

**Laboratory Record**  
**of : EAD**

**Roll No.:**

**Exp No.:** \_\_\_\_\_

**Sheet No.:** \_\_\_\_\_

**Date:** \_\_\_\_\_

**Aim: Create Jwt for authentication register and login .**

Create .env at project root:

PORT=4000

# Generate a long random secret once (do NOT commit to Git)

# In a terminal: node -e "console.log(require('crypto').randomBytes(64).toString('hex'))"

JWT\_SECRET=replace\_with\_a\_long\_random\_string

JWT\_EXPIRES\_IN=15m

Create src/utils/env.js to load env safely:

import dotenv from 'dotenv';

dotenv.config();

export const env = {

PORT: process.env.PORT || 4000,

JWT\_SECRET: process.env.JWT\_SECRET,

JWT\_EXPIRES\_IN: process.env.JWT\_EXPIRES\_IN || '15m',

};

if (!env.JWT\_SECRET) {

console.error('Missing JWT\_SECRET in .env');

process.exit(1);

}

Note: This lab uses in-memory storage for users to keep things simple. In a real app, use a database (e.g., MongoDB + Mongoose or PostgreSQL + Prisma).

#### 6) Minimal Express App

src/server.js

import express from 'express';

import cors from 'cors';

import { env } from './utils/env.js';

import authRouter from './routes/auth.routes.js';

const app = express();

app.use(cors());

app.use(express.json());

app.get('/', (req, res) => {

res.json({ message: 'JWT Lab API is running' });

});

app.use('/api/auth', authRouter);

app.use((req, res) => {

**Laboratory Record**  
**of : EAD**

**Roll No.:**

**Exp No.:** \_\_\_\_\_

**Sheet No.:** \_\_\_\_\_

**Date:** \_\_\_\_\_

```
res.status(404).json({ error: 'Route not found' });
});
```

```
app.listen(env.PORT, () => {
  console.log(`Server listening on http://localhost:${env.PORT}`);
});
```

---

**7) Auth Middleware (Protect Routes)**

src/middlewares/verifyToken.js

import jwt from 'jsonwebtoken';

import { env } from '../utils/env.js';

```
export function verifyToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  if (!authHeader) return res.status(401).json({ error: 'Missing Authorization header' });

  const [scheme, token] = authHeader.split(' ');
  if (scheme !== 'Bearer' || !token) {
    return res.status(401).json({ error: 'Invalid Authorization format. Use Bearer <token>' });
  }
}
```

```
try {
  const payload = jwt.verify(token, env.JWT_SECRET);
  req.user = payload; // e.g., { sub, email, iat, exp }
  next();
} catch (err) {
  console.error('JWT verify error:', err.message);
  return res.status(401).json({ error: 'Invalid or expired token' });
}
}
```

**8) Auth Routes (Register, Login, Protected)**

src/routes/auth.routes.js

import { Router } from 'express';

import bcrypt from 'bcryptjs';

import jwt from 'jsonwebtoken';

import { env } from '../utils/env.js';

import { verifyToken } from '../middlewares/verifyToken.js';

const router = Router();

*// In-memory users array for the lab*

*// Each user: { id, name, email, passwordHash }*

**Laboratory Record**  
**of : EAD**

**Roll No.:**

**Exp No.:** \_\_\_\_\_

**Sheet No.:** \_\_\_\_\_

**Date:** \_\_\_\_\_

```
const users = [];
let idCounter = 1;

function makeToken(user) {
  // Standard claims: sub (subject), email, iat, exp (set by sign options)
  const payload = { sub: user.id, email: user.email, name: user.name };
  return jwt.sign(payload, env.JWT_SECRET, { expiresIn: env.JWT_EXPIRES_IN });
}

// @route POST /api/auth/register
// @body { name, email, password }
router.post('/register', async (req, res) => {
  try {
    const { name, email, password } = req.body;
    if (!name || !email || !password) return res.status(400).json({ error: 'name, email, password are required' });

    const exists = users.find(u => u.email.toLowerCase() === email.toLowerCase());
    if (exists) return res.status(409).json({ error: 'Email already registered' });

    const passwordHash = await bcrypt.hash(password, 10);
    const user = { id: idCounter++, name, email, passwordHash };
    users.push(user);

    const token = makeToken(user);
    res.status(201).json({
      message: 'Registered successfully',
      token,
      user: { id: user.id, name: user.name, email: user.email }
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Server error' });
  }
});

// @route POST /api/auth/login
// @body { email, password }
router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    if (!email || !password) return res.status(400).json({ error: 'email and password are required' });
  }
});
```

## Laboratory Record

of : EAD

Roll No.:

