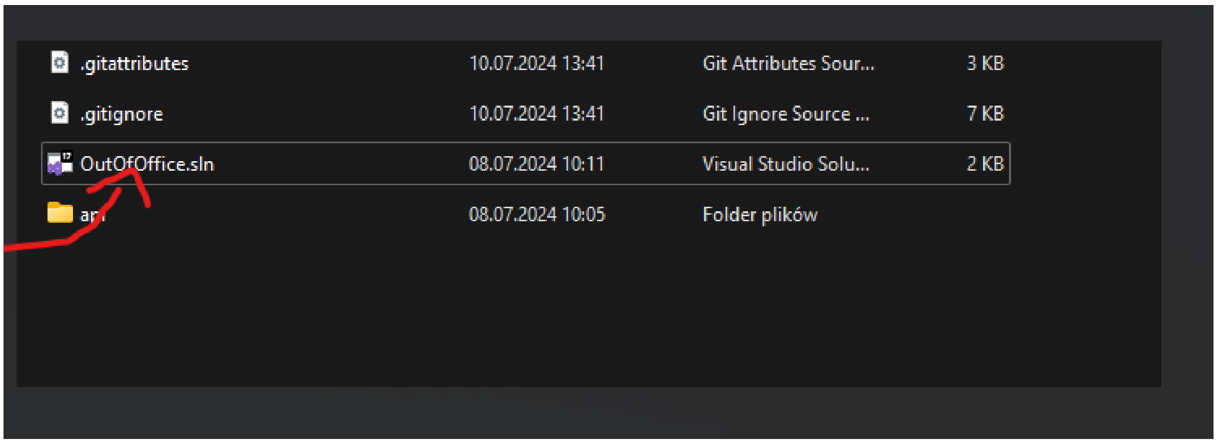
OutOfOffice Solution

OutOfOffice Solution was implemented in .Net v.7.0.12 technology using C#. TO collect Data I used SQlite3.

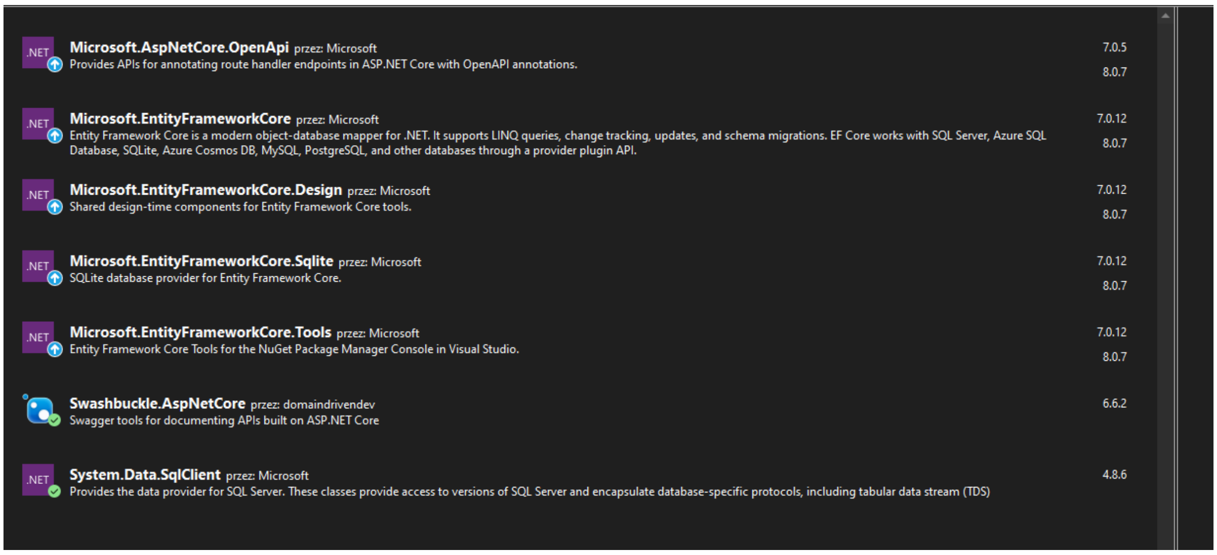
1. Launch a Project
   To launch a project you need to clone my repository from my github account. Link is able to click here:
   https://github.com/314otrek/OutOfOfficeSolution
   and also attached in the e-mail address. To clone repository open cmd console In particular folder and paste:
   git clone https://github.com/314otrek/OutOfOfficeSolution
   Wait until all files will download.
   After that you need a visual Studio. I am using 2022 version.
   After all repository will get download press OutOfOffice.sln and wait to Launch Visual Studio.



.gitattributes	10.07.2024 13:41	Git Attributes Sour...	3 KB
.gitignore	10.07.2024 13:41	Git Ignore Source ...	7 KB
OutOfOffice.sln	08.07.2024 10:11	Visual Studio Solu...	2 KB
ap	08.07.2024 10:05	Folder plików	

After program will launch correctly you can run a project or check if you have or needed dependencies to run a program. I suggest 2nd option to be sure app will run.

All used dependencies with version are shown below:



Microsoft.AspNetCore.OpenApi przez: Microsoft 7.0.5 8.0.7
Provides APIs for annotating route handler endpoints in ASP.NET Core with OpenAPI annotations.

Microsoft.EntityFrameworkCore przez: Microsoft 7.0.12 8.0.7
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.

Microsoft.EntityFrameworkCore.Design przez: Microsoft 7.0.12 8.0.7
Shared design-time components for Entity Framework Core tools.

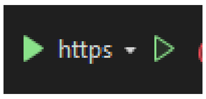
Microsoft.EntityFrameworkCore.Sqlite przez: Microsoft 7.0.12 8.0.7
SQLite database provider for Entity Framework Core.

