Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук

Кафедра програмної інженерії

КУРСОВА РОБОТА
ПОЯСНЮВАЛЬНА ЗАПИСКА
з навчальної дисципліни «Аналіз та рефакторинг коду програмного забезпечення»
Тема роботи: Програмна система обліку та доставки продуктів у корпоративних структурах

Студентгр.ПЗПІ-16-3 _____ Фастов М. М.
(підпис)
Керівник роботи _____ ст. викл. Сокорочук І. П.
(підпис)

Роботу захищено «__»_____2019 р.
з оцінкою _____

Комісія:

_____ доц. Лещинський В.О
(підпис)
_____ доц. Лещинська І.О.
(підпис)
_____ ст. викл. Сокорочук І. П.
(підпис)

Харків
2019 р.

Факультет *комп'ютерних наук* Кафедра *програмної інженерії*
Напрям підготовки *6.050103 - Програмна інженерія*
Курс *3* Семестр *5*
Навчальна дисципліна *Аналіз та рефакторинг коду програмного забезпечення*

ЗАВДАННЯ
НА КУРСОВУ РОБОТУ СТУДЕНТОВІ

Фастову Михайлу Максимовичу

1. Тема роботи: *«Програмна система обліку та доставку продуктів у корпоративних структурах»*

2. Термін узгодження завдання курсової роботи «__»_____ 2018 р.

3. Термін здачі студентом закінченої роботи «__»_____ 2019 р.

4. Вихідні дані до проекту (роботи): *В програмній системі передбачити: авторизацію, реєстрацію, додавання нової компанії, видалення компанії, задання інформації про компанію, додавання нового вендора, видалення вендора, задання інформації про вендора, створення накладних та додавання в них продуктів, модифікування їх статусу. Використовувати: ОС Windows 10, СКБД Microsoft SQL Server 2016, середовище розробки Microsoft VisualStudio 2017, платформу ASP.NET Web API 2, середовище розробки VisualStudioCode, середовище розробки AndroidStudio.*

5. Зміст пояснювальної записки: *вступ, аналіз предметної області, постановка задачі, проектування програмного проекту, структура бази даних, кодування програмного проекту, опис розробленої програмної системи, висновки, перелік посилань, додатки.*

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): *архітектура проекту, схема бази даних, діаграма варіантів використання, діаграма розгортання, діаграма діяльності, діаграма послідовності, діаграма класів, копії екранів («скріншоти») Веб і мобільного додатку.*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів курсової роботи	Термін виконання етапів роботи	Примітка
1	Функціональна специфікація програмного проекту		
2	Проектування програмного проекту		
3	Кодування програмного проекту		
4	Оформлення пояснювальної записки		
5	Захист курсової роботи		

Дата видачі завдання«__»_____2018 р.

Керівник _____ ст. викл. Сокорочук І. П.
(підпис)

Завдання прийняв до виконання
ст.гр.ПЗПІ-16-3_____ Фастов М. М.
(підпис)

РЕФЕРАТ

Пояснювальна записка до курсової роботи: 48 с., 8 рис., 1 табл., 6 додатків, 6 джерел.

COMPANY, DELIVERY, VENDOR, INVOICE, PRODUCT, PAYMENT, WEB API.

Об’єктом розробки є програмна система доставки та обліку продуктів у корпоративних структурах.

Метою курсової роботи є розробка програмної системи, що буде надавати можливість взаємодіяти постачальнику та компанії-покупцю, утверджувати доставку та ін.

Методи розробки базуються на платформі ASP.NET Web API 2 з використанням мови програмування C# для серверної частини, на HTML5, SCSS, AngularJS для клієнтського веб-додатку, у середовищі розробки VisualStudio з використанням мови програмування C# для смарт-девайсу, на Microsoft SQL Server 2016 для бази даних та Android SDK з використанням мови програмування Java для створення мобільного додатку.

У результаті курсової роботи здійснено реалізацію програмної системи, що дозволяє: авторизацію, реєстрацію, додавання нової компанії, видалення компанії, завдання інформації про компанію, додавання нового вендора, видалення вендора, завдання інформації про вендора, створення накладних та додавання в них продуктів, модифікування їх статусу. Програмна система складається з веб додатка, сервера, мобільного додатка та смартдевайса, що являє собою ArduinoNano.

ЗМІСТ

ВСТУП

Сьогодні у багатьох компаній, які мають велику кількість працівників, існує потреба у системі, яка могла б надати послуги у сфері доставки та обліку продуктів харчування. Точніше, необхідна система через яку компанія могла б заказувати певні продукти або готові страви у вендорів, а також вести їх облік. Система є корисною, тому що нагладжування усіх процесів, автоматизацію яких пропонує система, яке складним та не вигідним для самої компанії, так як це потребує багато фінансових на інтелектуальних ресурсів. Отже, було вирішено створити систему для спрощення описаних вище операцій.

Мета проекту - створення системи для взаємодії компанії та вендора, доставки їжі та обліку продуктів. Створення автоматизації, яка є виграшною для обох сторін.

Критерій успіху - вирішення проблеми наявності продуктів у компанії, а також повна автоматизація цієї галузі: налагоджування зв’язку з вендорами, доставка, облік та управління усім цим з особистого кабінету.

Основною бізнес-метою є реалізація робочого сервісу, який «клієнту» та «покупцю» он-лайн.

Сервіс повинен мати багаторівневу архітектуру, бути багатошаровим і масштабуватися, на клієнтському рівні мати веб-клієнт і мобільний додаток, а також розумний пристрій.

У якості основних інструментів розробки були використані середовища розробки VisualStudio 2017 та Arduino IDE. Для розробки серверної частини був використана технологія ASP.NET, для клієнтської - Angular.JS. Для збереження даних була використана СКБД MS SQL SERVER. Для реалізації IoT було використано ArduinoNano.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

У наші дні виробництво не може бути конкурентоспроможними без наявності програмного забезпечення, що полегшує його роботу. Дана проблема стосується будь-якої сфери діяльності людини, в тому числі і успішного функціонування системи доставки, що я вибрав в якості предметної області даного курсового проекту.

Компанія має на увазі під собою комерційну фірму, що володіє деякою кількістю філій, які мають потребу у системі, яка б охоплювала усю сферу споживання всередині компанії.

Персоналу конкретної компанії необхідно мати інформацію про те які продукти є на складі, які прибудуть з наступною поставкою, дату цієї поставки і т. д. Співробітникам з керівними посадами належить бути в курсі обсягу продажів, а також представляти загальну картину функціонування сфери споживання, для організації поставок певної кількості продуктів в певну філію.

На даний час не існує такого програмного забезпечення, яке б могло задовольнити усі потреби сучасних компаній та постачальників у сфері споживання. Існуючі програмні продукти є досить не ефективними та незручними у використанні. У деяких компаніях взагалі не має будь якої інформаційної системи, яка б полегшувала роботу.

Таким чином, проаналізувавши потреби користувачів і існуючі проблеми предметної області, надані вище, можна уявити потенційних користувачів системи і те, яким чином ця система може допомогти і спростити роботу людині. Потенційними користувачами системи є персонал конкретної компанії.

Щоб спростити роботу користувача і надати дійсно потрібні йому функції, система повинна надавати зручне відображення даних про компанії, вендорів, накладні, продукти, а також модифікацію власних підприємств і т. д.

Очевидно, що для зберігання десятків філій, а тим більше сотень найменувань продуктів, для системи необхідна наявність бази даних, яка дозволила б зберігати велику кількість інформації і надавати зручний і

швидкий доступ до них. Також база необхідна для зберігання даних користувачів, що працюють з системою.

Сьогодні не існує єдиної системи яка б охоплювали усі необхідні для компанії та вендора функції у сфері харчування. Тому найкращим вирішенням цієї проблеми є створення автоматизованої системи для діяльності у даній сфері. Така система дала б можливість надійно зберігати інформацію і надавати швидкий доступ до цієї інформації.

2 ПОСТАНОВКА ЗАДАЧІ

Мета курсового проекту - вирішення проблеми автоматизації та взаємодії компанії із постачальником продуктів. Основними компонентами рішення цієї проблеми є діючий сайт та серверна частина, де усі необхідні функції будуть реалізовані.

Рішення включає розробку сервера та програмного забезпечення для клієнтів, а саме вебсайт, мобайл та розумний пристрій.

По-перше, необхідно створити серверне програмне забезпечення, яке зберігає дані для всіх користувачів і здатне керувати ним, повинно бути розроблено, що контролювати усю систему.

По-друге, клієнтське програмне забезпечення повинно бути розроблено для синхронізації з сервером і виконувати роль клієнта.

2.1 Основний функціонал системи

Основний функціонал повинен включати в себе:

а) можливість створення компаній, вендорів;

б) можливість створення накладних, додавання в них продуктів;

в) управління статусом накладної де приймають участь вендор і компанія;

г) реєстрацію користувачів;

д) можливість управління світлом смарт-пристрою;

е) перегляд історії поставок;

Нижче наведено більш детальний опис функціоналу системи конкретно по кожній частині системи де його буде реалізовано:

а) серверна частина матиме наступний функціонал:

1) реєстрування та керування користувачами;
2) керування налаштуваннями існуючих пристроїв;
3) захист даних (через JWT токен);
4) реєстрування компаній, вендорів та накладних;
5) управління статусом накладної.

