DeployEase Backend - Authentication System

A robust Node.js backend with JWT-based authentication for the DeployEase application.

Features

- **User Registration & Login**: Secure signup and login with password hashing
- **JWT Authentication**: Token-based authentication with configurable expiration
- **Password Security**: Bcrypt hashing with salt rounds
- **Input Validation**: Comprehensive validation for all user inputs
- **Protected Routes**: Middleware for securing API endpoints
- **Profile Management**: User profile retrieval and updates
- **CORS Support**: Cross-origin resource sharing enabled
- **Environment Configuration**: Secure configuration management
- **Subdomain-based Deployment**: Deploy projects to custom subdomains (e.g., `myproject.sthara.fun`)
- **Static Site Hosting**: Host static websites with automatic routing
- **GitHub Integration**: Import and deploy projects directly from GitHub repositories

Tech Stack

- **Node.js** - Runtime environment
- **Express.js** - Web framework
- **MongoDB/Mongoose** - Database and ODM
- **JWT** - JSON Web Tokens for authentication
- **Bcrypt** - Password hashing
- **CORS** - Cross-origin resource sharing
- **Dotenv** - Environment variable management

Installation

1. Navigate to the Backend directory:

```
cd Backend
```

2. Install dependencies:

```
npm install
```

3. Set up environment variables:

```
cp .env.example .env
```

4. Update the `.env` file with your configuration:

```
PORT=5000
NODE_ENV=development
MONGODB_URI=mongodb://localhost:27017/deployease
JWT_SECRET=your-super-secret-jwt-key-change-this-in-production
JWT_EXPIRES_IN=7d
CORS_ORIGIN=http://localhost:5173
BASE_DOMAIN=sthara.fun
SESSION_SECRET=your-super-secret-session-key-change-this-in-production
```

5. Start MongoDB (if using MongoDB):

```
# On Windows with MongoDB installed
net start MongoDB

# On macOS with Homebrew
brew services start mongodb-community

# On Linux
sudo systemctl start mongod
```

6. Start the development server:

```
npm run dev
```

API Endpoints

Authentication Routes (`/api/auth`)

POST `/api/auth/signup`

Register a new user.

Request Body:

```
{
  "username": "johndoe",
  "email": "john@example.com",
  "password": "password123",
  "firstName": "John",
  "lastName": "Doe"
}
```

Response:

```
{
  "success": true,
  "message": "User registered successfully",
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "user": {
      "id": "user_id",
      "username": "johndoe",
      "email": "john@example.com",
      "firstName": "John",
      "lastName": "Doe",
      "role": "user"
    }
  }
}
```

POST `/api/auth/login`

Login with email/username and password.

Request Body:

```
{
  "identifier": "johndoe", // Can be email or username
  "password": "password123"
}
```

Response:

```
{
  "success": true,
  "message": "Login successful",
  "data": {
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "user": {
      "id": "user_id",
      "username": "johndoe",
      "email": "john@example.com",
      "firstName": "John",
      "lastName": "Doe",
      "role": "user",
      "lastLogin": "2025-08-28T11:00:00.000Z"
    }
```

```
  }
}
```

GET `/api/auth/profile`

Get current user profile (Protected).

Headers:

```
Authorization: Bearer <token>
```

Response:

```
{
  "success": true,
  "message": "Profile retrieved successfully",
  "data": {
    "user": {
      "id": "user_id",
      "username": "johndoe",
      "email": "john@example.com",
      "firstName": "John",
      "lastName": "Doe",
      "role": "user"
    }
  }
}
```

PUT `/api/auth/profile`

Update user profile (Protected).

Headers:

```
Authorization: Bearer <token>
```

Request Body:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "username": "johnsmith"
}
```

POST `/api/auth/change-password`

Change user password (Protected).

Headers:

```
Authorization: Bearer <token>
```

Request Body:

```
{
  "currentPassword": "oldpassword123",
  "newPassword": "newpassword123"
}
```

POST `/api/auth/logout`

Logout user (Protected).

Headers:

```
Authorization: Bearer <token>
```

Test Routes (`/api/auth-test`)

For testing without MongoDB, use the `/api/auth-test` endpoints which work with in-memory storage:

- `POST /api/auth-test/signup` - Register user (in-memory)
- `POST /api/auth-test/login` - Login user (in-memory)
- `GET /api/auth-test/profile` - Get profile (in-memory)
- `GET /api/auth-test/users` - Get all users (testing only)

Testing the API

Using PowerShell (Windows)

1. **Register a new user:**

