Experiment No - 07

Roll No.	52
Name	Kshitij Nangare
Class	D15B
Subject	Full Stack Development
Lab Outcome	L2, L3
Date of Performance / Submission	29/09/2025 06/10/2025
Signature & Grades	

Experiment 7

Aim : Validating RESTful APIs using Postman.

Code :

backend > app.js > ...

```javascript
import dotenv from 'dotenv';  7.2k (gzipped: 3.1k)
import express from 'express';
import cookieParser from 'cookie-parser';  5k (gzipped: 2k)
import cors from 'cors';  5k (gzipped: 2.1k)
import {connectDB} from './config/db.js';

import authRoutes from './routes/authRoutes.js';
import sessionRoute from './routes/sessionRoutes.js';

dotenv.config();

const app = express();
connectDB();

// Middleware
app.use(cors({ origin: process.env.FRONTEND_URL, credentials: true }));
app.use(cookieParser());
app.use(express.json());

app.use('/api/auth', authRoutes);
app.use('/api/session', sessionRoute);

// Server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});

```

Figure 1

backend > routes > authRoutes.js > ...

```javascript
import express from 'express';
import { register, login, changePassword } from '../controllers/authController.js';
import { authenticate } from '../middlewares/auth.js';

const router = express.Router();

// POST /api/auth/register
router.post('/register', register);

// POST /api/auth/login
router.post('/login', login);

router.get('/verify', authenticate, (req, res) => {
  // The authenticate middleware already verified the token
  res.json({user: {
    id: req.user._id,
    name: req.user.name,
    email: req.user.email
  }});
});

router.post('/change-password', authenticate, changePassword);


export default router;
```

Figure 2

backend > routes > sessionRoutes.js > ...

```javascript
import express from 'express';
import { authenticate } from '../middlewares/auth.js';

import {
    getAllPublishedSessions,
    getMySessions,
    createSession,
    updateSession,
    deleteSession,
    likeSession,
    getSessionById
} from '../controllers/sessionsController.js';

const router = express.Router();

// GET all published sessions
router.get('/get-all-sessions',authenticate, getAllPublishedSessions);


router.get('/get-session/:id',authenticate, getSessionById);

// GET logged-in user's sessions
router.get('/my-sessions',authenticate, getMySessions);

// POST create new session
router.post('/create',authenticate, createSession);

router.post('/like/:id',authenticate, likeSession);

// PATCH update a session
router.patch('/update/:id',authenticate, updateSession);

// DELETE a session
router.delete('/delete/:id',authenticate, deleteSession);

export default router;
```

Figure 3

backend > controllers > authController.js > register

```javascript
import { generateToken } from '../utils/jwtToken.js';
import bcrypt from 'bcryptjs';  20.1k (gzipped: 9k)

export const register = async (req, res) => {
  try {
    const { email,name, password } = req.body;

    // Validate input
    if (!email || !password || !name) {
      return res.status(400).json({ message: 'Email and password are required' });
    }

    // Check if user exists
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(409).json({ message: 'Email already in use' });
    }

    // Hash password
    const salt = await bcrypt.genSalt(10);
    const password_hash = await bcrypt.hash(password, salt);


    // Create new user
    const user = new User({ email,name, password: password_hash });
    await user.save();

    // Generate JWT
    const token = generateToken(user);

    res.status(201).json({
      token,
      user: {
        id: user._id,
        name: user.name,
        email: user.email
      }
    });
  } catch (error) {
    console.error('Registration error:', error);
    res.status(500).json({ message: 'Registration failed' });
  }
};
```

Figure 4

backend > controllers > authController.js > register

```javascript
export const changePassword = async (req, res) => {
    const { currentPassword, newPassword } = req.body;

    // 1. Basic validation
    if (!currentPassword || !newPassword) {
        return res.status(400).json({ message: 'Please provide current password and new password.' });
    }

    if (newPassword.length < 4) {
        return res.status(400).json({ message: 'New password must be at least 4 characters long.' });
    }

    try {
        // req.user will come from your authentication middleware (e.g., JWT verification)
        // It should contain the ID of the authenticated user
        const user = await User.findById(req.user._id).select('+password'); // Select password field explicitly

        if (!user) {
            return res.status(404).json({ message: 'User not found.' });
        }

        // 2. Compare current password
        const isMatch = await bcrypt.compare(currentPassword, user.password);

        if (!isMatch) {
            return res.status(401).json({ message: 'Current password is incorrect.' });
        }

        // 3. Check if new password is the same as current password
        if (newPassword === currentPassword) {
            return res.status(400).json({ message: 'New password cannot be the same as the current password.' });
        }

        // 4. Hash the new password and save
        const salt = await bcrypt.genSalt(10);
        user.password = await bcrypt.hash(newPassword, salt); // Hash with a salt round of 10
        await user.save();

        res.status(200).json({ message: 'Password changed successfully!' });

    } catch (error) {
        console.error('Error changing password:', error);
        res.status(500).json({ message: 'Server error. Could not change password.' });
    }
};
```