б) web клієнт матиме наступний функціонал:

1) відображення даних, отриманих від клієнтів;
2) реєстрація та вхід користувача;
3) підтримка локалізації (UA, EN);
4) захист даних (через JWT токен);
5) додавання та керування компаніями, вендорами та накладними;
6) перегляд історії.

в) Android клієнт матиме наступний функціонал:

1) вхід користувача;
2) додавання та керування компаніями, вендорами та накладними;
3) захист даних (через JWT токен).
4) підтримкалокалізації(UA, EN);

г) IoT пристрій матиме наступний функціонал:

1) реалізація API для прийому команд клієнта;
2) зміна яскравості світла;
3) увімкнути / вимкнути світло.

2.2 Допущення та залежності

Для коректної роботи усіх частин проекту наведений список допущень:

а) користувачі системи мають пристрій з доступом до Інтернету;

б) користувачі зацікавлені в основних функціях системи.

в) на одну компанію виділяється тільки один обліковий запис

Також існують наступні залежності:

а) додаток буде орієнтуватися на версії Android 4.4 і вище;

б) додаток буде орієнтуватися на новітні версії браузерів (IE 9+, Chrome 4.5+, Safari 4.0+);

в) телефон має надійне підключення до WiFi мережі або мати мобільний інтернет.

2.3 Користувацькі обмеження

Для коректного використання даного продукту та викреслення нерозуміння з боку користувачів, є деякі користувацькі обмеження:

а) користувач повинен пам’ятати логін та пароль;

б) користувач повинен авторизуватися для користування системою.

2.4 Бізнес - потреби та пріоритетність

Даний продукт може залучити користувачів своєю унікальністю та актуальною функціональністю (таблиця 2.1).

Зацікавлена особа	Інтереси	Обмеження
Користувач	Автоматизована система доставки	Бюджет
Розробник	Своєчасне виконання роботи та введення системи в експлуатацію	Терміни реалізації
Власник продукту	Своєчасне отримання готового продукту з документацією.	Бюджет

Таблиця 2.1 – Особи та їх інтереси

Час і бюджет - головні обмеження розробника, таким чином потрібно відокремити пріоритетні показники для того, щоб акцентувати свою увагу на них в процесі розробки і бути більш стійким до стресових ситуацій, пов'язаних з нестачею ресурсів.

Бюджет є також обмеження для власника продукту, адже існує деяку сума грошових коштів, які він готовий вкласти у проект.

2.6 Середа оточення

Серверна частина буде створена за допомогою ASP.NET Core 2.0, а саме ASP.NET CoreWebApi. Що працює за допомогою моделей та контролерів та нагадує паттерн MVC.

Як місце для розміщення серверної частини системи та веб-клієнту буде використовуватися продукт компанії Microsoft – веб сервер InternetInformation Server 7.0.IIS

Фронтенд буде зроблено з використанням фреймворку AngularJS.

Мобільний додаток буде реалізовано для операційної системи Android.

Розумний пристрій буде реалізовано на основі ArduinoNano.

3 АРХІТЕКТУРА ПРОЕКТУ

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

а) набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;

б) набір клієнтів, які використовують сервіси, що надаються серверами;

в) мережа, яка забезпечує взаємодію між клієнтами та серверами.

На рисунку 3.1 зображена діаграма розгортання проекту, що відображає архітектуру проекту:

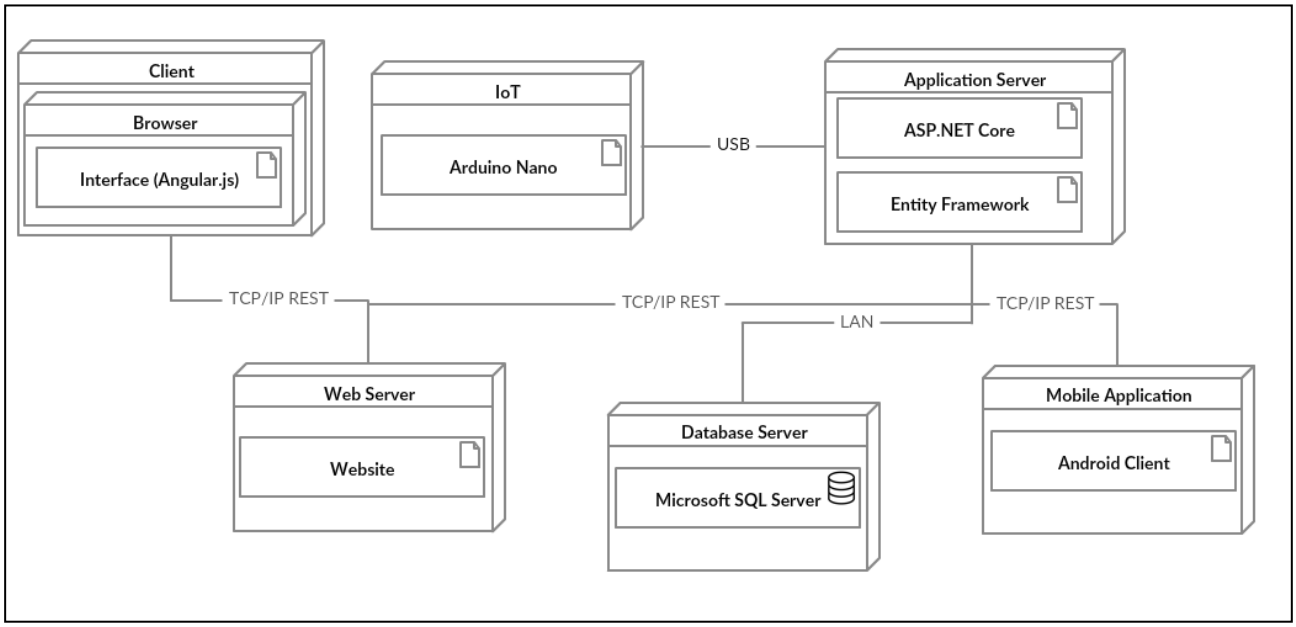

Рисунок 3.1 – Діаграма розгортання проекту

На діаграмі зображено взаємодію між різними компонентами системи. Так, для отримання результату з веб-браузеру користувача, front-end посилає

запит на back-end сервер. Back-end сервер в свою чергу валідує дані та звертається до бази даних, далі в зворотньому напрямку компоненти передають результат до користувача.

Усі компоненти системи, окрім IoT пристрою, було реалізовано за допомогою мови програмування C#, що дозволило повторно використовувати частину коду між різними компонентами та забезпечило кращою їх інтеграцію між собою та зменшило потенційну кількість помилок.У даній системі сервер використовується для синхронізації даних та часткової їх централізації. Він складається з MSSQL SERVERбази даних та ASP.NET CoreWeb API.

Веб-клієнт слугує для реєстрації нових користувачів, редагування свого профілю та користування функціями системи. Його було реалізовано за допомогою AngularJS. Усі данні для відображення він берез API, що реалізовано на сервері.

Розумний пристрій представляє собою ArduinoNano з світодіодом яка грає роль розумної лампи. Його логіку було реалізовано за допомогою мови C++ та Arduino IDE. Цей розумний пристрій є лише прикладом для демонстрації можливостей системи. Вона може підтримувати різні видирозумних пристроїв.

Мобільний додаток реалізовано за допомогою Andorid, який має тіж самі функції які є на веб-сайті.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

В даному проекті використовується багаторівнева архітектура проекту. Використовується класична трирівнева система, що складається з наступних рівнів:

а) API layer (рівень взаємодії з веб сервером): це той рівень, з яким взаємодіє клієнт, посилаючи запити на веб сервер.

б) Businesslayer (рівень бізнес-логіки): містить набір компонентів, які відповідають за обробку отриманих від рівня уявлень даних, реалізує всю необхідну логіку додатка, усі обчислення, взаємодіє з базою даних і передає рівнем подання результат обробки.

в) Data Access layer (рівень доступу до даних): зберігає моделі, що описують використовувані суті, також тут розміщуються специфічні класи для роботи з різними технологіями доступу до даних, наприклад, клас контексту даних BackendContext.

Також слід описати технологію RESTApi, завдяки якій працює застосування. REST стає загальним підходом для подання сервісів навколишнього світу. Причина його популярності полягає в його простоті, легкості використання, доступ через HTTP і інші. В Основі REST закладами принципи Функціонування Всесвітньої Павутина I, зокрема, можливості HTTP.

Дані повинні передаватися у виді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, що не повинен залежати від мережевого прошарку, що не винен зберігати інформацію про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабованість системи и дозволяє їй еволюціонувати з новими вимогами.

4 КОДУВАННЯ

4.1Back-endсервер

У якості мови розробки бекенду було обрано C#, технологія - ASP.NETCoreWebAPI, яка була використана у цьому проекті. Завдяки даному вибору розробка API була швидкою, а код вийшов компактним і читаним. У якості IDE було обраноVisualStudio 2017 найкращій продукт в своїй сфері [1].

Сервер являє собою набір контролерів, що використовуються функціоналом сайту, мобільного додатку та smart-девайсу. Сервер не має власного інтерфейсу, але він критично необхідний для коректного функціонування інших частин системи.

