**1.Project Overview**

**World News** is a full-stack web application designed to provide users with the latest news and allow them to interact by posting comments. The application features a modern, clean interface built with EJS templates, and it leverages Express.js for backend routing, MongoDB for data storage, and JWT for secure authentication.

The purpose of the project is to simulate a real-world news platform where:

* Visitors can browse and read news articles.
* Registered users can log in, add comments on news, and interact with others.
* Admins (optionally) can add or manage news posts.

This project was created as part of a university assignment focused on web development using the MERN-like stack (MongoDB, Express, Node.js).

**2. Technologies Used**

The **World News** application uses the following technologies:

**Frontend:**

* **EJS (Embedded JavaScript Templates)** — for rendering dynamic HTML pages.
* **CSS** — custom styles for layout, comments, forms, and responsive design.

**Backend:**

* **Node.js** — JavaScript runtime for server-side development.
* **Express.js** — web framework used for routing and middleware handling.
* **JWT (JSON Web Token)** — for user authentication and securing API endpoints.
* **cookie-parser** — to parse and manage cookies.

**Database:**

* **MongoDB** — NoSQL database for storing users, news articles, and comments.
* **Mongoose** — ODM (Object Data Modeling) library for MongoDB and Node.js.

**Development Tools:**

* **dotenv** — to manage environment variables securely (API keys, secrets).
* **Postman** — for testing API endpoints during development.
* **Nodemon** (optional) — for auto-restarting the server on changes during development.

**3. Features**

The **World News** application provides the following key features:

**User Authentication**

* Register a new user with a **username**, **email**, and **password**.
* Login functionality with **JWT token-based authentication**.
* Protected routes and pages only accessible by logged-in users.

**News Management**

* View a list of all published news articles (publicly accessible).
* Create new news articles (form-based creation).
* View a single news article along with user comments.

**Commenting System**

* Authenticated users can post comments on any news article.
* Comments display the author's name, content, and timestamp.
* All comments are stored in MongoDB and shown in reverse chronological order.

**Responsive & Scrollable UI**

* User-friendly layout using EJS templates and CSS.
* Comment box and list designed with a modern card-style layout.
* Scroll enabled within the comment section to avoid overflow.

**4. Project Structure**

world-news/

│

├── config/

│ └── db.js # MongoDB connection setup

│

├── middleware/

│ └── authMiddleware.js # JWT authentication middleware

│

├── models/

│ ├── User.js # User model schema

│ ├── News.js # News article schema

│ └── Comment.js # Comment schema

│

├── routes/

│ ├── authRoutes.js # Routes for login and registration

│ ├── newsRoutes.js # API routes for managing news

│ └── commentRoutes.js # API routes for posting comments

│

├── views/

│ ├── layout.ejs # Main EJS layout

│ ├── index.ejs # Home page with all news

│ ├── login.ejs # Login form

│ ├── register.ejs # Registration form

│ └── news.ejs # Single news view with comments

│

├── public/

│ └── css/ # Static stylesheets (optional)

│

├── .env # Environment variables (e.g. JWT secret, DB URI)

├── .gitignore # Ignores node\_modules and .env

├── package.json # Dependencies and scripts

└── server.js # Entry point and Express server setup

**5. Technologies Used**

The **World News** project was built using the following technologies:

* **Backend**
  + [Node.js](https://nodejs.org/): JavaScript runtime for server-side development
  + [Express.js](https://expressjs.com/): Web framework for Node.js
  + [MongoDB](https://www.mongodb.com/): NoSQL database
  + [Mongoose](https://mongoosejs.com/): MongoDB object modeling tool
  + [jsonwebtoken](https://github.com/auth0/node-jsonwebtoken): For creating and verifying JWT tokens
  + [dotenv](https://github.com/motdotla/dotenv): Loads environment variables from .env file
  + [cookie-parser](https://github.com/expressjs/cookie-parser): Parses cookies attached to client requests
* **Templating & Views**
  + [EJS](https://ejs.co/): Templating engine for generating HTML with embedded JavaScript
  + [express-ejs-layouts](https://www.npmjs.com/package/express-ejs-layouts): Layout support for EJS templates
* **Frontend**
  + HTML & CSS for the structure and styling of the pages
  + Responsive design and basic interactivity with CSS only
* **Other**
  + MVC architecture (Model-View-Controller)
  + RESTful API design

**6. API Endpoints**

The backend provides several RESTful API endpoints for handling authentication, news, and comments.

