Genovation AI - Back-End Developer Assignment: Task Management System API Form
Logging, Validation, Robust Testing

How to Use

dotnet run

Login `Cookie Authentication`

Testing in Postman with Cookies

Setup:

1. Go to **POST** /api/auth/login
2. Body:

json

```
{
  "username": "user",
  "password": "user123"
}
```

3. Send request
4. Cookie is **automatically saved** by Postman

Test Retrieve:

1. Go to **GET** /api/tasks/1
2. Cookie is **automatically sent**
3. Check response (200 OK if task belongs to user, 403 if not)

Testing in Swagger

Login `Cookie Authentication`

1. Navigate to https://localhost:5001/swagger
2. Find **POST /api/auth/login**
3. Click "Try it out"
4. Enter:

json

```
{
  "username": "user",
  "password": "user123"
}
```

5. Execute
6. Cookie is set in browser automatically

Test Endpoints

test all protected endpoints!

API Endpoints

Authentication

Users (Admin only)

Method	Endpoint	Description
GET	/api/users	Get all users
GET	/api/users/{id}	Get user by ID
POST	/api/users	Create new user
PUT	/api/users/{id}	Update user
DELETE	/api/users/{id}	Delete user

Tasks

Method	Endpoint	Auth	Description
GET	/api/tasks	All users	Get tasks (filtered by role)
GET	/api/tasks/{id}	All users	Get task by ID
POST	/api/tasks	Admin only	Create new task
PUT	/api/tasks/{id}	All users	Update task*
DELETE	/api/tasks/{id}	Admin only	Delete task

Admin can update all fields, Users can only update status of their own tasks.

Users Controller:

- POST /api/users → Creates new user dynamically
- GET /api/users → Returns seed data + any new users created
- PUT /api/users/{id} → Updates user (Admin only)
- DELETE /api/users/{id} → Deletes user (Admin only)

Tasks Controller:

- POST /api/tasks → Creates new task dynamically (Admin only)
- GET /api/tasks → Returns all tasks (Admin) or user's tasks (User)
- PUT /api/tasks/{id} → Updates task dynamically
- DELETE /api/tasks/{id} → Deletes task (Admin only)

Configuration

appsettings.json:

```
{
  "Authentication": {
    "Cookie": {
      "LoginPath": "/api/auth/login",
      "LogoutPath": "/api/auth/logout",
      "ExpireTimeMinutes": 120,
      "SlidingExpiration": true
    }
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Database

Type: In-Memory Database (EF Core)

- Data persists during runtime
- Resets on application restart
- Pre-seeded with 2 users and 3 tasks

Entities:

- Users (Id, Username, Email, PasswordHash, Role, CreatedAt)
- Tasks (Id, Title, Description, Status, AssignedUserId, DueDate, CreatedAt)

Technologies Used

- **Framework:** .NET 8
- **ORM:** Entity Framework Core (In-Memory)

- **Authentication:** Cookie Authentication
- **Password Hashing:** BCrypt
- **API Documentation:** Swagger/OpenAPI
- **Testing:** xUnit + Moq

Example Requests

1. Login

```
curl -X POST https://localhost:5001/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"admin123"}' \
  -c cookies.txt
```

Note: -c cookies.txt saves the authentication cookie

2. Get All Tasks (Admin) - Using Cookie

```
curl -X GET https://localhost:5001/api/tasks \
  -b cookies.txt
```

Note: -b cookies.txt sends the saved cookie

3. Create Task (Admin) - Using Cookie

```
curl -X POST https://localhost:5001/api/tasks \
  -b cookies.txt \
  -H "Content-Type: application/json" \
  -d '{
    "title": "New Task",
    "description": "Task description",
    "assignedUserId": 2,
    "status": 0
  }'
```

4. Update Task Status (User) - Using Cookie

```
curl -X PUT https://localhost:5001/api/tasks/1 \
  -b cookies.txt \
```

```
  -H "Content-Type: application/json" \
  -d '{"status": 2}'
```

5. Logout

```
curl -X POST https://localhost:5001/api/auth/logout \
  -b cookies.txt
```

Retrieve a Task

Scenario 1: Admin Views Any Task

Admin can view **any task** in the system.

