


## **Experiment No - 05**

Roll No.	52
Name	Kshitij Nangare
Class	D15B
Subject	Full Stack Development
Lab Outcome	L1, L2, L3
Date of Performance / Submission	18/08/2025 22/09/2025
Signature & Grades	

## Experiment 5

**Aim : Create secure, production-ready RESTful APIs**

**Code:**

```
backend >  app.js > ...
 1  import dotenv from 'dotenv'; 7.2k (gzipped: 3.1k)
 2  import express from 'express'; Calculating...
 3  import passport from 'passport'; 10k (gzipped: 3.1k)
 4  import cookieParser from 'cookie-parser'; 5k (gzipped: 2k)
 5  import cors from 'cors'; 5k (gzipped: 2.1k)
 6  import {connectDB} from './config/db.js';
 7
 8  import authRoutes from './routes/authRoutes.js';
 9  import userRoutes from './routes/userRoutes.js';
10  import projectRoutes from './routes/projectRoutes.js';
11
12  dotenv.config();
13
14  const app = express();
15  connectDB();
16
17  // Middleware
18  app.use(cors({ origin: process.env.FRONTEND_URL, credentials: true }));
19  app.use(cookieParser());
20  app.use(express.json());
21
22  app.use('/api/auth', authRoutes);
23  app.use('/api/admin', userRoutes);
24  app.use('/api', projectRoutes);
25
26  // Server
27  const PORT = process.env.PORT || 3000;
28  app.listen(PORT, () => {
29    console.log(`Server running on port ${PORT}`);
30  });
31
```

**Figure 1**

```
backend > routes > JS authRoutes.js > ...
 1  import express from 'express';
 2  import { getCurrentUser, login } from '../controllers/authController.js';
 3  import { authenticate } from '../middlewares/auth.js';
 4  import { updatePassword } from '../controllers/userController.js';
 5
 6  const router = express.Router();
 7
 8  router.post('/login', login);
 9
10  router.get('/me', authenticate , getCurrentUser);
11
12  // Update password
13  router.put('/users/password', authenticate, updatePassword);
14
15  export default router;
```

**Figure 2**

```
backend > routes > projectRoutes.js > ...
17
18 const router = express.Router();
19
20 // Get available users (admin/lead only)
21 router.get('/projects/available-users', getAvailableUsers);
22
23 // Get all projects (for authenticated users)
24 router.get('/projects', authenticate, getProjects);
25
26 // Get single project by ID
27 router.get('/projects/:id', authenticate, getProject);
28
29 // Create project (admin only)
30 router.post('/projects', authenticate, authorizeRoles('admin'), createProject);
31
32 // Update project (admin/lead only)
33 router.put('/projects/:id', authenticate, authorizeRoles('admin', 'lead'), updateProject);
34
35 // Delete project (admin only)
36 router.delete('/projects/:id', authenticate, authorizeRoles('admin'), deleteProject);
37
38 // Get all documents for a project (without binary data)
39 router.get('/projects/:projectId/documents', authenticate, getProjectDocuments);
40
41 // Download a specific document (with binary data)
42 router.get('/projects/:projectId/documents/:docId/download', authenticate, downloadDocument);
43
44 // Upload document link to project
45 router.post(
46   '/projects/:projectId/documents/link',
47   authenticate,
48   authorizeRoles('admin', 'lead'),
49   uploadDocumentLink
50 );
51
52
53 // Upload document to project
54 router.post(
55   '/projects/:projectId/documents',
56   authenticate,
57   authorizeRoles('admin', 'lead'),
58   upload.single('document'),
59   uploadDocument
60 );
```

