Міністерство освіти і науки України

Харківський національний Університет радіоелектроніки

Кафедра «Програмна інженерія»

**ЗВІТ**

* + лабораторної работи №2
* дисципліни «Аналіз та рефакторинг коду»

на тему: «Розробка серверної/back-end частини програмної системи»

Виконав: Перевірив:

ст. гр. ПЗПІ-20-3 ст. викл. каф. ПІ

Кулешов Микита Сокорчук І. П.

Харків 2022

**ВСТУП**

Метою лабораторної роботи є розробка серверної/back-end частини проекту за темою: «Програмна система для автоматизації вигулу домашніх тварин».

Хід лабораторної роботи:

1. опис прийнятих інженерних рішень;
2. загальна архітектура системи;
3. структура бази даних;
4. будова серверних компонентів системи, специфікація REST, методи взаємодії компонентів.

**1 ТЕХНОЛОГІЇ ТА АРХИТЕКТУРНІ РІШЕННЯ**

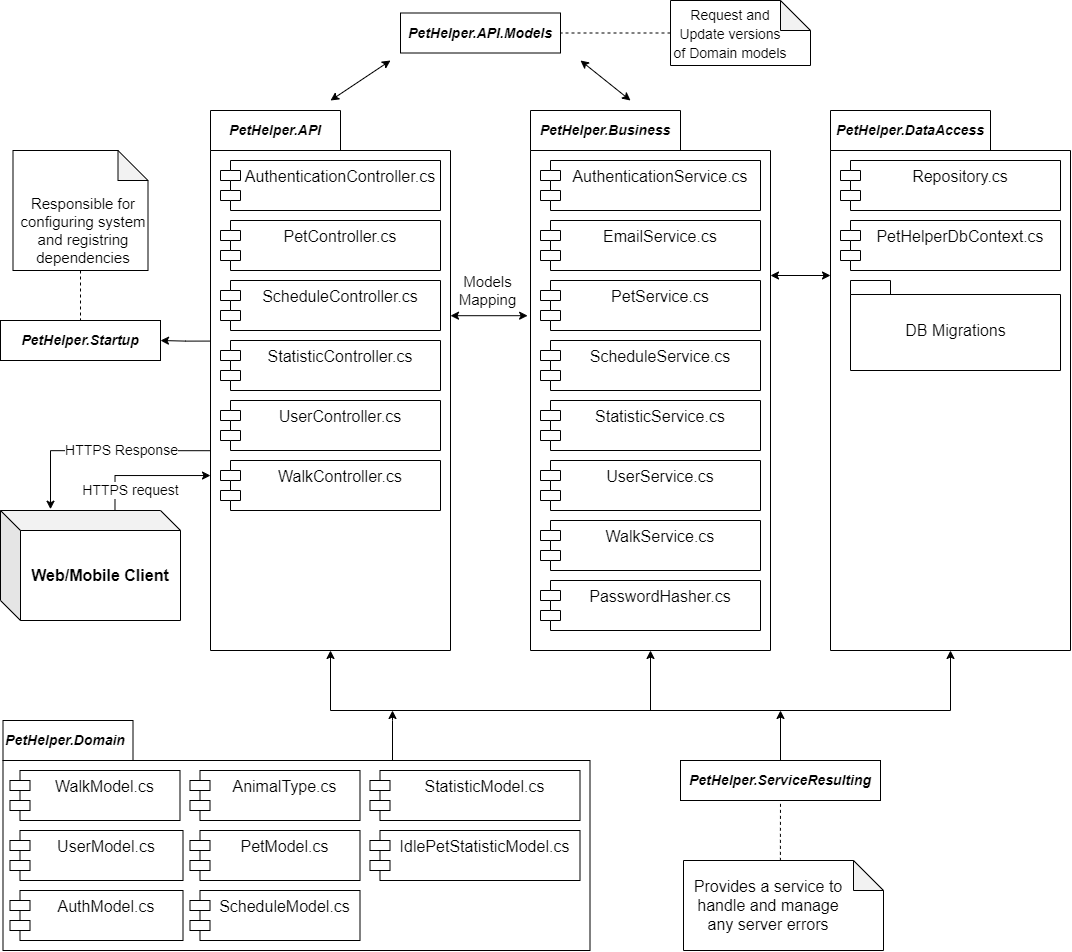
 На рисунку 1.1 наведено діаграму компонентів, що описує архітектуру програми.

Рисунок 1.1 – Діаграма компонентів

Для створення програмного забезпечення було обрано наступні інструменти розробки:

* серверна частина написана на мові C# з використанням технології .NET ASP.NET Core Web API з використанням стилю REST;
* база даних створена за допомогою технології MS SQL Server, доступ до неї здійснюється з використанням технології Entity Framework Core;

Як приклад, код контролера, що відповідає за виконання операцій над твариною наведено у додатку A. Відповідний сервіс для цього контролера наведено у додатку Б. Повний проект завантажено до хмарного сховища Google Disk [1].

