

1. Design patterns ( <a href="https://refactoring.guru/uk/design-patterns/catalog">https://refactoring.guru/uk/design-patterns/catalog</a> ).....	2
Завдання.....	2
Породжувальні (Фабричний).....	2
Структурні (Декоратор).....	4
Поведінкові (Стратегія).....	6
2. Implement simple (REST/graphql) API.....	8
- CRUD.....	8
- GET/POST/PUT/DELETE convention.....	11
- at least 3 total entities in total.....	14
- nested entities.....	19
- paging/sorting.....	21
3. Simple frontend application.....	21
- Authorization.....	21
- 3 pages.....	21
- the shared state between pages.....	21
4. Testing.....	21
- 30% code coverage.....	21
- performance testing at least 1 endpoint.....	21
* complex scenario testing (use endpoint out put as input to different call).....	21
* Scrap some data with Selenium (or similar), auth navigate to some page, scrap data..	21
5. Deployment.....	21
- deploy applications to some cloud (Azure/AWS/GCP/Heroku/DigitalOcean).....	21
- deploy as a container unit.....	21
- configure CI/CD.....	21

# 1. Design patterns

(<https://refactoring.guru/uk/design-patterns/catalog>)

## Завдання

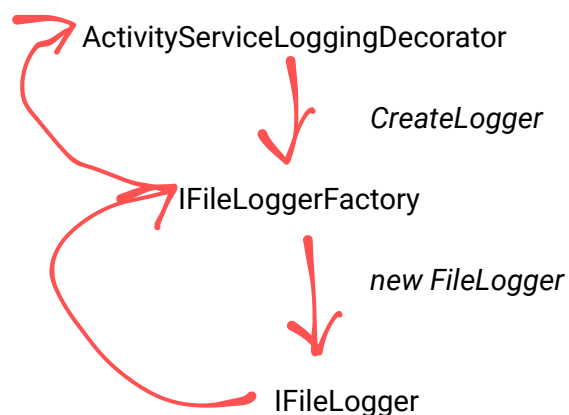
1. Design patterns (<https://refactoring.guru/uk/design-patterns/catalog>)
  - 1 pattern from each category
  - capable of explaining idea

## Породжувальні (Фабричний)

### Суть патерна

**Фабричний метод** — це породжувальний патерн проектування, який визначає загальний інтерфейс для створення об'єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об'єктів.

Патерн використовується для логування роботи сервісів. Було використано фабрику, яка створює логер.



```

4 references
public interface IFileLoggerFactory
{
    6 references
    IFileLogger CreateLogger();
}
2 references
public class FileLoggerFactory : IFileLoggerFactory
{
    private readonly string path = Path.Combine(Directory.GetCurrentDirectory(), "logfile.txt");
    0 references
    public FileLoggerFactory()
    {
    }

    6 references
    public IFileLogger CreateLogger()
    {
        return new FileLogger(path);
    }
}

```

Фабрика яка створює клас логера для запису логів у файл.

```

3 references
public interface IFileLogger : IDisposable
{
    19 references
    void WriteTextToFile(string message);
}
2 references
public class FileLogger : IFileLogger
{
    private readonly string filePath;

    1 reference
    public FileLogger(string filePath)
    {
        this.filePath = filePath;
    }

    0 references
    public void Dispose()
    {
    }

    19 references
    public void WriteTextToFile(string message)
    {
        lock (filePath)
        {
            File.AppendAllText(filePath, $"{DateTime.Now}: {message}{Environment.NewLine}");
        }
    }
}

```

Клас який виконує логування

```

3 references
public async Task<Activity> CreateActivityAsync(Activity newActivity)
{
    using (var logger = _loggerFactory.CreateLogger())
    {
        try
        {
            logger.WriteTextToFile("Creating a new activity.");
            var result = await _decoratedActivityService.CreateActivityAsync(newActivity);
            logger.WriteTextToFile($"Activity created successfully with ID: {result.ActivityID}");
            return result;
        }
        catch (Exception ex)
        {
            logger.WriteTextToFile($"Error creating activity {ex}");
            throw;
        }
    }
}

```

Приклад використання який пише логи у файл

## Структурні (Декоратор)

### Суть патерна

**Декоратор** — це структурний патерн проектування, що дає змогу динамічно додавати об'єктам нову функціональність, загортаючи їх у корисні «обгортки».

