Blog API Project Documentation

This guide will help you build a Blog API step by step using **Node.js, Express, and MongoDB**. It is written in simple language so you can follow it easily.

---

1. Project Setup

1. Create a new folder for your project.
2. Run:

```
npm init -y
npm install express mongoose bcryptjs jsonwebtoken multer cors dotenv
```

3. Create the following folders:

```
/config
/models
/routes
/controllers
/middleware
/uploads (for images)
```

4. Create a `server.js` file to start your app.

Example `server.js`:

```
const express = require('express');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const cors = require('cors');

dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());

// Routes will go here

mongoose.connect(process.env.MONGO_URI)
  .then(() => {
    console.log('MongoDB Connected');
```

```
    app.listen(5000, () => console.log('Server running on port 5000'));
  })
  .catch(err => console.error(err));
```

---

2. Models

User Model

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  username: { type: String, required: true, unique: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  bio: { type: String },
  profilePic: { type: String },
}, { timestamps: true });

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

Post Model

```
const mongoose = require('mongoose');

const postSchema = new mongoose.Schema({
  title: { type: String, required: true },
  content: { type: String, required: true },
  author: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  coverImage: { type: String },
  likes: [{ type: mongoose.Schema.Types.ObjectId, ref: 'User' }],
}, { timestamps: true });

module.exports = mongoose.model('Post', postSchema);
```

Comment Model

```
const mongoose = require('mongoose');

const commentSchema = new mongoose.Schema({
  content: { type: String, required: true },
  author: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  post: { type: mongoose.Schema.Types.ObjectId, ref: 'Post', required: true },
```

```
}, { timestamps: true });

module.exports = mongoose.model('Comment', commentSchema);
```

---

3. Authentication

- Use **bcryptjs** to hash passwords.
- Use **jsonwebtoken (JWT)** to create tokens for login.

Auth Flow

- **Register**: User creates account → password hashed → saved in DB.
- **Login**: User provides email + password → check password → return JWT.
- **Middleware**: Protect routes by checking JWT.

Example middleware:

```
const jwt = require('jsonwebtoken');

function authMiddleware(req, res, next) {
  const token = req.headers['authorization'];
  if (!token) return res.status(401).json({ message: 'Unauthorized' });

  try {
    const decoded = jwt.verify(token.split(' ')[1], process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(401).json({ message: 'Invalid Token' });
  }
}

module.exports = authMiddleware;
```

---

4. Routes

Auth Routes

- `POST /api/auth/register` → create user
- `POST /api/auth/login` → login user

User Routes

- `GET /api/users/:id` → get user profile
- `PUT /api/users/:id` → update user (protected)

Post Routes

- `POST /api/posts` → create post (protected)
- `GET /api/posts` → get all posts
- `GET /api/posts/:id` → get single post
- `PUT /api/posts/:id` → update post (only author)
- `DELETE /api/posts/:id` → delete post (only author)

Comment Routes

- `POST /api/posts/:id/comments` → add comment (protected)
- `DELETE /api/comments/:id` → delete comment (only author)

Like Route

- `POST /api/posts/:id/like` → like/unlike post (toggle)

---

5. File Uploads

Use **multer** for uploading profile pictures and cover images.

Example setup:

```
const multer = require('multer');

const storage = multer.diskStorage({
  destination: (req, file, cb) => cb(null, 'uploads/'),
  filename: (req, file, cb) => cb(null, Date.now() + '-' + file.originalname)
});

const upload = multer({ storage });

module.exports = upload;
```

Usage in routes:

```
router.post('/upload', upload.single('image'), (req, res) => {
  res.json({ imageUrl: `/uploads/${req.file.filename}` });
});
```

---

6. Extra Features

- **Pagination**: `/api/posts?page=1&limit=10`
- **Sorting**: `/api/posts?sort=latest` or `/api/posts?sort=mostLiked`
- **Admin Role**: allow admin to delete any post/comment.

---

7. Deployment

- Use **MongoDB Atlas** for database.
- Deploy backend on **Render, Railway, or Heroku**.
- Update `.env` with:

```
MONGO_URI=your_atlas_url
JWT_SECRET=your_secret
```

✅This completes the basic Blog API. Start small (auth + posts), then add comments, likes, and uploads. Keep testing with **Postman/Thunder Client** after every step.