

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Центр післядипломної підготовки
(повна назва)

Кафедра _____ Програмна інженерія
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський)

Програмне забезпечення підтримки роботи
відділу контролю якості підприємства
(тема)

Виконав:

студент 4 курсу, групи ПЗППз-18-1

Скиданенко Д. М.

(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного
забезпечення

(код і назва спеціальності)

Освітня програма Програмна інженерія

(повна назва освітньої програми)

Керівник ст. викл. Козел Н. Б.

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф.

Дудар З. В.

(підпис)

(прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет	Комп'ютерні науки
Кафедра	Програмна інженерія
Рівень вищої освіти	перший (бакалаврський)
Спеціальність	121 – Інженерія програмного забезпечення
	(код і повна назва)
Освітня програма	Програмна інженерія
	(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри

(підпис)

« ____ » _____ 20 ____ р.

ЗАВДАННЯ НА АТЕСТАЦІЙНУ РОБОТУ

студентові

Скиданенко Дмитру Михайловичу

(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Програмне забезпечення підтримки роботи відділу контролю якості підприємства

затверджена наказом по університету від « 15 » травня 2020 р. № 58 Стз

2. Термін подання студентом роботи до екзаменаційної комісії «15» червня 2020р.

3. Вхідні дані до роботи Офіційна документація системи управління якістю ISO 9001:2015; приклади комплексної інформаційної системи керування великим підприємством та малої системи, написаної та керованої вручну; статистика обсягу оброблюваних підприємством даних, приклади типових завдань менеджера з якості

4. Перелік питань, що потрібно опрацювати в роботі аналіз предметної галузі, формування вимог до програмного забезпечення, проектування архітектури програмного забезпечення, написання програмного забезпечення та тестів, аргументація та опис прийнятих програмних рішень.

КАЛЕНДАРНИЙ ПЛАН

[illegible]

Дата видачі завдання 20 р.

Студент _____ (підпис)

Керівник роботи _____
(підпис)

ст. викл. Козел Н. Б.
(посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи бакалавра: 61 сторінок, 2 таблиці, 20 рисунків, 4 додатки, 13 джерел.

ASP.NET CORE, C#, MVC, BOOTSTRAP, MS SQL SERVER, ISO 9001:2015, СИСТЕМА ЯКОСТІ, РЕКЛАМАЦІЯ, МЕТРОЛОГІЯ

Об'єктом дослідження є інформаційна система, створена для автоматизації деяких задач в роботі відділу управління якістю (на основі системі управління якістю ISO 9001:2015).

Метою роботи є проектування кросплатформеної модульної інформаційної системи з можливістю розширення в подальшому.

Методи розробки базуються на шаблоні проектування MVC та платформі ASP.NET Core 3.1, використовується база даних MS SQL Server. Розробка ведеться в програмному середовищі MS Visual Studio 2019, використовуючи засоби мови програмування C#. Зовнішнє оформлення «тонкого» клієнта велось за допомогою бібліотеки швидкого прототипування Bootstrap. Розробка бази велась за допомогою MySQL Workbench 8.0.16. та Microsoft SQL Server Management Studio 18.

В результаті роботи спроектовано програмне забезпечення підтримки роботи відділу контролю якості.

Explanatory note to bachelor's certification project, 61 pages, 2 tables, 20 figures, 4 annexes, 13 sources.

ASP.NET CORE, C#, MVC, BOOTSTRAP, MS SQL SERVER, ISO 9001:2015, QUALITY MANAGEMENT SYSTEM, CLAIM, METROLOGY

The purpose of this system's development is quality department's routine automatization (based on ISO 9001:2015 rev.). The work's main aim is development of cross-platform information system with scaling possibilities in the future.

Development methods are based on MVC design pattern within the scope of ASP.NET Core 3.1. Interaction with database was performed by using Entity Framework Core technology.

Server side is written with Visual Studio 2019 Community edition, the database was designed in MySQL Workbench 8.0.16 and tested in Microsoft SQL Server Management Studio 18.

Client side was developed using Visual Studio Code editor, Gulp.js task manager, Bootstrap 4 “rapid” prototyping library, CSSC and Pug.js pre-processors for CSS and html respectively.

As a result, quality department’s routine automatization software was designed and partially created.

ЗМІСТ

Вступ.....	8
1 Аналіз предметної галузі.....	10
1.1 Аналіз предметної галузі.....	10
1.2 Виявлення проблем та актуалізація рішень	12
1.3 Постановка задачі	18
2 Формування вимог до ПЗ	19
2.1 Вимоги до оточення.....	19
2.2 Функціональні вимоги.....	19
2.3 Нефункціональні вимоги.....	20
2.4 Можливі обмеження розробки	21
3 Архітектура та проектування ПЗ.....	22
3.1 Загальні відомості	22
3.2 Проектування архітектури ПЗ	22
3.3 Проектування системи зберігання даних	26
3.4 Створення дизайну системи.....	31
4 Опис прийнятих програмних рішень	33
4.1 Загальні відомості	33
4.2 Опис прийнятих програмних рішень	33
5 Тестування розробленого програмного забезпечення	49
5.1 Функціональне тестування.....	49
5.2 Нефункціональне тестування.....	51
Висновки	52
Перелік джерел посилання.....	53

Додаток А Повна схема таблиць бази даних.....	55
Додаток Б Скетчі дизайну клієнтської частини програми.....	56
Додаток В Лістинг коду усіх сутностей (Models) додатку	58
Додаток Г Слайди презентації до проекту	61

ВСТУП

Багато підприємств середнього та великого бізнесу існують впродовж значного періоду часу, навіть десятиріччями. Але, більш того, швидкість та інтенсивність життя не підпорядковується лінійній залежності, більш нагадуючи логарифмічну. Виникають нові підприємства аналогічного характеру та сфери інтересів, більш сучасні та технологічно розвинені. А це в багатьох випадках потребує від старих гравців на цьому ринку готовності до швидких та якісних змін.

Внаслідок цього з'являється потреба в автоматизації багатьох процесів, що раніше оброблялись вручну.

Більше за те, ці процеси постійно пришвидшуються, вдосконалюються та розвиваються, масштабуючись до завдань, які абсолютно неможливо обробляти вручну в штатному режимі.

Після набуття широким загалом доступу до мережі Internet (далі – Мережа), відбулася плавна, але незворотна якісна зміна, в результаті якої застосування інформаційних систем взагалі стало критерієм життєздатності підприємства.

Разом із цим, нового рівня сягнула культура спілкування з клієнтом, зокрема стандарти швидкості зворотного зв'язку та обробки запитів.

З іншої сторони, із часом все більше поширюється взаємна інтеграція країн у сфері торгівлі та комерційної або господарської діяльності. Дуже часто люди – представники різних держав, культур та взагалі світогляду – повинні взаємодіяти між собою для досягнення мети, що не завжди буває просто.

В якийсь мірі, для полегшення взаємодії створені стандарти систем якості, такі, як ISO 9001 [1]. Цей стандарт регламентує систему управління якістю. Із введенням цього стандарту на підприємстві стають прозорими всі процеси, їх можна обчислити та оцінити їх результати. Тим самим, ця система управління якістю стають необхідною складовою середніх та великих підприємств.

На сьогодні, Стандарт існує в імплементації 9001:2015, який вже значно м'якше, ніж 9001:2008 ставиться до документації. Стандарт пристосовується до

потреб сьогодні, і розуміючи, що сьогодні документація (оформлення звітів, зокрема паперових) вимагає значного обсягу часу, в порівнянні із іншими поточними справами. За новими правилами стандарту 9001:2015, процеси дозволяється не документувати, а просто вести записи [2].

І це для нас чудова новина, тому що для ведення записів/обліку в будь-яких процесах найкращим інструментом є саме інформаційні системи з базами даних. Стандарт в цілому регламентує велику кількість періодичних завдань, а усяди, де можна відслідкувати періодичність чи простежити залежності, процеси мають бути автоматизовані.

Отже, дана робота має на меті проектування програмного забезпечення підтримки роботи відділу контролю якості підприємства, що буде застосовано для автоматизації більшості завдань та вивільнення значного обсягу часу, як невідновлювального ресурсу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Основна задача для працівника відділу якості – це виявлення системних проблем, пошук причин їх виникнення та розробка рішень для їх усунення. Але крім цього є також велика кількість періодичних завдань (перевірка вимірювального інструмента, проходження атестацій для персоналу, розгляд звернень клієнтів (рекламації), керування складами, замовлення витратних матеріалів та ін. Значну частину цих завдань можна автоматизувати.