Figure 5

```
export const login = async (req, res) => {
  try {
    const { email, password } = req.body;

    // Validate input
    if (!email || !password) {
      return res.status(400).json({ message: 'Email and password are required' });
    }

    // Find user
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(401).json({ message: 'Invalid Email or Password' });
    }

    // Check password
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(401).json({ message: 'Invalid Email or Password' });
    }

    // Generate JWT
    const token = generateToken(user);

    res.json({
      token,
      user: {
        id: user._id,
        name: user.name,
        email: user.email
      }
    });
  } catch (error) {
    console.error('Login error:', error);
    res.status(500).json({ message: 'Login failed' });
  }
};
```

Figure 6

backend > controllers > sessionsController.js > ...

```javascript
import Session from "../models/Session.js";

// Get all published sessions (public) with search functionality
export const getAllPublishedSessions = async (req, res) => {
  try {
    const { search } = req.query; // Extract search term from query parameters

    let query = { status: 'published' };
    if (search) {
      // Create a case-insensitive regex for title or tags
      const searchRegex = new RegExp(search, 'i');
      query.$or = [
        { title: { $regex: searchRegex } },
        { tags: { $in: [searchRegex] } } // Search within the tags array
      ];
    }

    const sessions = await Session.find(query)
      .populate('user_id', 'name email') // Populate creator details
      .sort({ createdAt: -1 }); // Sort by creation date, newest first

    res.json(sessions);
  } catch (err) {
    console.error("Error in getAllPublishedSessions:", err); // Log the error
    res.status(500).json({ message: err.message || "Server error" });
  }
};

// Get logged-in user's sessions
export const getMySessions = async (req, res) => {
  try {
    const sessions = await Session.find({ user_id: req.user._id })
      .populate('user_id', 'name email')
      .sort({ createdAt: -1 });
    res.json(sessions);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

Figure 7

```
// Get logged-in user's sessions
export const getMySessions = async (req, res) => {
  try {
    const sessions = await Session.find({ user_id: req.user._id })
      .populate('user_id', 'name email')
      .sort({ createdAt: -1 });
    res.json(sessions);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};


export const getSessionById = async (req, res) => {
  try {
    const { id } = req.params; // Get the session ID from the URL parameters

    // Find the session and populate the user_id field to get creator's name and email
    const session = await Session.findById(id).populate('user_id', 'name email');

    if (!session) {
      return res.status(404).json({ message: 'Session not found.' });
    }

    res.status(200).json(session);
  } catch (error) {
    console.error('Error fetching session by ID:', error);
    // Handle specific CastError if ID format is invalid
    if (error.name === 'CastError') {
        return res.status(400).json({ message: 'Invalid session ID format.' });
    }
    res.status(500).json({ message: 'Server error.', error: error.message });
  }
};
```

Figure 7

```
// Create new session
export const createSession = async (req, res) => {

  const session = new Session({
    ...req.body,
    user_id: req.user._id
  });

  try {
    const newSession = await session.save();
    res.status(201).json(newSession);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
};

// Update a session
export const updateSession = async (req, res) => {
  try {
    const session = await Session.findOneAndUpdate(
      { _id: req.params.id, user_id: req.user._id },
      req.body,
      { new: true }
    ).populate('user_id', 'name email');
    if (!session) return res.status(404).json({ message: 'Session not found' });
    res.json(session);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
};
```

Figure 8

C:\Users\Roshan Yadav\jupyter_python\intern tasks\WellNest\backend\controllers\sessionsController.js