**Authentication Endpoints (/api/auth)**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /register | Register a new user |
| POST | /login | Login and receive JWT |

* Requires: username, email, and password
* Returns: JWT token in response

**News Endpoints (/api/news)**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | / | Get all news |
| POST | /create | Create a news article |
| GET | /:id | Get a single news article |

* Requires authentication for POST /create

**Comment Endpoints (/api/comments)**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /add | Add a comment to a news post |

* Requires: newsId, content
* Requires authentication

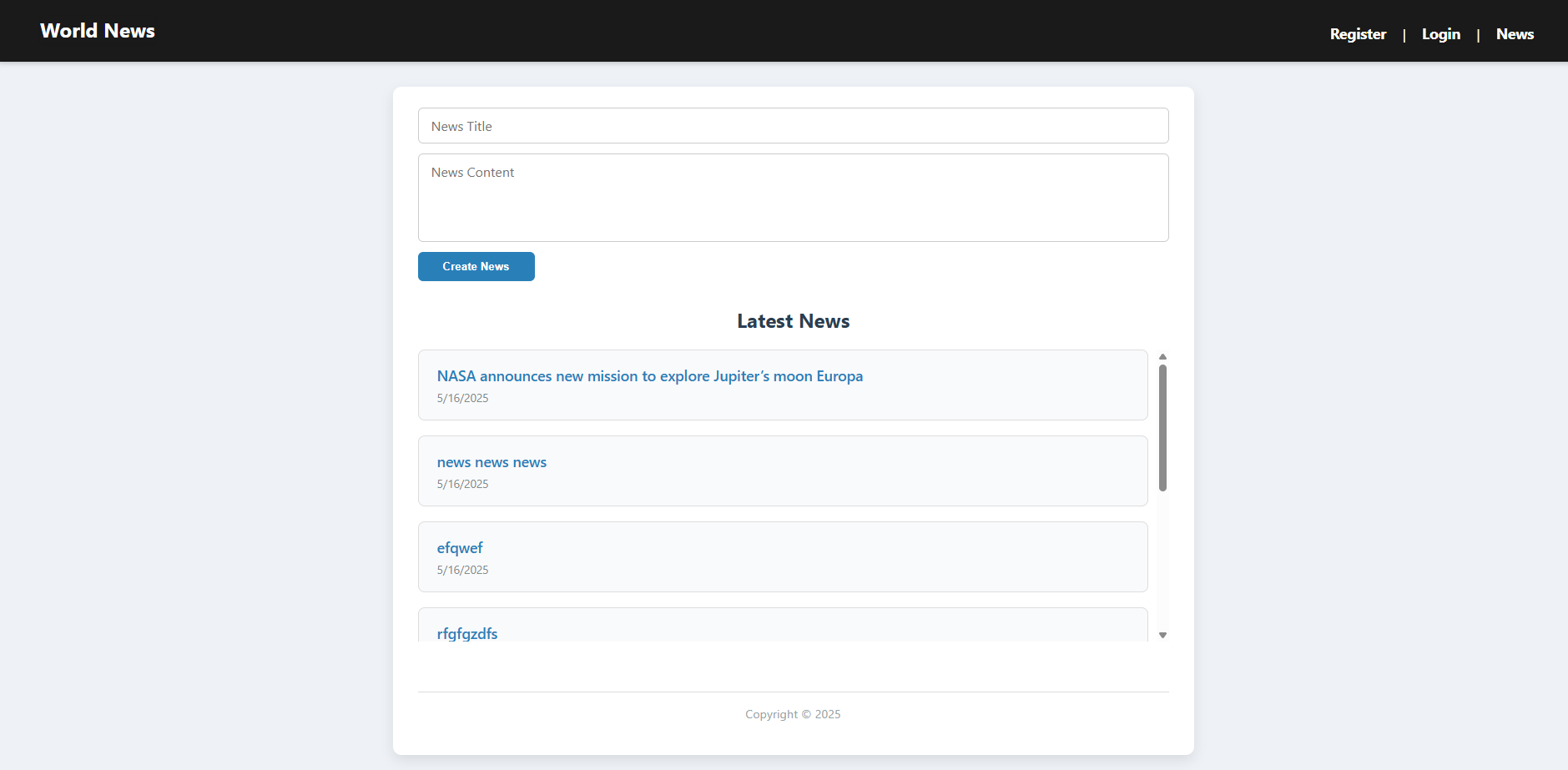
### Example Authenticated Request

To make authenticated requests, you must send a JW token in the Authorization

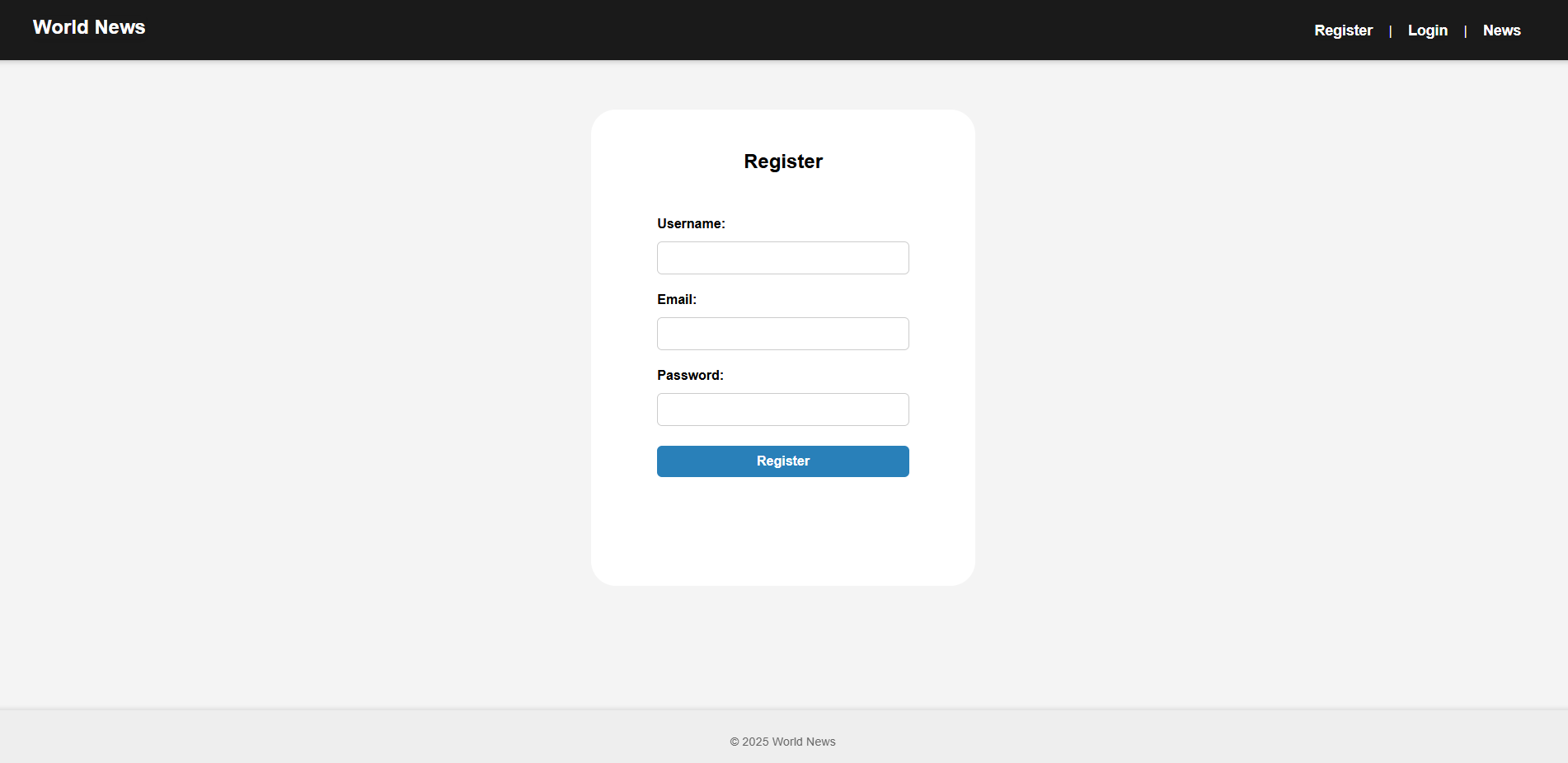
* Header:
* Authorization: Bearer <your\_token\_here>