Усього було розроблено вісім основних контролерів (AccountsController, AuthController, CompaniesController, DashboardController, InvoiceItemsController, InvoicesController, ProductsController, VendorsController).

AccountsController та AuthController відповідають за контроль облікових записів користувачів, а саме: реєстрацію, логін, вихід із системи та ін.Доступ до даних було реалізовано за допомогою REST API (див. додаток А), яке доступне для зареєстрованих користувачів. Механізм авторизації було виконано за допомогою JWT (JSON WebToken) [2].

CompaniesController відповідає за роботу із сутністю «Компанія», реалізує запити типу CRUD, а також перевірку чи належить компанія користувачу, та вивід компаній користувача, приклад коду наведено нижче:

```
        [Route("getMyCompanies")]
        [HttpGet]
publicIEnumerable<CompanyViewModel>getMyCompanies()
        {
            List<CompanyViewModel>myCompanies = new
List<CompanyViewModel>();
var Companies =  _context.Companies.Select(c =>newCompanyViewModel
            {
                ID = c.ID,
                Name = c.Name,
                Description = c.Description,
CustomerID=c.CustomerID
            })
```

```
                .ToList();

foreach (var company in Companies)
if (isOwner(company.ID,_caller,_context))
myCompanies.Add(company);
//myCompanies.Add
returnmyCompanies;
        }
```

VendorsController, ProductsControllerвиконує приблизно тіж самі функції тільки для вендорів та продуктів, а не компаній.

DashboardController відповідає за особистий кабінет користувача та вивід інформації про нього, код можна подивитися у додатку А.

InvoicesController та InvoiceItemsController відповідають за роботу накладних. Реалізують запити типу CRUD, та повернення даних про накладні для певного вендора чи компанію:

```
[Route("GetInvoicesVendor")]
        [HttpGet]
publicIEnumerable<InvoiceViewModel>GetInvoicesVendor()
        {
            List<VendorViewModel>myVendors = new
List<VendorViewModel>();
            List<InvoiceViewModel>myInvoices = new
List<InvoiceViewModel>();

var Vendors = _context.Vendors.Select(c =>newVendorViewModel
            {
                ID = c.ID,
                Name = c.Name,
                Description = c.Description,
CustomerID = c.CustomerID
            })
               .ToList();

foreach (var vendor in Vendors)
if (VendorsController.isOwner(vendor.ID, _caller, _context))
myVendors.Add(vendor);

varallInvoices = _context.Invoices.Select(c =>newInvoiceViewModel
            {
                ID = c.ID,
VendorID = c.VendorID,
CompanyID = c.CompanyID,
                Date = c.Date,
isAccepted = c.isAccepted,
isDelivered = c.isDelivered
```

```
            })
               .ToList();

for (int i = 0; i <allInvoices.Count(); i++)
            {
for (int j = 0; j <myVendors.Count(); j++)
                {
if (allInvoices[i].CompanyID == myVendors[j].ID)
myInvoices.Add(allInvoices[i]);
                }
            }
returnmyInvoices;
}
```

4.2 База даних

У якості СКБД у курсовому проекті було використано MSSQLSERVER. MSSQL SERVER- вільна реляційна система управління базами даних. MSSQL SERVERє рішенням для малих і середніх додатків. Саме з цієї причини було вирішено використати саме цю систему. Програмна мова C# має ефективний та зручний фреймоврк для роботи із базами даних - EntityFramework. Це є безсумнівним плюсом і приводом використовувати дану зв'язку.В даному курсовому проекті був використаний підхід code-first. Використовувалися моделі даних, наприклад:

```
publicclassCompany
    {
publicintID { get; set; }
publicstring Name { get; set; }
publicstring Description { get; set; }
}
```

Спочатку створюється модель даних, потім створюється контекст даних:

```
publicclassBackendContext :
Microsoft.AspNetCore.Identity.EntityFrameworkCore.IdentityDbContex
t<IdentityUser>//DbContext
    {
publicBackendContext(DbContextOptions<BackendContext> options) :
base(options) { }

publicDbSet<Company> Companies { get; set; }
publicDbSet<CompanyToProduct>CompanyToProducts { get; set; }
```

```
publicDbSet<Product> Products { get; set; }
publicDbSet<Customer> Customers { get; set; }
publicDbSet<Vendor> Vendors { get; set; }
publicDbSet<VendorToProduct>VendorToProducts { get; set; }
publicDbSet<Invoice> Invoices { get; set; }
publicDbSet<InvoiceItem>InvoiceItems { get; set; }

}
```

Після цього завдяки міграціям та EntityFramework, створюється база даних або таблиця у ній.

Як можна побачити, такий спосіб здобування даних с бази даних є легко читаним, гнучким і зручним для подальших змін, на відміну від сирих запросів.

Дані можуть розподілятися за кількома системами зберігання даних, в кожній з яких застосовуються свої протоколи, але навіть в додатках, що працюють з однією системою зберігання даних, необхідно підтримувати баланс між вимогами системи зберігання даних і вимог написання ефективного і зручного для обслуговування коду програми [3].

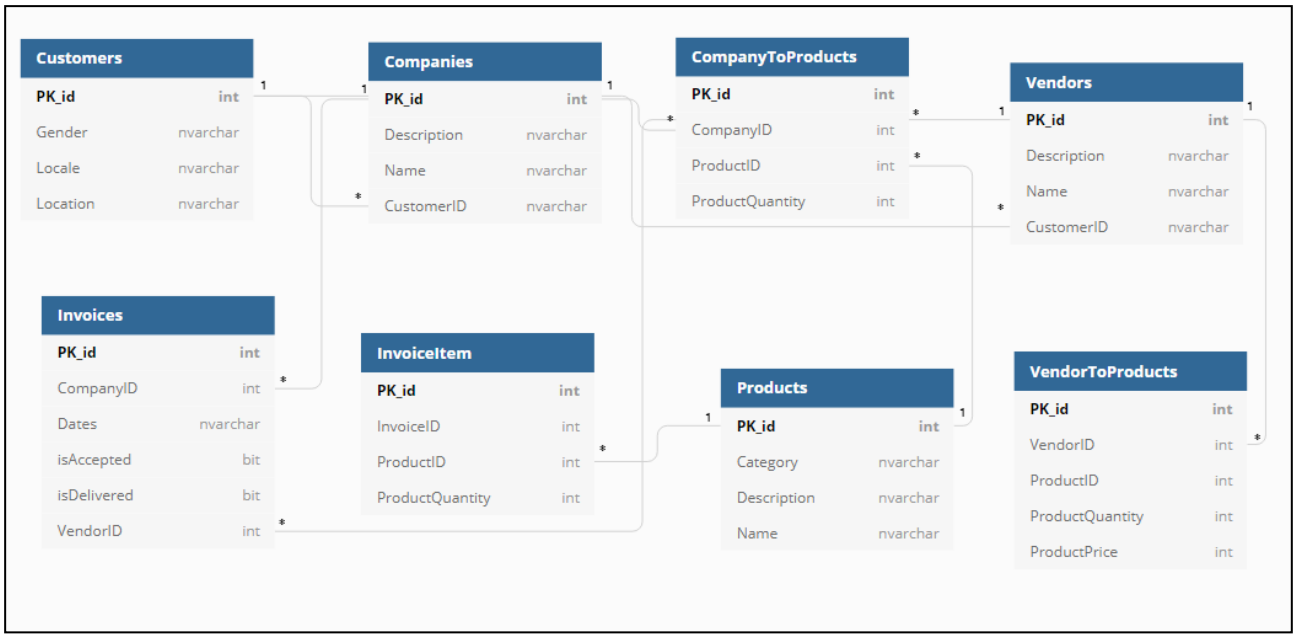

Рисунок 4.1 – Діаграма бази даних

На рисунку 4.1 зображена діаграма бази даних, що відображає таблиці, дані в них та зв'язки між таблицями. За допомогою вище описаного Entity

Framework та способу code-first, було створено базу даних для зберігання всієї необхідної інформації.

База даних містить в собі дані про користувачів, компанії, вендорів, накладні, продукти та допоміжні таблиці.

Вище наведена база даних знаходиться в третій нормальній формі [4]. Це означає, що вона нормалізована, тому ми можемо легко і швидко отримати всі необхідні дані.

4.3 Веб-клієнт

В даному курсовому проекті для розробки front-end був використаний AngularJSфреймворк. AngularJS - JavaScript-фреймворк з відкритим вихідним кодом для створення користувацьких інтерфейсів. Легко інтегрується в проекти з використанням інших JavaScript-бібліотек. Може функціонувати як веб-фреймворк для розробки багатосторінкових додатків в реактивному стилі. Оскільки при роботі з API потрібно використовувати так звані багатосторінкові додатки, було вирішено використовувати саме цей фреймворк. Він простий в засвоєнні і зручний в розробці, що є важливим аспектом для розробника.
Front-end існує для відображення даних у вигляді HTML сторінки. Коли потрібно, він звертається за даними до back-end серверу. З боку front-end використовується клас HttpClient, що дозволяє виконувати REST - запити.

На діаграмі прецедентів (див. рис. 4.2) можна побачити можливі дії користувача front-end. Можна побачити, що вебсайт має багато функцій:
Функції та можливості веб-клієнту:

а) відображення даних, отриманих від клієнтів;

б) створення компаній, вендорів, накладних

в) реєстрація та вхід користувача;

г) підтримка локалізації (UA, EN);

д) керування компаніями, вендорами та накладними;

е) перегляд історії.