```
    const userId = req.user._id; //  auth middleware sets req.user.id
    const { id } = req.params; // Session ID from the URL parameter

    try {
      const session = await Session.findById(id);

      if (!session) {
        return res.status(404).json({ message: 'Session not found.' });
      }

      let updatedSession;

      // Check if the user has already liked this session
      if (session.likedBy.includes(userId)) {
        // If already liked, UNLIKE IT: Decrement likes and remove user ID
        updatedSession = await Session.findByIdAndUpdate(
          id,
          {
            $inc: { likes: -1 },         // Atomically decrements the 'likes' count by 1
            $pull: { likedBy: userId }  // Removes the user's ID from the 'likedBy' array
          },
          {
            new: true,
            runValidators: true
          }
        ).populate('user_id', 'name email');;
        return res.status(200).json({ message: 'Session unliked successfully.', session: updatedSession });
      } else {
        // If not liked, LIKE IT: Increment likes and add user ID
        updatedSession = await Session.findByIdAndUpdate(
          id,
          {
            $inc: { likes: 1 },         // Atomically increments the 'likes' count by 1
            $push: { likedBy: userId }  // Adds the user's ID to the 'likedBy' array
          },
          {
            new: true,
            runValidators: true
          }
        ).populate('user_id', 'name email');;
        return res.status(200).json({ message: 'Session liked successfully.', session: updatedSession });
      }
    } catch (error) {
      console.error('Error processing like/unlike for session:', error);
```

Figure 9

Output :

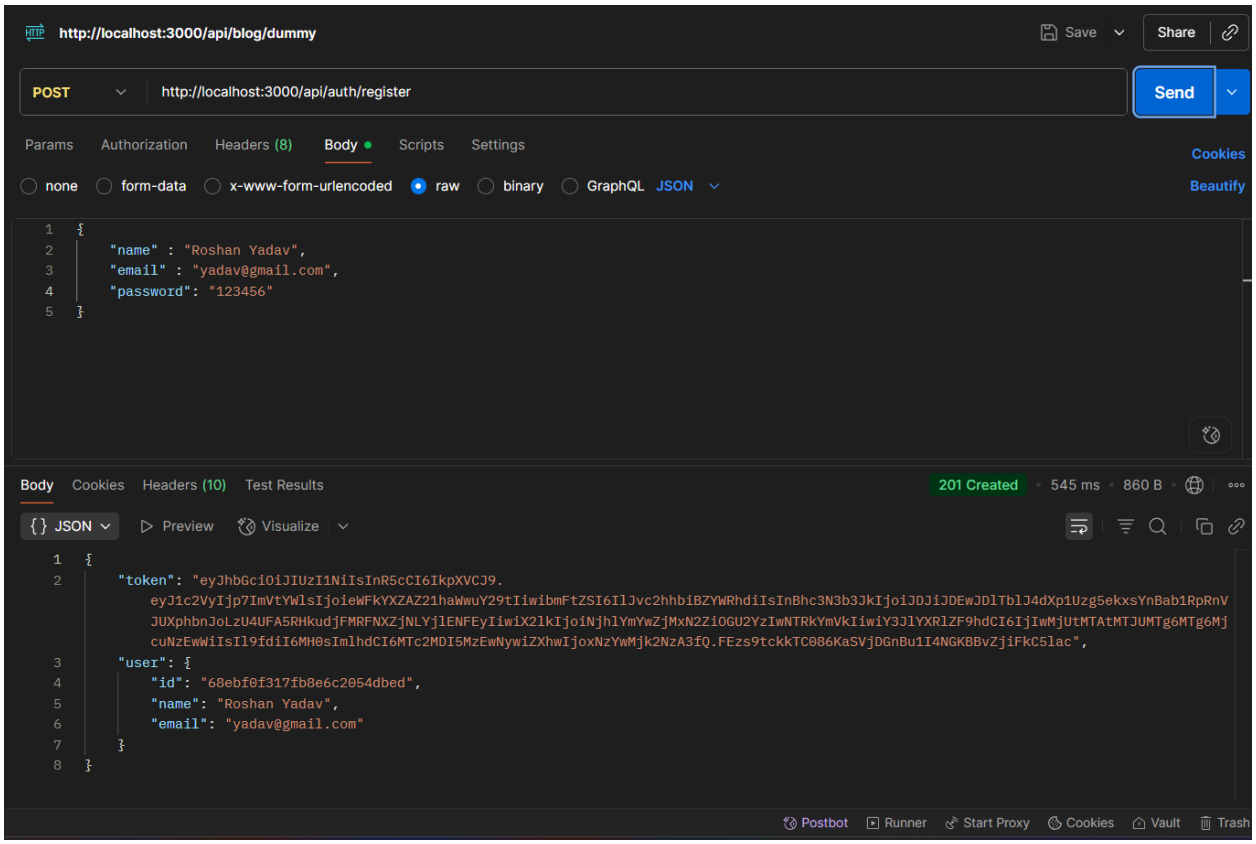

Figure 10

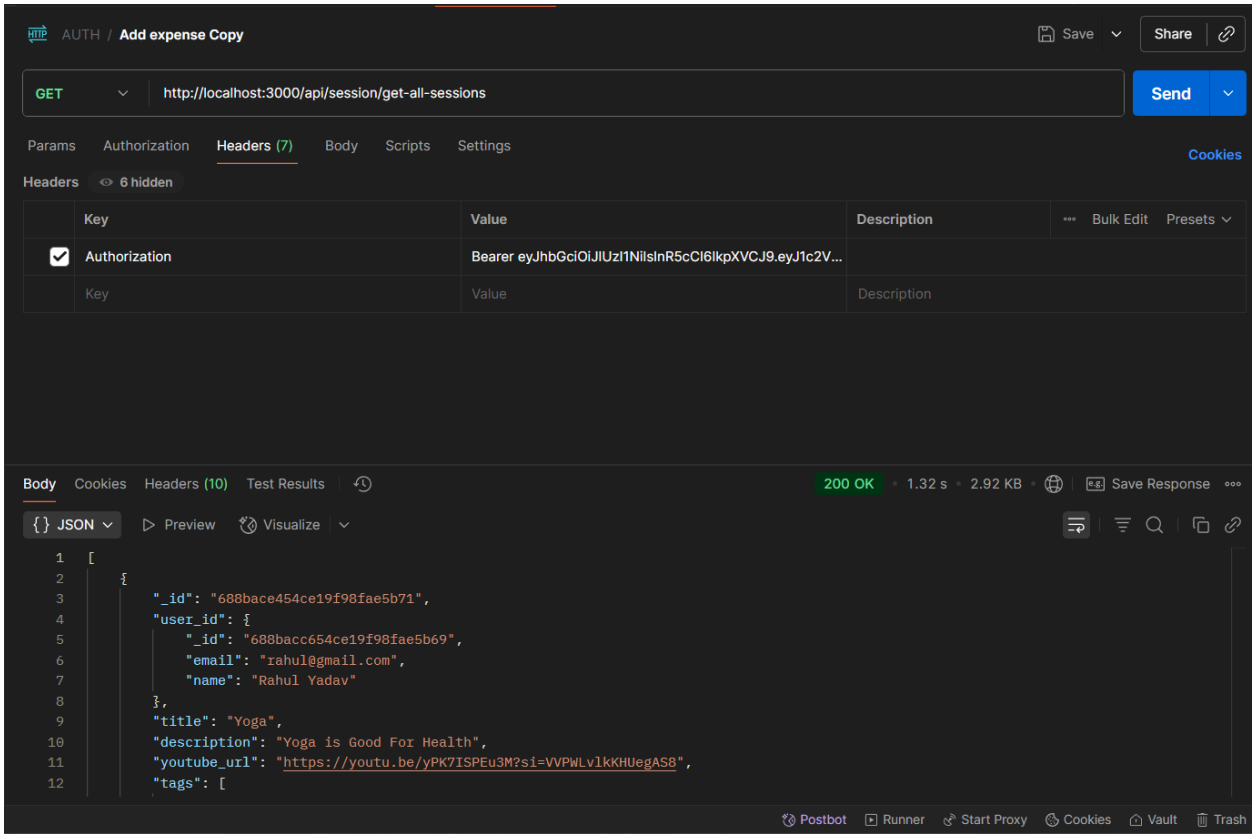
AUTH / Add expense Copy
Save
Share
GET
http://localhost:3000/api/session/get-all-sessions
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Headers
6 hidden
Key
Value
Description
Bulk Edit
Presets
Authorization