ActivityService це сервіс який декорується. Він має функціонал для операцій з активностями.

```

public class ActivityService : IActivityService
{
    private readonly ApplicationDbContext _context;

    0 references
    public ActivityService(ApplicationDbContext context)
    {
        _context = context;
    }

    3 references
    public async Task<Activity> CreateActivityAsync(Activity newActivity)
    {
        _context.Activities.Add(newActivity);
        await _context.SaveChangesAsync();
        return newActivity;
    }

    3 references
    public async Task<IEnumerable<Activity>> GetAllActivitiesAsync()
    {
        return await _context.Activities.ToListAsync();
    }
}

```

ActivityServiceLoggingDecorator це декоратор для попереднього сервісу. Він виконує усі дії сервісу та додає функціонал логування.

```

references
public class ActivityServiceLoggingDecorator : IActivityService
{
    private readonly IActivityService _decoratedActivityService;
    private readonly ILoggerFactory _loggerFactory;

    0 references
    public ActivityServiceLoggingDecorator(IActivityService decoratedActivityService, ILoggerFactory
    {
        _decoratedActivityService = decoratedActivityService;
        _loggerFactory = loggerFactory;
    }
}

```

У цьому прикладі використовується імплментація з декорованого класу та додається функціонал для запису у файл.

```

3 references
public async Task<IEnumerable<Activity>> GetAllActivitiesAsync()
{
    using (var logger = _loggerFactory.CreateLogger())
    {
        try
        {
            logger.WriteTextToFile("Retrieving all activities.");
            var result = await _decoratedActivityService.GetAllActivitiesAsync();
            logger.WriteTextToFile($"Retrieved all activities successfully. Count: {result.Count()}.");
            return result;
        }
        catch (Exception ex)
        {
            logger.WriteTextToFile($"Error retrieving activities. {ex}");
            throw;
        }
    }
}

```

Приклад запису у файл

```

3/11/2024 8:32:33 PM: Retrieving all activities.
3/11/2024 8:32:35 PM: Retrieved all activities successfully. Count: 1.

```

## Поведінкові (Стратегія)

### Суть патерна

**Стратегія** — це поведінковий патерн проектування, який визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна замінювати один на інший прямо під час виконання програми.

`IBookingPricingStrategy` це інтерфейс, який описує загальний метод для стратегії визначення ціни.

`StandardPricingStrategy` визначає стратегію для визначення ціни для не підписників.

`MemberPricingStrategy` визначає стратегію для визначення ціни для підписників. У цьому класі учасник(member) отримує знижку 5%(тобто ціна дешевша на 5%).

```

5 references
public interface IBookingPricingStrategy
{
    3 references
    double CalculatePrice(Booking booking);
}

1 reference
public class StandardPricingStrategy : IBookingPricingStrategy
{
    2 references
    public double CalculatePrice(Booking booking)
    {
        return booking.Price;
    }
}

1 reference
public class MemberPricingStrategy : IBookingPricingStrategy
{
    2 references
    public double CalculatePrice(Booking booking)
    {
        // підписники мають знижку
        return booking.Price * 0.95;
    }
}

```

Логіка для визначення чи є користувач підписником (member). Відповідно до статусу задається стратегія.

```

1 reference
private async Task<IBookingPricingStrategy> SelectPriceStrategy(Guid userId)
{
    var user = await userService.GetUserByIdAsync(userId);

    IBookingPricingStrategy pricingStrategy;
    // Якщо користувач є підписником сайту то він отримує знижку
    if (user.IsMember)
    {
        pricingStrategy = new MemberPricingStrategy();
    }
    else
    {
        pricingStrategy = new StandardPricingStrategy();
    }
    return pricingStrategy;
}

```

## 2. Implement simple (REST/graphQL) API

- CRUD

```
6 references
public interface IUserService
{
    2 references
    Task<User> CreateUserAsync(User newUser);
    2 references
    Task<IEnumerable<User>> GetAllUsersAsync();
    3 references
    Task<User> GetUserByIdAsync(Guid userId);
    2 references
    Task<User> UpdateUserAsync(Guid userId, User updatedUser);
    2 references
    Task<bool> DeleteUserAsync(Guid userId);
}
```



2 references

```
public class UserService : IUserService
{
    private readonly ApplicationDbContext _context;

    0 references
    public UserService(ApplicationDbContext context)
    {
        _context = context;
    }

    2 references
    public async Task<User> CreateUserAsync(User newUser)
    {
        _context.Users.Add(newUser);
        await _context.SaveChangesAsync();
        return newUser;
    }

    2 references
    public async Task<IEnumerable<User>> GetAllUsersAsync()
    {
        return await _context.Users.ToListAsync();
    }

    3 references
    public async Task<User> GetUserByIdAsync(Guid userId)
    {
        return await _context.Users.FirstOrDefaultAsync(u => u.UserId == userId);
    }
}
```

2 references

```
public async Task<User> UpdateUserAsync(Guid userId, User updatedUser)
{
    var user = await _context.Users.FirstOrDefaultAsync(u => u.UserId == userId);

    if (user == null)
    {
        return null;
    }

    user.FirstName = updatedUser.FirstName;
    user.LastName = updatedUser.LastName;
    user.Email = updatedUser.Email;
    user.PasswordHash = updatedUser.PasswordHash;
    // ... other properties

    _context.Users.Update(user);
    await _context.SaveChangesAsync();
    return user;
}
```

