Міністерство науки та освіти України
Харківський національний університет радіоелектроніки

Звіт до лабораторної роботи №2
з дисципліни «Аналіз та рефакторінг коду програмного забезпечення»

Виконав:
ст. гр. ПЗПІ-19-3
Логвінов О. В.

Перевірив:
ст. викл. каф. ПІ
Сокорчук І.П

Харків 2021

Мета: розробити серверну частину програмної системи "FireSaver", яка була описана в межах документу Vision&Scope.

Посилання на файл з кодом на гугл-диску:

https://drive.google.com/file/d/1YTILuMC_46eVfEycvwRcUzzfZbullTSv/view?usp=sharing

MD5 hash of ./ pzpi-19-3-lohvinov-oleksandr-lab2.zip:
AA433FD5041DD66135F05DAAE65E515F

Хід роботи:

Серверна частина реалізована за допомогою технології .Net Core 5. У якості бази даних використовувалася MySQL Server. Для взаємодії бізнес логіки та бази даних була застосована ORM технологія Entity Framework Core. Для впровадження роботи з web sockets, використовувалася технологія SignalR. Усі ці частини допомагають організувати взаємодію з такими зовнішніми клієнтами як веб клієнт, мобільний клієнт та IoT пристрій.

Серверна частина представляє собою трьохрівневу архітектуру, де в якості Presentation Layer виступають endpoints, Business Layer – сервіси, шо містять логіку, та Data Access Layer – рівень доступу до бази даних. Для реалізації автентифікації використовується JWT-токен – стандарт RFC-7519, що визначає один із способів безпечної передачі даних у вигляді JSON об'єктів. Для створення JWT-токену застосовується алгоритм HmacSha256 на основі секрету. Цей токен містить інформація про роль користувача та його ідентифікаційний номер у системі. Для зберігання інформації, як паролі користувачей, використовується алгоритм одностороннього хешування даних Sha256.

Основною задачею системи є допомога з евакуації людям у будівлі. Механізм евакуації працює наступним чином (див. рис. 1): мобільний клієнт

передає інформацію про приміщення, в якому знаходиться користувач, та його GPS координати. У разі відсутності інформації про приміщення, користувачу потрібно її внести за допомогою сканування выдповыдного QR-коду. Після чого клієнт отримує данні та відображає їх користувачу. Реалізація алгоритму представлена у додатку Д.

