Authentication

By: Puskar Adhikari

Introduction

Authentication is the process of **verifying who a user is**.

- It answers the question: *“Are you really who you say you are?”*
- Commonly done via:
  - Username & password
  - Tokens (JWT)
  - OAuth (Google, Facebook login)
  - Biometric methods (fingerprint, face ID)

Why?

Why do we need it?

1. **Protect user data** – Only the right person can access their account.
2. **Secure application resources** – Prevent unauthorized users from performing actions.
3. **Personalized experience** – Show user-specific content after login.

How?

How Authentication Works (Basic Flow)

1. **User submits credentials** (username & password).
2. **Backend verifies** credentials against a database.
3. **If valid**: backend generates a **token** (e.g., JWT) or session.
4. **Frontend stores the token** (localStorage, cookies).
5. **Future requests** include the token to prove the user is authenticated.

Gitlab Repo

Clone this repo:

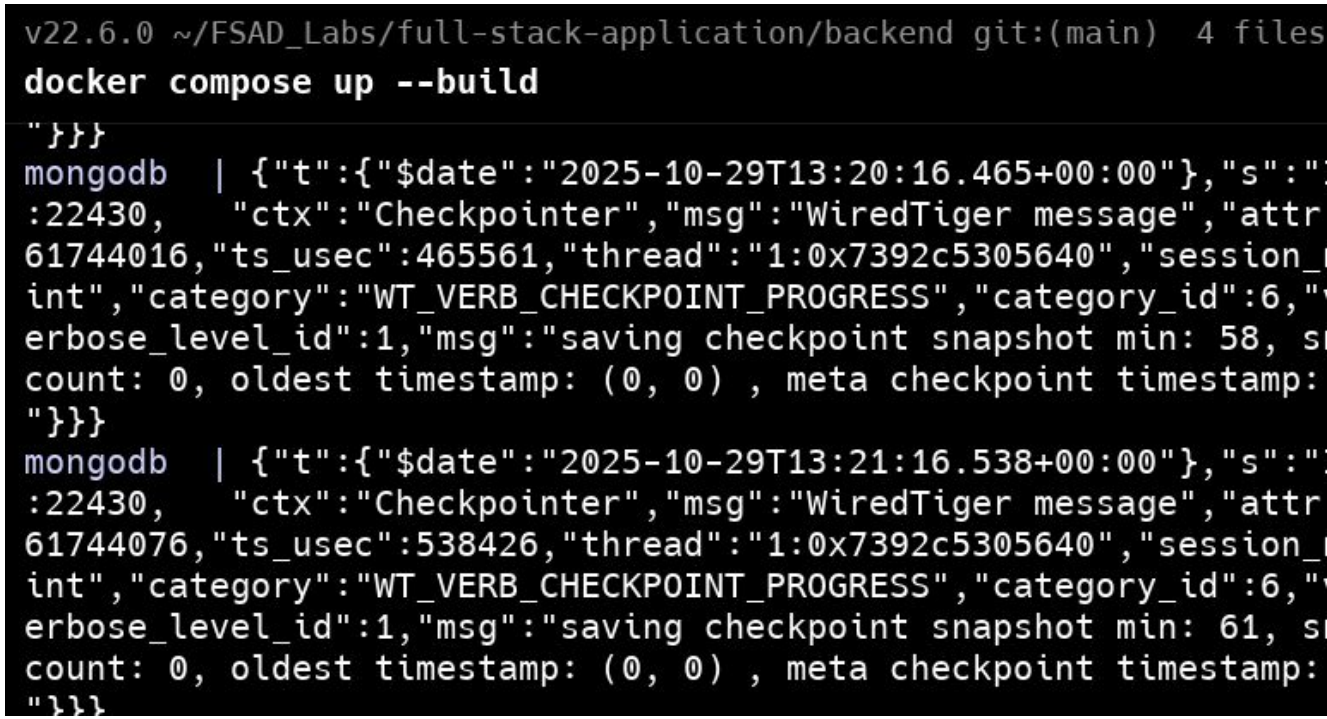
https://gitlab.com/ait-fsad-2025/labreference/fullstackapplication

Project Structure

Running the Application

To run the application, the db and the backend must be up and running first, only then the frontend can be run.