**2 ДІАГРАМА ПРЕЦЕНДЕНТІВ**

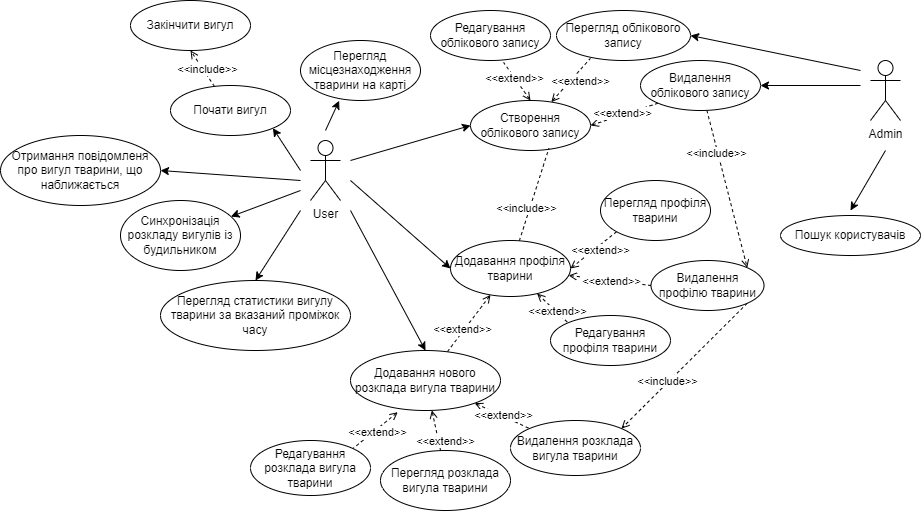
 В програмі наявні дві ролі: адміністратор та користувач. Щоб наглядно показати наявний функціонал та його доступність в залежності від ролі, було створено діаграму прецедентів (див. рис. 2.1).

Рисунок 2.1 – Діаграма прецедентів

**3 ДІАГРАМА РОЗГОРТАННЯ**

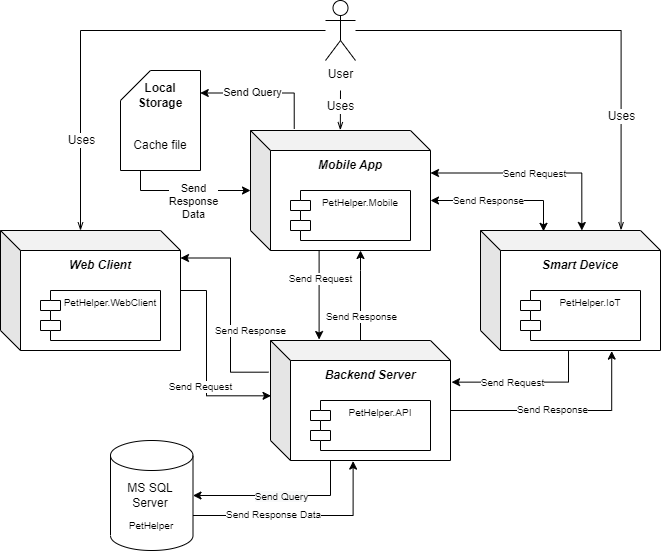
 На рисунку 3.1 представлена діаграма розгортання. Цілком система складається з шести частин, їх взаємодія відображана на рисунку.

Рисунок 3.1 – Діаграма розгортання

Всього є три функціональні частини, що тримають зв’язок із сервером – мобільний додаток, веб-клієнт та смарт девайс. Усі вони здійснюють обмін даними з сервером через інтерфейс REST, використовуючи HTTPS протокол.

Смарт девайс, тримає зв’язок з мобільним додатком, за допомогою якого з нього можна отримати певну інформацію, наприклад, поточну геолокацію тварини. В тей самий час, девайс також тримає зв’язок з сервером для можливості обмінюватись необхідними даними напряму, не використовуючи мобільний застосунок як посередника.

Мобільний додаток має додаткове локальне сховище у вигляді кеш файлу, в якому зберігаються часто потрібні дані, для оптимізації швидкості загрузки окремих вікон програми.