Figure 3

backend > routes >  userRoutes.js > ...

```
1  import express from 'express';
2  import { authenticate, authorizeRoles } from '../middlewares/auth.js';
3  import {
4    getUsers,
5    getUser,
6    createUser,
7    updateUser,
8    deleteUser,
9    updatePassword
10 } from '../controllers/userController.js';
11
12 const router = express.Router();
13
14 // Get all users (admin only)
15 router.get('/users', authenticate, authorizeRoles('admin'), getUsers);
16
17 // Create a new user (admin only)
18 router.post('/users', authenticate, authorizeRoles('admin'), createUser);
19
20 // Get a single user by ID (admin only)
21 router.get('/users/:id', authenticate, authorizeRoles('admin'), getUser);
22
23 // Update a user by ID (admin only)
24 router.put('/users/:id', authenticate, authorizeRoles('admin'), updateUser);
25
26 // Delete a user by ID (admin only)
27 router.delete('/users/:id', authenticate, authorizeRoles('admin'), deleteUser);
28
29 export default router;
30
```

```

backend > controllers > authController.js > ...
1  import User from '../models/User.js';
2  import bcrypt from 'bcryptjs'; 20.1k (gzipped: 9k)
3  import { generateToken } from '../utils/jwtToken.js';
4
5  export const login = async (req, res) => {
6    try {
7      const { email, password } = req.body;
8
9      // 1. Check if email and password exist
10     if (!email || !password) {
11       return res.status(400).json({
12         status: 'fail',
13         message: 'Please provide email and password'
14       });
15     }
16
17     // 2. Check if user exists && password is correct
18     const user = await User.findOne({ email });
19
20     // for first time login someone with new mongoDB URL
21     if(email == "admin@gmail.com" && password == "admin" && !user) {
22       const user = {
23         name: "Admin",
24         email: "admin@gmail.com:",
25         role: "admin",
26       };
27       const token = generateToken(user);
28       res.status(200).json({
29         status: 'success',
30         token,
31         data: {
32           user
33         }
34       });
35     }
36
37     if (!user || !(await bcrypt.compare(password, user.password))) {
38       return res.status(401).json({
39         status: 'fail',
40         message: 'Incorrect email or password'
41       });
42     }
43
44     if (!user ) {
45       return res.status(401).json({

```

**Figure 4**

backend > controllers > `authController.js` > ...

```
5   export const login = async (req, res) => {
51   // 3. If everything ok, send token to client
52   const token = generateToken(user);
53
54   // 4. Remove password from output
55   user.password = undefined;
56
57   // 5. Send response with token
58   res.status(200).json({
59     status: 'success',
60     token,
61     data: {
62       user
63     }
64   });
65
66   } catch (err) {
67     res.status(500).json({
68       status: 'error',
69       message: 'Something went wrong! Please try again later.'
70     });
71   }
72 };
73
74 export const getCurrentUser = async (req, res) => {
75   try {
76     const user = req.user; // User is set by the authenticate middleware
77
78     if (!user) {
79       return res.status(404).json({
80         status: 'fail',
81         message: 'User not found'
82       });
83     }
84
85     // Remove password from output
86     user.password = undefined;
87
88     res.status(200).json({
89       status: 'success',
90       data: {
91         user
92       }
93     });
94   } catch (err) {
95     res.status(500).json({
96       status: 'error',
97       message: 'Something went wrong! Please try again later.'
98     });
99   }
100 }
```

backend > controllers >  projectController.js > ...

```
72 export const getProjects = async (req, res) => {
73   try {
74     let query = {};
75     const { role, _id: userId } = req.user;
76
77     // Filter projects based on user role
78     if (role === 'developer') {
79       query = {
80         'team.userId': userId,
81         status: 'active' // Only active projects
82       };
83     } else if (role === 'lead') {
84       query = {
85         $and: [
86           { $or: [{ lead: userId }, { 'team.userId': userId }] },
87           { status: 'active' } // Only active projects
88         ]
89       };
90     }
91
92     const projects = await Project.find(query)
93       .populate('lead', 'name email')
94       .populate('team.userId', 'name email role')
95       .sort('-createdAt');
96
97     // Get document counts for each project
98     const projectsWithDocCount = await Promise.all(
99       projects.map(async (proj) => {
100         const docCount = await Document.countDocuments({ project: proj._id });
101         return { ...proj.toObject(), documentCount: docCount };
102       })
103     );
104
105     res.status(200).json({
106       status: 'success',
107       results: projectsWithDocCount.length,
108       data: projectsWithDocCount,
109     });
110   } catch (err) {
111     res.status(500).json({
112       status: 'error',
113       message: 'Failed to fetch projects'
114     });
115   }
116 };
```