```
# First, login as admin and save cookie

curl -X POST https://localhost:5001/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"admin","password":"admin123"}' \
  -c cookies.txt

# Then, retrieve task with ID 1 (works for any task)

curl -X GET https://localhost:5001/api/tasks/1 \
  -b cookies.txt
```

Response (200 OK):

json


```
{
  "id": 1,
  "title": "Setup project",
  "description": "Initial setup and configuration",
  "status": "Pending",
  "dueDate": "2026-01-15T00:00:00Z",
  "createdAt": "2026-01-08T00:00:00Z",
  "assignedUserId": 2,
  "assignedUserName": "user",
  "assignedUserEmail": "user@example.com"
}
```

Scenario 2: User Views Their Own Task

Regular user can only view tasks **assigned to them**.

```
# Login as regular user and save cookie

curl -X POST https://localhost:5001/api/auth/login \
  -H "Content-Type: application/json" \
```

```
  -d '{"username":"user","password":"user123"}' \
  -c user-cookies.txt

# Retrieve task assigned to this user (e.g., task ID 1)

curl -X GET https://localhost:5001/api/tasks/1 \
  -b user-cookies.txt
```

Response (200 OK):

json

```
{
  "id": 1,
  "title": "Setup project",
  "description": "Initial setup and configuration",
  "status": "Pending",
  "dueDate": "2026-01-15T00:00:00Z",
  "createdAt": "2026-01-08T00:00:00Z",
  "assignedUserId": 2,
  "assignedUserName": "user",
  "assignedUserEmail": "user@example.com"
}
```

Scenario 3: User Tries to View Another User's Task

Regular user tries to view a task **NOT assigned to them**.

```
# Login as user (ID = 2)
curl -X POST https://localhost:5001/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"username":"user","password":"user123"}' \
  -c user-cookies.txt

# Try to view task assigned to someone else (e.g., task ID 3 assigned to admin)
curl -X GET https://localhost:5001/api/tasks/3 \
  -b user-cookies.txt
```

Response (403 Forbidden):

json

```
{
  "message": "You can only view your own tasks"
}
```

Scenario 4: Retrieve Non-Existent Task

```
# Try to get task that doesn't exist
curl -X GET https://localhost:5001/api/tasks/999 \
  -b cookies.txt
```

Response (404 Not Found):

json

```
{
  "message": "Task not found"
}
```

Troubleshooting

Issue: "401 Unauthorized" on all requests

1. **Solution:** Make sure you've logged in and used the token in Authorization header
2. **Issue:** Swagger not loading
3. **Solution:** Ensure you're in Development mode and navigate to /swagger
4. **Issue:** "Username already exists"
5. **Solution:** Use different username or restart application to reset database

---

Project Architecture / Repository and Unit of Work

Controllers / API endpoints and HTTP handling

```
├── AuthController.cs
├── UsersController.cs
└── TasksController.cs
```

Models // DTOs for API requests/responses

```
├── LoginRequest.cs
├── LoginResponse.cs
├── UserDto.cs
├── CreateUserDto.cs
├── UpdateUserDto.cs
├── TaskDto.cs
├── CreateTaskDto.cs
└── UpdateTaskDto.cs
```

Entities // Database models (EF Core entities)

```
├── User.cs
├── TaskItem.cs
├── Role.cs
└── TaskStatus.cs
```

Services // Business logic layer

```
├── IAuthService.cs
├── AuthService.cs
├── IUserService.cs
├── UserService.cs
├── ITaskService.cs
```

└── TaskService.cs

Repositories // Data access layer
├── UserRepository.cs
├── IUserRepository.cs
├── ITaskRepository.cs
└── TaskRepository.cs

Data // EF Core DbContext and database configuration
└── AppDbContext.cs

Middleware // Custom middleware
└── RoleAuthorizationMiddleware.cs

Prepared By
Hamdi Yaseen
hamdi_yafa@live.com