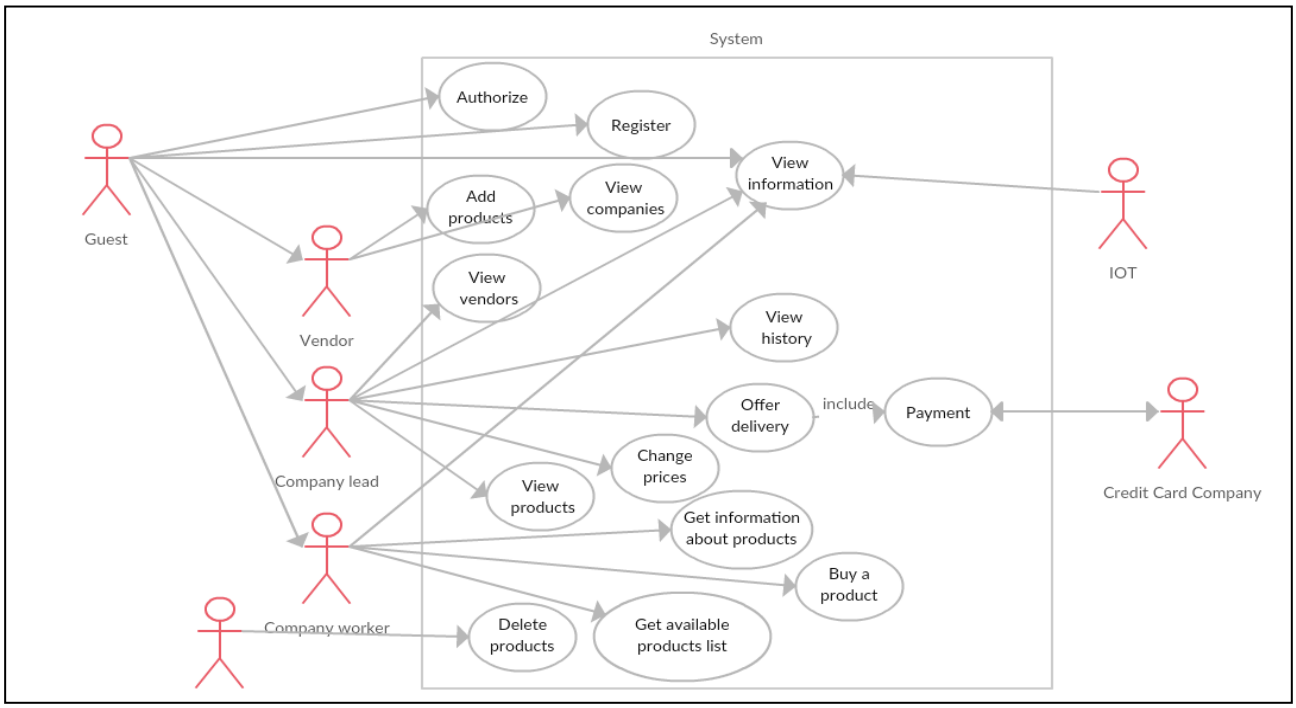

Рисунок 4.2 – Діаграма прецедентів

На рисунку 4.3 зображений користувацький інтерфейс для логіну, завдяки якому користувач може війти у систему.

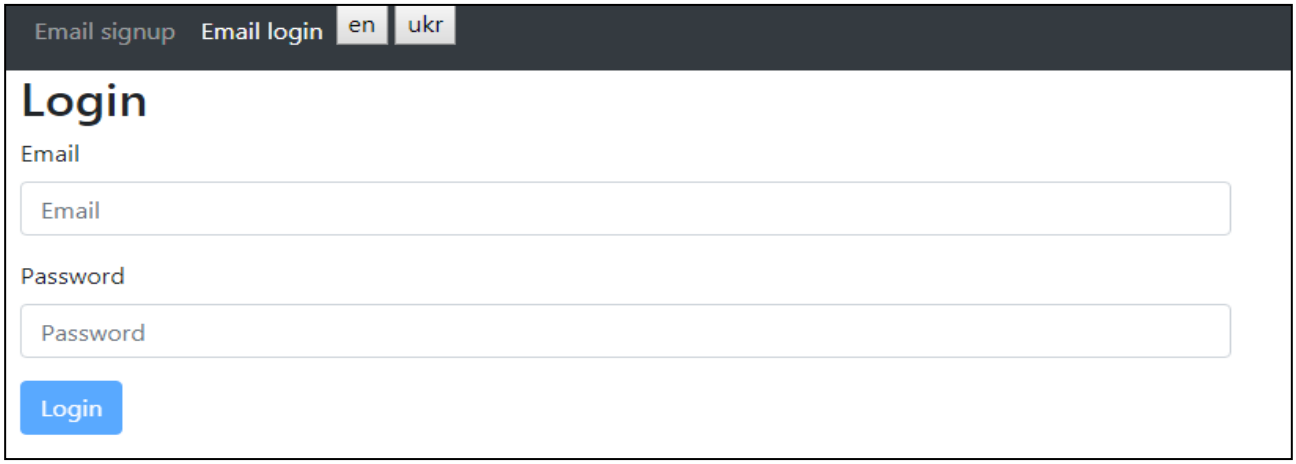

Рисунок 4.3 – Логін

На рисунку 4.4 зображений користувацький інтерфейс для реєстрації, завдяки якому користувач може зареєструватися у системі.

На рисунку 4.5 можна побачити користувацький інтерфейс перегляду усіх та власних компаній.

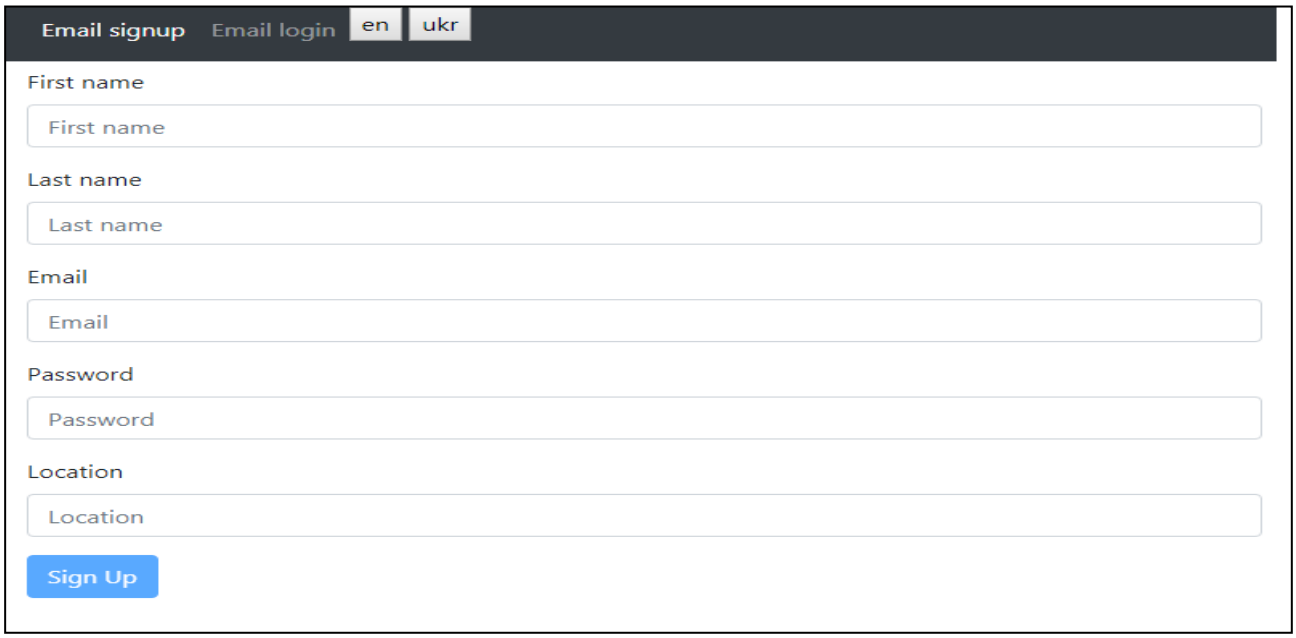

Рисунок 4.4 – Реєстрація

Користувач повинен ввести своє ім’я, імейл, пароль та локацію для того щоб зареєструватися у системі.