Figure 5

```
backend > controllers > projectController.js > ...
118 // @desc   Get single project
119 // @route   GET /api/projects/:id
120 // @access  Private
121 export const getProject = async (req, res) => {
122   try {
123     const project = await Project.findById(req.params.id)
124       .populate('lead', 'name email')
125       .populate('team.userId', 'name email role');
126
127     if (!project) {
128       return res.status(404).json({
129         status: 'fail',
130         message: 'Project not found'
131       });
132     }
133
134     // Check if user has access to this project
135     const { role, _id: userId } = req.user;
136     const isTeamMember = project.team.some(member => member.userId.equals(userId));
137
138     if (role !== 'admin' && !project.lead.equals(userId) && !isTeamMember) {
139       return res.status(403).json({
140         status: 'fail',
141         message: 'You do not have permission to view this project'
142       });
143     }
144
145     res.status(200).json({
146       status: 'success',
147       data: project
148     });
149   } catch (err) {
150     res.status(500).json({
151       status: 'error',
152       message: 'Failed to fetch project'
153     });
154   }
155 };
156
```

Figure 6

backend > controllers > JS projectController.js > ...

```
236 export const deleteProject = async (req, res) => {
237   try {
238     const project = await Project.findById(req.params.id);
239     if (!project) {
240       return res.status(404).json({
241         status: 'fail',
242         message: 'Project not found'
243       });
244     }
245
246     // Delete all related documents first
247     await Document.deleteMany({ project: req.params.id });
248
249     // Delete the project
250     await Project.findByIdAndDelete(req.params.id);
251
252     res.status(204).json({
253       status: 'success',
254       data: null
255     });
256   } catch (err) {
257     console.error(err);
258     res.status(500).json({
259       status: 'error',
260       message: 'Failed to delete project'
261     });
262   }
263 };
264
265 // @desc    Get users available for project (leads and developers)
266 // @route    GET /api/projects/available-users
267 // @access   Private (Admin/Lead)
268 export const getAvailableUsers = async (req, res) => {
269   try {
270     const users = await User.find({
271       role: { $in: ['lead', 'developer'] }
272     }).select('name email role');
273
274     res.status(200).json({
275       status: 'success',
276       data: users
277     });
278   } catch (err) {
279     console.error('Failed to fetch available users', err);
280     res.status(500).json({
```

```

backend > controllers > projectController.js > ...
7 // @success Private (name, lead)
8 export const createProject = async (req, res) => {
9   try {
10     const { name, description, deadline, status, lead, team } = req.body;
11
12     // Validate required fields
13     if (!name || !description || !deadline || !lead) {
14       return res.status(400).json({
15         status: 'fail',
16         message: 'Please provide name, description, deadline, and lead'
17       });
18     }
19
20     // Check if lead exists
21     const leadUser = await User.findById(lead);
22     if (!leadUser) {
23       return res.status(400).json({
24         status: 'fail',
25         message: 'Lead user not found'
26       });
27     }
28
29     // Check if team members exist
30     if (team && team.length > 0) {
31       const teamMembers = await User.find({
32         _id: { $in: team.map(member => member.userId) }
33       });
34       if (teamMembers.length !== team.length) {
35         return res.status(400).json({
36           status: 'fail',
37           message: 'One or more team members not found'
38         });
39       }
40     }
41
42     // Create project
43     const project = await Project.create({
44       name,
45       description,
46       deadline,
47       status: status || 'active',
48       lead,
49       team
50     });
51
52     // Populate lead and team for response