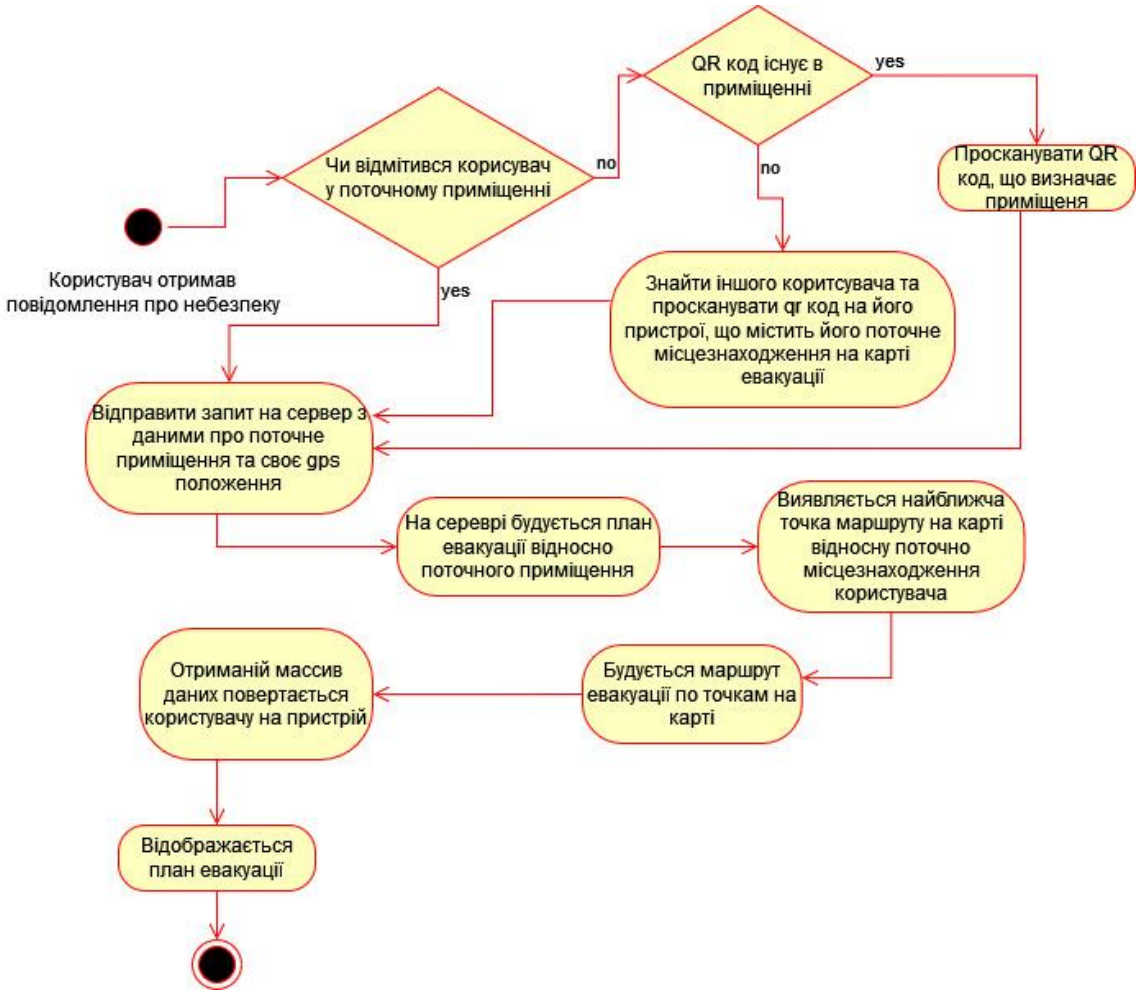

Рисунок 1 – діаграма діяльності алгоритму евакуації з приміщення

Також система повинна контролювати входження людей до приміщення (див. рис. 2) шляхом перевірки їх знань щодо правил безпеки у відповідному приміщенні. Для цього користувачу спочатку потрібно відсканувати QR-якій містить інформації про приміщення. Після чого користувачу надається правила

безпеки для цього приміщення. Якщо для цього приміщення встановлений тест, користувачу потрібно його пройти щонайменше на 80% правильних відповідей, після чого він допускається до приміщення. Код реалізації представлений у додатку Г.