**4 БАЗА ДАНИХ**

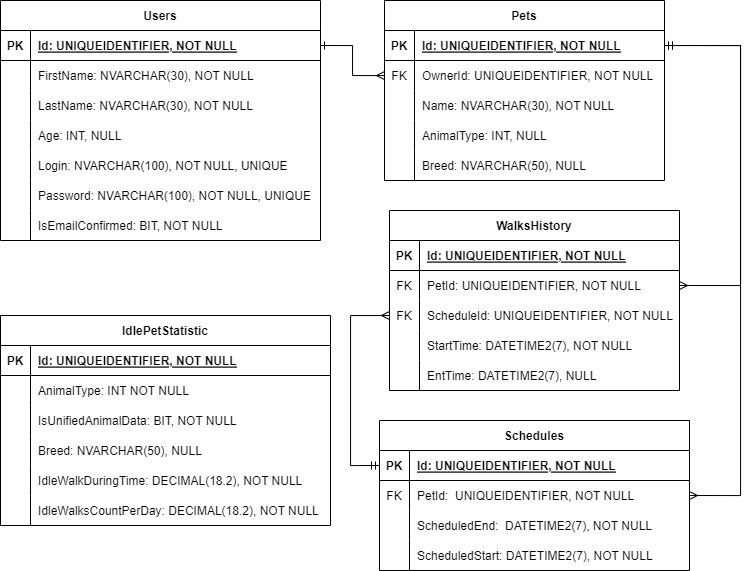
 База даних складається з 5 таблиць. На рисунку 4.1 представлена ER-діаграма бази даних, яка відображає таблиці, їх атрибути з обмеженнями та типами даних, а також зв’язки між таблицями.

Рисунок 4.1 – ER-діаграма

Таблиця IdlePetStatistic не має зв’язків з жодною із таблиць, оскільки вона використовується лише для збереження ідеальних статистичних даних для кожного типу та породи тварини, що підтримується системою, необхідних при запиті користувача відобразити статистику вигулів його окремої тварини за певний проміжок часу. Користувач не повинен мати можливості змінювати вміст таблиці або мати до неї доступ, вона використовується лише внутрьошньо в системі.

**5 СПЕЦИФІКАЦІЯ REST API**

Для полегшення взаємодії з API було створено її специфікацію, що наведено у таблиці 5.1.

Таблиця 5.1 – REST специфікація

|  |  |  |  |
| --- | --- | --- | --- |
| Контролер | HTTP-запит | Шлях | Призначення |
| Authentication | POST | /api/authentication/register | Реєстрація користувача |
| POST | /api/authentication/login | Аунтифікація та aвторизація користувача |
| PATCH | /api/authentication/confirmEmail/{key} | Підтвердження пошти користувача |
| GET | /api/authentication/logout | Вихід користувача із системи |
| Pet | GET | /api/pet/{petId} | Отримання тварини за ідентифікатором |
| POST | /api/pet | Додання нової тварини до сховища |
| PUT | /api/pet{petId} | Оновлення інформації про певну тварину |
| DELETE | /api/pet/{petId} | Видалення тварини з системи |

Продовження таблиці 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| Контролер | HTTP-запит | Шлях | Призначення |
| User | GET | /api/user/{userId} | Отримання інформації про користувача |
| DELETE | /api/user/{userId} | Видалення користувача із системи |
| PUT | /api/user/{userId} | Оновлення інформації про користувача |
| Schedule | POST | /api/schedule | Додання нового розкладу вигулу тварини |
| PUT | /api/schedule/{sheduleId} | Оновлення інформації про розклад вигулу тварини |
| GET | /api/schedule/{scheduleId} | Отримання інформації про певний розклад вигулу тварини |
| GET | /api/schedule/pet/{petId} | Отримання інформації про усі розклади вигулу для певної тварини |
| DELETE | /api/schedule/{scheduleId} | Видалення розклада вигула тварини |

Продовження таблиці 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| Контролер | HTTP-запит | Шлях | Призначення |
| Walk | POST | /api/walk | Додання інформації про певний вигул тварини |
| PATCH | /api/walk/{walkId} | Запис часу закінчення певного вигулу тварини |
| GET | /api/walk/{petId} | Отримання інформації про історію вигулів для певної тварини |
| Statistic | POST | /api/statistic | Розрахунок та отримання статистику вигулу для певної тварини |

**ВИСНОВКИ**

* ході виконання було розроблено серверна/back-end частина до проекту на тему « Програмна система для автоматизації вигулу домашніх тварин ».

Представлена загальна архітектура системи, показана будова її серверних конпонентів та структара бази даних. Представлені діаграма компонентів, діаграма розгортання, діаграма прецедентів та ER-діаграма, надано REST специфікацію API.

Запис з демонстрацією роботи серверної частини можна подивитись за посиланням до хмарного сховища Google Disk [2].

**ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ**

1. Код проекту. Google Disk. URL: [https://drive.google.com](https://drive.google.com/drive/folders/11QPZSDeoS4KTSkMEiV7mbhUmAyypcs54?usp=share_link) (дата звернення: 3.11.2022).
2. Відео демонстрація роботи. Google Dist. URL: (дата звернення: 3.11.2022).