Наведемо їх нижче:

- перевірка та заміна вимірювального інструменту;
- щомісячні заявки на замовлення нового обладнання;
- відновлення періодичних сертифікатів;
- склади: списання;
- робота з рекламаціями.

Як ми бачимо, багато з цих завдань вочевидь можна автоматизувати, наприклад, замовлення нових приладів та обладнання (в залежності від кількості на складах) та пам'ятки з відновлення періодичних сертифікатів (налаштовуються на проміжок часу до закінчення строку дії сертифікату).

Також слід враховувати зручність користування системою, де всі дані зберігаються в одному місці - це значно полегшує та прискорює проходження будь-яких аудитів. Більш того, дуже часто аудит другої сторони закінчується тим, що аудитор інспектує деякі показники в системі та лишається тим задоволений: система діє автоматично, тому це гарантує послідовність та неперервність дій, що є дуже важливим критерієм якості.

Але є деякі сфери діяльності, автоматизування яких не є настільки очевидним. Наприклад, робота за рекламаціями майже завжди передбачає збирання та обробку даних в межах досліджень, і популярними методами це вже

автоматизувати неможливо, адже дослідницька робота завжди передбачає елементи творчості. Треба дослідити цю галузь детальніше.

Отже, традиційно робота з рекамаціями передбачає наступні етапи:

- подання рекамації клієнтом;
- отримання рекамації відповідальною особою;
- зворотній зв'язок клієнту про отримання рекамації та прийняття в роботу;
- збирання даних;
- відновлення картини, дослідження випадку;
- приймання рішення та інформування клієнта про рішення;
- призначення коригуючих дій та відповідальних осіб;
- визначення матеріальних зобов'язань сторін та їх впровадження;
- закриття рекамації та інформування сторін.

В даному випадку, отримання рекамації, надання зворотного зв'язку клієнтам та (частково) призначення відповідальних сторін можна автоматизувати. Ще кращим є те, що всю інформацію стосовно рекамації можна зберігати в одному місці, не витрачаючи час на оформлення документації (що дозволяє ISO 9001:2015).

За потребою, можна отримати всю інформацію у вигляді звіту, який можна використовувати для офіційного обігу документації. Наприклад, як шаблон можна використати доволі зручний інструмент, розроблений Г. Фордом – "8D звіт" [3]. Ця форма звітності прийнята як стандарт якості на багатьох підприємствах провідних країн світу.

Також слід відмітити, що рекамації прийнято розподіляти по типах. Найчастіше це бувають рекамації зовнішні (від клієнтів), внутрішні (між підрозділами підприємства), транспортні, рекамації постачальникам, тощо. Розподіляють їх тому, що кожна має свою послідовність дій, а це – ще один спосіб автоматизувати їх ведення.

Наприклад, транспортна рекамація містить записи про час, обсяг та якість (результат) завантаження / вивантаження, а також реквізити сторін. Основою цього

типу рекламаций може бути товаро-транспортна накладна, яка зазвичай і містить усі потрібні дані. Вона же є й офіційним документом, достатнім для документообігу.

Внутрішня рекламація є аналогічною до зовнішньої, але дещо спрощена – частина даних заздалегідь відома усім та незмінна. На її супровід найчастіше витрачається набагато менше часу, а її документальне оформлення може реалізувати в собі "4D звіт" за авторством того ж Г. Форда.

Ще один аспект діяльності відділу якості – це опис, регламентація та впровадження виробничих процедур та процесів підприємства. І хоча ISO 9001:2015 не зобов'язує описувати всі процеси, практика доводить, що наявність та доступність цих описів значно спрощує взаємодію підрозділів підприємства, робить більш чіткими та прозорими процеси ні підприємстві, а співробітникам дозволяє значно скоріше адаптуватися на новому місці.

1.2 Виявлення проблем та актуалізація рішень

Проведемо аналіз цільової аудиторії.

В малих підприємствах рекламації оброблюються здебільшого вручну, часто просто в телефонному режимі. Кількість рекламаций або незначна, або їх зміст заздалегідь відомий та порядок дій відпрацьований. В такому випадку, необхідності для ведення запису та процедур немає. Винятком є малі підприємства, діяльність яких може бути пов'язана, наприклад, з проектуванням, невірне виконання якого призводить до великих збитків, значно перевищуючих вартість самих робіт з проектування. Виконавець зі свого боку хоче бути захищений, тому документує все, що може бути пов'язане із робочим процесом.

В середніх підприємствах отримання рекламації відбувається через менеджера відділу продаж та в ручному режимі передається менеджеру з якості. Далі також в ручному режимі оброблюється та завершується. Та якщо для

порівняно невеликих підприємств (або невеликої кількості звернень по рекламаціях) це є нормальним, то зі збільшенням обсягу можуть трапитись деякі колізії, пов'язані з банальним людським фактором, наприклад: забув, не встиг, недоотримав дані, не втримав на контролі і т. ін. Це вже є неприпустимим, адже рекламації – це питання найвищого пріоритету. В цих випадках необхідно використовувати інструменти, що забезпечують втримання справи на контролі та її доведення до кінця.

Великі підприємства теж мають свої особливості. По-перше, майже всі вони вже використовують системи документообігу чи навіть повного циклу виробництва. Це величезні, багатофункціональні, з високим рівнем абстракції та дуже гнучкі в налаштуваннях "комбайни", які потребують значних зусиль та знань для адаптації та запуску під конкретне підприємство. Дуже часто в таких системах вже закладено деякі інструменти керування системою якості. Не всі вони є зручним, але здебільшого, покривають всі потреби підприємства.

Підсумовуючи все, наша головна цільова аудиторія – це середні за розмірами підприємства, що мають потребу в автоматизації процесів, пов'язаних з якістю, але ще не прив'язані до якоїсь конкретної системи управління підприємством. Такі підприємства не хочуть використовувати великі та потужні бізнес-рішення через їх велику вартість та складність налаштування. Якщо для великих компаній використання таких комплексних систем – це необхідність, то для середніх та малих – невиправдана витрата ресурсів та ускладнення процесу. До того ж, дуже часто ці рішення є комплексними во всіх розуміннях, і через систему необхідно пускати абсолютну більшість процесів, інакше вона не працюватиме, або не буде давати очікуваний результат.

Також, з точки зору невеликих підприємств, ці системи є занадто ускладненими. Це є наслідок гнучкості та високого рівня абстракції.

Розглянемо приклад такої системи, в якості якого виступатиме дуже розповсюджена в державах Європи (зокрема, Скандинавії та Балтики) система шведських розробників Monitor ERP [4]:



Рисунок 1.1 – Комплексна система керування підприємством "Monitor"

Для відображення цікавих для нас розділів Системи ми повинні перейти безпосередньо до вікна створення екземпляру рекламації.

В цій системі рекламація заповнюється на основі існуючого в межах системи замовлення, тому спробуємо проаналізувати пусте діалогове вікно, що зображено на рисунку 2.2.

Як ми бачимо, ця форма внаслідок своєї універсальності є занадто складною та надлишково інформативною. А це означає, що вона не виконує одну з важливіших покладених на неї функцій: не прискорює процес. Те ж саме відбувається і з наступними процесами – вони оптимізовані під максимальну кількість типів задач для великих підприємств.

Більше того, щоб зберегти інформацію в цій формі, необхідно вже заздалегідь провести міні-збір інформації, інакше логіка форми не дозволить зберегти дані.

А це значить, що до закінчення етапу збору треба все тримати в голові або записнику, що знов таки ставить під сумнів доцільність такого підходу для застосування малим та середнім підприємствам.