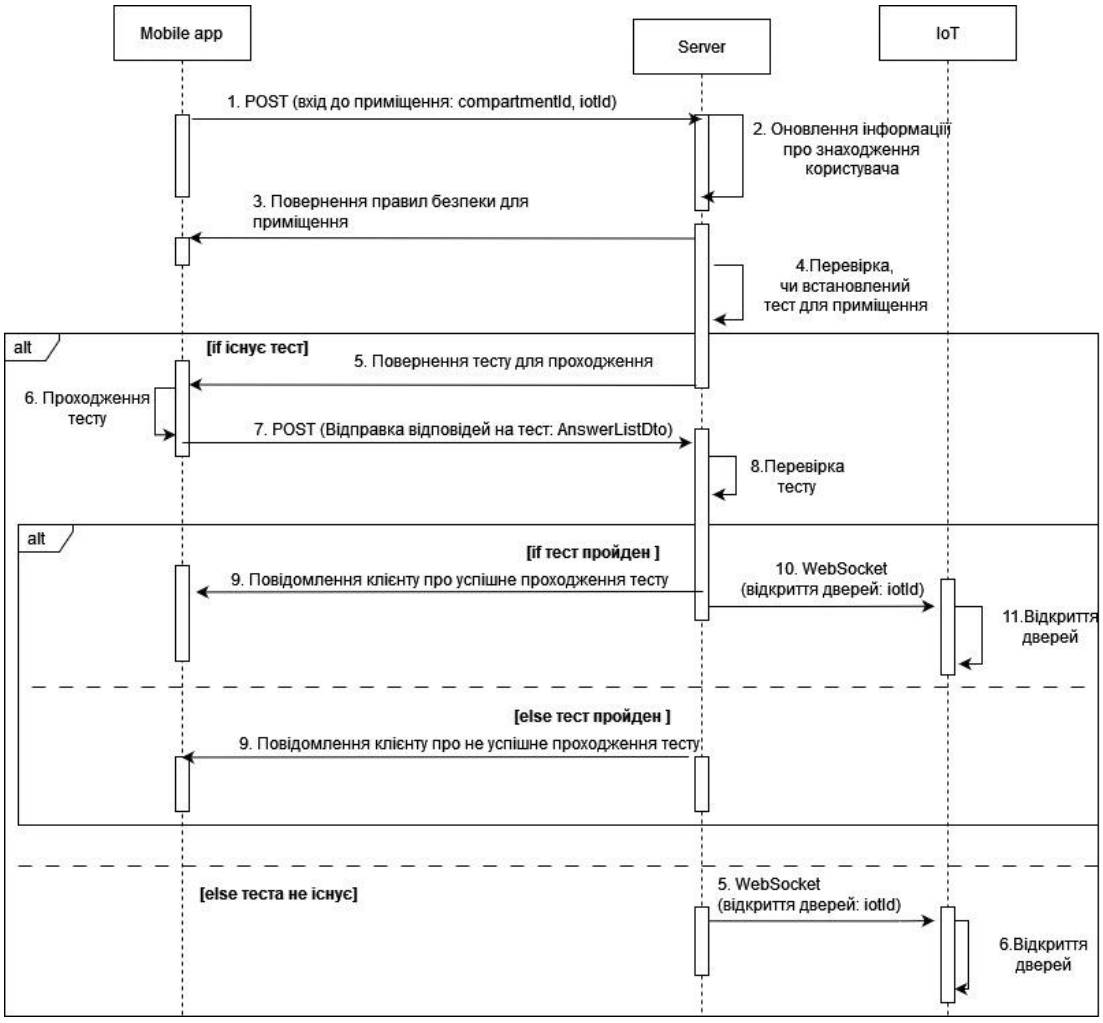

Рисунок 2 – діаграма послідовності допуску людини до приміщення

Серверна частина складається з 10 контролерів, кожний з яких містить Endpoints. Усього налічується 35 API функцій (див. табл. 1), кожна з яких відповідає збереження, видалення, зміну та відображення інформації з таблиць бази даних, виконання алгоритмів бізнес-логіки. Кожний endpoint має свою політику доступу.

На сервері існує три політики:

1) Admin – може виконувати будь-яку операцію, також відповідає за додавання нових пристроїв до бази даних та сворення резервних копій бази даних.
2) AuthorizedUser – може бути назначеним відповідною особою для будівлі. У цьому разі йому потрібно завантажувати та налаштовувати плани евакуації для будівлі. Також, може добавляти інформацію про себе для того, щоб у разі екстреної ситуації з ним можна було зв'язатися.
3) Guest – не може вносити інформацію про себе та назначатися відповідальною особою. Тимчасова інформація знищується через 24 години після входу до системи.

Таблиця 1 - Ендпоїнти

Посилання на endpoint	Метод	Політика доступу	Параметри
/user/newuser	POST	Guest	name: string, surname: string, patronymic: string mail: string, password: string, telNumber: string dob: DateTime
/user/auth	POST	Guest	mail: string, password: string
/user/getGuestToken	GET	Guest	

Продовження таблиці 1

/user/:id	GET	AuthorizedUser, Admin	
/user/updateInfo	PUT	AuthorizedUser, Admin	userId: int, name: string, surname: string, patronymic: string telephoneNumber: string dob: DateTime
/user/setWorldPostion	POST	AuthorizedUser, Admin	userId: int, Longtitude: double, Latitude: double
/user/enterCompartment	POST	AuthorizedUser, Admin, Guest	userId: int, compartmentId: int, iotId: int?
/user/evacuate	GET	AuthorizedUser, Admin, Guest	
/building/new	POST	Admin	address: string. info: string
/building/adduser/:userId /:buildingId	GET	AuthorizedUser, Admin	userId: int, buildingId: int
/building/removeuser/:userId	DELETE	AuthorizedUser, Admin	userId: int