Microsoft.EntityFrameworkCore.Tools przez: Microsoft 7.0.12 8.0.7
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Swashbuckle.AspNetCore przez: domaindrivendev 6.6.2
Swagger tools for documenting APIs built on ASP.NET Core

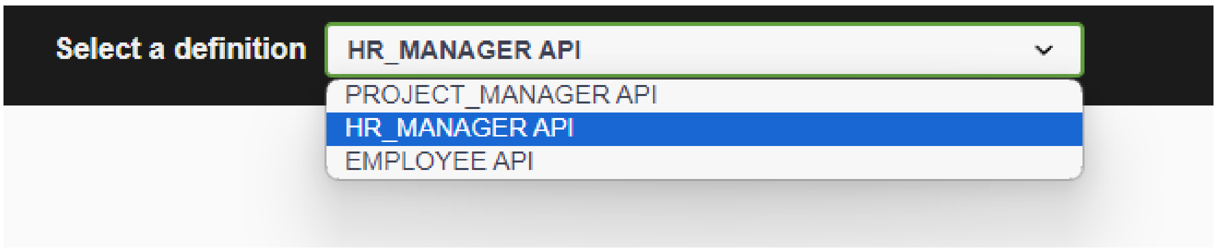
System.Data.SqlClient przez: Microsoft 4.8.6
Provides the data provider for SQL Server. These classes provide access to versions of SQL Server and encapsulate database-specific protocols, including tabular data stream (TDS)

When you are sure that all dependencies are in project you can test application by clicking:

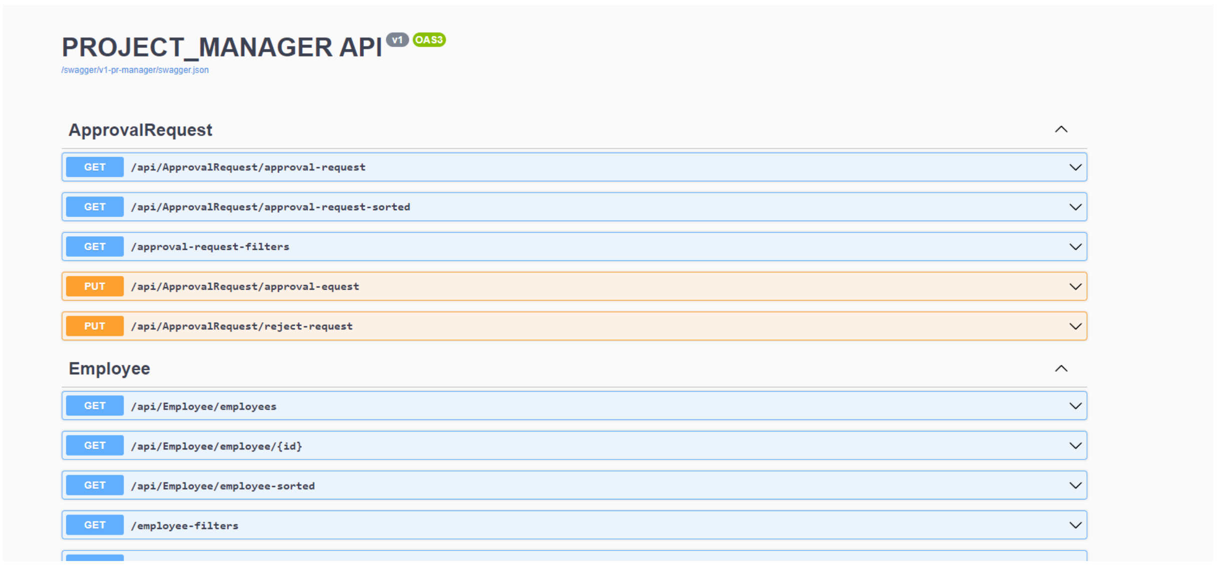


Or use CTRL + F5

2. View o Rest endpoints
To demonstrate application Endpoints, I used Swagger. In the up right panel you can choose view for roles like Employee, Hr-Manager and Project Manager where are shown selected function for particular role.



Sample view in Swagger:



PROJECT_MANAGER API v1 OAS3
/swagger/v1-pr-manager/swagger.json

ApprovalRequest

```
GET  /api/ApprovalRequest/approval-request
GET  /api/ApprovalRequest/approval-request-sorted
GET  /approval-request-filters
PUT  /api/ApprovalRequest/approval-equest
PUT  /api/ApprovalRequest/reject-request
```

Employee

```
GET  /api/Employee/employees
GET  /api/Employee/employee/{id}
GET  /api/Employee/employee-sorted
GET  /employee-filters
```

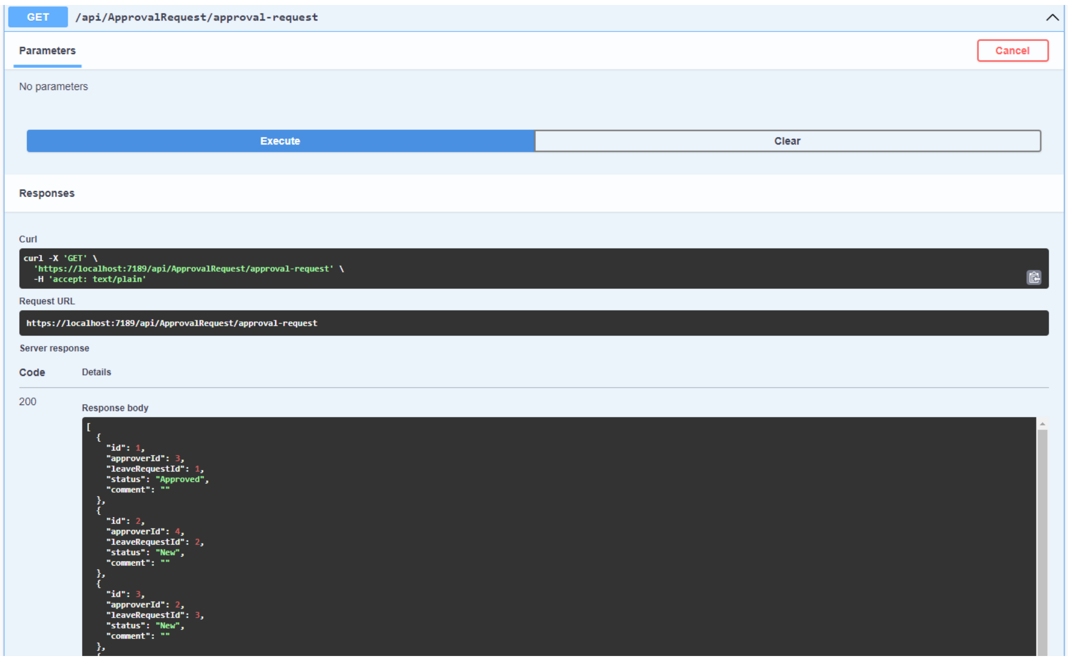
3. Implemented methods:

Below I listed all endpoints with short descriptions. In some of them I attached screenshots.

Approval Request Endpoints (Only for Project Manager and HR-MANAGER View)

GET Endpoints for Approval Request:

- approval-requests – list approved request



```
GET  /api/ApprovalRequest/approval-request
```

No parameters

Curl

```
curl -X 'GET' \
  'https://localhost:7189/api/ApprovalRequest/approval-request' \
  -H 'accept: text/plain'
```

Request URL

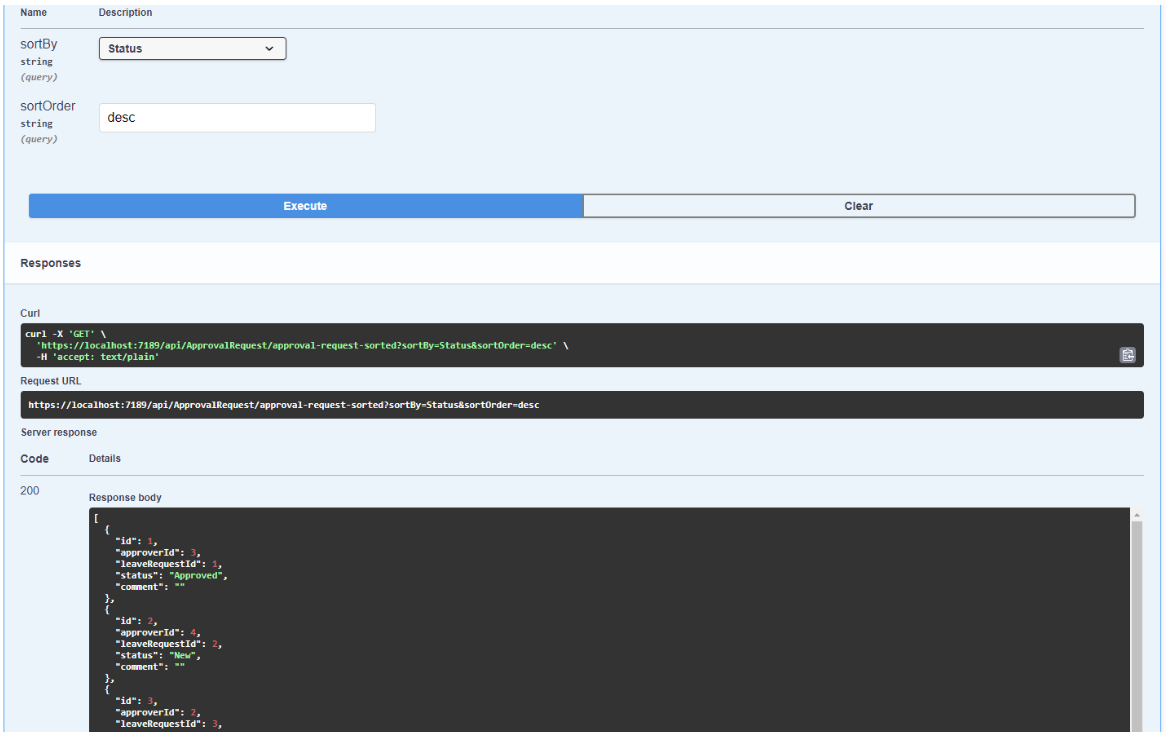
```
https://localhost:7189/api/ApprovalRequest/approval-request
```

Server response

Code 200 — Details: Response body

```
[
  {
    "id": 1,
    "approverId": 3,
    "leaveRequestId": 1,
    "status": "Approved",
    "comment": ""
  },
  {
    "id": 2,
    "approverId": 4,
    "leaveRequestId": 2,
    "status": "New",
    "comment": ""
  },
  {
    "id": 3,
    "approverId": 2,
    "leaveRequestId": 3,
    "status": "New",
    "comment": ""
  },
```

- approval-request-sorted - list approved requests selected method sort (default asc) and using selected column



Name | Description

sortBy string (query): Status

sortOrder string (query): desc

Curl

```
curl -X 'GET' \
  'https://localhost:7189/api/ApprovalRequest/approval-request-sorted?sortBy=Status&sortOrder=desc' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7189/api/ApprovalRequest/approval-request-sorted?sortBy=Status&sortOrder=desc
```