2 references

```
public async Task<bool> DeleteUserAsync(Guid userId)
{
    var user = await _context.Users.FirstOrDefaultAsync(u => u.UserId == userId);

    if (user == null)
    {
        return false;
    }

    _context.Users.Remove(user);
    await _context.SaveChangesAsync();
    return true;
}
```

## - GET/POST/PUT/DELETE convention

Users		^
GET	/Users	▼
POST	/Users	▼
GET	/Users/{id}	▼
PUT	/Users/{id}	▼
DELETE	/Users/{id}	▼

```
// GET: /Users
[HttpGet]
0 references
public async Task<IActionResult> GetUsers()
{
    var users = await _userService.GetAllUsersAsync();
    return Ok(users);
}

// GET: /Users/{id}
[HttpGet("{id}")]
1 reference
public async Task<IActionResult> GetUser(Guid id)
{
    var user = await _userService.GetUserByIdAsync(id);
    if (user == null)
    {
        return NotFound();
    }
    return Ok(user);
}
```

```
// POST: /Users
[HttpPost]
0 references
public async Task<IActionResult> CreateUser([FromBody] User user)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    var createdUser = await _userService.CreateUserAsync(user);
    return CreatedAtAction(nameof(GetUser), new { id = createdUser.UserId }, createdUser);
}
```

```
// PUT: /Users/{id}
[HttpPut("{id}")]
0 references
public async Task<IActionResult> UpdateUser(Guid id, [FromBody] User updatedUser)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    var user = await _userService.UpdateUserAsync(id, updatedUser);
    if (user == null)
    {
        return NotFound();
    }
    return Ok(user);
}
```

```
// DELETE: /Users/{id}
[HttpDelete("{id}")]
0 references
public async Task<IActionResult> DeleteUser(Guid id)
{
    var result = await _userService.DeleteUserAsync(id);
    if (!result)
    {
        return NotFound();
    }
    return NoContent();
}
```

CommunityCenterApi / Users / /Users Create Standard User Copy

POST

{{(baseUrl)}Users

Send

ParamsAuthorizationHeaders (9)BodyPre-request ScriptTestsSettingsCookies

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

```
1 {
2   "userId": "3fa85f64-5717-4562-b3fc-2c963f66afa9",
3   "firstName": "Test1",
4   "lastName": "Standard",
5   "email": "user1@example.com",
6   "dateOfBirth": "2024-03-11T16:57:11.646Z",
7   "isMember": false,
8   "passwordHash": "string",
9   "userPermissions": [],
10  "activities": [],
11  "bookings": []
12 }
```

BodyCookiesHeaders (5)Test ResultsStatus: 201 CreatedTime: 198 msSize: 557 BSave as example

PrettyRawPreviewVisualizeJSON

```
1 {
2   "$id": "1",
3   "userId": "3fa85f64-5717-4562-b3fc-2c963f66afa9",
4   "firstName": "Test1",
5   "lastName": "Standard",
6   "email": "user1@example.com",
7   "dateOfBirth": "2024-03-11T16:57:11.646Z",
8   "isMember": false,
9   "passwordHash": "string",
10  "userPermissions": {
11    "$id": "2",
```

CommunityCenterApi / Users / /Users Get All

GET

{{(baseUrl)}Users

Send

ParamsAuthorizationHeaders (6)BodyPre-request ScriptTestsSettingsCookies

noneform-datax-www-form-urlencodedrawbinaryGraphQL











This request does not have a body

BodyCookiesHeaders (4)Test ResultsStatus: 200 OKTime: 119 msSize: 970 BSave as example

PrettyRawPreviewVisualizeJSON

```
24   "isMember": false,
25   "passwordHash": "string",
26   "userPermissions": null,
27   "activities": null,
28   "bookings": null
29 },
30 {
31   "$id": "4",
32   "userId": "3fa85f64-5717-4562-b3fc-2c963f66afa9",
33   "firstName": "Test1",
34   "lastName": "Standard",
35   "email": "user1@example.com",
36   "dateOfBirth": "2024-03-11T16:57:11.646Z",
37   "isMember": false,
38   "passwordHash": "string",
39   "userPermissions": null,
40   "activities": null,
41   "bookings": null
42 }
43 ]
44 }
```