Продовження таблиці 1

/building/updateBuilding /:buidlingId	PUT	AuthorizedUser, Admin	buidlingId: int, address: string. info: string
/floor/addFloorToBuilding/ :buildingId	POST	AuthorizedUser, Admin	buildingId: int, Level: int, Name: string, Description: string SafetyRules:string
/floor/changeFloorInfo/ :floorId	PUT	AuthorizedUser, Admin	floorId: int, Level: int, Name: string, Description: string SafetyRules:string
/floor/:floorId	GET	AuthorizedUser, Admin, Guest	floorId: int
/floor/:floorId	DELETE	AuthorizedUser, Admin	floorId: int
/room/addRoomToFloor /:floorId	POST	AuthorizedUser, Admin	florId: int, Name: string, Description: string SafetyRules:string

Продовження таблиці 1

/room/changeRoomInfo/:roomId	PUT	AuthorizedUser, Admin	roomId: int, Name: string, Description: string SafetyRules:string
/room/:roomId	DELETE	AuthorizedUser, Admin	roomId: int
/room/:roomId	GET	AuthorizedUser, Admin	roomId: int
/evacuationPlan/ :compartmentId/newEvacPlan	POST	AuthorizedUser, Admin	compartmentId: int, evacPlanImage: IFormFile
/evacuationPlan/ :evacId/updateEvacPlan	PUT	AuthorizedUser, Admin	evacId: int, evacPlanImage: IFormFile
/ evacuationPlan /:compartmentId	GET	AuthorizedUser, Admin, Guest	compartmentId: int
/ evacuationPlan /evacuationPlans	GET	AuthorizedUser, Admin, Guest	

Продовження таблиці 1

/scalePoints/newPos/ :compartmentId	POST	AuthorizedUser, Admin	compartmentId: int, MapPosition:{ Longtitude: double, Latitude: double}, WorldPostion:{ Longtitude: double, Latitude: double}
/routeBuilder/:compartmentId /newroute	POST	AuthorizedUser, Admin	compartmentId: int, parntRoutePointId: int?, PointPosition: { Longtitude: double, Latitude: double }
/routeBuilder/newpoint	POST	AuthorizedUser, Admin	parntRoutePointId: int?, PointPosition: { Longtitude: double, Latitude: double }

Продовження таблиці 1

/routeBuilder/newpoint /:strtRoutePointId	GET	AuthorizedUser, Admin	strtRoutePointId:int
/test/addCompartmentTest /:compartmentId	POST	AuthorizedUser, Admin	tryCount: int, Questions: [{ content:string, answerList: string[], pssblAnswrs: string[] }]
/testanswerCompartmentTest	POST	AuthorizedUser, Admin, Guest	testId: int, Answers: { QuestionID, Anser:string}[]
/test/updateCompartmentTest /:testId	PUT	AuthorizedUser, Admin	tryCount: int, Questions: [{ content:string, answerList: string[], pssblAnswrs: string[]}]

Продовження таблиці 1

/iot/newIot	POST	Admin	IoTIdentifier: string
/iot/newIotToCompartment	POST	AuthorizedUser, Admin	IoTIdentifier: string, CompartmentId: int
/admin/backup	GET	Admin	
/admin/restore	GET	Admin	

Бізнес логіка керує базою даних використовуючи технологію EntityFrameworkCore, яка має зручний та ефективний спосіб роботи доступу до реляційної бази даних MySQL. Згідно з предметною областю, була розроблена ER-діаграма (див. додаток А). На цій діаграмі зображені усі сутності, що необхідні для роботи інформаційної системи а саме:

- User – таблиця, що описуэ користувача.
- Roles – таблиця, що описує існуючі ролі користувача
- UserRoles – суміжна таблиця що описує які ролі має певний користувач
- Building – таблиця, що містить дані про будівлю, яка використовує систему FireSaver.
- Compartment – таблиця, що описує приміщення у будівлі. Існує два види приміщення – кімната та поверх. Кімната характеризується значенням "Room" у стовпчику CompartmentType. Для поверха цей атрибут дорывнює "Floor". На одному поверсі може бути декілька кімнат.
- EvacuationPlan – таблиця, що описує зображення плана евакуації для приміщення.
- ScaleModel – таблиця, що містить дані для зображення позицій користувачей на плані евакуації.
- ScalePoint – таблиця, що описує точки, які потрібні для формування ScaleModel для плану евакуації

- RoutePoints – таблиця, що описує точки, які формують шлях евакуації з приміщення.
- IoTs – таблиця, що описує розумні пристрої, які використовує приміщення
- Test – таблиця що описує тести, які потрібно пройти перед входженням у приміщення.
- Questions – запитання, які присутні у тестах та їх відповіді.