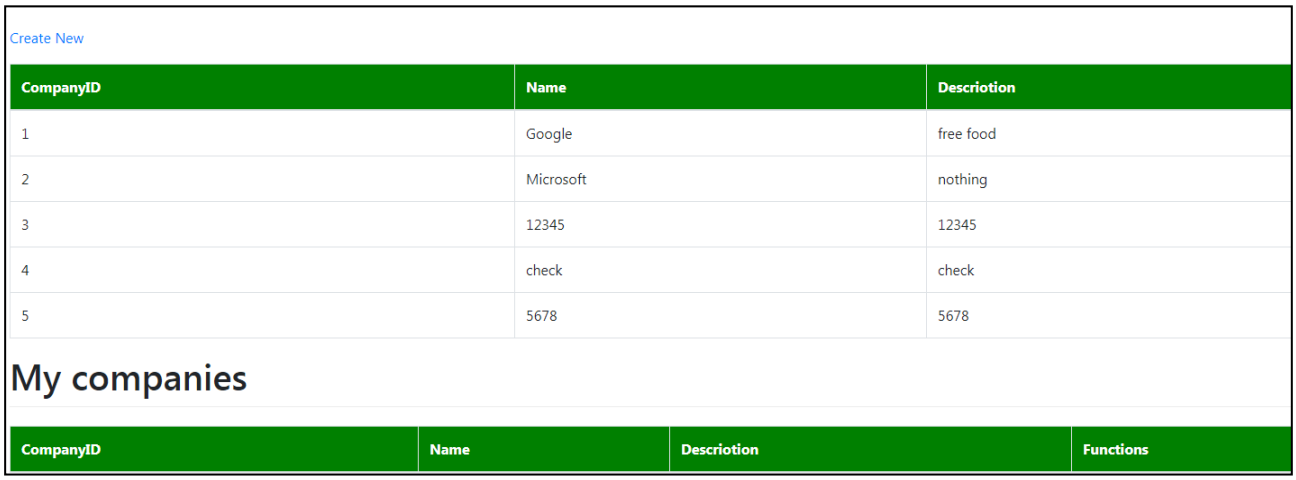

Рисунок 4.5 – СторінкаКомпанії

Для роботи з даними, необхідними для функціоналу використовується “CompaniesController” код якого наведено у додатку Б.

4.4 Мобільний клієнт

В мобільному додатку доступні функції реєстрації та авторизації.

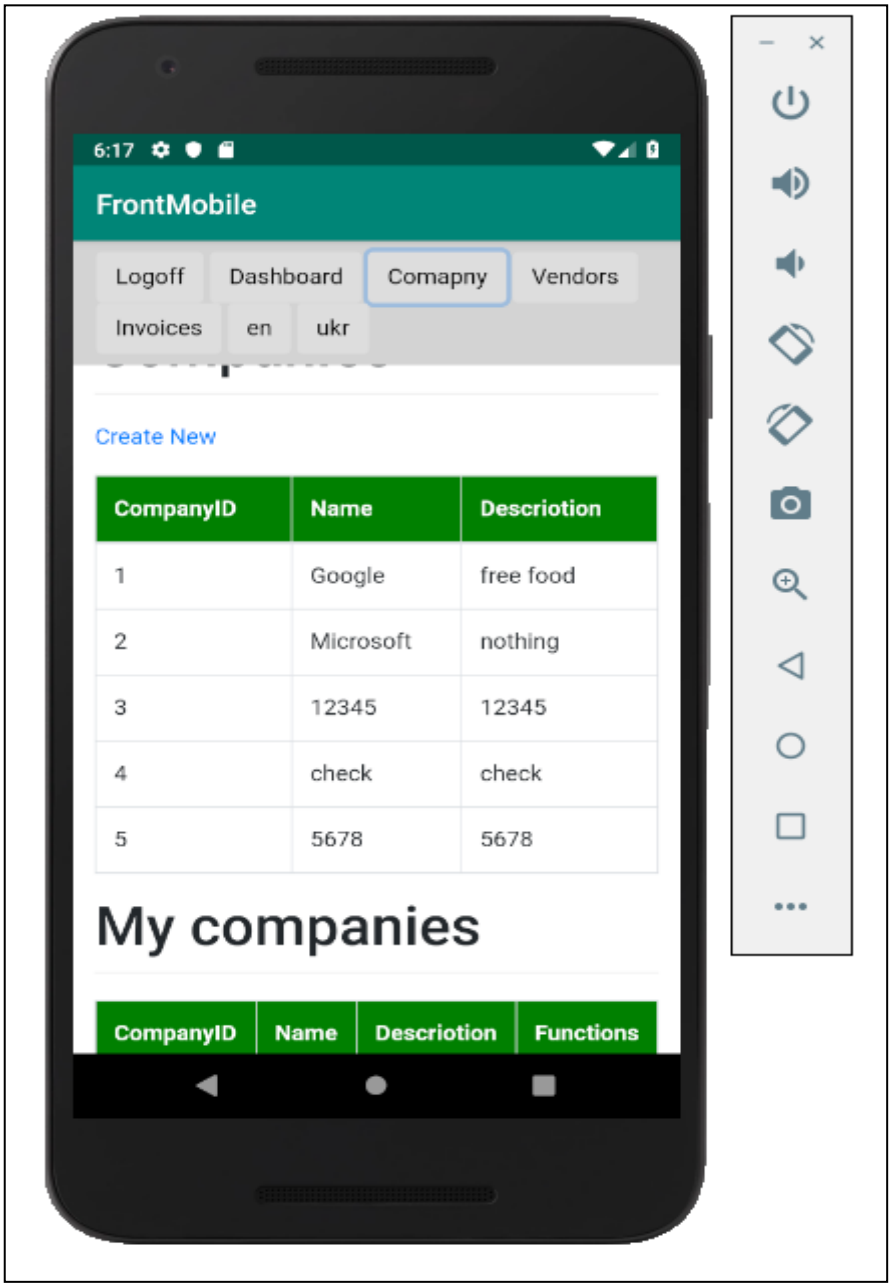

Рисунок 4.6 – Сторінка Компанії

Так само, як і у веб-додатку, для них виділено окремі форми, які не наведено через їх схожість і загальну незначущість. Код програми написано на Java [5].

Після успішної авторизації або реєстрації користувач потрапляє на сторінку профілю, звітки може переміщатися до будь-якої сторінки. Інтерфейс сторінки компаній зображено на рисунку 4.6

Як можна побачити, функціонал мобільного додатку дублює відповідний функціонал веб-додатку, щоб забезпечити максимально схожу функціональність для усіх користувачів незалежно від платформи.

На рисунку 4.7 наведено діаграму діяльності мобільного додатку, яка також може бути застосована і для вебсайту.

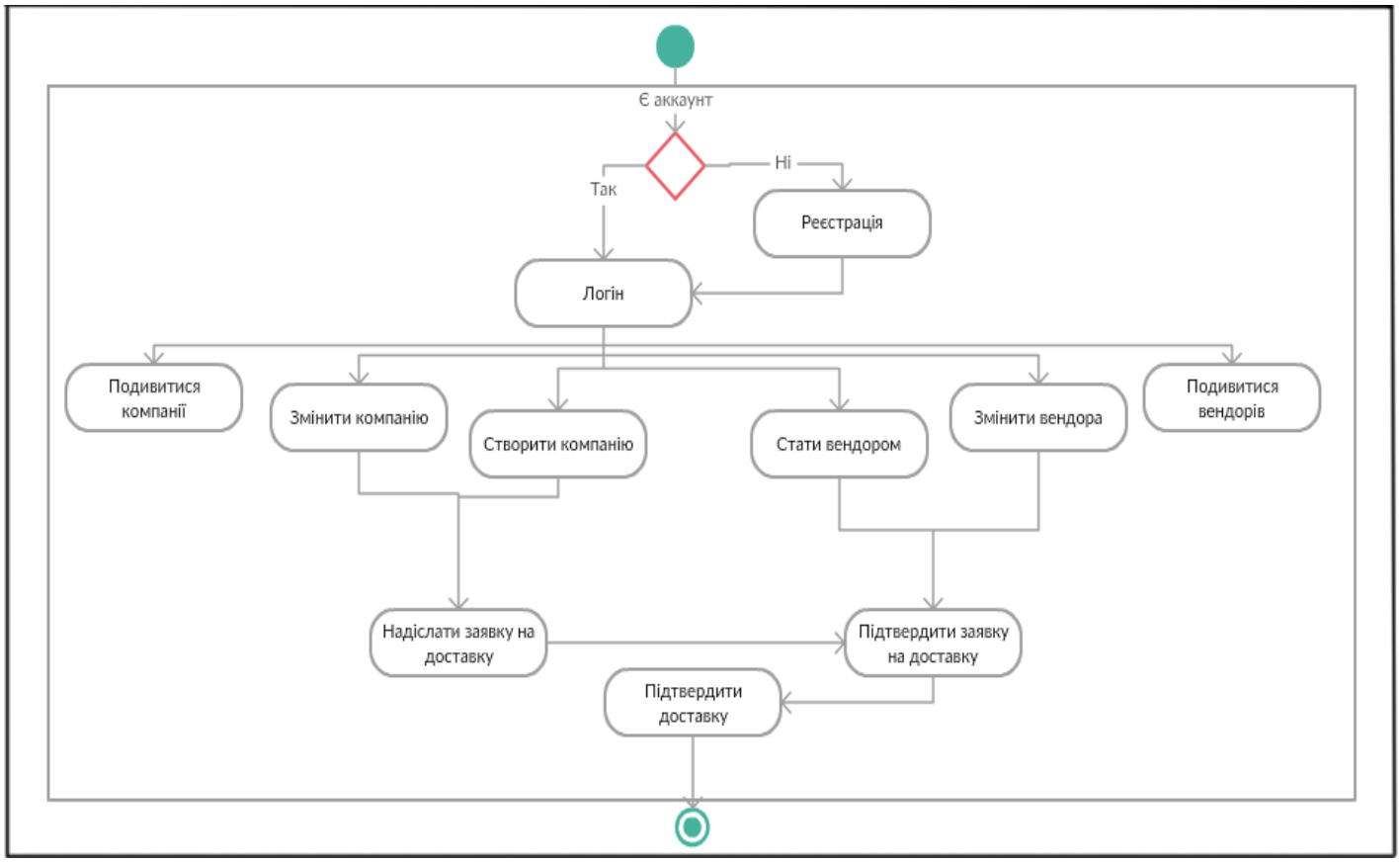

Рисунок 4.7 – Діаграма діяльності

Мобільний додаток можна запустити на будь-якому мобільному пристрої із системою Android 4.0.3+ та доступом до мережі Інтернет. Також, для використання мобільного додатку, не обов’язково треба бути зареєстрованим користувачем. Фрагмент коду мобільного додатка, який відповідає за функціонал екрану реєстрації наведено в додатку Д.