- at least 3 total entities in total

- [-]  Tables
  - [+]  System Tables
  - [+]  FileTables
  - [+]  External Tables
  - [+]  Graph Tables
  - [+]  dbo.\_\_EFMigrationsHistory
  - [+]  dbo.Activities
  - [+]  dbo.Bookings
  - [+]  dbo.UserPermissions
  - [+]  dbo.Users

```
User {  
  userId > [...]  
  firstName* > [...]  
  lastName* > [...]  
  email* > [...]  
  dateOfBirth* > [...]  
  isMember* > [...]  
  passwordHash* > [...]  
  userPermissions > [...]  
  activities > [...]  
  bookings > [...]  
}
```

```

UserPermission v {
  userPermissionId      > [...]
  permissionName        > [...]
  userId                > [...]
  user                  User v {
    userId              > [...]
    firstName*          > [...]
    lastName*           > [...]
    email*              > [...]
    dateOfBirth*        > [...]
    isMember*           > [...]
    passwordHash*       > [...]
    userPermissions     > [...]
    activities           > [...]
    bookings            > [...]
  }
}

```

```

Booking ∨ {
  bookingId      > [...]
  date           > [...]
  status         > [...]
  price          > [...]
  userId         > [...]
  user           User ∨ {
    userId       > [...]
    firstName*   > [...]
    lastName*    > [...]
    email*       > [...]
    dateOfBirth* > [...]
    isMember*    > [...]
    passwordHash* > [...]
    userPermissions > [...]
    activities   > [...]
    bookings    > [...]
  }
  activityId     > [...]
  activity       Activity ∨ {
    activityId   > [...]
    activityName > [...]
    description  > [...]
    userId       > [...]
    user         > {...}
    bookings    > [...]
  }
}

```



```

Activity ▾ {
  activityId          > [...]
  activityName        > [...]
  description         > [...]
  userId             > [...]
  user
    User ▾ {
      userId          > [...]
      firstName*      > [...]
      lastName*       > [...]
      email*          > [...]
      dateOfBirth*   > [...]
      isMember*       > [...]
      passwordHash*   > [...]
      userPermissions > [...]
      activities      > [...]
      bookings        > [...]
    }
  bookings            > [...]
}

```

- nested entities

```
Activity ▾ {
  activityId          > [...]
  activityName        > [...]
  description         > [...]
  userId             > [...]
  user
    User ▾ {
      userId          > [...]
      firstName*      > [...]
      lastName*       > [...]
      email*          > [...]
      dateOfBirth*    > [...]
      isMember*       > [...]
      passwordHash*   > [...]
      userPermissions > [...]
      activities      > [...]
      bookings        > [...]
    }
  bookings
    > [...]
}
```

```

Booking ∨ {
  bookingId      > [...]
  date           > [...]
  status         > [...]
  price          > [...]
  userId         > [...]
  user           User ∨ {
    userId       > [...]
    firstName*   > [...]
    lastName*    > [...]
    email*       > [...]
    dateOfBirth* > [...]
    isMember*    > [...]
    passwordHash > [...]
    userPermissions > [...]
    activities   > [...]
    bookings    > [...]
  }
  activityId     > [...]
  activity       Activity ∨ {
    activityId   > [...]
    activityName > [...]
    description  > [...]
    userId       > [...]
    user         > [...]
    bookings    > [...]
  }
}

```

- paging/sorting

```

[HttpGet("GetAllActivitiesOrderedByDate")]
0 references
public async Task<ActionResult<IEnumerable<Activity>>> GetAllActivitiesOrderedByDate()
{
    var activities = await _activityService.GetAllActivitiesAsync();
    var orderedActivities = activities.OrderBy(a => a.Date).ToList();
    return Ok(orderedActivities);
}

```

### 3. Simple frontend application

#### - Authorization

Контролер для авторизації на сервері. Контроллер аутентифікує користувача та генерує JWT токен та надсилає його клієнту

```
[HttpPost("login")]
0 references
public async Task<IActionResult> Login([FromBody] UserLoginRequest request)
{
    var user = await _userService.Authenticate(request.Username, request.Password);

    if (user == null)
        return BadRequest(new { message = "Username or password is incorrect" });

    var token = GenerateJwtToken(user);

    return Ok(new
    {
        UserId = user.UserId,
        Username = user.Email,
        Token = token
    });
}
```

Токен генерується за допомогою бібліотеки System.IdentityModel.Tokens.Jwt

```
1 reference
private string GenerateJwtToken(User user)
{
    var tokenHandler = new JwtSecurityTokenHandler();
    var key = Encoding.ASCII.GetBytes(_configuration["Jwt:Key"]);
    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(new[]
        {
            new Claim("id", user.UserId.ToString()),
            new Claim(ClaimTypes.Name, user.Email)
        }),
        Expires = DateTime.UtcNow.AddDays(7),
        SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
    };

    var token = tokenHandler.CreateToken(tokenDescriptor);
    return tokenHandler.WriteToken(token);
}
```

Авторизація працює наступним чином:

1. Отримання користувача за поштою
2. Порівняння хешу паролей

```
2 references
public async Task<User> Authenticate(string username, string password)
{
    var user = await _context.Users
                                .AsNoTracking()
                                .SingleOrDefaultAsync(u => u.Email == username);

    if (user == null)
    {
        return null;
    }

    var hashedPassword = user.PasswordHash;

    var verificationResult = PasswordHasher.CheckPassword(hashedPassword, password);

    if (!verificationResult.Verified)
    {
        return null;
    }

    return user;
}
```