Для зображення того, що має система виконувати та як з нею взаємодіяти, була розроблена діаграма прецедентів (див. додаток Б). Згідно з діаграмою ми бачимо, що системою користуються адміністратор, зареєстрований та не зареєстрований користувач, а також розумний пристрій.

Для ролі розумного пристрою доступний такий функціонал:

- Відкриття дверей у разі успішного проходження тесту на знання правил безпеки для приміщення
- Повідомлення про небезпеку людей, що знаходяться у будівлі
- Серверна частина приймає тільки авторизовані запроси, тому перед початком роботи, пристрою потрібно авторизуватися
- Стеження за вмістом шкідливих речовин у приміщенні

Для ролі незареєстрованого користувача доступний такий функціонал як

- Реєстрація
- Отримання плану евакуації

Авторизований користувач може

- Бути назначеним відповідальною особою для приміщення
- Якщо користувач є відповідальною особою то
  1) додавати або змінювати інформацію про приміщення чи будівлю
  2) налаштовувати плани евакуації
  3) Викликати тривогу у разі необхідності
  4) Назначати іншого користувача відповідальною особою

Адміністратор у системі має наступні обов'язки

- Реєстрація нових розумних пристроїв
- Керування базою даних
- Перегляд інформації про будь-яку будівлю чи приміщення у системі

Також було розроблено діаграму розгортання, що зображує те, як система буде виглядати на апаратному середовище (див. додаток В).

Висновки

Під час лабораторної роботи була створена серверна частина інформаційної системи FireSaver, отримані навички з побудови Use-Case діаграм, діаграм розгортання, ER-діаграм, діаграм послідовностей та діяльностей.