```

Figure 7

backend > controllers > JS userController.js > ...

```
7   export const getUsers = async (req, res) => {
8     try {
9       const users = await User.find().select('-password').sort({ name: 1 }); ;
10
11      res.status(200).json({
12        status: 'success',
13        results: users.length,
14        data: {
15          users
16        }
17      });
18    } catch (err) {
19      res.status(500).json({
20        status: 'error',
21        message: 'Failed to fetch users'
22      });
23    }
24  };
25
26  // @desc    Get single user
27  // @route    GET /api/users/:id
28  // @access   Private/Admin
29  export const getUser = async (req, res) => {
30    try {
31      const user = await User.findById(req.params.id).select('-password');
32
33      if (!user) {
34        return res.status(404).json({
35          status: 'fail',
36          message: 'User not found'
37        });
38      }
39
40      res.status(200).json({
41        status: 'success',
42        data: {
43          user
44        }
45      });
46    } catch (err) {
47      res.status(500).json({
48        status: 'error',
49        message: 'Failed to fetch user'
50      });
51    }
52  }
53 }
```

```
backend > controllers > JS userController.js > ...
56 // @access Private/Admin
57 export const createUser = async (req, res) => {
58   try {
59     const { name, email, role } = req.body;
60     const password = role
61     // 1. Check if email already exists
62     const existingUser = await User.findOne({ email });
63     if (existingUser) {
64       return res.status(400).json({
65         status: 'fail',
66         message: 'Email already in use'
67       });
68     }
69
70     // 2. Hash password
71     const hashedPassword = await bcrypt.hash(password, 12);
72
73     // 3. Create user
74     const newUser = await User.create({
75       name,
76       email,
77       password: hashedPassword,
78       role
79     });
80
81     // 4. Remove password from output
82     newUser.password = undefined;
83
84     res.status(201).json({
85       status: 'success',
86       data: {
87         user: newUser
88       }
89     });
90   } catch (err) {
91     res.status(500).json({
92       status: 'error',
93       message: 'Failed to create user'
94     });
95   }
96 };
97
```

Figure 8

```

backend > controllers > JS userController.js > ...
98 // @desc    update user
99 // @route    PUT /api/users/:id
100 // @access   Private/Admin
101 export const updateUser = async (req, res) => {
102   try {
103     const { name, email, role } = req.body;
104     const fieldsToUpdate = { name, email, role };
105
106
107     const updatedUser = await User.findByIdAndUpdate(
108       req.params.id,
109       fieldsToUpdate,
110       {
111         new: true,
112         runValidators: true
113       }
114     ).select('-password');
115
116     if (!updatedUser) {
117       return res.status(404).json({
118         status: 'fail',
119         message: 'User not found'
120       });
121     }
122
123     res.status(200).json({
124       status: 'success',
125       data: {
126         user: updatedUser
127       }
128     });
129   } catch (err) {
130     res.status(500).json({
131       status: 'error',
132       message: 'Failed to update user'
133     });
134   }
135 };
136

```

Figure 9

```
backend > controllers > JS userController.js > ...
136
137
138
139 export const updatePassword = async (req, res) => {
140   const { currentPassword, newPassword } = req.body;
141   if (!currentPassword || !newPassword) {
142     return res.status(400).json({ message: 'Old password and new password are required' });
143   }
144
145   try {
146     const user = await User.findById(req.user._id);
147     if (!user) {
148       return res.status(404).json({ message: 'User not found' });
149     }
150
151     // Compare old password with stored hash
152     const isMatch = await bcrypt.compare(currentPassword, user.password);
153     if (!isMatch) {
154       return res.status(400).json({ message: 'Incorrect old password' });
155     }
156
157     // Hash new password and update
158     const hashedNewPassword = await bcrypt.hash(newPassword, 12);
159     user.password = hashedNewPassword;
160     await user.save();
161
162     res.status(200).json({ message: 'Password updated successfully' });
163   } catch (error) {
164     console.error('Password update error:', error);
165     res.status(500).json({ message: 'Server error' });
166   }
167 };
168
```

Figure 10

```
backend > controllers > js userController.js > ...
168
169
170 // @desc    Delete user
171 // @route    DELETE /api/users/:id
172 // @access   Private/Admin
173 export const deleteUser = async (req, res) => {
174   try {
175     // Prevent admin from deleting themselves
176     if (req.user.id === req.params.id) {
177       return res.status(400).json({
178         status: 'fail',
179         message: 'You cannot delete yourself'
180       });
181     }
182
183     const user = await User.findByIdAndDelete(req.params.id);
184
185     if (!user) {
186       return res.status(404).json({
187         status: 'fail',
188         message: 'User not found'
189       });
190     }
191
192     res.status(204).json({
193       status: 'success',
194       data: null
195     });
196   } catch (err) {
197     res.status(500).json({
198       status: 'error',
199       message: 'Failed to delete user'
200     });
201   }
202 };
```