Сервіс для авторизації на стороні клієнта. Відправляє запит у якому є логін та пароль. У раз успішного виконання, аутентифікаційний токен зберігається у пам'яті браузера.,

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable } from 'rxjs';
import { map } from 'rxjs/operators';
import { environment } from '../environments/environment';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private currentUserSubject: BehaviorSubject<any>;
  public currentUser: Observable<any>;

  constructor(private http: HttpClient) {
    this.currentUserSubject = new BehaviorSubject<any>(
      JSON.parse(localStorage.getItem('currentUser') || '{}')
    );
    this.currentUser = this.currentUserSubject.asObservable();
  }

  public get currentUserValue() {
    return this.currentUserSubject.value;
  }

  login(username: string, password: string) {
    return this.http.post<any>(`${environment.apiUrl}api/auth/login`, { username, password })
      .pipe(map(user => {
        // store user details and jwt token in local storage to keep user logged in between page refreshes
        localStorage.setItem('currentUser', JSON.stringify(user));
        this.currentUserSubject.next(user);
        return user;
      }));
  }

  logout() {
    // remove user from local storage to log user out
    localStorage.removeItem('currentUser');
    this.currentUserSubject.next(null);
  }
}

```

Клас для того, щоб кожен запит мав аутентифікаційний токен

```

import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor
} from '@angular/common/http';
import { Observable } from 'rxjs';
import { AuthService } from '../authService';

@Injectable()
export class JwtInterceptor implements HttpInterceptor {
  constructor(private authenticationService: AuthService) {}

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    // add authorization header with jwt token if available
    let currentUser = this.authenticationService.currentUserValue;
    if (currentUser && currentUser.token) {
      request = request.clone({
        setHeaders: {
          Authorization: `Bearer ${currentUser.token}`
        }
      });
    }