Exp No.: \_\_\_\_\_

Sheet No.: \_\_\_\_\_

Date: \_\_\_\_\_

```
const user = users.find(u => u.email.toLowerCase() === email.toLowerCase());
if (!user) return res.status(401).json({ error: 'Invalid credentials' });
```

```
const ok = await bcrypt.compare(password, user.passwordHash);
if (!ok) return res.status(401).json({ error: 'Invalid credentials' });
```

```
const token = makeToken(user);
res.json({
  message: 'Logged in',
  token,
  user: { id: user.id, name: user.name, email: user.email }
});
} catch (err) {
  console.error(err);
  res.status(500).json({ error: 'Server error' });
}
});
```

```
// @route GET /api/auth/profile (protected)
router.get('/profile', verifyToken, (req, res) => {
  // req.user was set by verifyToken
  const user = users.find(u => u.id === req.user.sub);
  if (!user) return res.status(404).json({ error: 'User not found' });
  res.json({ id: user.id, name: user.name, email: user.email });
});
```

export default router;

### Postman:

#### 1. Register

- Method: POST
- URL: <http://localhost:4000/api/auth/register>
- Body (JSON):

```
{
  "name": "Alice",
  "email": "alice@example.com",
  "password": "Pass@1234"
}
```
- Expect: 201, response includes token.

#### 2. Login

- Method: POST
- URL: <http://localhost:4000/api/auth/login>
- Body (JSON):

## Laboratory Record of : EAD

Roll No.:

Exp No.: \_\_\_\_\_

Sheet No.: \_\_\_\_\_

Date: \_\_\_\_\_

```
{  
  "email": "alice@example.com",  
  "password": "Pass@1234"  
}
```

- Expect: 200, response includes token.

### 3. Access Protected Route

- Method: GET
- URL: `http://localhost:4000/api/auth/profile`
- Header: Authorization: Bearer <paste\_token\_here>
- Expect: 200 with user profile JSON.

### Output:

HTTP `http://localhost:4000/api/auth/register` Save Share

**POST** `http://localhost:4000/api/auth/register` Send

Params Auth Headers (8) **Body** Scripts Tests Settings Cookies Beautify

raw **JSON**

```
1 {  
2   "name": "Alice",  
3   "email": "alice@example.com",  
4   "password": "Pass@1234"  
5 }  
6
```

Body `201 Created` • 220 ms • 577 B •

**JSON** Preview Visualize

```
2 "message": "Registered successfully",  
3 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
  eyJzdWUiOiJEsImVtYWlsIjoiYWxpY2VAZXhhbXBsZS5jb20iLCJuY  
  W1lIjoiQWxpY2UiLCJpYXQiOiJE3NjA2MzA2MTIsImV4cCI6MTc2MD  
  YzMTUxMn0.  
  BUeRdc-UMsnfms5H0a1jhmyqZBZLWrGqCob_GYscNGc",  
4 "user": {  
5   "id": 1,  
6   "name": "Alice",  
7   "email": "alice@example.com"  
8 }  
9 }
```

**Roll No.:** \_\_\_\_\_  
**Exp No.:** \_\_\_\_\_  
**Sheet No.:** \_\_\_\_\_  
**Date:** \_\_\_\_\_

```
body Cookies Headers (8) Test Results 200 OK 83 ms 558 B
{} JSON Preview Visualize
1 {
2   "message": "Logged in".
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJEsImVtYSI6IjoiYXNpY2VhZGhhbXBzSS5ib291CjUyYWIiOiJpXWxpY2U1LCJpcyYXQ1IjE3NjA2MzA2NDMmImV4cCI6MTc2MDYzMTU0M30.
4   "user": {
5     "id": 1,
6     "name": "Alice",
7     "email": "alice@example.com"
8   }
9 }
```

**GET** ☐ `http://localhost:4000/api/auth/profile` **Send** ☐

The image shows the Swagger UI interface. At the top, there are tabs: 'Params', 'Auth', 'Headers (7)', 'Body', 'Scripts', 'Tests', and 'Settings'. The 'Auth' tab is selected and underlined. Below the tabs, on the left, is the 'Auth Type' dropdown menu, which is currently set to 'Bearer Token'. To the right of this is the 'Token' field, which is highlighted with a yellow border. Inside the 'Token' field, there is a series of dots representing a masked token, followed by an eye icon and a lock icon. Below the 'Auth Type' dropdown, there is a text block that reads: 'The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.'

```
Body  [v] [refresh] 200 OK • 14 ms • 318 B • [globe] [three dots]
[JSON] [v] [Preview] [Visualize] [v]
1 {
2   "id": 1,
3   "name": "Alice",
4   "email": "alice@example.com"
5 }
```