4.5 Розумний пристрій

В даному курсовому проекті розумний пристрій реалізований за допомогою ArduinoNano [6].

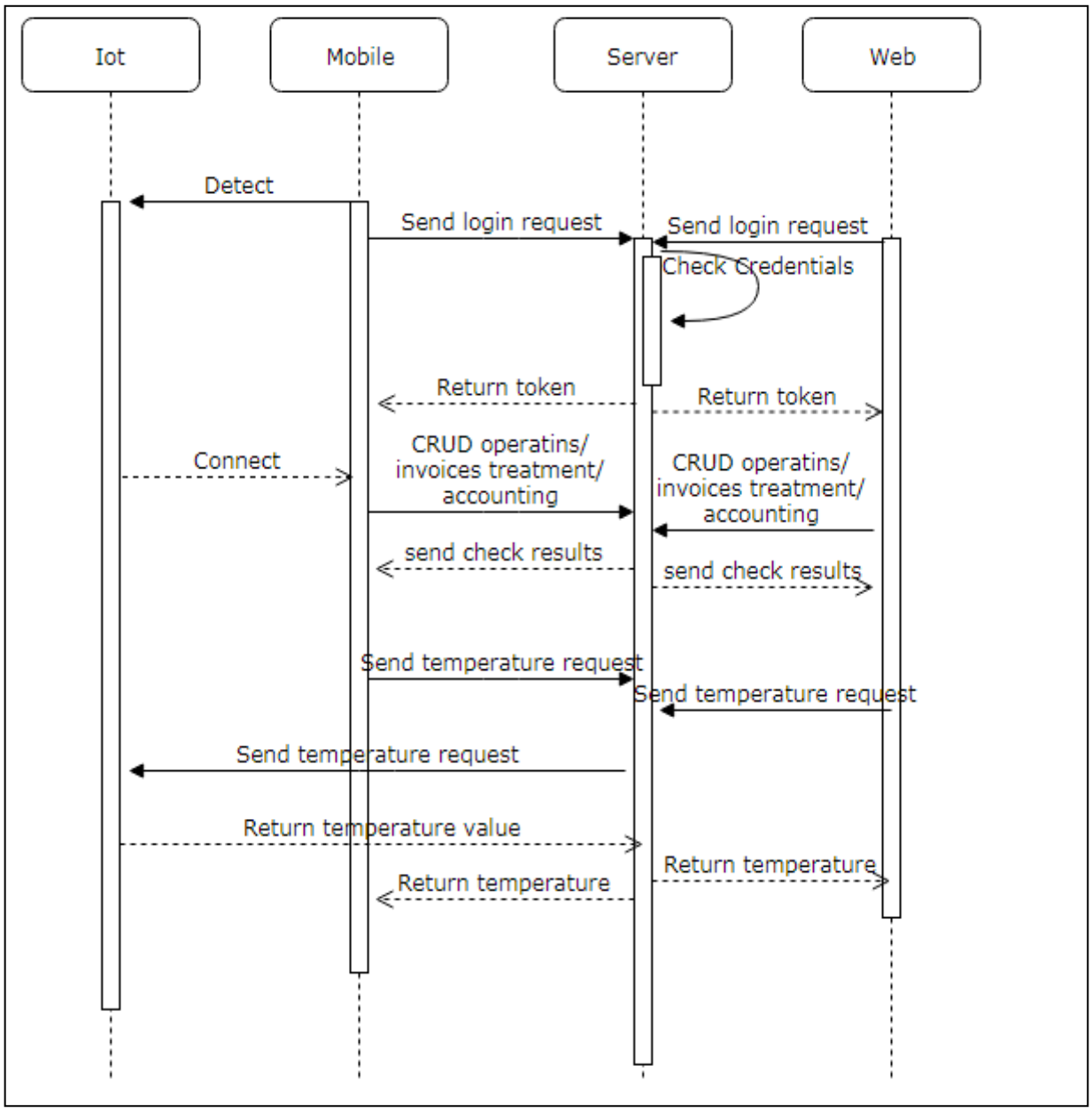

Рисунок 4.8 – Діаграма послідовності

Повністю відкрита архітектура системи дозволяє вільно копіювати або доповнювати лінійку продукції Arduino. Arduino може використовуватися як

для створення автономних об'єктів автоматики, так і підключатися до програмного забезпечення на комп'ютері через стандартні дротові і бездротові інтерфейси, що робить її придатною до використання у проекті.

У даному курсовому проекті розумний пристрій відповідає за запит даних освітлення та вимір температури

Принцип взаємодії між розумним пристроєм та сервером можна побачити на діаграмі послідовності проекту (див. рис. 4.8).

Для запалення діоду використовувалися команда digitalWrite, код з використанням якої наведене нижче:

```
if (val == '1')
    {
digitalWrite(red, HIGH);
delay(time);
digitalWrite(red, LOW);
}
```

Повний код розумного пристрою наведений у додатку В

ВИСНОВКИ

В результаті роботи було розроблено програмну систему для комунікації певної компанії із певним вендором, доставку та обліку. Вона складається з чотирьох частин: серверу, сайту, мобільного додатка та smart-пристрою.

Розробка велась з використанням платформи ASP.NET CoreWeb API 2 з використанням мови програмування C# для серверної частини програмної системи. Сайт було розроблено з використанням HTML5, CSS3, AngularJS та мови JavaScript. Для розробки мобільного додатку використовувалася середовище розробки AndroidStudio та мова програмування Java. Логіка роботи smart-девайсу була написана мовою програмування C# у середовищі розробки VisualStudio. В якості СКБД було обрано MS SQL Server 2016, а в якості технології доступу до даних –Entity Framework.

Розроблена програмна система є лише прототипом та не може в поточному стані використовуватися в реальних умовах. Перше, що необхідно доробити – доповнити систему більшим об’ємом функцій, наприклад: функції повідомлень вендора та компаній, додати більше інформації до бази даних та вебсайту. Також треба зробити інтерфейс програми більш зрозумілим для користувача.

Незважаючи на це, розвиток системи є досить перспективним, тому що відкриває можливість співпраці з різними компаніями, полегшує роботу компаній та вендорів зацікавлених у налагоджуванні сфери споживання. Система дозволяє виконувати базові функції такі, як авторизація, реєстрація, Створення компаній, вендорів, накладниї, додавання у накладні продуктів, маніпуляція статусом накладної.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Адаптивный код на C#. Проектированиеклассов и интерфейсов, шаблоны и принципы SOLID [Текст]: пер. с англ / – М.: ООО «И.Д. Вильямс», 2015. – 432 с.

2. C# 6.0. Справочник. Полноеописаниеязыка. [Текст]: пер.сангл / – М.: ООО «И.Д. Вильямс», 2016. – 1040 с.

3. Дейт, К. Дж. Введення у системи баз Даних [Текст] / К. Дж. Дейт.- 7е вид. - М.: Вид. дім «Вільямс», 2001. - 846 с.

4. ЛарриУльман. MySQL: Руководство по изучениюязыка / ЛарриУльман., 2018. – 250 с.

5. BruceEckel.Thinking in Java 4th Editon., 2015. –334 с.

6. GettingStartedwithArduino [Електронний ресурс] – Режим доступу до ресурсу: https://www.arduino.cc/en/Guide/HomePage (дата звернення: 28.12.2018).