    return next.handle(request);
  }
}

```

Сторінка авторизації

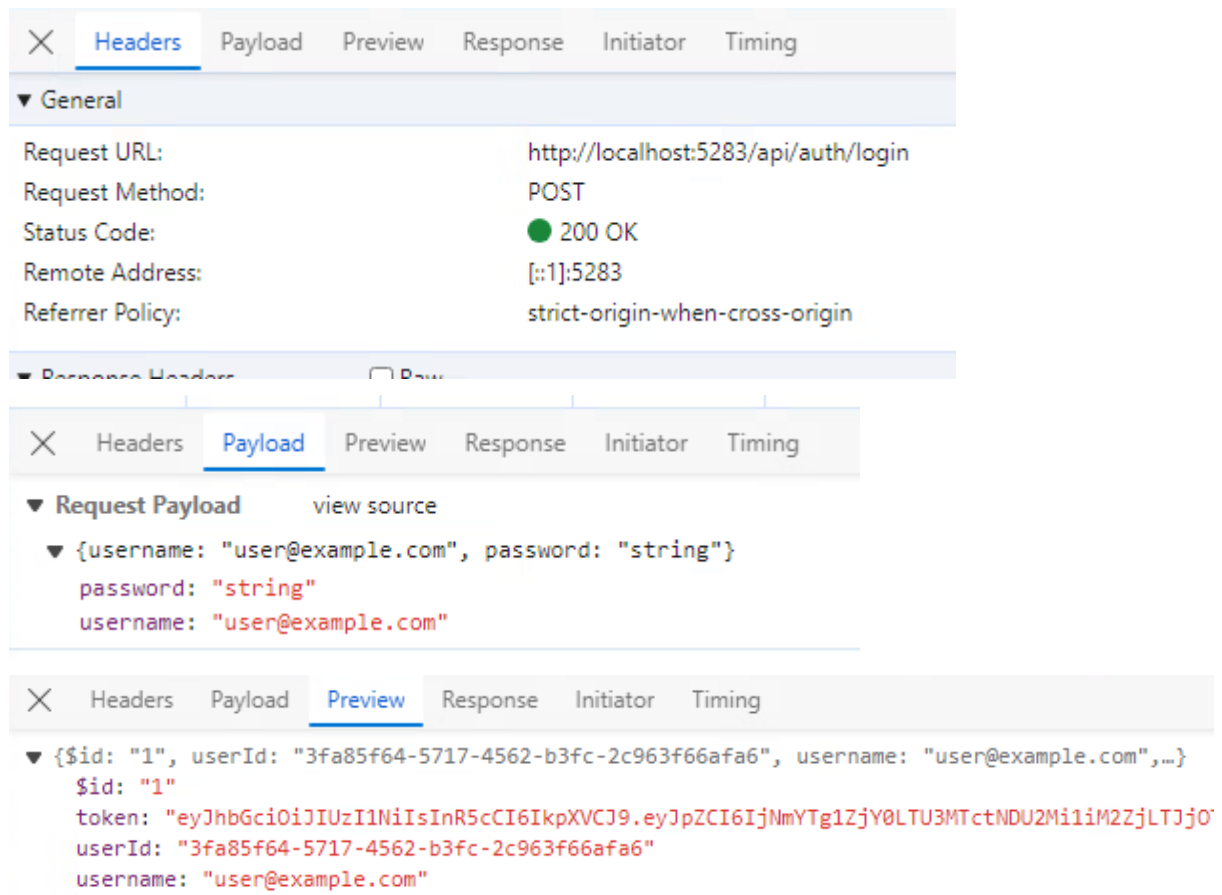
Activities Add New Activity User Profile

Username:

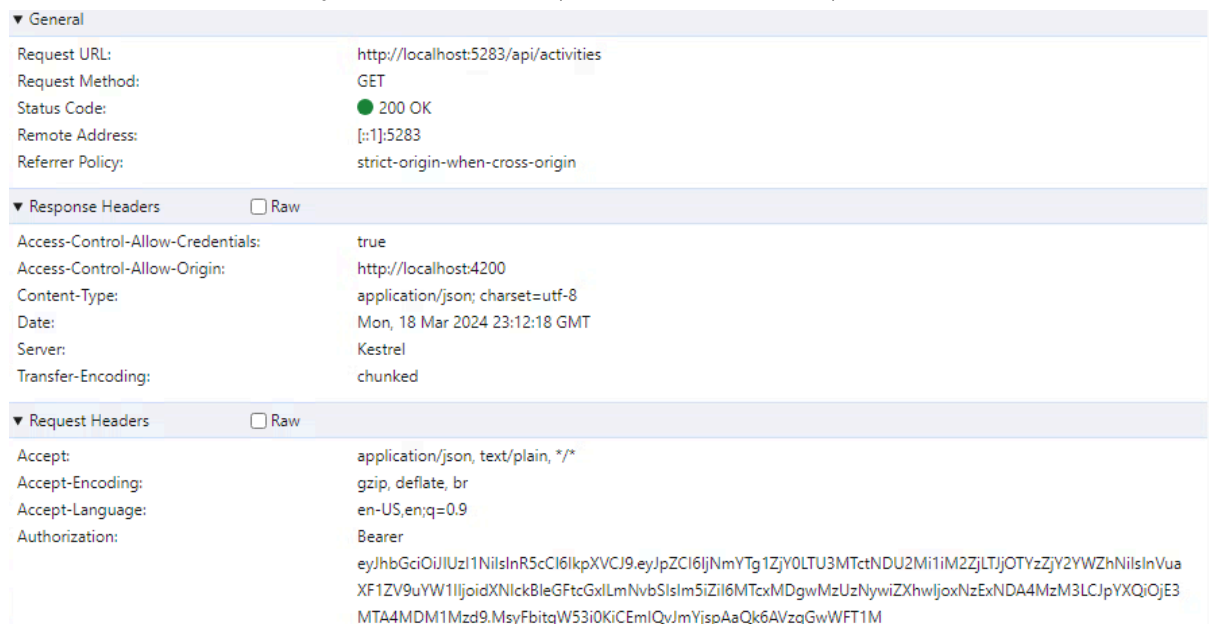
Password:


Login

Запит зі сторони клієта який авторизує користувача



Далі токен використовується в запитах(секція Authorization)



- 3 pages

## Сторінка логіну



**Username:**

123@1.1

**Password:**

.....

Login

Register

Сторінка реєстрації

**First Name:**

TestSQL

**Last Name:**

Testing

**Email:**

test@test.test

**Password:**

.....

**Date of Birth:**

05 / 16 / 2003

**Is Member:**

☐

Register

Сторінка активностей

Activities Add New Activity Bookings		
Activity List		Search activities...
Test	something 1231 123142 Date: Mar 19, 2024	Booked
The created	grgdgrg Date: Apr 19, 2024	Booked
The created	grgdgrg Date: Apr 30, 2024	Booked
The created	faefefef Date: Apr 30, 2024	Booked
Test	something 1231 123142 Date: Apr 30, 2024	Book
Test	something 1231 123142 Date: Apr 30, 2024	Book
Test	something 1231 123142 Date: Apr 30, 2024	Book
Test	something 1231 123142 Date: Apr 30, 2024	Book

Створення активності

Name:

new activity

Description:

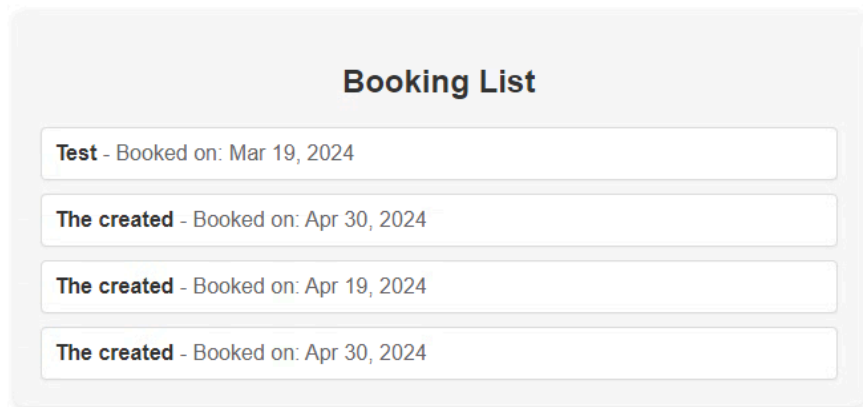
23124234234

Start Date:

05/18/2024 11:46 AM

Create Activity

Список зарезервованих активностей



## - the shared state between pages

Для імплементації використовується localStorage

Сервіс AuthService забезпечує управління аутентифікацією користувачів у Angular застосунку. Він використовує HttpClient для взаємодії з API аутентифікації і зберігає дані користувача у localStorage, що дозволяє підтримувати стан входу користувача при перезавантаженні сторінки.

У конструкторі сервісу ініціюється BehaviorSubject з поточним користувачем, де значення береться з localStorage. Якщо у localStorage нічого немає, використовується пустий об'єкт {}.

