Vivekanand Education Society's Institute of Technology
An Autonomous Institute Affiliated to University of Mumbai
Department of Computer Engineering

Year: 2024-2025

Name of the Course : Full Stack Web Development (Lab)

Year/Sem/Class : S.E.(COMP) / Sem IV / D7B **Code:**NCMVS41

Faculty Incharge : Pradnya Raut

Roll No: 67	**Name:** Manish Raje	
Exp No.: 06	**Title:** **Building a RESTful API with Express.**	
DOP: 26-03-2025	**DOS:**	
Grade:	**Course Outcomes:**	**Signature:**

AIM: Building a RESTful API with Express.

Experiment: Create routes for CRUD operations (Create, Read, Update, Delete) on a mock dataset. Use Express middleware for request handling and validation.

Theory

This Express.js application is a simple REST API that manages a collection of tours stored in a JSON file. The API allows users to perform CRUD (Create, Read, Update, Delete) operations on tour data.

1. Middleware Functions

Middleware functions are used to handle requests before they reach the main route handlers.

- The first middleware logs a message for every incoming request.
- The second middleware adds a requestTime property to the request object, storing the current timestamp.

These middleware functions ensure that additional information is available before processing a request.

2. Handling Tour Data

The tour data is stored in a JSON file (tours-simple.json). This data is read synchronously when the server starts and is used for processing requests.

3. Route Handlers

The API supports the following endpoints:

a) Get All Tours (GET /api/v1/tours)

- Returns a list of all tours.
- Includes metadata such as the number of tours and the request timestamp.

b) Create a New Tour (POST /api/v1/tours)

- Receives tour details in the request body.
- Assigns a new ID to the tour and adds it to the JSON file.
- Returns the newly created tour in the response.

c) Get a Single Tour (GET /api/v1/tours/:id)

- Retrieves a specific tour based on the provided id.
- If no tour is found, returns an error message.

d) Update a Tour (PATCH /api/v1/tours/:id)

- Allows partial updates to a tour's details.

- Merges the new data with existing tour data.
- Returns the updated tour.

e) Delete a Tour (DELETE /api/v1/tours/:id)

- Removes a tour based on its ID.
- Filters the tour list and rewrites the JSON file without the deleted entry.
- Returns the updated list of tours.

4. Express Route Chaining

Routes with the same base URL (/api/v1/tours) are grouped using Express's route() method.

- GET and POST requests for all tours are handled together.
- GET, PATCH, and DELETE requests for a specific tour ID are grouped.

5. Server Initialization

- The server listens on port 8000.
- A message is logged when the server starts successfully.
-