ДОДАТОК А

Код DashboardController.cs

```
namespaceBackend.Controllers
{
[Authorize(Policy = "Admin")]
[Authorize(Policy = "ApiUser")]
    [Route("api/[controller]/[action]")]
publicclassDashboardController : Controller
    {
privatereadonlyClaimsPrincipal _caller;
privatereadonlyBackendContext _appDbContext;
privatereadonlyIHttpContextAccessor _httpContextAccessor;

publicDashboardController(UserManager<AppUser>userManager,
BackendContextappDbContext,
IHttpContextAccessorhttpContextAccessor)
        {
            _caller = httpContextAccessor.HttpContext.User;
            _appDbContext = appDbContext;
            _httpContextAccessor = httpContextAccessor;
        }

// GET api/dashboard/home

        [HttpGet]
publicasync Task<IActionResult> Home()
        {
varuserId = _caller.Claims.Single(c =>c.Type == "id");

var customer = await _appDbContext.Customers.Include(c
=>c.Identity).SingleAsync(c =>c.Identity.Id == userId.Value);

returnnewOkObjectResult(new
            {
                customer.ID,
                Message = "This is secure API and user data!",
customer.Identity.FirstName,
customer.Identity.LastName,
customer.Identity.PictureUrl,
customer.Identity.FacebookId,
customer.Location,
customer.Locale,
customer.Gender
});
        }

    }

}
```

ДОДАТОК Б

Код CompaniesController.cs

```
namespaceBackend.Controllers
{
//[Authorize(Policy = "Admin")]
//[Authorize(Policy = "ApiUser")]
    [Produces("application/json")]
    [Route("api/Companies")]
publicclassCompaniesController : Controller
    {
privatereadonlyBackendContext _context;
privatereadonlyClaimsPrincipal _caller;

publicCompaniesController(BackendContext context,
IHttpContextAccessorhttpContextAccessor)
{
            _context = context;
            _caller = httpContextAccessor.HttpContext.User;
        }


publicstaticboolisOwner(intcompanyId, ClaimsPrincipal _caller,
BackendContext _context)
        {
varuserId = _caller.Claims.Single(c =>c.Type == "id");
var customer = _context.Customers.Include(c =>c.Identity).Single(c
=>c.Identity.Id == userId.Value);

var company = _context.Companies.SingleOrDefault(m => m.ID ==
companyId);
if (customer.ID == company.CustomerID) returntrue;
returnfalse;
        }

        [Route("getMyCcompanies")]
        [HttpGet]
publicIEnumerable<CompanyViewModel>getMyCompanies()
        {
            List<CompanyViewModel>myCompanies = new
List<CompanyViewModel>();
var Companies =  _context.Companies.Select(c =>newCompanyViewModel
            {
                ID = c.ID,
                Name = c.Name,
                Description = c.Description,
CustomerID=c.CustomerID
            })
               .ToList();
```

```
foreach (var company in Companies)
if (isOwner(company.ID,_caller,_context))
myCompanies.Add(company);
//myCompanies.Add
returnmyCompanies;
        }

// GET: api/Companies
        [HttpGet]
publicIEnumerable<CompanyViewModel>GetCcompanies()
        {
return _context.Companies.Select(c =>newCompanyViewModel
            {
                ID = c.ID,
                Name = c.Name,
                Description = c.Description
            })
               .ToList();
        }

// GET: api/Companies/5
        [HttpGet("{id}")]
publicasync Task<IActionResult>GetCompany([FromRoute] int id)
        {
if (!ModelState.IsValid)
            {
returnBadRequest(ModelState);
            }

var company = await _context.Companies.SingleOrDefaultAsync(m =>
m.ID == id);

if (company == null)
            {
returnNotFound();
            }

return Ok(company);
        }

// PUT: api/Companies/5
        [HttpPut("{id}")]
publicasync Task<IActionResult>PutCompany([FromRoute] int id,
[FromBody] Company company)
        {
if (!ModelState.IsValid)
            {
returnBadRequest(ModelState);
            }

if (id != company.ID)
            {
```

```
returnBadRequest();
            }

            _context.Entry(company).State = EntityState.Modified;

try
            {
await _context.SaveChangesAsync();
            }
catch (DbUpdateConcurrencyException)
            {
if (!CompanyExists(id))
                {
returnNotFound();
                }
else
                {
throw;
                }
            }

returnNoContent();
        }

// POST: api/Companies
        [HttpPost]
publicasync Task<IActionResult>PostCompany([FromBody] Company
company)
        {
if (!ModelState.IsValid)
            {
returnBadRequest(ModelState);
            }


varuserId = _caller.Claims.Single(c =>c.Type == "id");
var customer = _context.Customers.Include(c =>c.Identity).Single(c
=>c.Identity.Id == userId.Value);
company.CustomerID = customer.ID;

            _context.Companies.Add(company);
await _context.SaveChangesAsync();

returnCreatedAtAction("GetCompany", new { id = company.ID },
company);
        }

// DELETE: api/Companies/5
        [HttpDelete("{id}")]
publicasync Task<IActionResult>DeleteCompany([FromRoute] int id)
        {
if (!ModelState.IsValid)
            {
```

```
returnBadRequest(ModelState);
            }

var company = await _context.Companies.SingleOrDefaultAsync(m =>
m.ID == id);
if (company == null)
            {
returnNotFound();
            }

            _context.Companies.Remove(company);
await _context.SaveChangesAsync();

return Ok(company);
        }

privateboolCompanyExists(int id)
        {
return _context.Companies.Any(e => e.ID == id);
}
    }
}
```

ДОДАТОК В

Код скетчу для Arduino

```
///set pin numbers for leds

//power indicator led
intpwr = 6;

//controlled leds

int red = 13;


///set baudrate for serial connection
int baud = 9600;

///set delay time: 1 second
int time = 5000;

void setup()
{
  //set pinmode and turn on power led
pinMode(pwr, OUTPUT);
digitalWrite(pwr, HIGH);

  //start serial
Serial.begin(baud);

  //set pinmode for controlled leds
pinMode(red, OUTPUT);

}
```

```
// the loop routine runs over and over and over and over
void loop()
//loop constantly scans for serial input
{
  //when connection is more than 1 begin if statements
if (Serial.available() > 0)
  {
    //read serial input value
intval = Serial.read();
    //begin if statement

if (val == '1')
    {
digitalWrite(red, HIGH);
delay(time);
digitalWrite(red, LOW);
    }

if (val == '2')
    {
digitalWrite(red, HIGH);
delay(500);
digitalWrite(red, LOW);
delay(500);
digitalWrite(red, HIGH);
delay(500);
digitalWrite(red, LOW);
delay(500);
digitalWrite(red, HIGH);
delay(500);
digitalWrite(red, LOW);
delay(500);
digitalWrite(red, HIGH);
delay(500);
digitalWrite(red, LOW);
delay(500);
```

```
digitalWrite(red, HIGH);
delay(500);
digitalWrite(red, LOW);
    }

    //flush the serial value
Serial.flush();
}
}
```

ДОДАТОК Г

Код Логіну

```
import { Subscription } from 'rxjs';
import { Component, OnInit,OnDestroy } from '@angular/core';
import { Router, ActivatedRoute } from '@angular/router';

import { Credentials } from
'../../shared/models/credentials.interface';
import { UserService } from '../../shared/services/user.service';
import { Globals } from '../../globals';
@Component({
selector: 'app-login-form',
templateUrl: './login-form.component.html',
styleUrls: ['./login-form.component.scss']
})

export class LoginFormComponent implements OnInit, OnDestroy {

private subscription: Subscription;

brandNew: boolean;
errors: string;
isRequesting: boolean;
submitted: boolean = false;
credentials: Credentials = { email: '', password: '' };

constructor(private userService: UserService, private router:
Router,privateactivatedRoute: ActivatedRoute) { }

ngOnInit() {

    // subscribe to router event
this.subscription = this.activatedRoute.queryParams.subscribe(
```

```
      (param: any) => {
this.brandNew = param['brandNew'];
this.credentials.email = param['email'];
      });
  }

ngOnDestroy() {
    // prevent memory leak by unsubscribing
this.subscription.unsubscribe();
  }

login({ value, valid }: { value: Credentials, valid: boolean }) {
this.submitted = true;
this.isRequesting = true;
this.errors='';
if (valid) {
this.userService.login(value.email, value.password)
        .finally(() =>this.isRequesting = false)
        .subscribe(
result => {
if (result) {
            //global.email = value.email;
this.router.navigate(['/dashboard/home']);
}
        },

error =>this.errors = error);
    }
  }
}
```

ДОДАТОК Д

Фрагмент відображення webview

```
privateWebViewwebview;

    @Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

webview =(WebView)findViewById(R.id.webView);

webview.setWebViewClient(new WebViewClient());
webview.getSettings().setJavaScriptEnabled(true);
webview.getSettings().setDomStorageEnabled(true);
webview.setOverScrollMode(WebView.OVER_SCROLL_NEVER);
webview.loadUrl("http://10.0.2.2:4200");

//webview.loadUrl("http://192.168.1.101:45455/api/companies");
    }
```

ДОДАТОК Е
Специфікація ПЗ

1 ВСТУП

1.1 Огляд ПЗ

Для кращого розуміння системи необхідний приклад. Кожна компанія з великою кількістю співробітників повинна мати систему харчування. Власникові компанії важко створювати та маніпулювати такою системою, і це вимагає дорогоцінного часу. Отже, проект необхідний для полегшення роботи продовольчої сфери у великих компаніях. Адміністратор Компанії може замовити необхідну їжу від місцевих постачальників для швидкої доставки, тому не потрібно самостійно виконувати пошук. Незважаючи на це, система включає власну функціональність оплати та інформування. Таким чином, кожен користувач має можливість купувати продукти зі зручною формою оплати і отримувати повідомлення про ціни. Головною метою проекту є економія часу та грошей для приватних компаній.

1.2 Мета

Послуга вирішує проблему часу та ресурсів, витрачених на планування, збір інформації про постачальників, підписання контрактів тощо. Клієнти мають можливість робити замовлення і навіть оформляти регулярні замовлення без зайвих рухів, що призводить до помилок і які сьогодні не потрібні.

Оскільки система є багатофункціональною, є конкуренти в різних сферах.