ДОДАТОК А

ER-діаграма

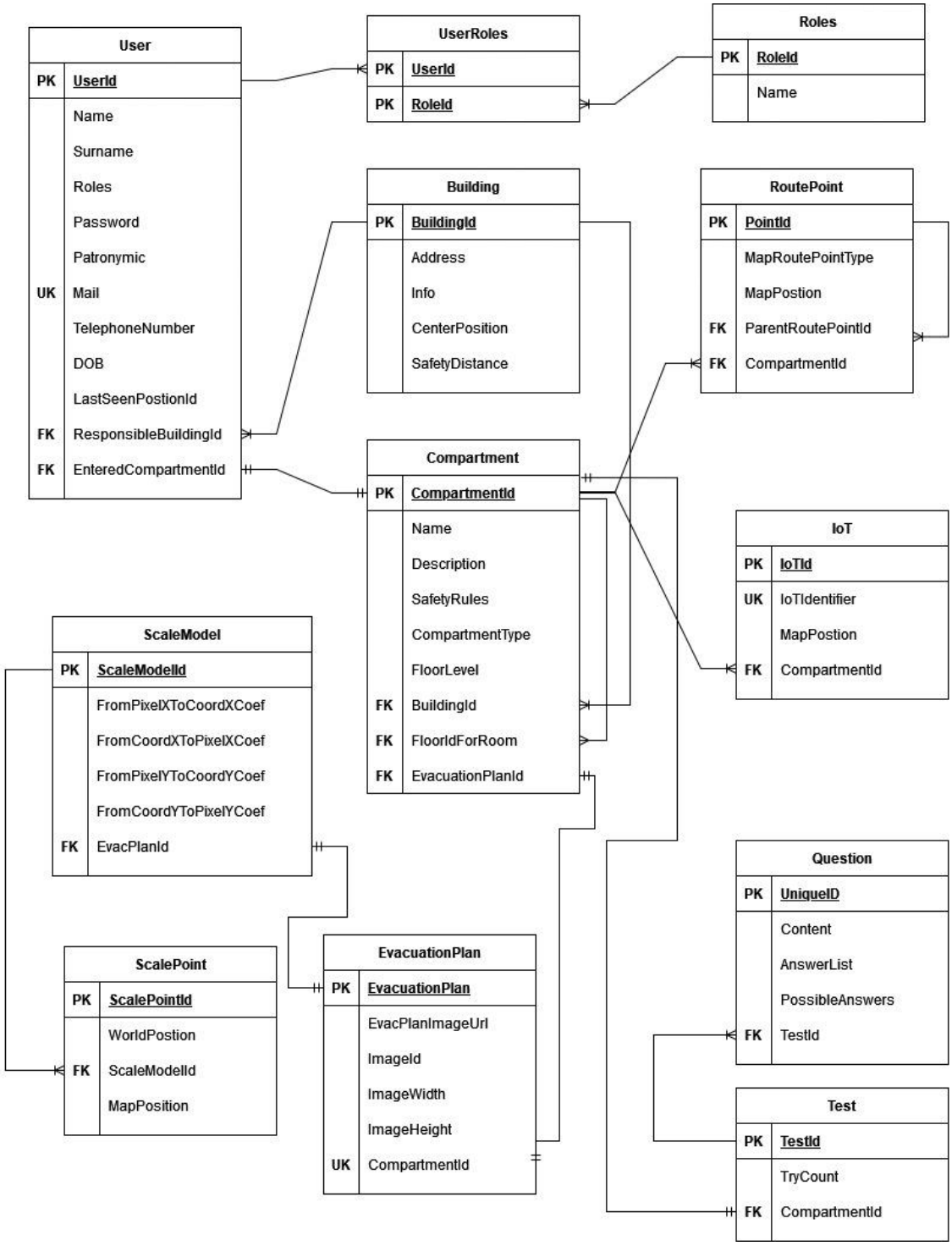

Рисунок А.1 – ER-діаграма FireSaver

ДОДАТОК Б

Use-Case діаграма

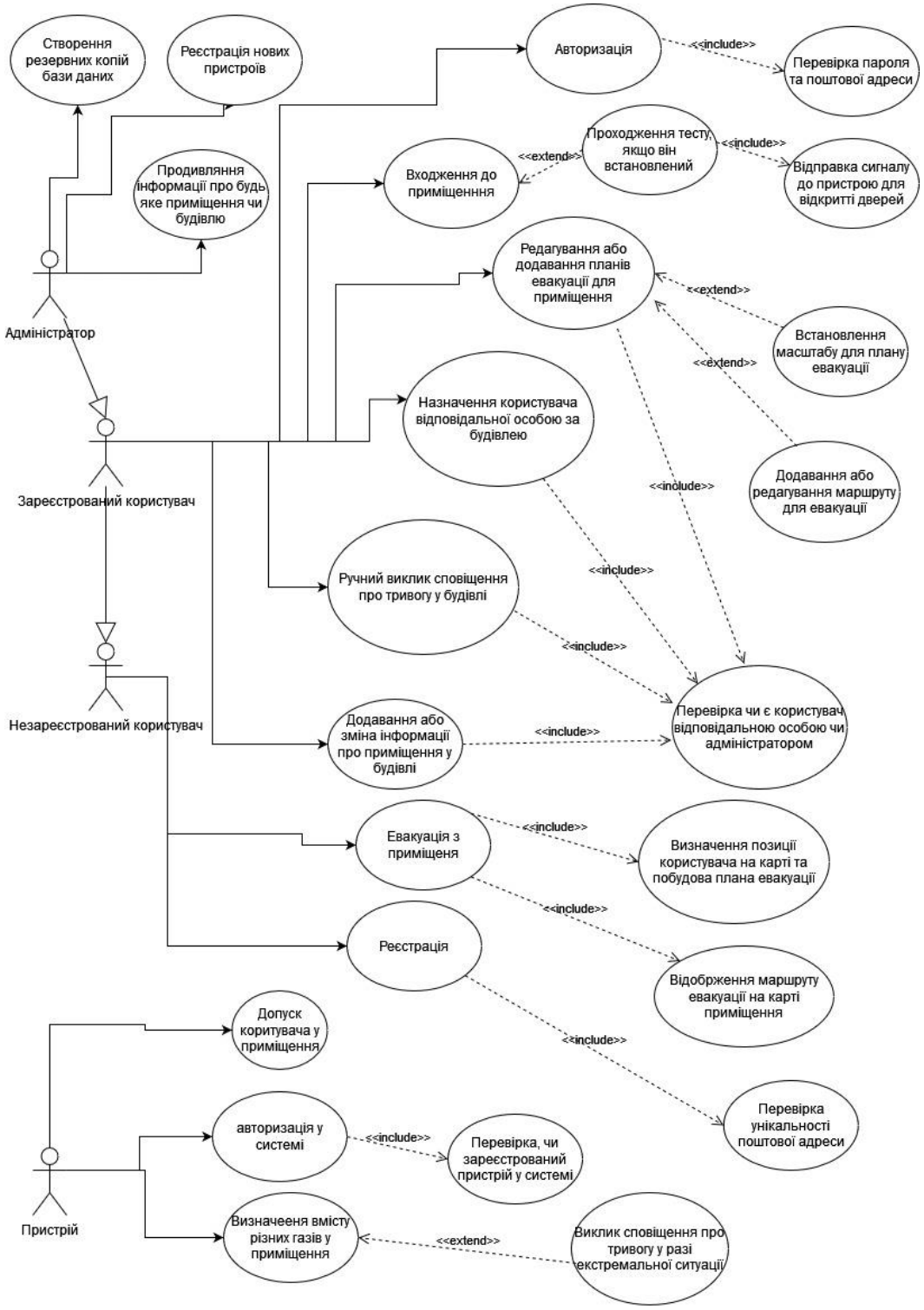

Рисунок Б.1 - Use-Case діаграма

ДОДАТОК В

Діаграма розгортання

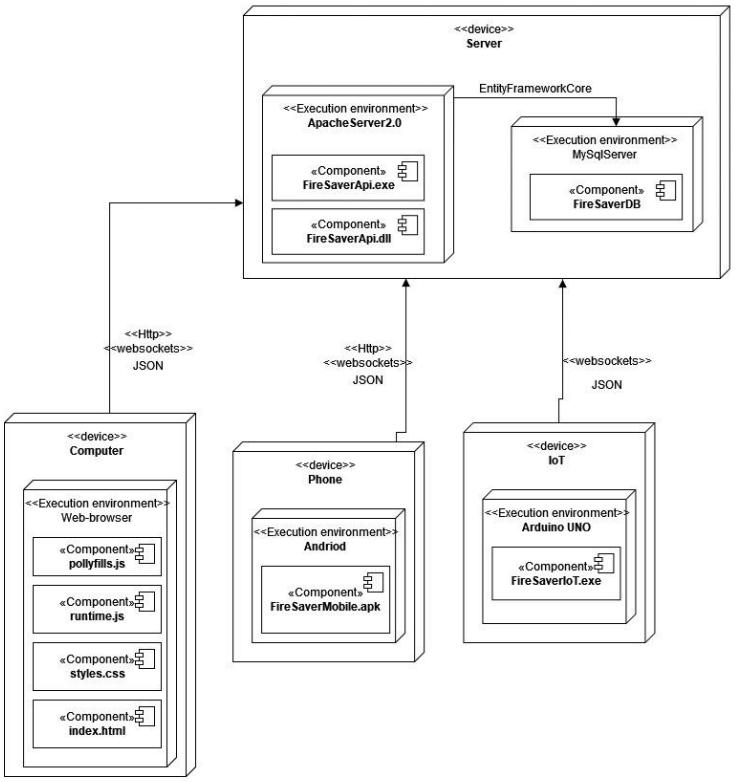

Рисунок В.1 – Діаграма розгортання інформаційної системи FireSaver

ДОДАТОК Г

Реалізація допущення користувача до приміщення

```
public async Task<TestOutputDto> EnterCompartmentById(int
userId, int compartmentId, int? iotId)
{
    var compartment = await context.Compartment
                      .Include(t => t.CompartmentTest)
                      .ThenInclude(q => q.Questions)
                      .FirstOrDefaultAsync(c => c.Id ==
compartmentId);


    if (compartment.CompartmentTest != null)
    {
        var testToComplete = await
testService.GetTestInfo(compartment.CompartmentTest.Id);

        var testOutput =
mapper.Map<TestOutputDto>(testToComplete);
        return testOutput;
    }
    else
    {

        var user = await GetUserById(userId);
        if (iotId != null)
        {
        await socketService.OpenDoorWithIot(iotId.Value);
        }

        user.CurrentCompartment = compartment;
        context.Update(user);
        await context.SaveChangesAsync();

        return null;
    }
}
```