The screenshot shows a software interface for registering complaints. The title bar reads 'Registruoti neatitiktį'. The main area is divided into several sections:

- Top Bar:** Includes 'Reklamac.Nr.', 'Rekl. tipas:' (set to 'Kliento reklamacija'), and 'Atsakingas:' (set to 'ADMIN').
- Navigation Bar:** Contains icons and labels for 'Registruoti neatitiktį', 'Veiksmai', 'Kaštai', 'Patvirtinimas', 'Atsakym.', and 'Neatitikties Aktas'.
- Artikulas Section:** Fields for 'Artik. Nr.', 'Revizija', 'Neatitikties kodas', 'Neat. kiekis', 'Partijos Nr.', 'Prist. kiek.', 'Užsakymo nr.', 'Operacija', 'Projektas', 'Gamyb.data', and 'Neatit.data' (set to '5/13/2020').
- Kliento/tiekėjo informacija Section:** Fields for 'Jūsų Neatitikt.nr.', 'Užs. Nr.', 'Jūsų atstovas:', and 'El. paštas'.
- Pagrindinis artikulas Section:** Fields for 'Užs. Nr.', 'Gamybos data', 'Operacija:', and 'Proj.:'.
- Klaidinga priežastis Section:** Fields for 'B-kodas:', 'Tiekėjas:', 'DC:', 'Pad:', and 'Priežasties art'.
- Derinimas/Gaminimas Section:** A table with columns 'Eil.', 'Operacija', and 'G-užs. Nr.'. The first row shows '1' in the 'Eil.' column.
- Konstatuota Section:** Fields for 'B-kodas:' and 'B-kiekis:'.
- Išvairūs Section:** Fields for 'Neatikt. kat.:', 'Pradinis s.f. nr.', 'Subst.order:', 'Lydraštis:', 'Kred. s.f.nr.', 'Nauja s.f.:', 'Vieta:', 'Kalba:', 'Stat.:', and 'Priorit.:' (set to '9').
- Buttons:** 'Vidinė pastaba', 'Išorinė pastaba', 'Paveikti likučius...', 'Įrašų žurnalas..', 'Rodyti kaštus', 'Sukūrė...', 'Pakeista...', and 'Komentaras...'.

Рисунок 1.2 – Діалогове вікно реєстрації рекламації в системі Monitor

В контексті вищенаведеного, для цільової аудиторії найкраще підійшла би більш спрощена форма з мінімумом пре-реквізитів. А ще краще, в наступному автоматизувати подачу рекламаций та делегувати цей процес на сторону клієнта, щоб він міг надати необхідну інформацію заздалегідь.

Також, ця система повинна працювати з мінімумом налаштувань та якомога автономніше, незалежно від інших можливих модулів (таких як бухгалтерія, склади, закупівлі, продажі, CRM-системи та ін.)

Також, слід відмітити, що правильним рішенням було би забезпечення модульності цього продукту.

Та досить часто відбувається так, що підприємство не хоче прив'язуватися до великого комерційного продукту, але обсяги виробництва вже вимагають додаткової роботи з даними. В таких підприємствах ще з давніх давен для зберігання даних та елементарних обчислень використовують табличні процесори, найчастіше – MS Excel. Ці таблиці згодом зростають, ускладнюються, наповнюються даними та часто – помилками. Інколи, в залежності від наявності більш компетентного персоналу, в таблицях з'являються VBA скрипти, а потім – як наслідок, – форми.

```
Sub ImportItems()

    Dim i As Integer, j As Integer, k As Integer, s As Integer
    ioffset = ThisWorkbook.Sheets("inventura").Cells(1, 1).Value
    ion = ThisWorkbook.Sheets("inventura").Cells(2, 1).Value + ioffset - 1
    joffset = ThisWorkbook.Sheets("Source").Cells(1, 1).Value
    jon = ThisWorkbook.Sheets("Source").Cells(2, 1).Value + joffset - 1

    For i = ioffset To ion
        For j = joffset To jon
            If ThisWorkbook.Sheets("inventura").Cells(i, 2).Value = ThisWorkbook.Sheets("Source").Cells(j, 6).Value Then
                ThisWorkbook.Sheets("Source").Cells(j, 6).Value = ThisWorkbook.Sheets("Source").Cells(j, 6).Value + 1
                ThisWorkbook.Sheets("inventura").Cells(i, 5).Value = "OK"
            Else
                ThisWorkbook.Sheets("inventura").Cells(i, 5).Value = "Not Found"
            End If
        Next j
    Next i

End Sub
```

Рисунок 1.3 – VBA-скрипт внесення результатів інвентаризації в систему

Як наслідок, згодом десятки окремих таблиць об'єднуються в одну систему, а як відомо, із підвищенням складності системи падає показник її надійності [5].

Річ в тім, що здебільшого жоден з цих файлів не створювався для роботи в системі. В них є величезна кількість слабких місць та вразливостей, усі файли створені особами з кардинально різним навиком роботи з процесором та різними уявленнями не тільки про спосіб отримання бажаного результату, а й про сам результат.

Також існує проблема паралельного доступу до даних: дуже часто функція паралельного доступу до файлу (share workbook) конфліктує з виконанням VBA-

скриптів, тому для їх запуску потрібно відключити паралельний доступ, виконати скрипт, а потім знову роздати доступ для паралельної роботи у файлі. При цьому, якщо в системі декілька робочих файлів та один з них відкрито кимось іншим, виконання скрипту може перерватися, що в кращому випадку загрожує втратою опрацьовуваних даних, в гіршому – втратою чи псуванням всього файлу.

Подивимось на приклад форми реєстрації невідповідності в такій системі.

Рисунок 1.4 – Форма реєстрації невідповідності в таблицях Excel

Як ми бачимо, ця форма вже менш навантажена непотрібними в даному випадку подробицями, але брак функціоналу таблиць та ненадійність систем в цілому залишається.

Ціль проектування – розробити систему, яка водночас буде нескладною в обслуговуванні та досить функціональною. Окреслимо задачу детальніше.

1.3 Постановка задачі

Раніше ми визначили цільову аудиторію для розробки ПЗ, а саме - середні за розміром підприємства, які не мають змоги або бажання переходити на великі комерційні комплексні системи, та вже не мають можливості керувати процесами вручну. Для них насамперед важливо отримати від ПЗ наступні можливості:

- автоматизація усіх можливих рутинних завдань;
- підвищення швидкості внесення та обробки запитів;
- агрегація усіх даних в одному місці;
- швидке розгортання та налаштування системи;
- автономність та незалежність від інших модулів/сервісів;
- якомога менше займатися підтримкою системи, по можливості взагалі передати на аутсорсинг;
- можливість масштабувати систему та доповнювати іншими модулями за необхідності.

Також слід врахувати різні платформи, на яких клієнт може забажати використовувати ПЗ, іншими словами, воно також повинне бути кросплатформовим.

Виходячи з вищенаведеного, найбільше для цього проекту підійде ASP.NET Core 3.1 [6] (це актуальна на середину 2020р. версія) з MVC [7], [8] підходом до розробки. Буде зручно розташувати таку систему в хмарі Azure Cloud.

2 ФОРМУВАННЯ ВИМОГ ДО ПЗ

2.1 Вимоги до оточення

Обладнання. Клієнт – будь-який девайс, здатний відобразити веб-сторінку та маючий засоби введення текстової інформації та взаємодії з елементами. Сервер: хмара, найбільш імовірно, Azure cloud.

Програмне оточення. Браузер: IE 8.0+, Firefox 21.0+, Chrome 27.0+, з дозволом на використанням cookies. Сервер: підтримка ASP.NET MVC Core 3.1, MS SQL Server.

З'єднання. Клієнт спілкується із сервером за протоколом TCP (http/https), але для завантаження медіаконтента можливе використання UDP.

2.2 Функціональні вимоги

Авторизація / реєстрація користувача. Користувач має доступ до всіх функцій ПЗ тільки після успішної авторизації. Після реєстрації усі користувачі набувають роль клієнта, яка надає обмежені права (оформлення рекламаций). Ролі з більшими правами надаються вручну адміністратором системи.

Заповнення рекламаций. Кожен авторизований користувач має право сформулювати та надіслати рекламацию.

Робота з рекламациями. Відповідальна за роботу з рекламациями особа може призначати завдання та відповідальних за їх виконання осіб. Відповідальні особи в той же час мають можливість зробити звіт з роботи (у вигляді коментаря з медіаконтентом), який підтверджує або відправляє на доопрацювання ініціатор завдання.

Робота з фінансами. В межах будь-якого завдання (чи рекламациї) можливо отримувати та виписувати рахунки-фактури для їх подальшої оплати та аналізу картини витрат на якість в цілому.

Робота із вимірювальними приладами та складами. Кожному типу вимірювального приладу надається період повірки, ближче до закінчення якого періодично (раз на місяць) формується звіт з планом повірок на наступний місяць. Кожний екземпляр вимірювального приладу набуває інвентарний номер та призначається до використання/зберігання конкретним робітником. Під час повірки та в разі її не проходження, формується звіт на списання приладів, вони утилізуються по документах та зініціюється перевірка наявності цих приладів в достатній кількості на складах. У разі відсутності / недостатньої наявності приладів на складах, формується звіт на їх придбання. Звіт повинен формуватися, виходячи з динаміки використання / списання в межах останнього року та з урахуванням щомісячних тенденцій.

Робота з сертифікатами. Усі сертифікати, їх продовження, перевидання та модифікації, а також строки дії повинні бути введені в систему. Система повинна заздалегідь сповіщати про закінчення строку дії сертифіката його володаря, керівника володаря та менеджера з якості.