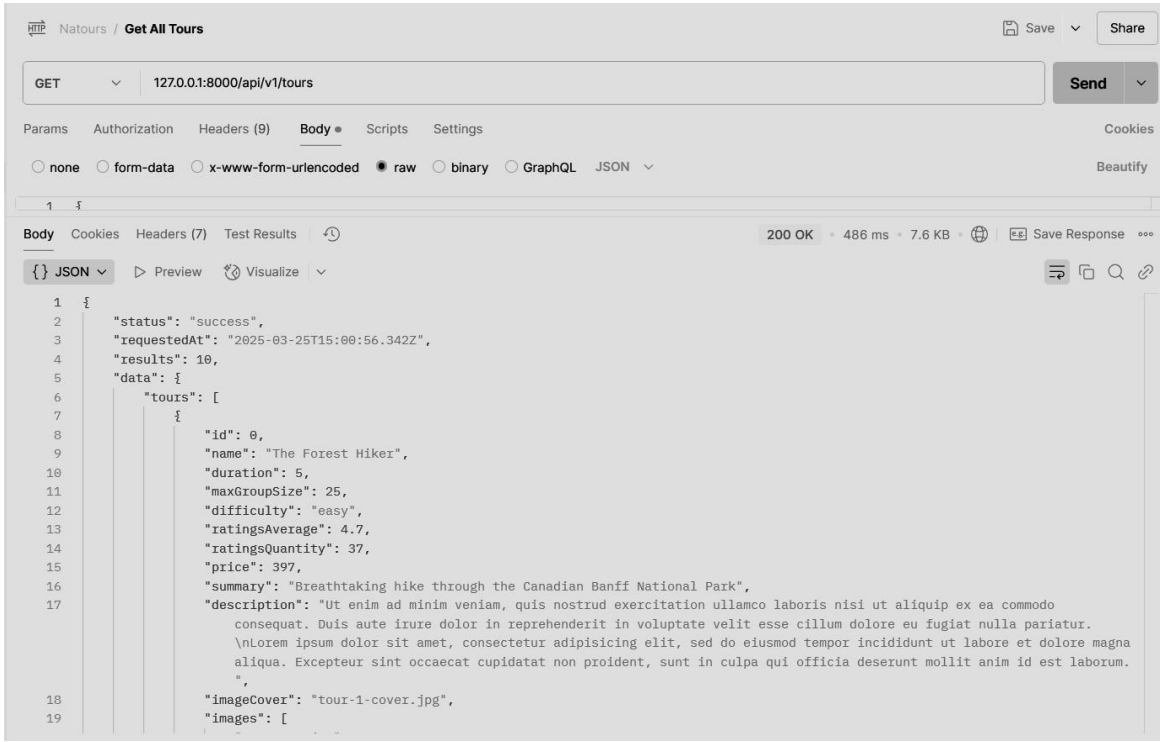
```
const fs = require('fs');
const express = require('express'); const
app = express(); app.use(express.json());
app.use((req, res, next)=>{
  console.log('This is middleware function'); next();
})
app.use((req, res, next)=>{
  req.requestTime = new Date().toISOString(); next();
})
const tours = JSON.parse(
fs.readFileSync(`${__dirname}/dev-data/data/tours-simple.json`,

'utf-8')
);
const getAllTours=(req, res) => {
  console.log(req.requestTime)
  res.status(200).json({
    status: 'success',
    requestedAt: req.requestTime, results:
    tours.length,
    data: {
      tours,
    },
  });
}
const createTour=(req, res) => {
  const newId = tours[tours.length - 1].id + 1;
  const newTour = Object.assign({ id: newId }, req.body);
  tours.push(newTour);
  fs.writeFile(
    `${__dirname}/dev-data/data/tours-simple.json`, JSON.stringify(tours),
    (err) => {
      res.status(201).json({
      status: 'success', data: {
          tour: newTour,
        },
      });
    }
  );
}
const getTour = (req, res) => {
  const tour = tours.find((el) => el.id === parseInt(req.params.id)); if
  (!tour) {
    return res.status(404).json({
      status: 'fail',
      message: 'Invalid ID',
    });
  }
  res.status(200).json({
    status: 'success', data: {
      tour,
    },
  });
}
const updateTour=(req, res) => {
```

```
  const id = parseInt(req.params.id); if
  (id > tours.length || id < 1) { return
  res.status(404).json({
      status: 'fail', message:
      'Invalid ID'
    });
  }
  // Find the tour and update it
  const tourIndex = id - 1; // Assuming IDs are 1-based and
correspond to array indexes
  const updatedTour = { ...tours[tourIndex], ...req.body }; // Merge
updates
  tours[tourIndex] = updatedTour; // Save changes res.status(200).json({
    status: 'success',
    data: {
      tour: updatedTour
    }
  });
}
const deleteTour=(req,res)=>{  const id
  = parseInt(req.params.id); if (id >
  tours.length || id < 1) { return
  res.status(404).json({
      status: 'fail',
      message: 'Invalid ID'
    });
  }
  const newTours = tours.filter(tour=>tour.id !== id); fs.writeFile(
    `${__dirname}/dev-data/data/tours-simple.json`,
    JSON.stringify(newTours),
    (err) => {
      res.status(201).json({
      status: 'success', data: {
          tours: newTours,
        },
      });
    }
  );
}
app.route('/api/v1/tours').get(getAllTours).post(createTour);
app.route('/api/v1/tours/:id').get(getTour).patch(updateTour).delet
e(deleteTour);
const PORT = 8000;
app.listen(PORT, () => {
  console.log(`App running on port ${PORT}`);
});
```