```
constructor(private http: HttpClient) {  
  this.currentUserSubject = new BehaviorSubject<any>(  
    JSON.parse(localStorage.getItem('currentUser') || '{}')  
  );  
  this.currentUser = this.currentUserSubject.asObservable();  
}
```

### Методи

- login(username: string, password: string):
  - Аутентифікує користувача за допомогою API, відправляючи ім'я користувача та пароль.

- У разі успішної відповіді з API, деталі користувача та JWT токен зберігаються у localStorage. Це дозволяє підтримувати стан користувача після перезавантаження сторінки.
- Потім ці дані передаються через next() метод BehaviorSubject, оновлюючи спостерігачів про новий стан користувача.

```
login(username: string, password: string) {
  return this.http.post<any>(`${environment.apiUrl}api/auth/login`, { username, password })
    .pipe(map(user => {
      // store user details and jwt token in local storage to keep user logged in between page refreshes
      localStorage.setItem('currentUser', JSON.stringify(user));
      this.currentUserSubject.next(user);
      return user;
    }));
}
```

- logout():
  - Видаляє дані користувача з localStorage і очищає поточний стан у BehaviorSubject.

```
logout() {
  // remove user from local storage to log user out
  localStorage.removeItem('currentUser');
  this.currentUserSubject.next(null);
}
```

#### Метод currentUserValue

- Опис: Повертає поточний об'єкт користувача з BehaviorSubject.
- Використання: Цей метод використовується для доступу до повного об'єкта користувача, який містить усі дані, збережені під час аутентифікації, такі як userId, username, токен доступу та інші можливі атрибути.

#### Метод currentUserId

- Опис: Повертає userId поточного користувача.
- Використання: Використовується, коли потрібен лише ID користувача, наприклад, для запитів до API, де необхідно передати ID користувача.

#### Метод currentUsername

- Опис: Повертає ім'я користувача (username) поточного користувача.
- Використання: Це зручно, коли потрібно відображати ім'я користувача на UI або використовувати його в логічних перевірках у вашому застосунку.