Статистика. Можливе виведення звітів кількості рекламаций по підрозділах, клієнтах, типах витрат та ін., а також по сумах на компенсації. Такі ж звіти потрібні по вимірювальних приладах.

2.3 Нефункціональні вимоги

Архітектура. Продукт повинен бути модульним. Підключення додаткового модуля не повинен впливати на працездатність системи в цілому.

Доступність. Даний програмний продукт повинен бути кросплатформовим, із базою даних, як головним сховищем інформації. Програмну частину не

обов'язково ділити на клієнта та сервер. Продукт повинен бути доступний цілодобово, сім днів на тиждень, але основне навантаження на нього очікується в робочі часи.

Безпека. ПЗ повинно містити модуль авторизації на не повинне надавати доступу незареєстрованим користувачам. З'єднання для передачі даних повинне бути шифрованим. Всі помилки повинні виловлюватись та записуватись в лог. Критичні помилки, що впливають на працездатність, відображаються клієнту.

Обслуговування. ПЗ найкраще розмістити в хмарах, де воно буде автоматично обслуговуватись без втручання клієнта.

Сховище даних. Уся інформація та посилання на медіаконтент зберігається в базі даних, сам медіаконтент – на файловому сховищі. Також БД повинна підтримувати використання процедур та тригерів (для формування автоматичних звітів). Резервне копіювання БД – раз на добу, медіаконтента – раз на тиждень.

Швидкодія. Якихось специфічних вимог до швидкодії нема, але для комфортної роботи час відгуку системи не повинен перевищувати логічного порогу, наприклад, в одну секунду.

2.4 Можливі обмеження розробки

Недостатній час на проектування. Проектування може проводитись етапами (спрінтами), в яких поступово будуть додаватися нові модулі та покращуватися вже імплементовані.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПЗ

3.1 Загальні відомості

Як вже було наведено раніше, інформаційна система розроблюється для запуску на будь-яких пристроях, що здатні забезпечити відображення веб-контенту та інтерактивну взаємодію з ним. Система має бути створена за допомогою кросплатформової технології від Microsoft ASP.NET Core 3.1 та із застосуванням шаблону проектування MVC.

Розробка цього продукту розділена на спрінти, тому впродовж першого спрінта частина функціоналу не буде реалізована в класах, але для зменшення вірогідності колізій під час додаткових міграцій, база даних буде побудована одразу максимально повною.

3.2 Проектування архітектури ПЗ

Обрана технологія передбачає використання одного з двох підходів проектування: Razor pages [9] та MVC. Паттерн проектування MVC передбачає поділ системи на три взаємодіючі частини: модель даних (що взаємодіє з БД), подання (відображення даних користувачеві) та контролер (інтерпретація дій користувача). Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (подання) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Перевага використання саме цього шаблону – гнучкий дизайн програмного забезпечення, який забезпечить модульність продукту та повинен полегшувати його доопрацювання чи розширення.

Тепер створимо діаграми послідовності для прояснення деяких неочевидних взаємодій, таких як супровід рекламцій або розрахунок необхідного обсягу закупівлі вимірювального інструмента.

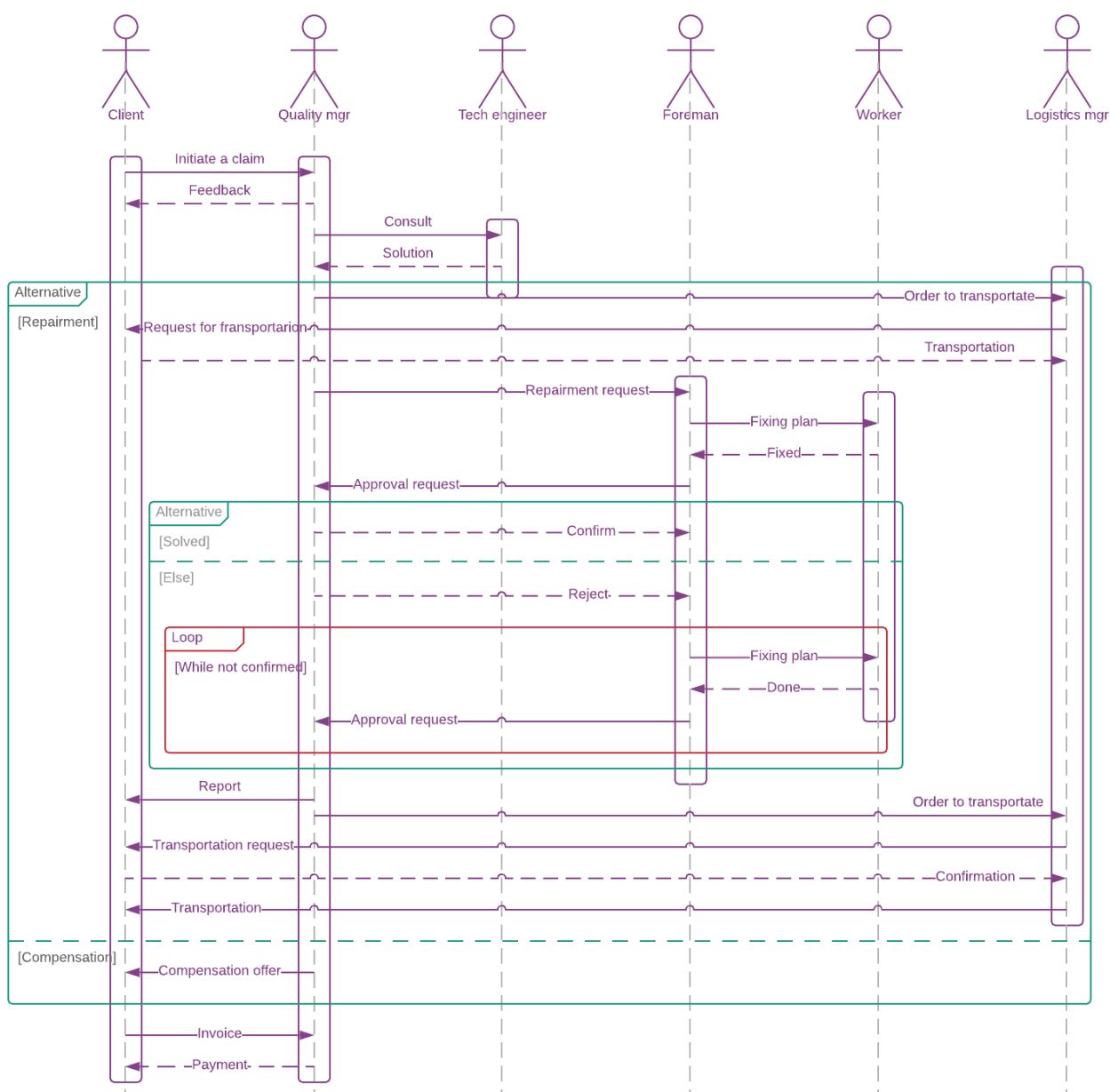


Рисунок 3.2 – Діаграма послідовностей сутності «рекламація»

Із цієї діаграми ми вже бачимо, як вимальовується майбутній клас, – одна із ключових сутностей в системі. Вже бачимо два оператора розгалуження та один вкладений цикл.

Тепер задля того, щоб розуміти, що діється із рекламацією, як абстрактною сутністю, побудуємо діаграму станів.