Для перевірки відповідей на тест використовується функція CheckTestAnswers.

```
public async Task<bool> CheckTestAnwears(AnswerListDto
answears)
{
    int userId = userContextService.GetUserContext().Id;
    CheckIfUserIsNotBanned(userId, answears.TestId);

    var test = await dataContext.Tests.Include(t =>
t.Questions).FirstOrDefaultAsync(t => t.Id ==
answears.TestId);
    if (test == null)
    {
        throw new System.Exception("Test is not found");
    }

    if (answears.Answears.Count != test.Questions.Count)
    {
       throw new System.Exception("Take test again
please");
    }
    int wrongCount = 0;
    for (int i = 0; i < answears.Answears.Count; i++)
    {
         var trueAnswer = test.Questions.Where(q => q.Id
== answears.Answears[i].QuestionId).FirstOrDefault();
         if (trueAnswer == null)
         {
             throw new System.Exception("Take test again
please");
         }

         var realAnswers =
trueAnswer.AnswearsList.ToLower().Split(',');
         var inputAnswers =
answears.Answears[i].Answear.ToLower().Split(',');
         Array.Sort(realAnswers);
         Array.Sort(inputAnswers);

```

```
            if (!realAnswers.SequenceEqual(inputAnswers))
            {
                wrongCount++;
            }
        }

        double passThreshold = 80;
        double currentThreshold =
    ((double)(answears.Answears.Count - wrongCount) /
    (double)answears.Answears.Count) * 100d;
        if (currentThreshold < passThreshold)
        {
        timerService.IncreaseFailedUserTestFaledCount(userId,
    answears.TestId, test.TryCount);

        return false;
        }

        timerService.ClearUSerFailedTest(userId,
    answears.TestId);
        return true;
    }
```

ДОДАТОК Д

Реалізація алгоритму побудови шляху евакуації для користувача

```
public async Task<List<RoutePointDto>>
BuildEvacuationRootForCompartment(int userId)
{
    var user = await GetUserById(userId);

    if (user.CurrentCompartment == null)
    {
        throw new Exception("Current compartment for user
is not set");
    }
    var compartmentPoints = await context.RoutePoints
                                         .Where(p =>
p.Compartment.Id == user.CurrentCompartment.Id)
                                         .ToListAsync();
    if (compartmentPoints.Count == 0)
    {
        throw new Exception("No points for the
compartment found");
    }

    var worldPosition =
mapper.Map<PositionDto>(user.LastSeenBuildingPosition);

    var mappedWorldPostion = await
locationService.WorldToImgPostion(worldPosition,
user.CurrentCompartment.Id);

    RoutePoint closestPoint =
GetClosestPoint(compartmentPoints, mappedWorldPostion);

    var rootPointFotCurrentRoutePoint = await
routebuilderService.GetRootPointForRoutePoint(closestPoin
t.Id);
    var exitPoints = compartmentPoints.Where(p =>
p.RoutePointType == RoutePointType.EXIT ||
       p.RoutePointType ==
RoutePointType.ADDITIONAL_EXIT).ToList();

```

```
    if (exitPoints.Count == 0)
    {
        return new List<RoutePointDto>()
        {
             mapper.Map<RoutePointDto>(await
routebuilderService.GetAllRoute(rootPointFotCurrentRouteP
oint.Id))
        };
     }
     else
     {
        var possibleEvacuationList = new
List<RoutePointDto>();
        for (int i = 0; i < exitPoints.Count; i++)
        {
           var evacuationRoute = await
routebuilderService.GetRouteBetweenPoints(closestPoint.Id
, exitPoints[i].Id);

           possibleEvacuationList
      .Add(mapper.Map<RoutePointDto>(evacuationRoute));

        }

        return possibleEvacuationList
}

```