## **8. Screenshots / Demo**

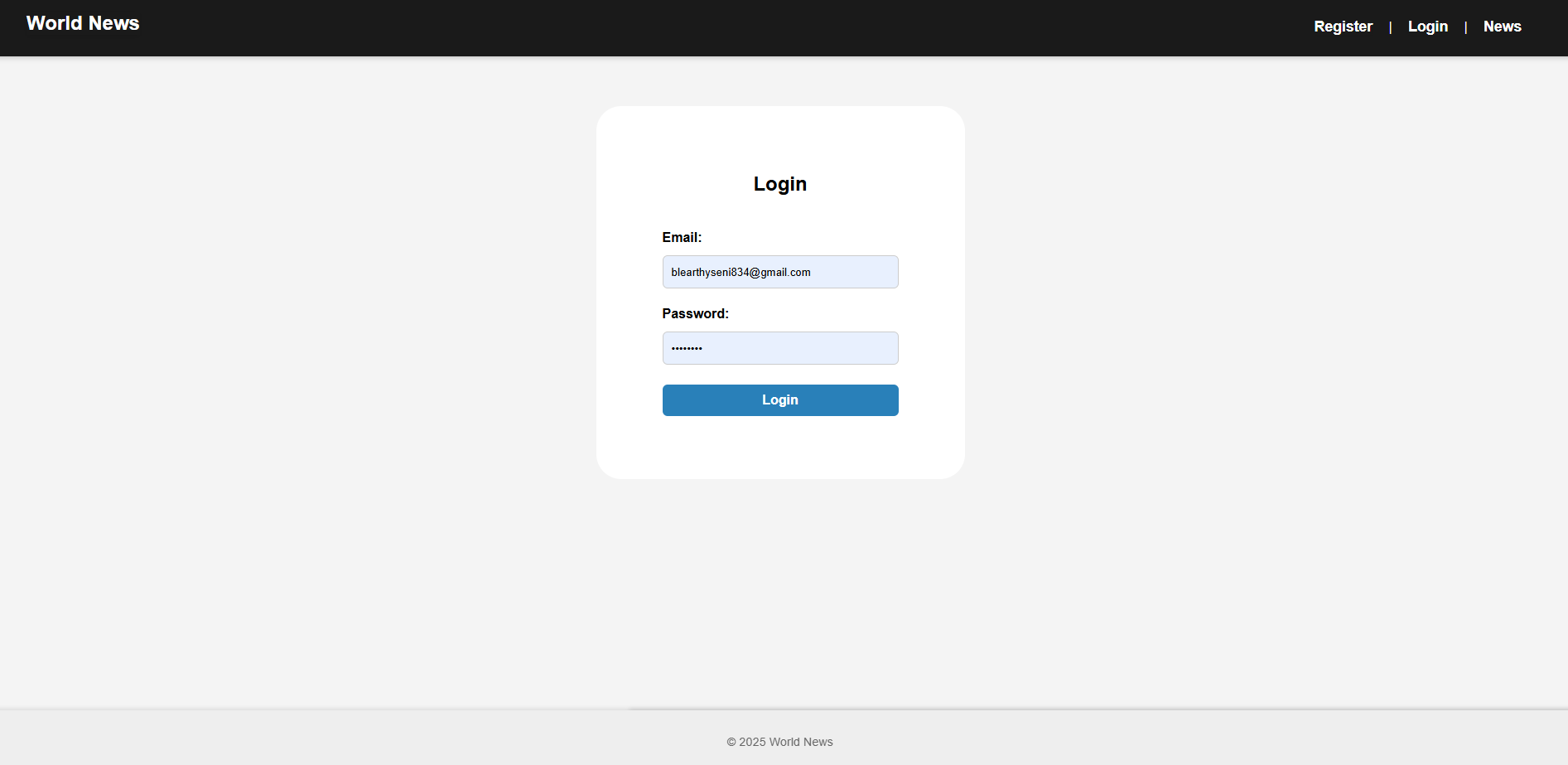
### 1Homepage (News List)