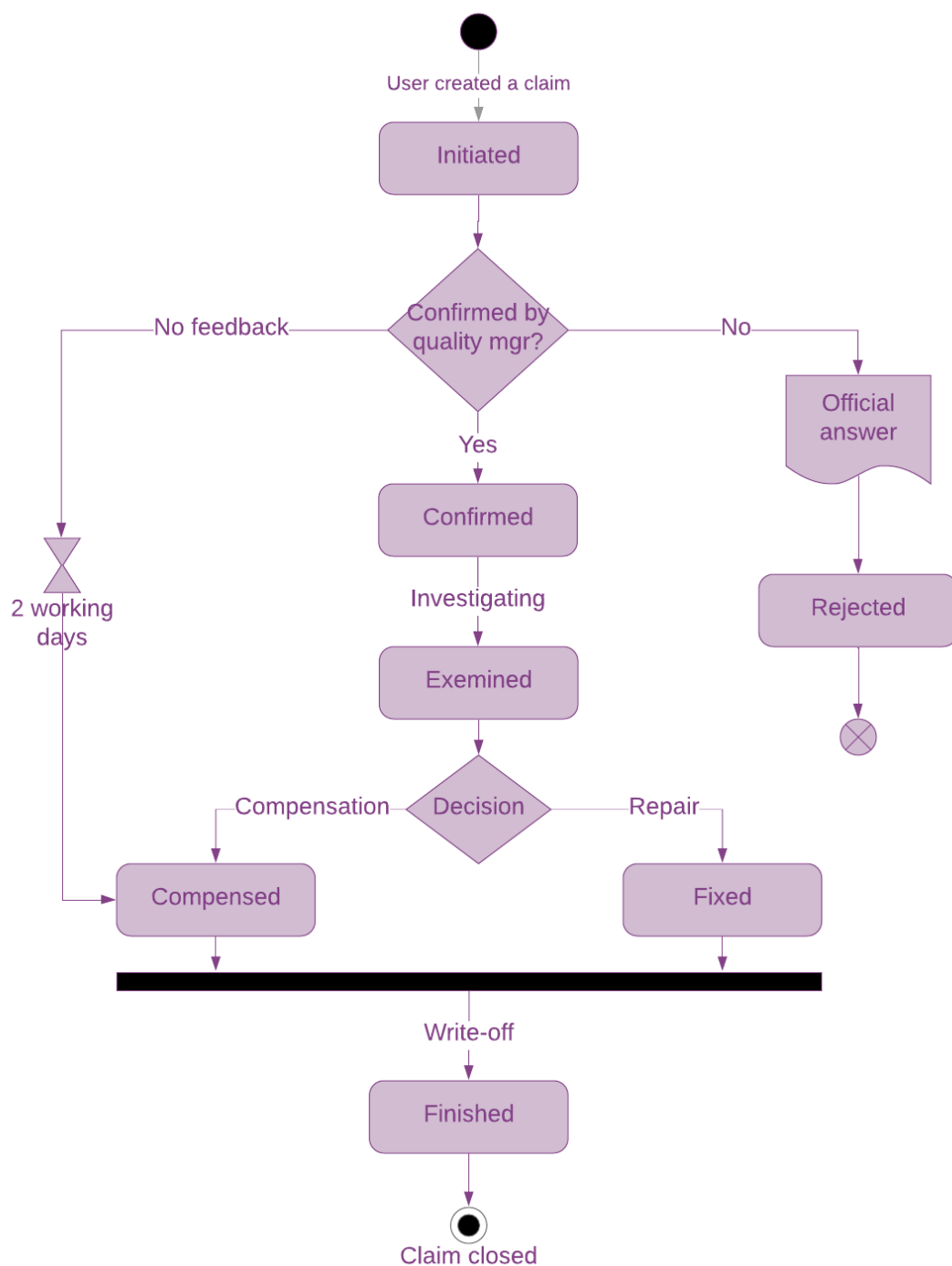


Рисунок 3.3 – Діаграма станів для сутності «рекламація»

Тепер вже можна створювати клас реклаमाції. Але перш за все, кожна дія в межах системи виконується учасниками над замовленням, тож є сенс створити базові абстрактні класи для замовлення та учасника дії.

А далі, виходячи з концепції MVC, створюємо класи по сутностям в базі даних, одна таблиця – один клас-модель [10]. Отже, час перейти до проектування бази даних.

3.3 Проектування системи зберігання даних

Приймаючи до уваги особливості розробки систем за патерном MVC, відобразимо структуру моделей в базі даних. Як вже було сказано вище, базу даних бажано проектувати з урахуванням подальших розширень функціоналу продукту, враховуючи навіть ті можливості, що поки що не передбачаються ані розробниками, ані замовниками. В крайньому випадку, невикористані таблиці можна буде видалити, і це буде набагато легше та дешевше, ніж робити міграцію для додавання нових сутностей в базу даних.

Перш за все, вносимо в базу сутності, що відповідають за опис та структуру підприємства. Це таблиця підрозділів, яка спирається на таблиці складів та таблицю внутрішніх віртуальних рахунків підрозділів підприємства. До неї також підключені таблиці штатного розпису та (опціонально) фізичних осіб. Окрім вищеназваних, є ще дві таблиці, що закладені на майбутнє: матриця компетенцій (підключається до списку фізичних осіб) та шаблон компетенцій (підключається до списку департаментів).

Ці таблиці в теорії повинні відповідати за нарахування надбавок до заробітної платні робітникам. В матрицю компетенцій вписують рівень усіх компетенцій робітника, такі, як володіння мовами, освіта, опит роботи, навички читання креслень, робочі навички, як то робота на кранах, користування завантажувачем- "автокарою" або зварювання (в конкретних ступенях дозволу, наприклад, EXC2 або EXC3).

Шаблон компетенцій, в свою чергу, окреслює важливість або не суттєвість конкретних навиків для роботи в конкретному підрозділі, тому підключаємо його до таблиці з підрозділами.

Ми окреслили базовий набір таблиць, які необхідні для функціонування системи в цілому. І вже до цього набору ми будемо підключати інші таблиці, які необхідні для реалізації завдань безпосередньо системи управління якістю.

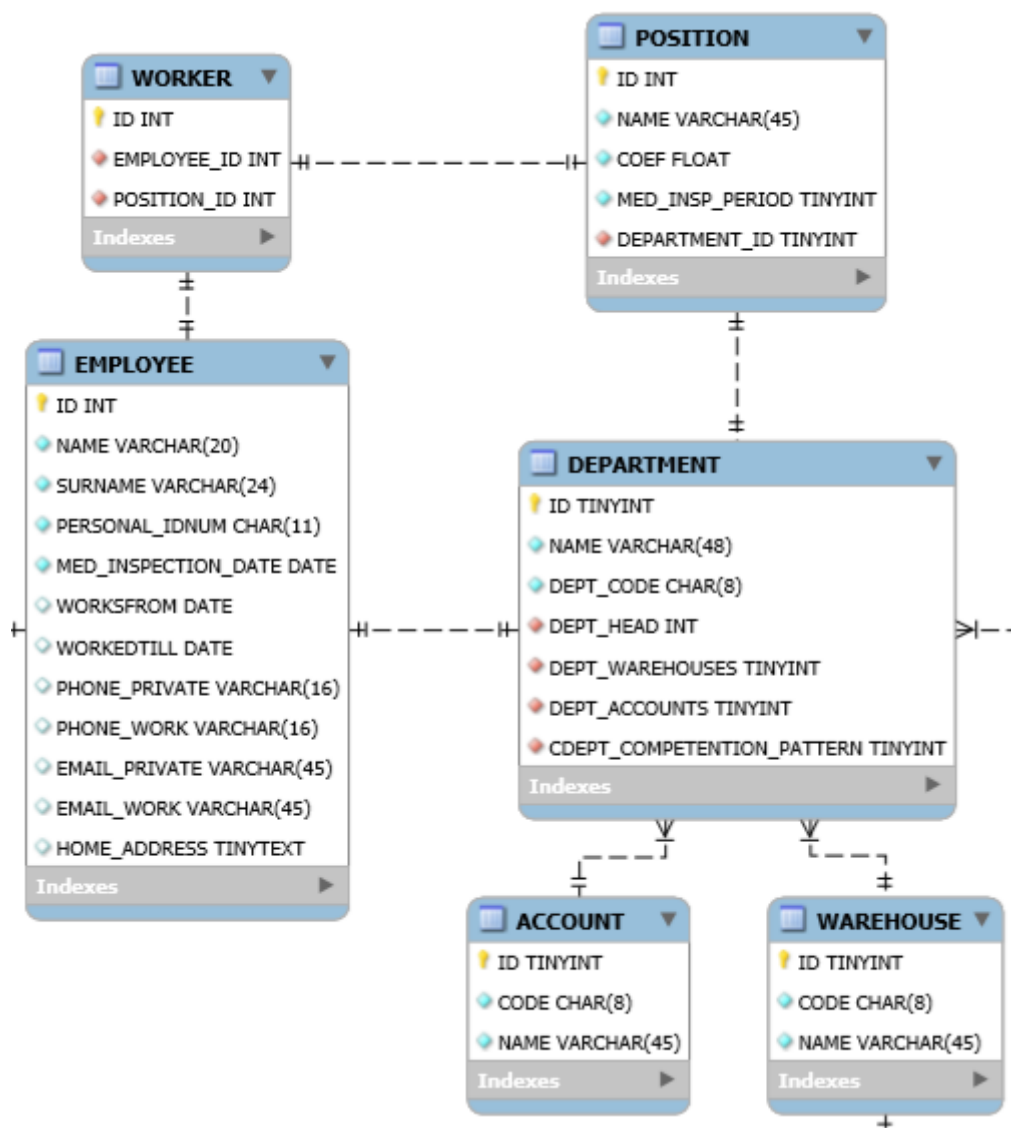


Рисунок 3.4 – Базовий набір таблиць БД підприємства

відобразимо таблицю замовлення, як одну з ключових сутностей. Ця таблиця базується на таблиці типів замовлень, таблиці асортименту продукції, таблиці зі списком клієнтів та має ще декілька полів: дата замовлення, дата відвантаження, ціна замовлення, коментар.

