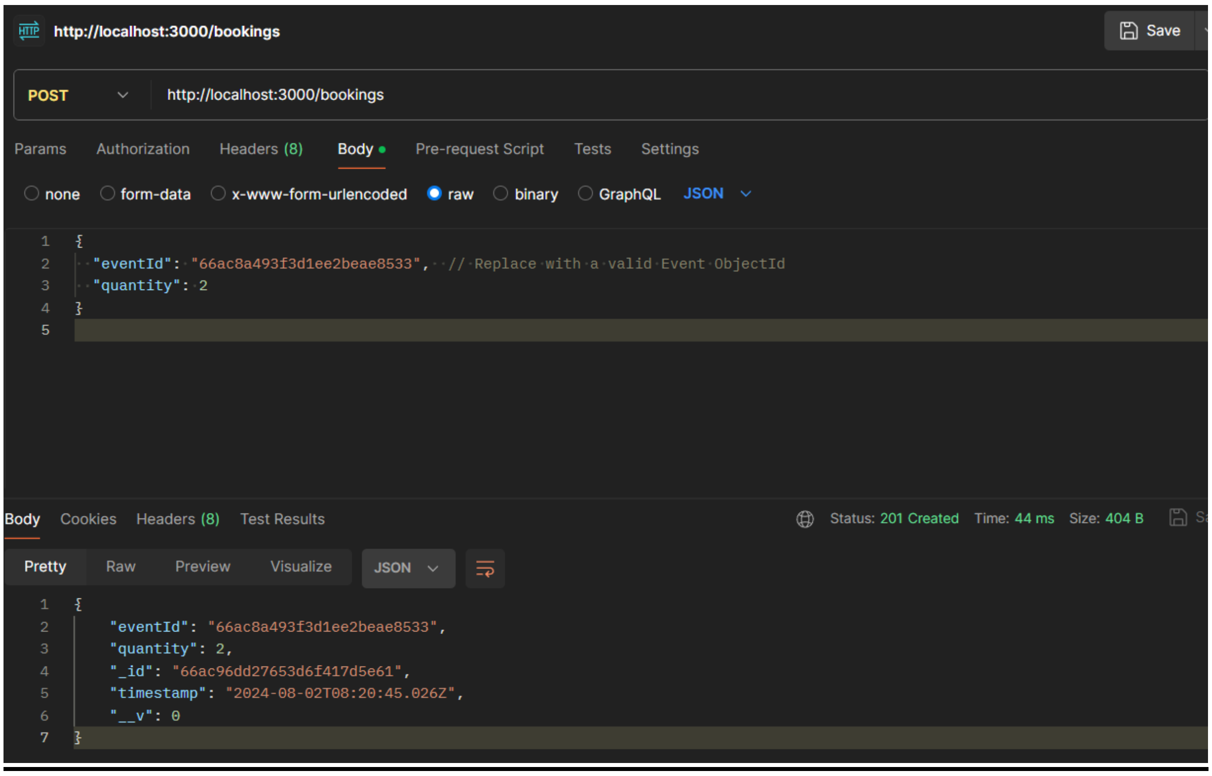
```
{
    "name": "Mokshith",
    "date": "2024-09-01T19:00:00.000Z",
    "totalTickets": 100,
    "bookedTickets": 0,
    "_id": "66ac94d80aac39c64d908d32",
    "__v": 0
}
```



http://localhost:3000/events

GET http://localhost:3000/events

```
[
    {
        "_id": "66ac8a493f3d1ee2beae8533",
        "name": "Concert",
        "date": "2024-09-01T19:00:00.000Z",
        "totalTickets": 100,
        "bookedTickets": 0,
        "__v": 0
    },
    {
        "_id": "66ac94d80aac39c64d908d32",
        "name": "Mokshith",
        "date": "2024-09-01T19:00:00.000Z",
        "totalTickets": 100,
        "bookedTickets": 0,
        "__v": 0
    }
]
```



http://localhost:3000/bookings

POST http://localhost:3000/bookings

```
{
  "eventId": "66ac8a493f3d1ee2beae8533",  // Replace with a valid Event ObjectId
  "quantity": 2
}
```

Status: 201 Created Time: 44 ms Size: 404 B

```
{
    "eventId": "66ac8a493f3d1ee2beae8533",
    "quantity": 2,
    "_id": "66ac96dd27653d6f417d5e61",
    "timestamp": "2024-08-02T08:20:45.026Z",
    "__v": 0
}
```

Conclusion

In this project, we developed an Event Booking System using Node.js, TypeScript, Express.js, and MongoDB. The system includes functionalities for creating events, booking tickets, canceling bookings, and retrieving event details.

We designed a schema to manage events and bookings, implemented the necessary API endpoints, and ensured proper error handling. The system was tested with Postman to confirm its functionality and reliability.

The project is documented with clear setup instructions and usage guidelines, making it easy to deploy and use. This system demonstrates effective application of modern web development practices.