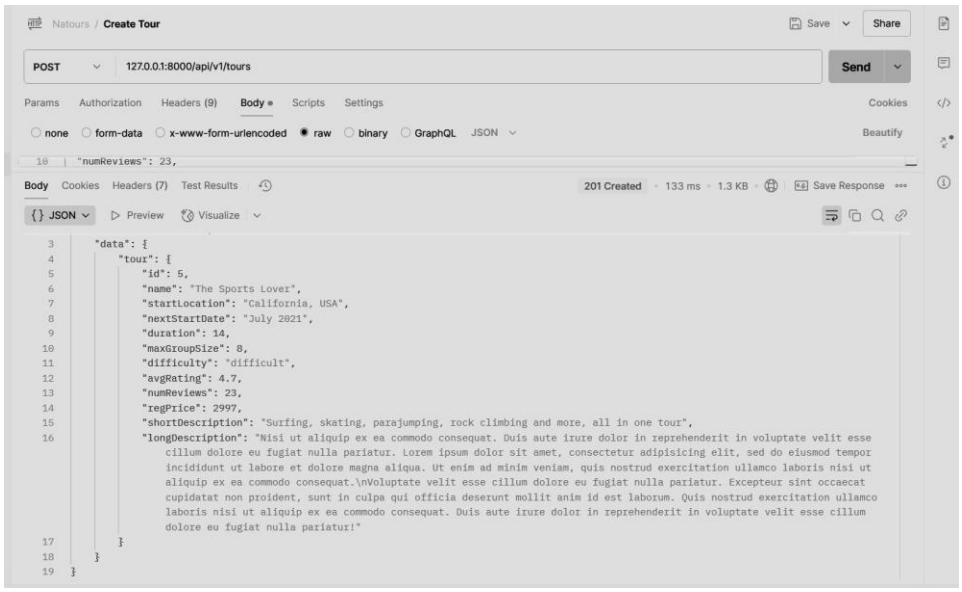
Output:-

1. Get All Tours

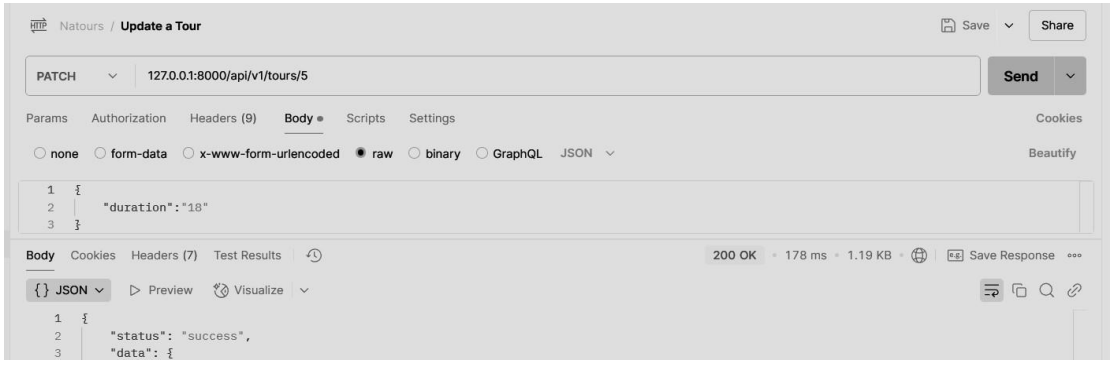


2. Get a particular tour

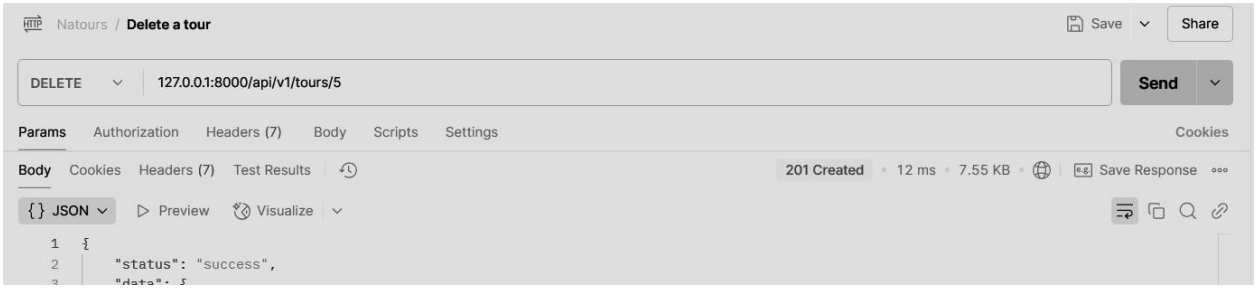
3. Create a tour



4. Updating a tour's duration



5. Deleting a tour



Conclusion

This API demonstrates fundamental Express.js concepts, including middleware usage, route handling, request processing, and file-based data storage. It provides a basic yet functional system for managing tour data without a database.