****

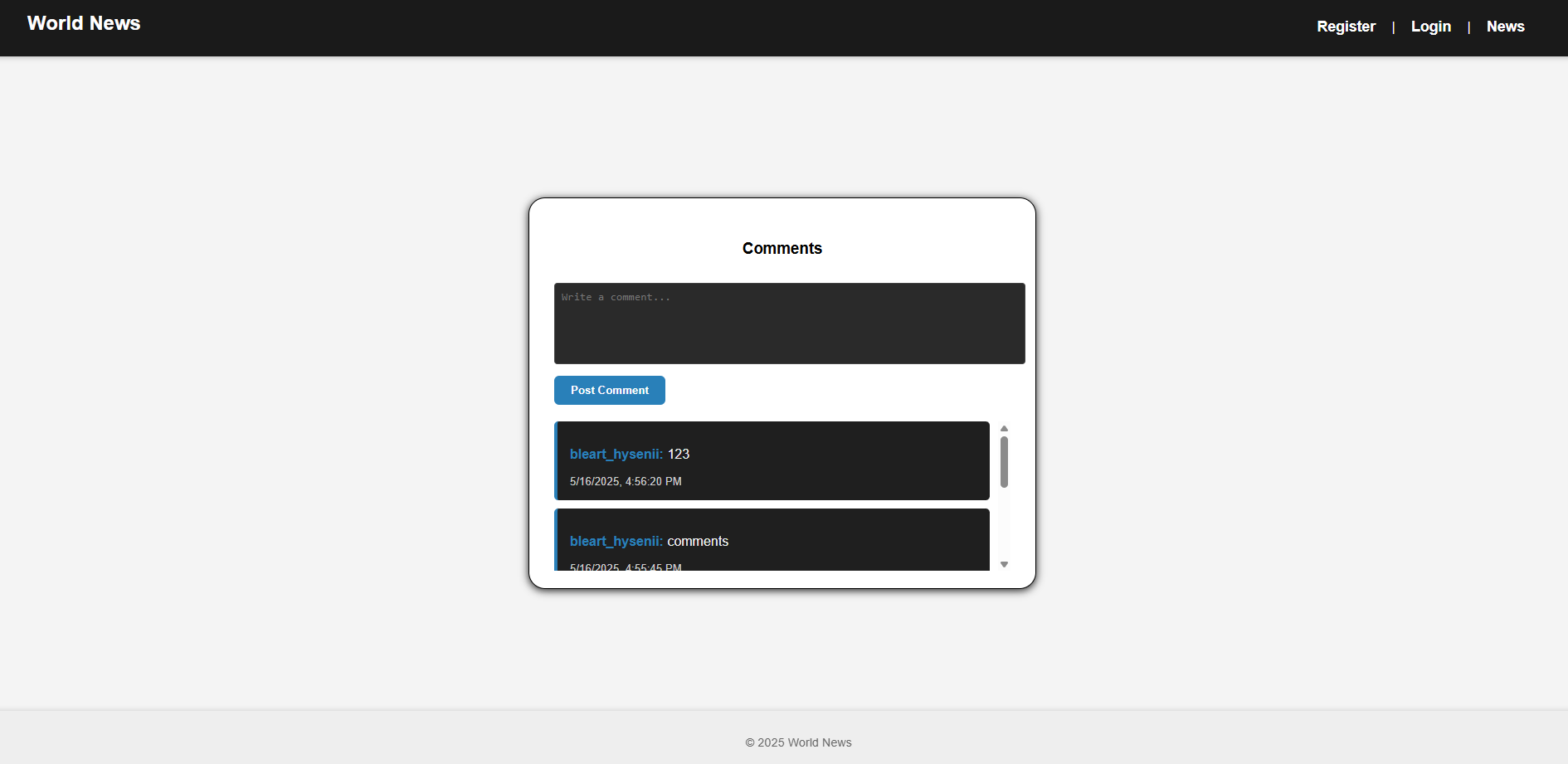
**2.Registration Page**

****

**3.** **Login Page**

****

**4.News Article Page**

****

**9. Backend Code**

**1.Server.js**

//server.js

const express = require('express');

const path = require('path');

const connectDB = require('./config/db');

const authRoutes = require('./routes/authRoutes');

const newsRoutes = require('./routes/newsRoutes');

const commentRoutes = require('./routes/commentRoutes');

const expressLayouts = require('express-ejs-layouts');

const cookieParser = require('cookie-parser');

const jwt = require('jsonwebtoken');

const authMiddleware = require('./middleware/authMiddleware');

require('dotenv').config();

const app = express();