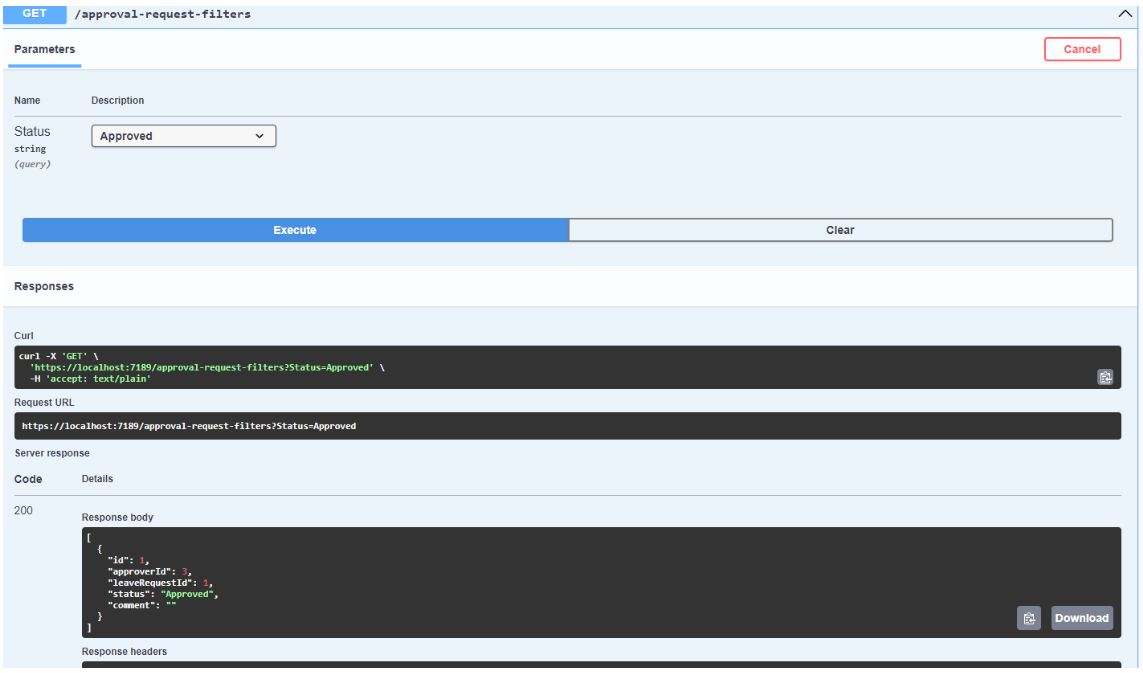
Server response

Code 200

Response body

```
[
  {
    "id": 1,
    "approverId": 3,
    "leaveRequestId": 1,
    "status": "Approved",
    "comment": ""
  },
  {
    "id": 2,
    "approverId": 4,
    "leaveRequestId": 2,
    "status": "New",
    "comment": ""
  },
  {
    "id": 3,
    "approverId": 2,
    "leaveRequestId": 3,
```

- approval-requests-filter – list approval request using selected fields of object



GET /approval-request-filters

Parameters

Name | Description

Status string (query): Approved

Curl

```
curl -X 'GET' \
  'https://localhost:7189/approval-request-filters?Status=Approved' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7189/approval-request-filters?Status=Approved
```

Server response

Code 200

Response body

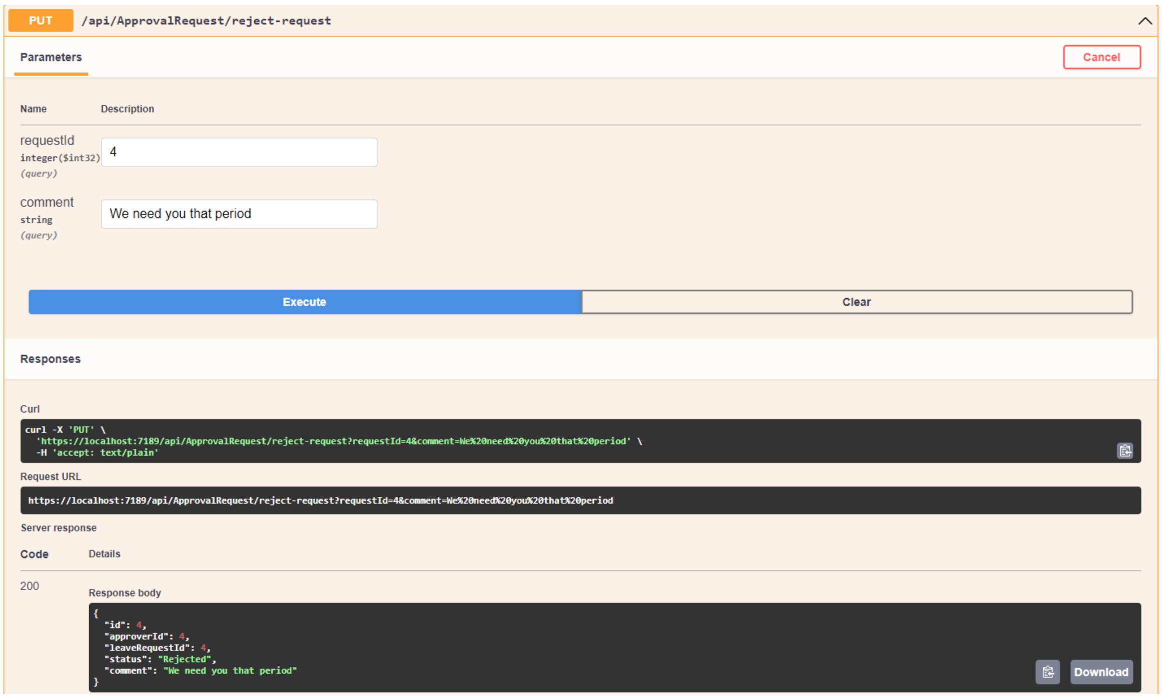
```
[
  {
    "id": 1,
    "approverId": 3,
    "leaveRequestId": 1,
    "status": "Approved",
    "comment": ""
  }
]
```

Response headers

PUT Endpoint for Approval Requests:

- approval-request – approve request with provided id

- reject-request – reject requests with provided id



/api/ApprovalRequest/reject-request

Name	Description
requestId integer($int32) (query)	4
comment string (query)	We need you that period

Responses

Curl

```
curl -X 'PUT' \
  'https://localhost:7189/api/ApprovalRequest/reject-request?requestId=4&comment=We%20need%20you%20that%20period' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7189/api/ApprovalRequest/reject-request?requestId=4&comment=We%20need%20you%20that%20period
```