Загалом цей вузол з усіма локальними прямими зв'язками показаний на рисунку нижче:

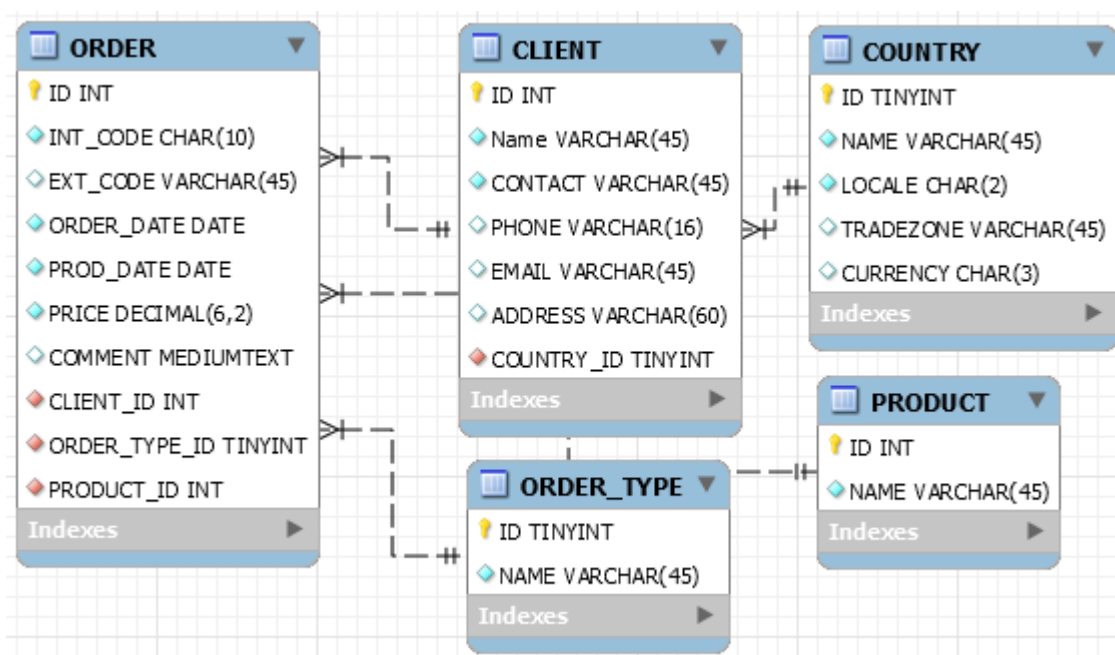


Рисунок 3.5 – Вузол опису замовлення

На підставі цього вузла будемо будувати все, що стосується рекламаций. Для цього створюємо наступні таблиці з ідентифікатором замовлення в якості зовнішнього ключа: картка якості, рекламація.

Тут слід розуміти, що на одне замовлення може бути відкрито декілька рекламаций. Також слід відмітити, що об'єкт замовлення може змінювати власника без відома виробника, внаслідок чого подавець рекламації не обов'язково повинен співпадати з початковим замовником, отже мусимо додавати дані до таблиці з рекламациями поле «клієнт» повторно. Гарним кроком було би автоматичне заповнення форми діалогового вікна значенням початкового замовника, але з можливістю корекції.

Почнемо роботу над таблицею рекламаций. Для кожної рекламації ми хочемо мати історію дій та коментарів. Ці дії повинні включати в себе сформульовану ціль, відповідальну за дію особу, дату початку та дату закінчення, а також результат у вигляді коментарю. Відповідальну особу підключаємо із таблиці з працівниками, решта даних оформлюємо просто полями.

характеристик інструмента та базовану на них таблицю екземплярів інструментів. Також до останньої буде підключено таблицю робітників (власники / зберігачі). Інтервал перевірки буде записано в таблиці типу інструмента.

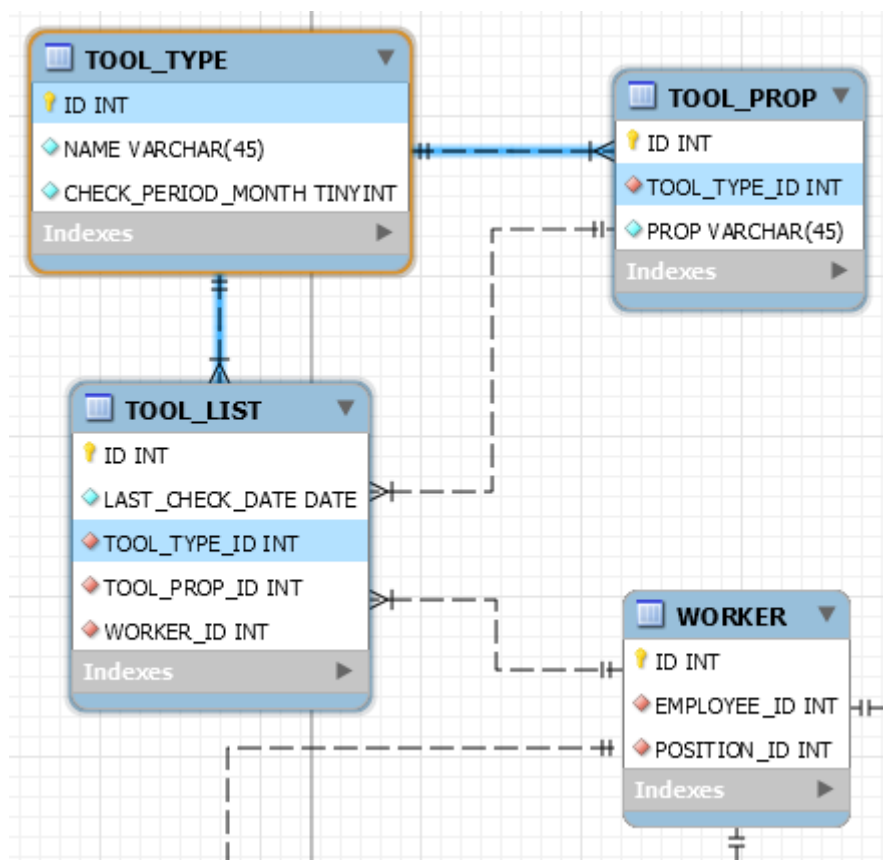


Рисунок 3.7 – Блок таблиць використання вимірювальних приладів

За тим же самим принципом створимо таблицю сертифікатів:

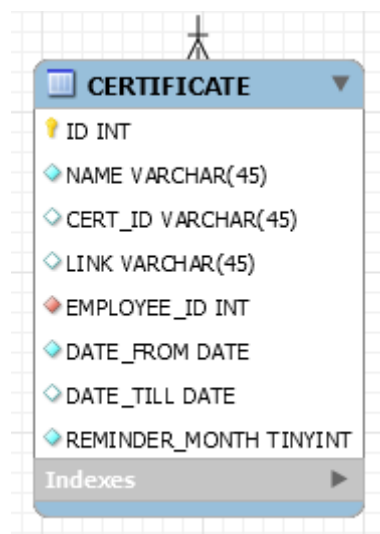


Рисунок 3.8 – Таблиця сертифікатів

Для сертифікатів обмежимося однією таблицею з вписаним зовнішнім ключем фізичної особи (адже сертифікати частіше бувають іменні).

3.4 Створення дизайну системи

Інтерфейс користувача повинен бути чутливим (responsive), надавати максимально можливий обсяг для робочого простору, бути спокійним (не перевантажувати погляд) та обов'язково бути інтуїтивно зрозумілим. Саме тому була обрана класична компоновка, яку можна зустріти в настільних додатках та системах: колонка інструментів зліва та велике робоче поле правіше від нього.