Figure 11



## Output :

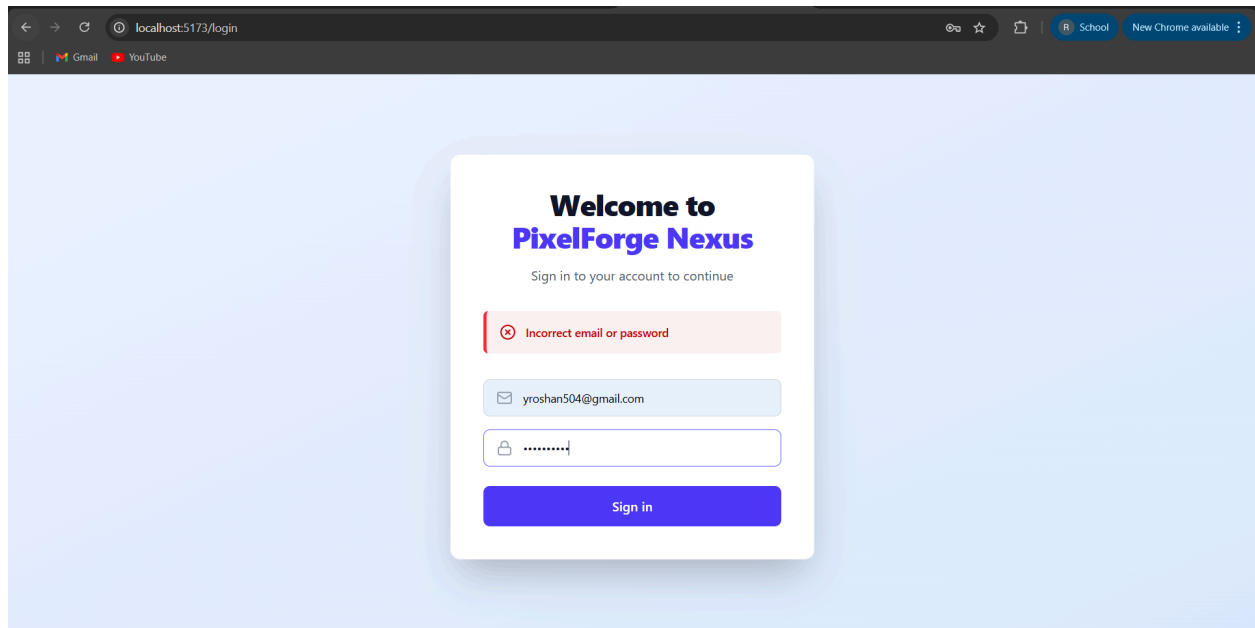


Figure 12