```

public get currentUserValue() {
    return this.currentUserSubject.value;
}

public get currentUserId() {
    return this.currentUserSubject.value.userId;
}

public get currentUsername() {
    return this.currentUserSubject.value.username;
}

```

Ці методи дуже зручні для використання у компонентах Angular, де потрібно реагувати на зміни стану користувача або просто отримати доступ до даних користувача без необхідності щоразу звертатися до сервера. Вони дозволяють ефективно розділити логіку управління даними користувача та їх відображення у компонентах, сприяючи чистоті коду та зменшенню залежностей.

## 4. Testing

### - 30% code coverage

Використані бібліотеки

1. NUnit - це популярна бібліотека для юніт-тестування в .NET, яка дозволяє структурувати тестові кейси та легко проводити асертації.
2. Moq - це бібліотека для мокування, яка використовується для створення легких, налаштовуваних тестових подвійників. Вона дуже корисна для імітації залежностей у тестуванні класів, що дозволяє ізолювати поведінку, яка перевіряється.
3. Entity Framework In-Memory Database Provider - використовується для імітації бази даних під час тестування, дозволяючи виконувати інтеграційні тести без необхідності звертання до справжньої бази даних.
4. xUnit - ще одна популярна тестова бібліотека, аналогічна NUnit, яка часто використовується для написання тестів у проектах на .NET.
5. Coverlet - бібліотека для вимірювання покриття коду в .NET, яка інтегрується з .NET Core та використовується для генерації звітів про покриття коду під час виконання тестів.
6. ReportGenerator - інструмент, який перетворює звіти про покриття коду, сгенеровані інструментами, такими як Coverlet, у детальніші, легкі для читання HTML-звіти.

## Summary

### Information

Parser: Cobertura  
Assemblies: 1  
Classes: 31  
Files: 32  
Coverage date: 5/6/2024 - 12:40:23 PM

### Line coverage

Covered lines: 525  
Uncovered lines: 1141  
Coverable lines: 1666  
Total lines: 2550  
Line coverage: 31.5%

### Branch coverage

Covered branches: 62  
Total branches: 86  
Branch coverage: 72%

- Покриття рядків (Line coverage): Звіт показує, що покриття рядків становить 31%, що вказує на те, що лише приблизно третина коду в проекті випробована за допомогою автоматичних тестів. Це важливий показник, який визначає, яку частину коду було виконано під час тестування.
- Покриття гілок (Branch coverage): Звіт вказує на покриття гілок 72%, що демонструє ефективність тестування у виявленні можливих шляхів виконання в різних умовних операторах і циклах у коді.

## Coverage

Line coverage		Branch coverage	
▼ Covered	▼ Uncovered	▼ Covered	▼ Total
525	1141	62	86
▼ Percentage	▼ Percentage	▼ Percentage	▼ Percentage
31.5%	69.5%	72%	83.3%
CommunityCenterApi			
CommunityCenterApi.BookingPricing.MemberPricingStrategy	3	0	3
CommunityCenterApi.BookingPricing.StandardPricingStrategy	3	0	3
CommunityCenterApi.Controllers.ActivitiesController	52	3	55
CommunityCenterApi.Controllers.ActivityDto	4	0	4
CommunityCenterApi.Controllers.ActivityModel	1	0	1
CommunityCenterApi.Controllers.AuthController	33	0	33
CommunityCenterApi.Controllers.BookingDto	5	0	5
CommunityCenterApi.Controllers.BookingsController	58	20	78
CommunityCenterApi.Controllers.UsersController	37	2	39
CommunityCenterApi.DB.ApplicationDbContext	27	0	27
CommunityCenterApi.Helpers.PasswordHasher	31	0	31
CommunityCenterApi.Log.FileLogger	0	12	12
CommunityCenterApi.Log.FileLoggerFactory	0	7	7
CommunityCenterApi.Migrations.AddBookings	0	203	203
CommunityCenterApi.Migrations.AddDateToActivity	0	192	192
CommunityCenterApi.Migrations.AddPricing	0	192	192
CommunityCenterApi.Migrations.AddUserPermissionsAndActivities	0	166	166
CommunityCenterApi.Migrations.ApplicationDbContextModelSnapshot	0	170	170
CommunityCenterApi.Migrations.InitialCreate	0	60	60
CommunityCenterApi.Models.Activity	7	0	7
CommunityCenterApi.Models.Booking	7	0	7
CommunityCenterApi.Models.User	7	3	10
CommunityCenterApi.Models.UserLoginRequest	2	0	2
CommunityCenterApi.Models.UserPermission	4	0	4
CommunityCenterApi.Services.Implementations.ActivityService	38	3	41
CommunityCenterApi.Services.Implementations.ActivityServiceLoggingDecorator	42	46	88
CommunityCenterApi.Services.Implementations.BookingService	44	18	62
CommunityCenterApi.Services.Implementations.UserPermissionService	37	0	37
CommunityCenterApi.Services.Implementations.UserService	52	0	52
Program	0	37	37
UserPermissionsController	31	7	38

- Загальний огляд: Звіт включає рядки для кожного класу чи файлу в проекті, такі як контролери, моделі, міграції, сервіси, та інші компоненти.
- Покриття рядків (Line coverage): Відображає кількість рядків коду, які були виконані під час тестування, порівняно з загальною кількістю рядків, які можуть бути покриті. Наприклад, для ActivityService покриття складає 47.7%.
- Покриття гілок (Branch coverage): Ілюструє ефективність тестів у перевірці різних логічних шляхів у коді. Наприклад, для BookingsController покриття гілок становить 90%.
- Індикатори покриття: Кожен рядок звіту містить графічний індикатор (зелений для покритих рядків, червоний для непокритих), що дозволяє швидко оцінити стан тестового покриття для кожного компонента.

- performance testing at least 1 endpoint
- \* complex scenario testing (use endpoint out put as input to different call)
- \* Scrap some data with Selenium (or similar), auth navigate to some page, scrap data

## 5. Deployment

- deploy applications to some cloud (Azure/AWS/GCP/Heroku/DigitalOcean)
- deploy as a container unit
- configure CI/CD