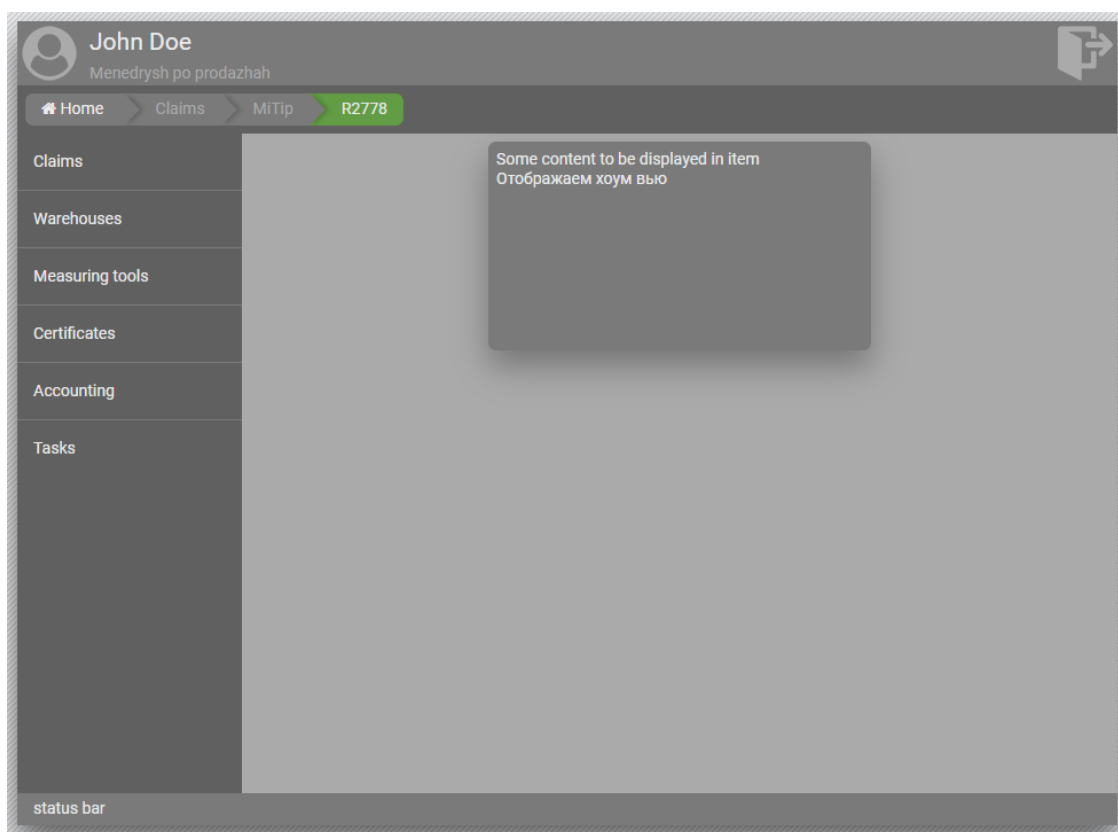


Рисунок 3.9 – компоновка макету додатку

Розробка якісного з точки зору UX/UI дизайну передбачено на другому спринті, коли буде створюватись кабінет клієнта та реалізовуватись можливість

самому користувачу оформлювати та направляти на розгляд, а також слідкувати за етапами вирішення рекламації. Скетчі дизайну проекту наведені в додатку Б.

Під час першого спрінту буде реалізовано тільки базовий інтерфейс взаємодії співробітників на базі фреймворку-бібліотеки Bootstrap [11] (яка підключається в якості NuGet пакету до проекту), а також за допомоги таких засобів розробки для фронт-енду, як пре-процесору таблиць стилів SCSS/Sass, пре-процесору розмітки Pug.js та таск-менеджеру Gulp.js, який інтерпретує код пре-процесорів, об'єднує модульні файли, мінімізує код та готує скомпільовані файли до публікації.

На даному етапі макет системи створено за допомогою флекс-боксів (flexbox), але в подальшому основний каркас системи та структури, відповідні за компоновку відносно параметрів екрану пристрою, було б доцільно зробити за допомогою CSS Grid. Однак, елементи навігації та відображення сутностей в робочому полі доцільно залишити в реалізації flexbox через дуже корисні в цьому випадку властивості “flex-grow”, “flex-wrap”, “flex-shrink” та ін.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Загальні відомості

Програмний продукт в цілому створений для автоматизації завдань та збільшення комфорту користувача, також для зниження виміг вхідного порогу для користувачів. Гарна система та, якою можна користуватися, не замислюючись над особливостями специфічного програмного оточення, над інтерфейсом чи "точкою входу" для користувача. Вона повинна бути простою в сприйнятті, доступною будь-де та будь-коли та не вимагати чогось специфічного від користувача.

4.2 Опис прийнятих програмних рішень

Система має клієнт-серверний вигляд, при чому клієнт – "тонкий", на основі HTML5 та CSS3 з додаванням за необхідності ES5 (в крайньому випадку – ES6 + Babel). Це надає змогу бути мобільним та гнучким – адже такий "тонкий" клієнт може бути запущений в будь-якому смартфоні майже десятирічної давнини, а по факту навіть більше, адже самий "новий" елемент такої розмітки – flexbox – має підтримку вже з жовтня 2012 року (для Opera Mobile 12.1) та часткову підтримку (з присутнім тут синтаксисом – цілком достатньо) з вересня того ж 2012 року для IE 10. Загалом підтримка цього синтаксису дорівнює 98,05%, що досить переконливо свідчить про актуальність технології [12].

Для прототипування клієнта було використано препроцесори SCSS (для каскадних таблиць) та Pug.js (для розмітки гіпертексту). Кожний логічний блок оформлення та розмітки створено окремим модулем (для забезпечення зручності підтримки коду), однак для зменшення обсягу передачі даних та кількості звернень до сервера для завантаження клієнта, використано конкатенацію файлів та їх мініфікація. Задачі конкатенації модулів, підключення бібліотек (в разі потреби),

компіляції коду препроцесорів та мініфікації файлів покладено на менеджер завдань Gulp.js. Це дуже зручний консольний інструмент з підтримкою конвейеру-пайплайну.

Для цього треба створити такий конвейер. Принцип роботи цього конвейеру наступний:

- збираємо список SCSS файлів, крім часткових (що починаються із символу підкреслення "_");
- компілюємо SCSS в класичний CSS;
- створюємо вендорні префікси;
- вивантажуємо в каталог наступного рівня (staging);
- збираємо всі не-часткові pug-файли та компілюємо їх в html;
- переносимо в каталог наступного рівня;
- збираємо всі необхідні js-бібліотеки та з'єднуємо їх в один окремий файл бібліотек;
- забираємо свої скрипти та додаємо їх до бібліотек;
- пережимаємо файл скриптів та вивантажуємо в каталог наступного рівня;
- створюємо завдання синхронізації браузера з каталогом наступного рівня;
- створюємо завдання-"шпигуна" (watch), який відслідковує зміни в файлах та автоматично, не перезавантажуючи сторінку браузера, "на льоту" змінює код сторінки та впроваджує відображення змін.

Також, окремо створимо завдання (build) для вивантаження повністю готового мініфікованого коду для клієнта. Нижче наведено код, що автоматизує всі ці завдання:

```
var gulp      = require('gulp'),
    sass      = require('gulp-sass'),
    browserSync = require('browser-sync'),
    concat    = require('gulp-concat'),
    uglify    = require('gulp-uglify'),
    cssnano   = require('gulp-cssnano'),
    rename    = require('gulp-rename'),
    del       = require('del'),
    imagemin  = require('gulp-imagemin'),
    pngquant  = require('imgemin-pngquant'),
    cache     = require('gulp-cache'),
```

```

    autoprefixer    = require('gulp-autoprefixer'),
    pug             = require('gulp-pug');
gulp.task('sass', function() {
  return gulp.src([
    'src/scss/*.+(sass|scss)',
    '!src/scss/_*.*'
  ])
    .pipe(sass())
    .pipe(autoprefixer(['last 15 versions', '> 1%', 'ie 8',
'ie 7'], { cascade: true } ))
    .pipe(gulp.dest('src/preprod/css'))
    .pipe(browserSync.reload({stream: true}))
  });
gulp.task('css-libs', ['sass'], function() {
  return gulp.src([
    'src/assets/libs/normalize.css/normalize.css',
    'src/assets/mods/**/*.css'
  ])
    .pipe(concat('libs.min.css'))
    .pipe(cssnano())
    .pipe(gulp.dest('src/preprod/css'));
});
gulp.task('pug', function() {
  return gulp.src([
    'src/pug/*.pug',
    '!src/pug/_*.*'
  ])
    .pipe(pug())
    .pipe(gulp.dest('src/preprod'))
  });
gulp.task('scripts', function() {
  return gulp.src([
    'src/assets/libs/jquery/dist/jquery.min.js'
  ])
    .pipe(concat('libs.min.js'))
    .pipe(uglify())
    .pipe(gulp.dest('src/preprod/js'));
});
gulp.task('browser-sync', function() {
  browserSync({
    server: {
      baseDir: 'src/preprod'
    },
    notify: false
  });
});
gulp.task('jayass', function(){
  return gulp.src([
    'src/js/*.js'
  ])
    .pipe(gulp.dest('src/preprod/js'));
});
gulp.task('imgs', function(){
  return gulp.src([
    'src/img/*.*'
  ])
    .pipe(gulp.dest('src/preprod/img'));
});