Server response

Code	Details
200	Response body

```
{
  "id": 4,
  "approverId": 4,
  "leaveRequestId": 4,
  "status": "Rejected",
  "comment": "We need you that period"
}
```

Employee Endpoints

GET Endpoints for Employee:

- employees - list employees sorted by id
- employee-id – get employee with particular id
- employee-sorted – works like approval-request-sorted
- employee-filters – works like approval-request-filters
- employee-search-by-name list employees found by phrase related to FullName
- projects-of-employees- list project of particular employee by his id(able only in employee view)

PUT Endopoint for Employee:

- add-employee-to-project – add employee to a project
- deactive-employee – set employee status to deactive(HR-Manager view)

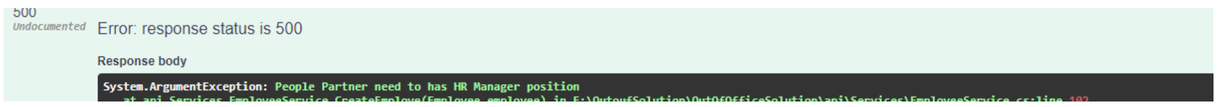
- update-employee – update data about particular employee (HR-Manager View)

POST Ednpoint for Employee:

- create-employee – create employee (HR-Manaager View)

Example of invalid create-employee request and response in Swagger

```
{
  "fullName": "Mark Zuckerberg",
  "subdivision": "IT",
  "position": "EMPLOYEE",
  "status": "Active",
  "peoplePartnerId": 2,
  "outOfOfficeBalance": 20
}
```



```
500
Undocumented  Error: response status is 500

              Response body
              System.ArgumentException: People Partner need to has HR Manager position
                 at api.Services.EmployeeService.CreateEmploye(Employee employee) in E:\OutoufSolution\OutOfOfficeSolution\api\Services\EmployeeService.cs:line 102
```

For Leave Request Table (View able for)

GET Endopoint for LeaveRquest Table:

- leave-request – list all leaveRequests
- leave-request-sorted – works like approval-request-sorted
- leave-request-filters- works like approval-request-sorted

PUT Endpoint for LeaveRequest Table:

- leave-request-submit – submit leave request (able also in HR_MANAGER and EMPLOYEE API)
- leave-request-cancel – cancel created earlier request (able also in HR_MANAGER and EMPLOYEE API)

POST Endpoint for LeaverRequest Table:

- leave-request – create Leave Request (able also in HR_MANAGER and EMPLOYEE API)

Example of post JSON to create a LeaveRequest:

```
{
  "employeeId": 2,
  "absenceReason": "VACATION",
  "startDate": "2024-07-19",
  "endDate": "2024-07-23",
  "comment": "string",
  "status": "New"
}
```

GET Endpoints for Project Table:

- projects – list all projects
- project-sorted – works like approval-request-sorted
- project-filters – works like approval-request-filters
- project-of-employee list projects which are related to particular id which is provided in request

PUT Endpoints for Project Table:

- project-deactive – deactive project using ID
- project-update – update data in project

POST Endpoint for Project Table:

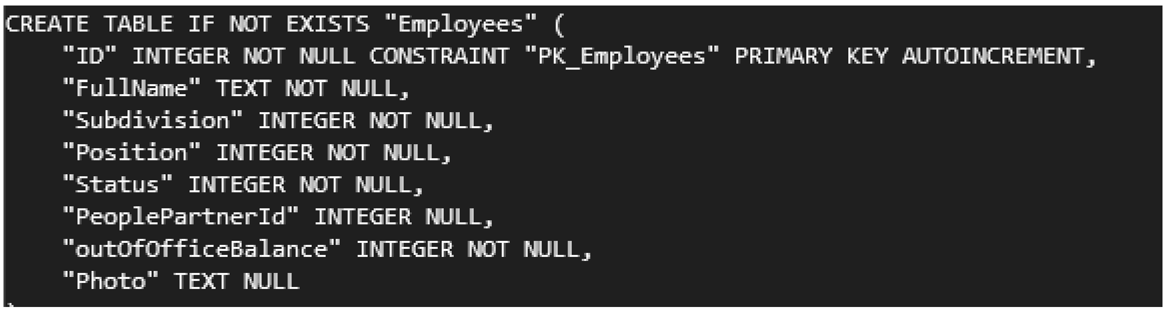
- project-create

Example of project create json :

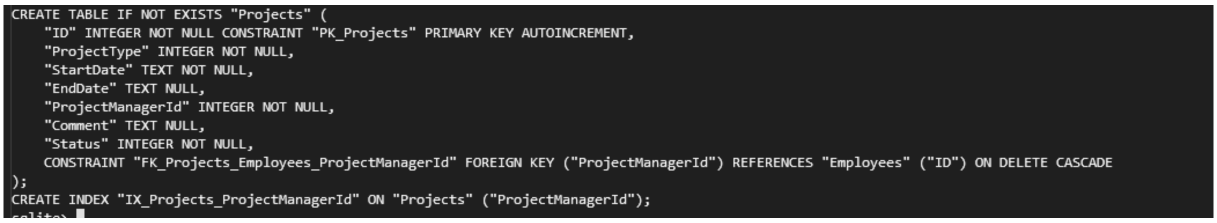
```
{
  "projectType": "WEB_DEVELOPMENT",
  "startDate": "2024-07-12",
  "projectManagerId": 4,
  "comment": "string",
    "status": "Active"
  }
```