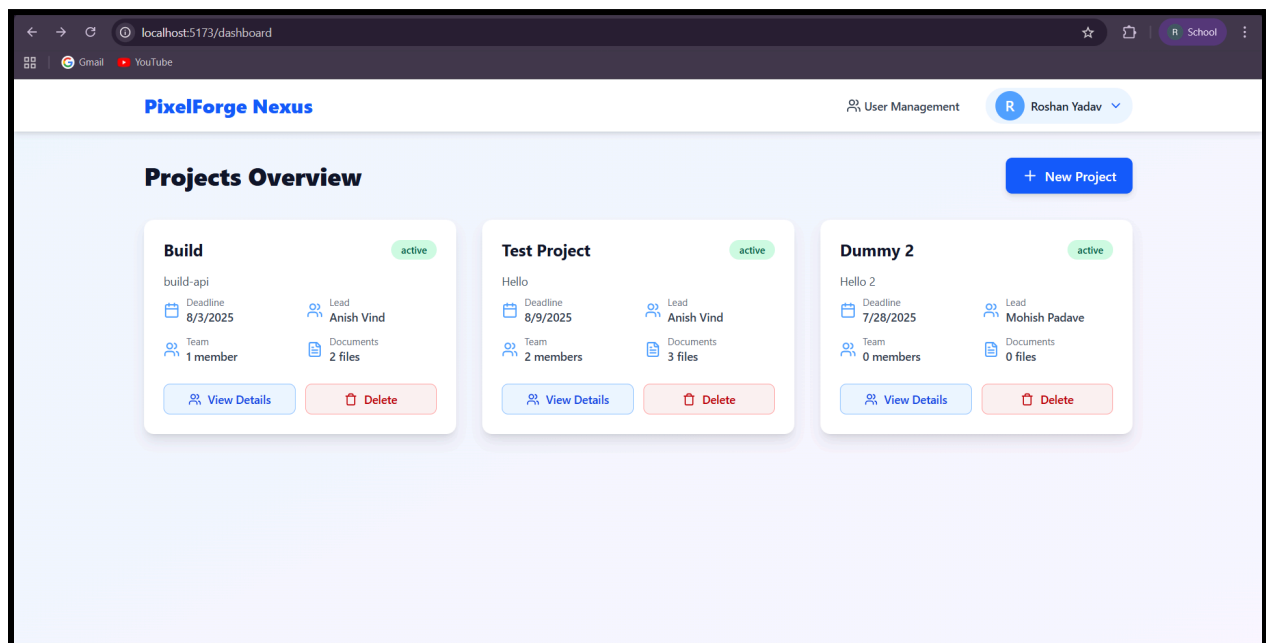


Figure 13

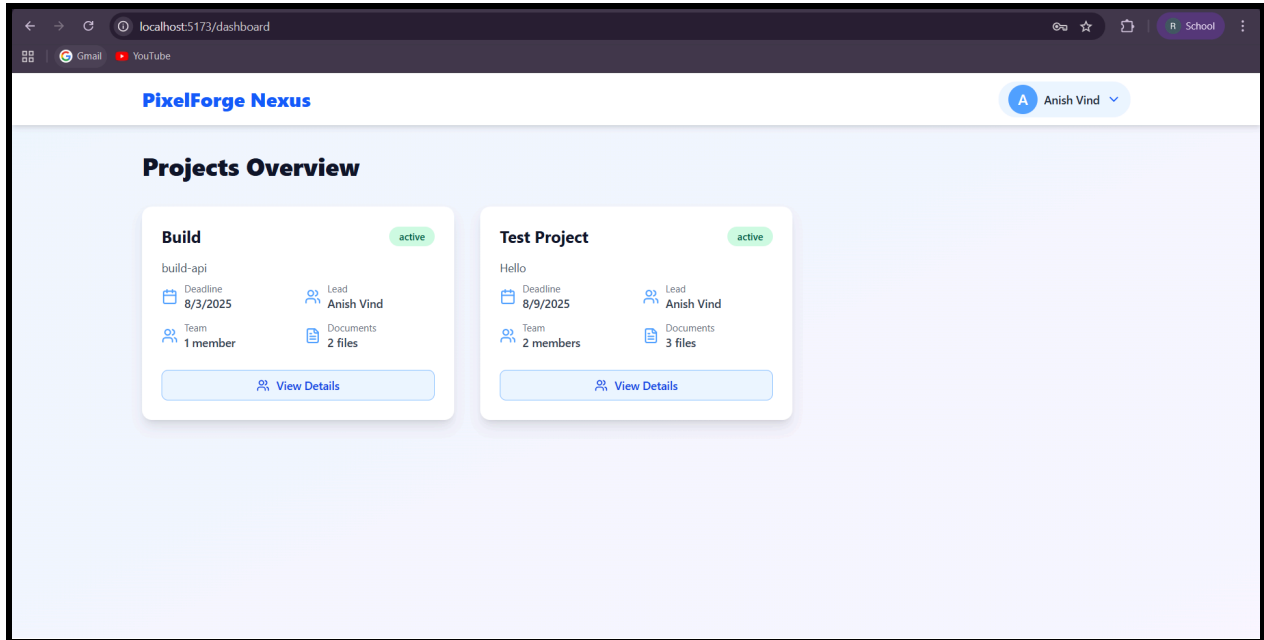


Figure 14