- cd backend/
- docker compose up --build //for first time use



MongoDB

Schema: defines the structure, rules, and behavior of documents.

Model: class based on schema to interact with MongoDB collection.

Mongoose: makes working with MongoDB easier with validation, hooks, and methods.

backend > models > JS user.js > ...

```
1   const mongoose = require('mongoose')
2   const bcrypt = require('bcrypt')
3
4   const userSchema = new mongoose.Schema({
5       email: {type: String, required: true},
6       password: {type: String, required: true}
7   })
8
9   userSchema.pre('save', async function(next) {
10    if (!this.isModified('password')) return next();
11    this.password = await bcrypt.hash(this.password, 10);
12    next();
13  });
14
15  userSchema.methods.comparePassword = async function(password) {
16    return await bcrypt.compare(password, this.password);
17  };
18
19  module.exports = mongoose.model('User', userSchema)
20
```

MongoDB

Schema: defines the structure, rules, and behavior of documents.

Model: class based on schema to interact with MongoDB collection.

Mongoose: makes working with MongoDB easier with validation, hooks, and methods.

backend > models > JS user.js > ...

```
1   const mongoose = require('mongoose')
2   const bcrypt = require('bcrypt')
3
4   const userSchema = new mongoose.Schema({
5       email: {type: String, required: true},
6       password: {type: String, required: true}
7   })
8
9   userSchema.pre('save', async function(next) {
10    if (!this.isModified('password')) return next();
11    this.password = await bcrypt.hash(this.password, 10);
12    next();
13  });
14
15  userSchema.methods.comparePassword = async function(password) {
16    return await bcrypt.compare(password, this.password);
17  };
18
19  module.exports = mongoose.model('User', userSchema)
20
```

/users/register api

- The body will have new email and password.
- Check for existing email
- Save if everything checks out