4. Database

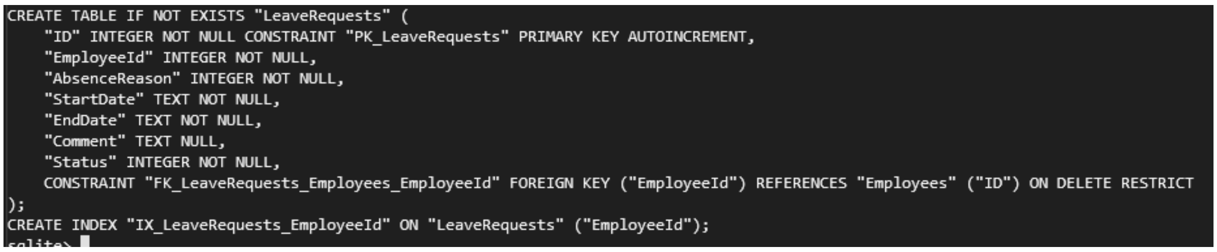
The database I used is Sqlite3. By using Entity Framework the tables was auto created. Below are shown how they can be created:



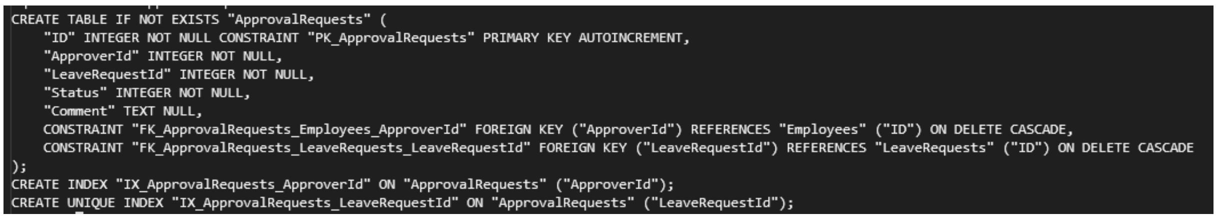
```
CREATE TABLE IF NOT EXISTS "Employees" (
    "ID" INTEGER NOT NULL CONSTRAINT "PK_Employees" PRIMARY KEY AUTOINCREMENT,
    "FullName" TEXT NOT NULL,
    "Subdivision" INTEGER NOT NULL,
    "Position" INTEGER NOT NULL,
    "Status" INTEGER NOT NULL,
    "PeoplePartnerId" INTEGER NULL,
    "outOfOfficeBalance" INTEGER NOT NULL,
    "Photo" TEXT NULL
```



```
CREATE TABLE IF NOT EXISTS "Projects" (
    "ID" INTEGER NOT NULL CONSTRAINT "PK_Projects" PRIMARY KEY AUTOINCREMENT,
    "ProjectType" INTEGER NOT NULL,
    "StartDate" TEXT NOT NULL,
    "EndDate" TEXT NULL,
    "ProjectManagerId" INTEGER NOT NULL,
    "Comment" TEXT NULL,
    "Status" INTEGER NOT NULL,
    CONSTRAINT "FK_Projects_Employees_ProjectManagerId" FOREIGN KEY ("ProjectManagerId") REFERENCES "Employees" ("ID") ON DELETE CASCADE
);
CREATE INDEX "IX_Projects_ProjectManagerId" ON "Projects" ("ProjectManagerId");
```

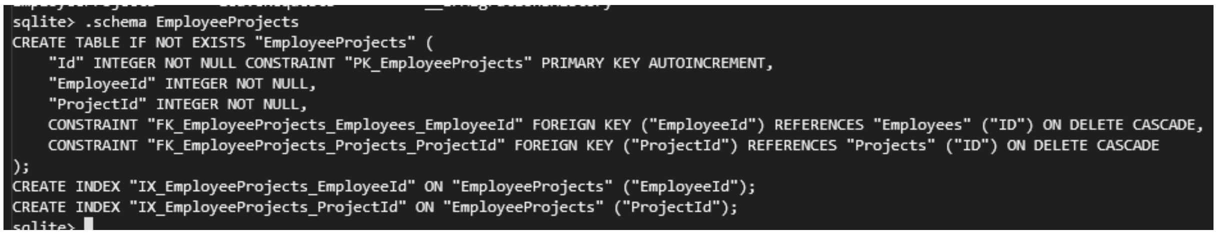


```
CREATE TABLE IF NOT EXISTS "LeaveRequests" (
    "ID" INTEGER NOT NULL CONSTRAINT "PK_LeaveRequests" PRIMARY KEY AUTOINCREMENT,
    "EmployeeId" INTEGER NOT NULL,
    "AbsenceReason" INTEGER NOT NULL,
    "StartDate" TEXT NOT NULL,
    "EndDate" TEXT NOT NULL,
    "Comment" TEXT NULL,
    "Status" INTEGER NOT NULL,
    CONSTRAINT "FK_LeaveRequests_Employees_EmployeeId" FOREIGN KEY ("EmployeeId") REFERENCES "Employees" ("ID") ON DELETE RESTRICT
);
CREATE INDEX "IX_LeaveRequests_EmployeeId" ON "LeaveRequests" ("EmployeeId");
```



```
CREATE TABLE IF NOT EXISTS "ApprovalRequests" (
    "ID" INTEGER NOT NULL CONSTRAINT "PK_ApprovalRequests" PRIMARY KEY AUTOINCREMENT,
    "ApproverId" INTEGER NOT NULL,
    "LeaveRequestId" INTEGER NOT NULL,
    "Status" INTEGER NOT NULL,
    "Comment" TEXT NULL,
    CONSTRAINT "FK_ApprovalRequests_Employees_ApproverId" FOREIGN KEY ("ApproverId") REFERENCES "Employees" ("ID") ON DELETE CASCADE,
    CONSTRAINT "FK_ApprovalRequests_LeaveRequests_LeaveRequestId" FOREIGN KEY ("LeaveRequestId") REFERENCES "LeaveRequests" ("ID") ON DELETE CASCADE
);
CREATE INDEX "IX_ApprovalRequests_ApproverId" ON "ApprovalRequests" ("ApproverId");
CREATE UNIQUE INDEX "IX_ApprovalRequests_LeaveRequestId" ON "ApprovalRequests" ("LeaveRequestId");
```