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2V...
Key
Value
Description
Body
Cookies
Headers (10)
Test Results
200 OK
1.32 s
2.92 KB
Save Response
JSON
Preview
Visualize
[
{
"_id": "688bace454ce19f98fae5b71",
"user_id": {
"_id": "688bacc654ce19f98fae5b69",
"email": "rahul@gmail.com",
"name": "Rahul Yadav"
},
"title": "Yoga",
"description": "Yoga is Good For Health",
"youtube_url": "https://youtu.be/yPK7ISPEu3M?si=VVPWLvlkKHUegAS8",
"tags": [
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

Figure 11

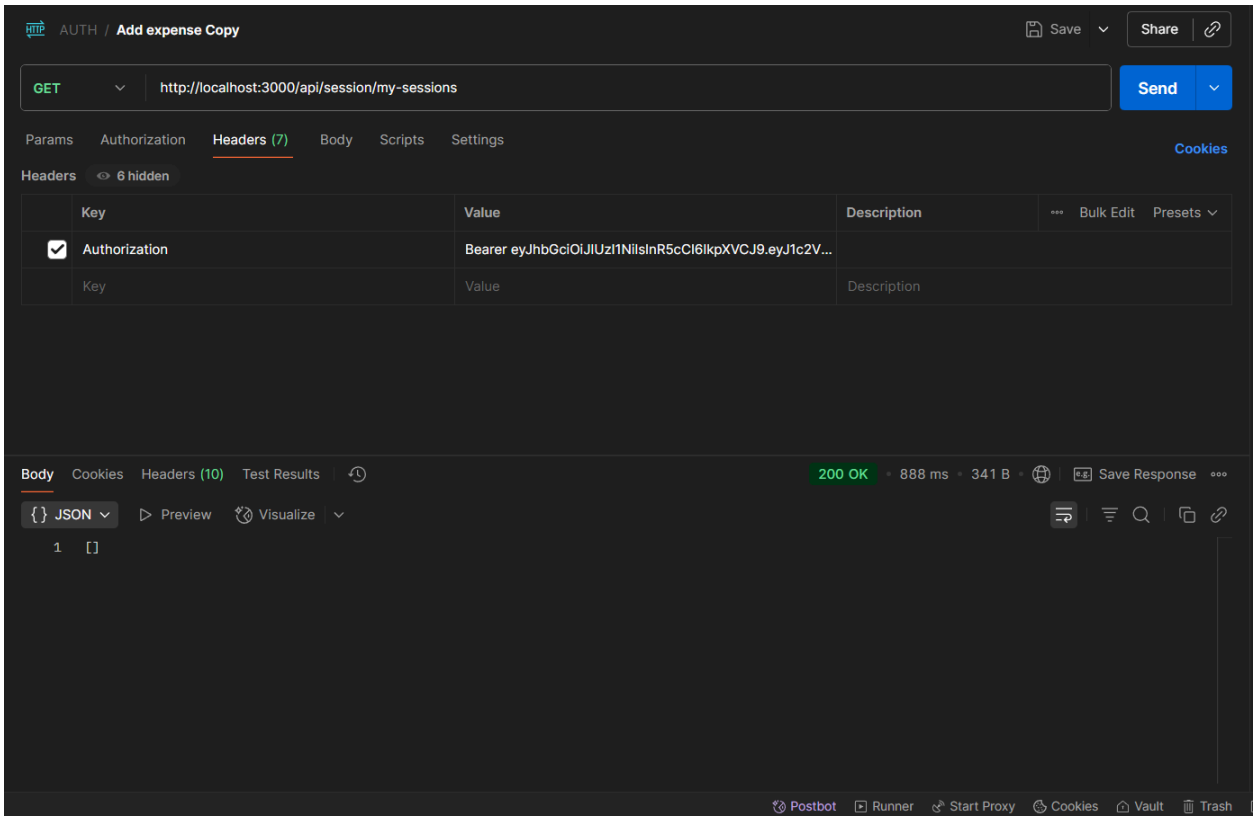
AUTH / Add expense Copy
Save
Share
GET
http://localhost:3000/api/session/my-sessions
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Headers
6 hidden
Key
Value
Description
Bulk Edit
Presets
Authorization
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2V...
Key
Value
Description
Body
Cookies
Headers (10)
Test Results
200 OK
888 ms
341 B
Save Response
JSON
Preview
Visualize
1 []
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

Figure 12