// Connect to MongoDB

connectDB();

// Middleware

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

const attachUser = (req, res, next) => {

  const token = req.cookies.token;

  if (!token) return next();

  try {

    const decoded = jwt.verify(token, process.env.JWT\_SECRET);

    req.user = decoded;

    res.locals.user = decoded;

  } catch (err) {

  }

  next();

};

app.use(attachUser);

// Set EJS and Layouts

app.use(expressLayouts);

app.set('view engine', 'ejs');

app.set('views', path.join(\_\_dirname, 'views'));

app.set('layout', 'layout');

// Static folder

app.use(express.static(path.join(\_\_dirname, 'public')));

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

app.use('/api/news', newsRoutes);

app.use('/api/comments', authMiddleware, commentRoutes);

const News = require('./models/News');

const Comment = require('./models/Comment');

// ROOT path => redirect to /news

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

  res.redirect('/news');

});

// Register page

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

  res.render('register', { title: 'Register' });

});

// Login page

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

  res.render('login', { title: 'Login' });

});

// Home page with news list

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

  try {

    const news = await News.find().sort({ createdAt: -1 }).lean();

    res.render('index', { title: 'Home', news });

  } catch (err) {

    console.error(err);

    res.status(500).send('Server error');

  }

});

// Create news

app.post('/news/create', async (req, res) => {

  try {

    const { title, content } = req.body;

    if (!title || !content) {

      return res.status(400).send('Title and content are required');

    }

    const newNews = new News({

      title,

      content,

      createdAt: new Date(),

    });

    await newNews.save();

    res.redirect('/news');

  } catch (err) {

    console.error(err);

    res.status(500).send('Server error');

  }

});

// Show single news article with comments

app.get('/news/:id', async (req, res) => {

  try {

    const news = await News.findById(req.params.id).lean();

    if (!news) return res.status(404).send('News not found');

    const comments = await Comment.find({ newsId: news.\_id })

      .populate('authorId', 'username')

      .sort({ createdAt: -1 })

      .lean();

    res.render('news', { title: news.title, news, comments, user: res.locals.user });

  } catch (err) {

    console.error(err);

    res.status(500).send('Server error');

  }

});

// Start server

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

**2. MongoDB Model Example**(models/News.js)

// models/News.js

const mongoose = require('mongoose');

const NewsSchema = new mongoose.Schema({

  title: { type: String, required: true },

  content: { type: String, required: true },

  createdAt: { type: Date, default: Date.now },

  updatedAt: { type: Date, default: Date.now }

});

module.exports = mongoose.model('News', NewsSchema);

**3. Express Routes Example** (routes/newsRoutes.js)

//newRoutes.js

const express = require('express');

const News = require('../models/News');

const authMiddleware = require('../middleware/authMiddleware');

const router = express.Router();

// CREATE news

router.post('/', authMiddleware, async (req, res) => {

  try {

    const { title, content, category } = req.body;

    if (!title || !content || !category) {

      return res.status(400).json({ message: 'Please provide title, content, and category' });

    }

    const news = new News({

      authorId: req.user.id, // nga tokeni

      title,

      content,

      category,

    });

    await news.save();

    res.status(201).json(news);

  } catch (err) {

    console.error("Error in POST /news:", err);

    res.status(500).json({ message: 'Server error' });

  }

});

// UPDATE news

router.put('/:id', authMiddleware, async (req, res) => {

  try {

    const news = await News.findById(req.params.id);

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

    if (!news.authorId) {

      return res.status(400).json({ message: 'News is missing authorId' });

    }

    if (news.authorId.toString() !== req.user.id) {

      return res.status(403).json({ message: 'Not authorized' });

    }

    const { title, content, category } = req.body;

    news.title = title || news.title;

    news.content = content || news.content;

    news.category = category || news.category;

    news.updatedAt = new Date();

    await news.save();

    res.json(news);