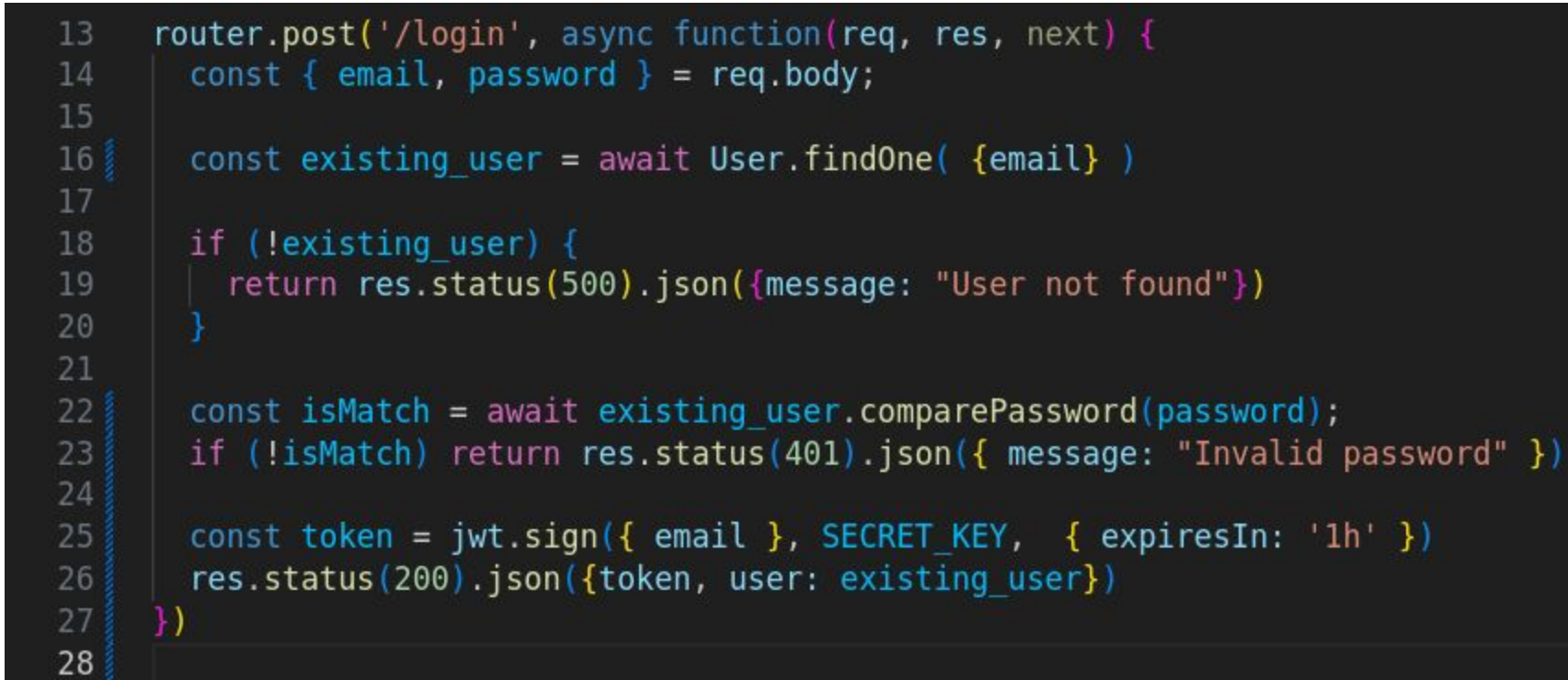
```
router.post('/register', async function(req, res, next) {
  const { email, password } = req.body;

  if (!email || !password) {
    res.status(500).json({message: "Empty fields!"})
  }

  const existing_user = await User.findOne( {email} )

  if (existing_user) {
    return res.status(500).json({message: "Email already in use."})
  }

  const new_user = new User({email, password})
  await new_user.save()

  res.status(200).json({ new_user })
})
```

JWT

What is JWT Authentication?

JWT

When the user does the login, the token should be sent along with other infos to the frontend. The token now tells the frontend the user is there/logged in.



```
13  router.post('/login', async function(req, res, next) {
14    const { email, password } = req.body;
15
16    const existing_user = await User.findOne( {email} )
17
18    if (!existing_user) {
19      return res.status(500).json({message: "User not found"})
20    }
21
22    const isMatch = await existing_user.comparePassword(password);
23    if (!isMatch) return res.status(401).json({ message: "Invalid password" })
24
25    const token = jwt.sign({ email }, SECRET_KEY,  { expiresIn: '1h' })
26    res.status(200).json({token, user: existing_user})
27  })
28
```

Frontend Integration

- Same approach for both react and angular
- Call the users/login api and save the token in localstorage
- While trying to access the **protected** route/link/page, check for token.
- If the token exists, the user is logged in

React

- Since we are not using advanced techniques for state management, lets use useEffect hook to check authentication status.
- In the useEffect hook, check whether there exists token in localstorage to confirm the login status of the user.

```
13  const handleSubmit = async() => {
14    console.log("Login button clicked", email, password)
15    if (!email || !password) {
16      setError('Please fill all the fields.')
17    }
18
19    const res = await fetch('http://localhost:3000/users/login', {
20      method: 'POST',
21      headers: { 'Content-Type': 'application/json' },
22      body: JSON.stringify({ email: email, password: password }),
23    });
24    const data = await res.json();
25
26    if (res.status === 200) {
27      localStorage.setItem('token', data.token)
28      navigator('/')
29    }
30    else {
31      setError('Something went wrong.')
32    }
33  }
```

```
7   function App() {
8     const navigate = useNavigate();
9
10    useEffect(() => {
11      const token = localStorage.getItem('token');
12      if (!token) {
13        navigate('/login');
14      }
15    }, [ navigate ]);
16
```

Angular

- Login Component
- Protect all/any routes
- Middleware
- canActivate
- Implement logout feature