Основна функціональна база в сервісі розроблена для швидких замовлень і доставки їжі до офісу. Тут конкурентами є продовольчі магазини (Сільпо, АТБ, Fozzy), магазини з оптовими закупівлями та інші послуги доставки їжі (інтернет-магазини). Послуги з доставки їжі не можуть конкурувати з послугою через високі ціни на продукти та доставку, отже, для великих компаній неефективно витрачати ресурси марно. Продовольчі магазини та магазини з оптовими закупівлями не пропонують поставки, але магазини з оптовими закупівлями можуть запропонувати кращі ціни. Тим не менш, в цих структурах немає постачання, тому компанія повинна доставляти продукцію сама або наймати спеціальних агентів. На завершення цього пункту можна сказати, що наш сервіс краще майже за всіма параметрами

Другий тип конкурентів - це конкуренти в області програмного забезпечення. Більше конкретно - системи з різними бухгалтерськими функціями, які дозволяють вести облік доступних продуктів, необхідних продуктів, реалізованої продукції тощо. Це означає, що у наших клієнтів немає необхідності наймати фахівців у цій сфері або створювати декілька баз даних для зберігання дат і створення спеціальної бізнес-логіки для обліку всіх необхідних подій, оскільки вона вже створена системою. Тому ефективніше витрачати гроші один раз на сервер і не витрачати гроші кожен раз без гарантованого результату.

Третій тип - ресторани. Зазвичай, в самому офісі є кілька місць для їжі, і працівник повинен узяти велику перерву (1-2 години), щоб знайти місце для їжі. Надана послуга може компенсувати її, поставляючи вже готову продукцію (бутерброди, гамбургери, салати тощо), щоб задовольнити побажання клієнта. Також є інші способи організації продуктової сфери:

а) доставляти продукти харчування для місцевих ресторанів;

б) доставляти продукти для ресторанів компанії, якщо вони існують;

в) створювати знижки для отримання клієнтської бази і т.д.

Великою перевагою послуги є всі описані функції в одному пакеті, що економить ресурси компанії.

1.3 Межі

Основний потік доходу від послуги походить від підписки, яку компанія повинна придбати для використання послуги. Оскільки проект тільки виходить на ринок, ціна підписки не буде занадто високою. Важливо зазначити, що оскільки ми є онлайн-проектом, нашій компанії не потрібно платити орендну плату. Протягом часу ми надаємо всі функції, які були описані пізніше, компанії та постачальники будуть зацікавлені в існуванні компанії.

Узагальнюючи написане раніше, проект будуть мати постійний грошовий потік після того, як отримає клієнтську базу, що є основним фактором успіху. Клієнтська база повинна бути достатньо великою, щоб компанії та постачальники могли взаємодіяти, а для нас - досягти точки беззбитковості. Щоб ми могли стабілізуватися на ринку і поступово рости.

Коли наша організація буде виходити на ринок, вона повинна бути зосереджена на залученні клієнтів, що є основною метою бізнесу. Після того, як компанія зможе забезпечити себе, це означає, що ця мета задоволена. Потім може знадобитися певний час на розширення можливостей і т.д. Після того, як всі клієнти будуть задоволені (наступна мета), ми зможемо зайняти нову позицію на ринку і витратити фінансовий ресурс, щоб отримати більшу клієнтську базу (кінцева мета).

1.4 Потреби користувачів

Потребам типових клієнтів є замовлення спеціального виду продукції за найнижчою ціною і з швидкою доставкою, з однієї сторони і системою обліку з іншого боку. Зайнятий час і ймовірність відмови також є важливим фактором для клієнта. Наш сервіс орієнтований на конкретні мережі, які, можливо,

захочуть зробити регулярне замовлення і мати постійного постачальника для полегшення його бізнесу.

Здатність регулярних замовлень, можливість комунікації між провайдером і клієнтом, доставка, облік і наявність всіх функцій в одному пакеті не реалізується в будь-якій існуючій системі. Тому наша служба буде важливою для наших майбутніх клієнтів. Клієнти зможуть використовувати систему для зв'язку з провайдерами або покупцями, здійснювати швидкі та безпечні замовлення, отримувати знижки та регулярні замовлення або регулярні поставки.

Для використання нашого сервісу клієнт повинен мати оновлений Інтернет-браузер і доступ до Інтернету;

Вимоги користувача:

а) реєстрація, авторизація, перевірка;

б) можливість надання запиту на постачання;

в) додавання продукції;

г) видалення продукції;

д) обробка рахунків;

е) модифікація цін на продукцію компанії;

є) пошук постачальника;

ж) модифікація цін на продукцію компанії для конкретного покупця;

з) створення груп покупців з метою зміни ціни для групи;

и) модифікація права;

і) запит на замовлення;

ї) договір для тривалого замовлення;

й) огляд наявних продуктів;

к) огляд історії системи;

л) створення концесії;

м) відкриття інформації про наявність товарів постачальникам;

н) повідомлення.

2 ЗАГАЛЬНИЙ ОПИС

2.1 Перспективи ПЗ

Узагальнюючи мету і написане раніше, ми можемо сказати, основна мета полягає в тому, щоб полегшити роботу компаній і корпорацій у харчовій галузі, а також полегшити роботу постачальників, які працюють у цій галузі.

Послуга надає багато функцій, які компанії або постачальники зазвичай роблять самі. Постачальники отримують велику клієнтську базу, яка допоможе їм отримувати нові замовлення та розповсюджувати рекламу для компаній. На той час компанії мають можливість вибирати постачальників з найкращими цінами і сервісом взагалі. Крім того, вони отримують систему бухгалтерського обліку, здатність маніпулювати нею і здійснення платежів, що в одному реченні є автоматизованим в області продовольства. Таким чином, компанії та постачальники можуть полегшити її роботу після придбання програми.

Висновок: це - автоматизована система харчування, яку зараз потребує ринок

2.2 Функції ПЗ

Система матиме такі функції:

а) Можливість реєстрації на сайті;

б) Повна автоматизована система з автоматизованими обчисленнями;

в) Можливість створення регулярних замовлень;

г) підтвердження електронної пошти при реєстрації;

д) Інтернет-служба підтримки для надання допомоги та відповіді на запитання;

е) Автоматизація практично всіх процесів;

є) Можливість отримання знижок;

ж) Зручний інтерфейс.

з) підтримка локалізації (UA, EN);

2.3 Характеристики користувачів

Користувач може додати, змінити або видалити свою компанії або вендора, але перед цим йому потрібно або зареєструватися, або ввійти в систему. Також користувач може створювати накладні, та змінювати їх статус. Якщо користувач увійшов у систему, він може вийти з неї.

Адміністратор може входити в систему, управляти користувачами, додавати нові продукти, налагоджувати систему, якщо це потрібно.

2.4 Загальні обмеження

Система не зможе працювати без доступу до Інтернету. Для правильної роботи системи потрібен сервер із підтримкою версії ASP.NET Core та IIS.

2.5 Припущення й залежності

AS-1: Система є автоматичною, тому вона ставить на себе відповідальність за замовлення та збереження продуктів на собі.

DE-1: Це повинно збільшити вільний час для клієнтів і зменшити проблеми.

AS-2: Використовуючи можливість комп'ютерних обчислень, наша система робить всі самі розрахунки.

DE-2: це зменшить кількість помилок користувача.

AS-3: Система завжди повідомляє клієнта про необхідні продукти.

DE-3: Клієнт завжди буде знати поточну інформацію.

3 КОНКРЕТНІ ВИМОГИ

3.1 Вимоги до зовнішніх інтерфейсів

Серверна частина:

а) реєстрування та керування користувачами;

б) керування налаштуваннями існуючих пристроїв;

г) захист даних (через JWT токен);

д) реєстрування компаній, вендорів та накладних;

е) управління статусом накладної.

Web клієнт:

а) відображення даних, отриманих від клієнтів;

б) реєстрація та вхід користувача;

в) підтримка локалізації (UA, EN);

г) захист даних (через JWT токен);

д) додавання та керування компаніями, вендорами та накладними;

е) перегляд історії.

Android клієнт:

а) вхід користувача;

б) додавання та керування компаніями, вендорами та накладними;

в) захист даних (через JWT токен);

г) підтримка локалізації(UA, EN).

IoT пристрій:

а) реалізація API для прийому команд клієнта;

б) зміна яскравості світла;

в) увімкнути / вимкнути світло.

3.2 Властивості ПЗ

Система буде розроблена для клієнтів, які не відносяться до певної країни або континенту, а буде розповсюджена на весь світ. Клієнти та користувачі матимуть доступ до системи в будь який час, не залежно від часового поясу. Дані, які система оброблятиме про користувачів, будуть зберігатися на віддалених серверах, які будуть розміщені в кожній країні, де ця система буде втілена. Для якісного сервісу системи повинна бути чітка робота, без переривань. Також клієнт і користувач повинні будуть мати постійний доступ до програмного продукту. Для захисту облікових записів користувачів буде використовуватися JWT токен. У базі даних інформація про користувача буде зберігатися в закодованому стані з використанням алгоритму шифрування «BCryptPassword». Під час розробки будуть використовуватися такі технології: ASP.NET Core, AngularJS, MS SQL SERVER, IIS, JavaAndroid.

3.3 Зміст подальших релізів

Система буде мати поетапну еволюцію. До наступних випусків відійдуть такі функції:

а) більше варінатів платіжних систем;

б) додання локалізації для більшості розповсюджених мов;

в) створення сайтів та інших веб-ресурсів на основі описаного сервісу.