  } catch (err) {

    console.error("Error in PUT /news/:id:", err);

    res.status(500).json({ message: 'Server error' });

  }

});

// DELETE news

router.delete('/:id', authMiddleware, async (req, res) => {

  try {

    const news = await News.findById(req.params.id);

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

    if (!news.authorId) {

      return res.status(400).json({ message: 'News is missing authorId' });

    }

    if (news.authorId.toString() !== req.user.id) {

      return res.status(403).json({ message: 'Not authorized' });

    }

    await news.remove();

    res.json({ message: 'News deleted successfully' });

  } catch (err) {

    console.error("Error in DELETE /news/:id:", err);

    res.status(500).json({ message: 'Server error' });

  }

});

// GET all news

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

  try {

    const news = await News.find().sort({ createdAt: -1 });

    res.json(news);

  } catch (err) {

    res.status(500).json({ message: 'Server error' });

  }

});

// GET one news by ID

router.get('/:id', async (req, res) => {

  try {

    const news = await News.findById(req.params.id);

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

    res.json(news);

  } catch (err) {

    res.status(500).json({ message: 'Server error' });

  }

});

module.exports = router;

Server.js

const newsRoutes = require('./routes/newsRoutes');

app.use('/api/news', newsRoutes);

**5.config/db.js**

require('dotenv').config();

console.log('MONGO\_URI =', process.env.MONGO\_URI);

const mongoose = require('mongoose');

const connectDB = async () => {

  try {

    await mongoose.connect(process.env.MONGO\_URI, {

      useNewUrlParser: true,

      useUnifiedTopology: true,

    });

    console.log('MongoDB connected');

  } catch (err) {

    console.error('MongoDB connection error:', err.message);

    process.exit(1);

  }

};

module.exports = connectDB;

**6. models/User.js**

//User.js

const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({

  username: { type: String, unique: true, required: true },

  email: { type: String, unique: true, required: true },

  passwordHash: { type: String, required: true },

  createdAt: { type: Date, default: Date.now }

});

module.exports = mongoose.model('User', UserSchema);

**8. routes/authRoutes.js**

const express = require('express');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const User = require('../models/User');

const router = express.Router();

// Render Register form

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

  res.render('register', { title: 'Register' });

});

// Register POST

router.post('/register', async (req, res) => {

  try {

    const { username, email, password } = req.body;

    const existingUser = await User.findOne({ email });

    if (existingUser) {

      return res.render('register', { title: 'Register', error: 'Email already in use' });

    }

    const hashedPassword = await bcrypt.hash(password, 10);

    const user = new User({ username, email, passwordHash: hashedPassword });

    await user.save();

    res.redirect('/api/auth/login');

  } catch (err) {

    res.render('register', { title: 'Register', error: 'Server error' });

  }

});

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

  res.render('login', { title: 'Login' });

});

// Login POST

router.post('/login', async (req, res) => {

  try {

    const { email, password } = req.body;

    const user = await User.findOne({ email });

    if (!user) return res.render('login', { title: 'Login', error: 'Invalid credentials' });

    const isMatch = await bcrypt.compare(password, user.passwordHash);

    if (!isMatch) return res.render('login', { title: 'Login', error: 'Invalid credentials' });

    const token = jwt.sign({ id: user.\_id, username: user.username }, process.env.JWT\_SECRET, {

      expiresIn: '1h',

    });

    res.cookie('token', token, { httpOnly: true });

    res.redirect('/');

  } catch (err) {

    res.render('login', { title: 'Login', error: 'Server error' });

  }

});

module.exports = router;

**9. middleware/authMiddleware.js**

//authMiddleware.js

const jwt = require('jsonwebtoken');

const authMiddleware = (req, res, next) => {

  const token = req.cookies.token;

  if (!token) {

    req.user = null;

    return next();

  }

  try {

    const decoded = jwt.verify(token, process.env.JWT\_SECRET);

    req.user = decoded;

    next();

  } catch (err) {

    req.user = null;

    next();

  }

};

module.exports = authMiddleware;

**10.ENV**

PORT=5000

MONGO\_URI=mongodb://localhost:27017/worldnews

JWT\_SECRET=your\_jwt\_secret