relation many to many between Project and Employee



```
sqlite> .schema EmployeeProjects
CREATE TABLE IF NOT EXISTS "EmployeeProjects" (
    "Id" INTEGER NOT NULL CONSTRAINT "PK_EmployeeProjects" PRIMARY KEY AUTOINCREMENT,
    "EmployeeId" INTEGER NOT NULL,
    "ProjectId" INTEGER NOT NULL,
    CONSTRAINT "FK_EmployeeProjects_Employees_EmployeeId" FOREIGN KEY ("EmployeeId") REFERENCES "Employees" ("ID") ON DELETE CASCADE,
    CONSTRAINT "FK_EmployeeProjects_Projects_ProjectId" FOREIGN KEY ("ProjectId") REFERENCES "Projects" ("ID") ON DELETE CASCADE
);
CREATE INDEX "IX_EmployeeProjects_EmployeeId" ON "EmployeeProjects" ("EmployeeId");
CREATE INDEX "IX_EmployeeProjects_ProjectId" ON "EmployeeProjects" ("ProjectId");
sqlite>
```

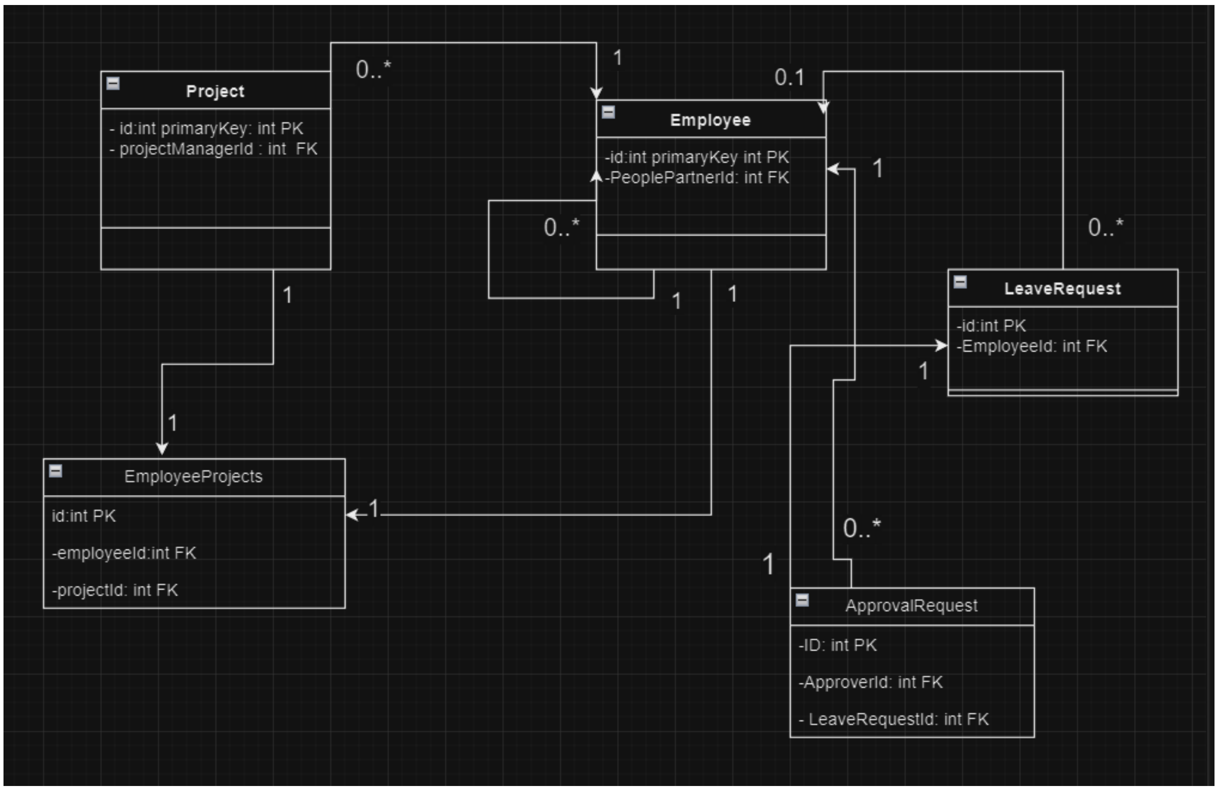
This table contain primary key Id , employeeId which is FK to Employee table and projectId which is FK to Project Table

Others relations between tables:

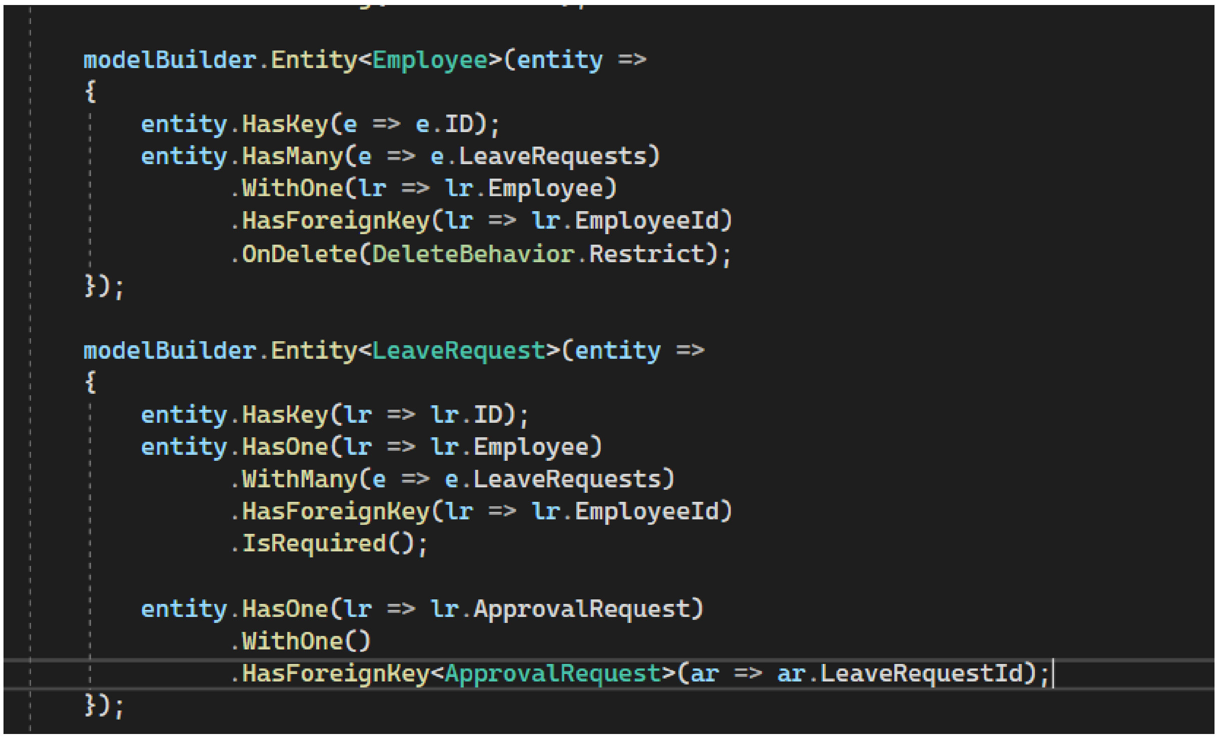
- LeaveRequest:
  One to many relations between Employee and LeaveRequest Table. LeaveRequest element contains FK key which is employeeId. Employee element contain List of LeaveRequests
- ApprovalRequest:
  One to many Relation between Epprover as a Employee Object and List of ApprovalRequestes. ApprovalRequest element contain FK approverId which is Employee PK.
- Project:
  One to many Relation between project_manager and list of projects. Employee object contain list of projects and Project contain field ProjectManagerId which is FK to Employee PK employeeId

5. Comments.
   Above I descripted all project I created. I know that project need some more validations but I am sure I showed some basic knowledge of programming. I spent much of my evenings time to do it and fact that its my 3rd full application in .Net (on the university), I am happy I increased my knowledge about this technology and thanks for the chance to show off.

UML DIAGRAM:



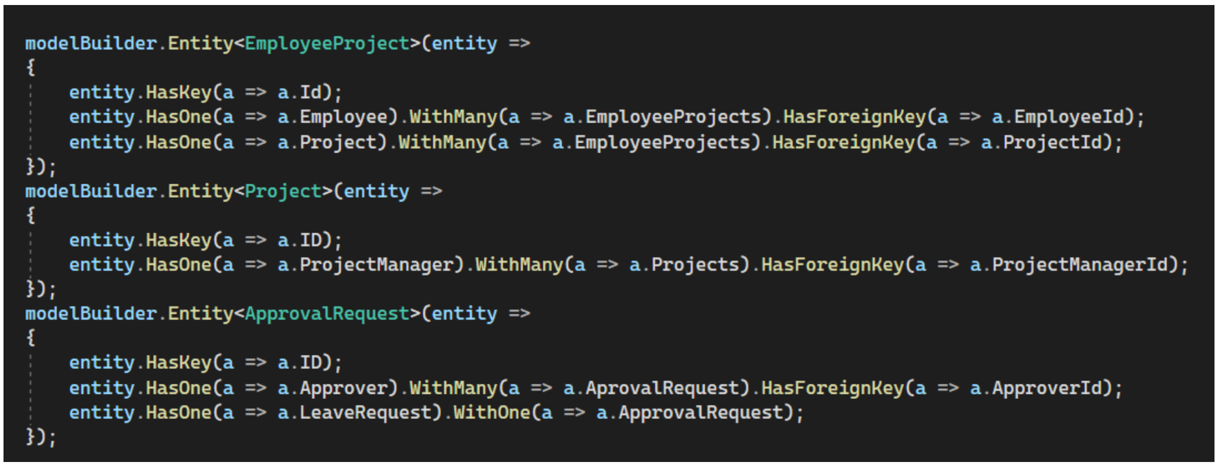
Below you can find how relation was defined in ApplicationDbContext.cs:



```
modelBuilder.Entity<Employee>(entity =>
{
    entity.HasKey(e => e.ID);
    entity.HasMany(e => e.LeaveRequests)
          .WithOne(lr => lr.Employee)
          .HasForeignKey(lr => lr.EmployeeId)
          .OnDelete(DeleteBehavior.Restrict);
});

modelBuilder.Entity<LeaveRequest>(entity =>
{
    entity.HasKey(lr => lr.ID);
    entity.HasOne(lr => lr.Employee)
          .WithMany(e => e.LeaveRequests)
          .HasForeignKey(lr => lr.EmployeeId)
          .IsRequired();

    entity.HasOne(lr => lr.ApprovalRequest)
          .WithOne()
          .HasForeignKey<ApprovalRequest>(ar => ar.LeaveRequestId);
});
```



```
modelBuilder.Entity<EmployeeProject>(entity =>
{
    entity.HasKey(a => a.Id);
    entity.HasOne(a => a.Employee).WithMany(a => a.EmployeeProjects).HasForeignKey(a => a.EmployeeId);
    entity.HasOne(a => a.Project).WithMany(a => a.EmployeeProjects).HasForeignKey(a => a.ProjectId);
});
modelBuilder.Entity<Project>(entity =>
{
    entity.HasKey(a => a.ID);
    entity.HasOne(a => a.ProjectManager).WithMany(a => a.Projects).HasForeignKey(a => a.ProjectManagerId);
});
modelBuilder.Entity<ApprovalRequest>(entity =>
{
    entity.HasKey(a => a.ID);
    entity.HasOne(a => a.Approver).WithMany(a => a.AprovalRequest).HasForeignKey(a => a.ApproverId);
    entity.HasOne(a => a.LeaveRequest).WithOne(a => a.ApprovalRequest);
});
```