**ДОДАТОК А**

PetController.cs

1. using Microsoft.AspNetCore.Authorization;
2. using Microsoft.AspNetCore.Mvc;
3. using PetHelper.Api.Models.RequestModels.Pets;
4. using PetHelper.Business.Pet;
5. using PetHelper.Domain.Pets;
6. using PetHelper.ServiceResulting;
7. namespace PetHelper.Api.Controllers
8. {
9. [Authorize]
10. public class PetController : ResultingController
11. {
12. private readonly IPetService \_petService;
13. public PetController(IPetService petService) => \_petService = petService;
14. [HttpPost("")]
15. public async Task<ServiceResult<Empty>> AddPet([FromBody] PetRequestModel petModel)
16. => await RunWithServiceResult(async () =>
17. {
18. var userId = GetUserIdFromToken();
19. return await \_petService.AddPet(petModel, userId);
20. });
21. [HttpPut("{petId:guid}")]
22. public async Task<ServiceResult<Empty>> UpdatePet(Guid petId, [FromBody] PetUpdateRequestModel petModel)
23. => await RunWithServiceResult(async () => await \_petService.UpdatePet(petModel, petId));
24. [HttpGet("{petId:guid}")]
25. public async Task<ServiceResult<PetModel>> GetPet(Guid petId)
26. => await RunWithServiceResult(async () => await \_petService.GetPet(x => x.Id == petId));
27. [HttpDelete("{petId:guid}")]
28. public async Task<ServiceResult<Empty>> RemovePet(Guid petId)
29. => await RunWithServiceResult(async () => await \_petService.RemovePet(petId));
30. }
31. }

**ДОДАТОК Б**

PetService.cs

1. using AutoMapper;
2. using PetHelper.Api.Models.RequestModels.Pets;
3. using PetHelper.Business.Extensions;
4. using PetHelper.Business.User;
5. using PetHelper.DataAccess.Repo;
6. using PetHelper.Domain.Exceptions;
7. using PetHelper.Domain.Pets;
8. using PetHelper.Domain.Properties;
9. using PetHelper.ServiceResulting;
10. using System.Linq.Expressions;
11. namespace PetHelper.Business.Pet
12. {
13. public class PetService : DataAccessableService<PetModel>, IPetService
14. {
15. private readonly IMapper \_mapper;
16. private readonly IUserService \_userService;
17. public PetService(IUserService userService, IMapper mapper, IRepository<PetModel> repo) : base(repo)
18. {
19. \_mapper = mapper;
20. \_userService = userService;
21. }
22. public async Task<ServiceResult<Empty>> AddPet(PetRequestModel petRequestModel, Guid userId)
23. {
24. var serviceResult = new ServiceResult<Empty>();
25. if(petRequestModel == null || userId == Guid.Empty)
26. {
27. return serviceResult.FailAndThrow(Resources.InvalidDataFound);
28. }
29. var petOwnerModel = await \_userService.GetUser(x => x.Id == userId);
30. //TODO добавить Policy и удалить
31. if(!petOwnerModel.Value.IsEmailConfirmed)
32. {
33. return serviceResult.FailAndThrow(Resources.EmailConfirmationIsNeeded);
34. }
35. var petDomainModel = \_mapper.Map<PetModel>(petRequestModel);
36. petDomainModel.OwnerId = userId;
37. var result = await \_repository.Insert(petDomainModel);
38. return result.CatchAny();
39. }
40. public async Task<ServiceResult<PetModel>> GetPet(Expression<Func<PetModel, bool>> predicate)
41. {
42. var result = await \_repository.FirstOrDefault(predicate);
43. return result.Catch<EntityNotFoundException>(Resources.TheItemDoesntExist)
44. .CatchAny();
45. }
46. public async Task<ServiceResult<Empty>> UpdatePet(PetUpdateRequestModel petRequestModel, Guid petId)
47. {
48. var petModel = await GetPet(x => x.Id == petId);
49. petModel.Value = \_mapper.MapOnlyUpdatedProperties(petRequestModel, petModel.Value);
50. var result = await \_repository.Update(petModel.Value);
51. return result.CatchAny();
52. }
53. public async Task<ServiceResult<Empty>> RemovePet(Guid petId)
54. {
55. var serviceResult = new ServiceResult<Empty>();
56. if (await PetExists(petId))
57. {
58. var userModel = await GetPet(x => x.Id == petId);
59. var removeResult = await \_repository.Remove(userModel.Value);
60. return removeResult.CatchAny();
61. }
62. return serviceResult.FailAndThrow(Resources.TheItemDoesntExist);
63. }
64. private async Task<bool> PetExists(Guid petId)
65. {
66. var result = await \_repository.Any(x => x.Id == petId);
67. return result.CatchAny().Value;
68. }
69. }
70. }