```
Invoke-RestMethod -Uri "http://localhost:5000/api/auth-test/signup" -Method
POST -ContentType "application/json" -Body
'{"username":"testuser","email":"test@example.com","password":"password123","fi
rstName":"Test","lastName":"User"}'
```

2. **Login:**

```
Invoke-RestMethod -Uri "http://localhost:5000/api/auth-test/login" -Method POST
-ContentType "application/json" -Body
'{"identifier":"testuser","password":"password123"}'
```

3. **Access protected route:**

```
$response = Invoke-RestMethod -Uri "http://localhost:5000/api/auth-test/login"
-Method POST -ContentType "application/json" -Body
'{"identifier":"testuser","password":"password123"}'
$token = $response.data.token
Invoke-RestMethod -Uri "http://localhost:5000/api/auth-test/profile" -Method
GET -Headers @{Authorization="Bearer $token"}
```

Using curl (Linux/macOS)

1. **Register a new user:**

```
curl -X POST http://localhost:5000/api/auth-test/signup \
  -H "Content-Type: application/json" \
  -d
'{"username":"testuser","email":"test@example.com","password":"password123","fi
rstName":"Test","lastName":"User"}'
```

2. **Login:**

```
curl -X POST http://localhost:5000/api/auth-test/login \
  -H "Content-Type: application/json" \
  -d '{"identifier":"testuser","password":"password123"}'
```

3. **Access protected route:**

```
TOKEN=$(curl -s -X POST http://localhost:5000/api/auth-test/login \
  -H "Content-Type: application/json" \
  -d '{"identifier":"testuser","password":"password123"}' | jq -r
'.data.token')

curl -X GET http://localhost:5000/api/auth-test/profile \
  -H "Authorization: Bearer $TOKEN"
```

Project Structure

```
Backend/
├── models/
│   └── User.js              # User model with Mongoose
├── routes/
│   ├── auth.js              # Authentication routes (MongoDB)
│   └── auth-test.js         # Test routes (in-memory)
├── middleware/
│   └── auth.js              # JWT authentication middleware
├── .env                     # Environment variables
├── .env.example             # Environment variables template
├── .gitignore               # Git ignore file
├── package.json             # Dependencies and scripts
├── server.js                # Main server file
└── README.md                # This file
```

Security Features

- **Password Hashing**: Bcrypt with 12 salt rounds
- **JWT Tokens**: Secure token generation with configurable expiration
- **Input Validation**: Comprehensive validation for all inputs
- **CORS Protection**: Configurable cross-origin resource sharing
- **Environment Variables**: Secure configuration management
- **Error Handling**: Proper error responses without sensitive data leakage

Development

- **Hot Reload**: Nodemon for automatic server restart during development
- **Environment**: Separate development and production configurations
- **Logging**: Comprehensive error logging for debugging

Subdomain-based Deployment

DeployEase supports subdomain-based deployment for static sites, similar to Vercel and Render:

How It Works

1. **Project Creation**: Create a project with a unique name
2. **Deployment**: Deploy your static site
3. **Subdomain Access**: Access your site at `projectname.sthara.fun`

Configuration

1. **DNS Setup** (Production): Configure wildcard DNS `*.sthara.fun` → your server IP
2. **Environment Variable**: Set `BASE_DOMAIN=sthara.fun` in your `.env` file
3. **Project Types**: Works best with `projectType: 'static'` for static sites

Example Deployment Flow

```
// Create project
POST /api/projects
{
  "name": "my-awesome-site",
  "repositoryUrl": "https://github.com/user/repo",
  "projectType": "static"
}

// Deploy
POST /api/projects/{id}/deploy

// Access at: https://my-awesome-site.sthara.fun
```

Local Development

For local testing, you can:

- Use tools like `ngrok` to create temporary subdomains
- Modify hosts file for local subdomain testing
- Test with path-based URLs during development

Production Deployment

1. Set `NODE_ENV=production` in your environment
2. Use a strong, unique `JWT_SECRET`
3. Configure your production MongoDB URI
4. Set up proper CORS origins
5. **Configure wildcard DNS** `*.sthara.fun` → your server IP
6. Set `BASE_DOMAIN=sthara.fun` in your environment
7. Use HTTPS in production
8. Consider rate limiting and additional security measures

Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Test thoroughly
5. Submit a pull request

License

This project is licensed under the ISC License.