

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

1. Design patterns

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

Завдання

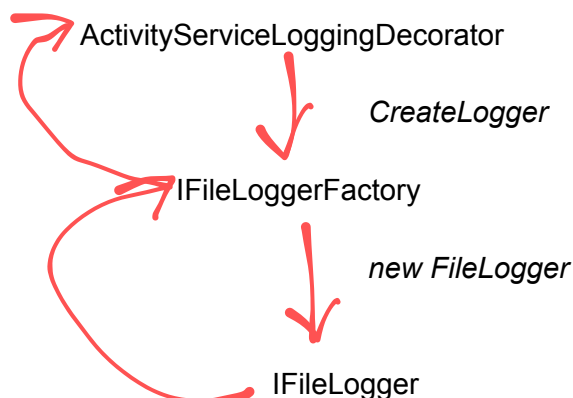
1. Design patterns (<https://refactoring.guru/uk/design-patterns/catalog>)
 - 1 pattern from each category
 - capable of explaining idea

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 імплементація 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 замінити 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 ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

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": []

}

BodyCookiesHeaders (5)Test Results

Status: 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 ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

This request does not have a body

BodyCookiesHeaders (4)Test Results

Status: 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": "2024-03-11T16:57:11.646",

37

"isMember": false,

38

"passwordHash": "string",

39

"userPermissions": null,

40

"activities": null,

41

"bookings": null

42

}

43











]

44

}

13

- 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 ▾ {
  userPermissionId      > [...]
  permissionName        > [...]
  userId                > [...]
  user                  User ▾ {
    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

3. Simple frontend application

- Authorization
- 3 pages
- the shared state between pages

4. Testing

- 30% code coverage
- performance testing at least 1 endpoint
- * complex scenario testing (use endpoint output as input to different call)
- * Scrape some data with Selenium (or similar), authenticate, navigate to some page, scrape data

5. Deployment

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