```

```

    gulp.task('watch', ['browser-sync', 'sass', 'css-libs', 'pug',
'scripts', 'jayass', 'imgs'], function() {
    gulp.watch('src/scss/*.+(scss|sass)', ['sass']);
    gulp.watch('src/pug/*.pug', ['pug']);
    gulp.watch('src/js/*.js', ['jayass']);
    gulp.watch('src/img/*..*', ['imgs']);
    gulp.watch('src/preprod/*.html', browserSync.reload);
    gulp.watch('src/preprod/js/*.js', browserSync.reload);
    gulp.watch('src/preprod/img/*..*', browserSync.reload);
    });
    gulp.task('clean', function() {
    return del.sync('public');
    });
    gulp.task('img', function() {
    return gulp.src('src/img/**/*.*)
        .pipe(cache(imagemin({
            interlaced: true,
            progressive: true,
            svgoPlugins: [{removeViewBox: false}],
            use: [pngquant()]
        })))
        .pipe(gulp.dest('public/img'));
    });
    gulp.task('build', ['clean', 'img', 'sass', 'pug', 'jayass',
'scripts'], function() {
    var buildCss = gulp.src([
        'src/preprod/css/libs.min.css',
        'src/preprod/css/styles.css',
        'src/preprod/css/svg.css'
    ])
        .pipe(concat('styles.css'))
        .pipe(gulp.dest('public/css'));

    var buildJs = gulp.src([
        'src/preprod/js/scripts.js'
    ])
        .pipe(concat('scripts.js'))
        .pipe(gulp.dest('public/js'));

    var buildHtml = gulp.src('src/preprod/*.html')
        .pipe(gulp.dest('public'));

    });
    gulp.task('clear', function () {
    return cache.clearAll();
    })
    gulp.task('default', ['watch']);

```

Відносна складність в налаштуванні роботи менеджера завдань Gulp повністю компенсується швидкістю та зручністю роботи: керування менеджером відбувається за допомоги двох команд: стеження (gulp watch) та кінцева компіляція (gulp build).

```

MINGW64:/d/NURE/Bachelor/04_program/Frontend_GULP
Owl@Codebay MINGW64 /d/NURE/Bachelor/04_program/Frontend_GULP (master)
$ gulp watch
[23:22:14] Using gulpfile D:\NURE\Bachelor\04_program\Frontend_GULP\gulpfile.js
[23:22:14] Starting 'browser-sync'...
[23:22:14] Finished 'browser-sync' after 683 ms
[23:22:14] Starting 'sass'...
[23:22:14] Starting 'pug'...
[23:22:14] Starting 'jayass'...
[23:22:14] Starting 'imgs'...
[23:22:14] Finished 'jayass' after 28 ms
[23:22:15] Finished 'pug' after 94 ms
[23:22:15] Finished 'imgs' after 107 ms
[Browsersync] 2 files changed (styles.css, svg.css)
[23:22:15] Finished 'sass' after 367 ms
[23:22:15] Starting 'watch'...
[23:22:15] Finished 'watch' after 20 ms
[Browsersync] Access URLs:
-----
    Local: http://localhost:3000
    External: http://192.168.56.1:3000
-----
    UI: http://localhost:3001
    UI External: http://localhost:3001
-----
[Browsersync] Serving files from: src/preprod

```

Рисунок 4.1 – Робота скрипту в таск-менеджері Gulp

Дуже корисною функцією Gulp є можливість транслювати скомпільовану сторінку в межах локальної мережі.

Це робиться для швидкої перевірки працездатності макету на різних пристроях. Для цього слід перейти за вказаною зовнішньою адресою, в цьому випадку – 192.168.56.1:3000.

Для створення серверної частини було обрано кросплатформну технологію ASP.NET Core актуальної на сьогодні (середина 2020 року) версії: 3.1.4.

Додаток будується за допомогою підходу MVC, а також із використанням СКБД MS SQL Server, що дозволяє скористатися також перевагою такої технології, як Entity Framework Core. Ця технологія дбає про зв'язок логіки та даних, розділюючи їх, та дозволяє розробнику звертатися до бази даних, як до звичайних поля/властивості класу. Це значно спрощує написання коду бізнес-логіки, його обслуговування та розуміння, але, як і в випадку з front-end, вимагає написання додаткового коду для налаштування. Як вже говорилося раніше, кожна сутність-

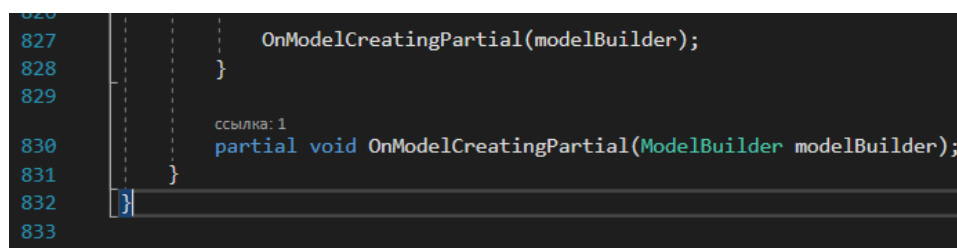
таблиця бази даних при застосуванні цього підходу описується спеціальним класом моделі, а взаємодія цих класів із базою даних описується в т.з. контекстному класі.

Технологія Entity Framework Core надає можливість застосувати два підходи для забезпечення взаємодії логіки та БД [13]: code-first (найпоширеніший підхід для нових рішень) та database-first (у випадку наявності БД). З огляду на те, що нами вже розроблена досить складна база даних, використаємо database-first підхід. Для цього застосуємо Scaffold-DbContext команду в сукупності з параметрами підключення до БД та налаштуваннями експорту сутностей. Результатом роботи скрипту є створені моделей сутностей БД, наприклад:

```
public partial class Product
{
    public Product()
    {
        Order = new HashSet<Order>();
    }
    public int Id { get; set; }
    public string Name { get; set; }
    public virtual ICollection<Order> Order { get; set; }
}
```

Також створено клас контексту для регламентації взаємодії коду з БД, і тут ми бачимо адекватність рішення про застосування database-first підходу. Файл містить більш 800 строк коду.

Загалом було створено 26 файлів з кодом. Однак такий код, на мою думку, ще вимагає доопрацювання. Наприклад, треба додатково визначити властивості комірок БД, як то [REQUIRED] для ключів-ідентифікаторів.



```
827         OnModelCreatingPartial(modelBuilder);
828     }
829
830     ссылка: 1
831     partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
832 }
833
```

Рисунок 4.2 – Результат роботи команди Scaffold-DbContext

З іншого боку, створення бази даних засобами EF Core із застосуванням code-first підходу є стандартом та однією з головних переваг використання цього фреймворку, тому реалізуємо код для створення необхідних для забезпечення роботи головної сутності (рекламація) таблиць.

Для цього створюємо класи відповідно до розроблених таблиць, а в класах пишемо властивості необхідного типу. Ключовими (первинними) таблицями будуть "Department", "Client", "ToolType". Вони міститимуть основні властивості та списки зв'язаних з ними сутностей.

```
public class ToolType
{
    public int ID { get; set; }
    public string Name { get; set; }
    public int CheckIntervalWeeks { get; set; }
    public ICollection<Tool> Tools { get; set; }
}

public class Department
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string? Account { get; set; }
    public string? Warehouse { get; set; }
    public ICollection<Employee> Employees { get; set; }
    public ICollection<ClaimExpense> ClaimExpenses { get; set; }
}

public class Client
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string? Country { get; set; }
    public string? Address { get; set; }
    public string? Email { get; set; }
    public string? Phone { get; set; }
    public ICollection<Claim> Claims { get; set; }
}
```

Далі на цих сутностях будуються залежні від них, наприклад, одна з основних, "Claim".

```
public class Claim
{
    public int ID { get; set; }
    public int ClientID { get; set; }
```

```

public string Title { get; set; }
public string? Description { get; set; }
public string? Dirpath { get; set; }
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}",
    ApplyFormatInEditMode = true)]
public DateTime DateStart { get; set; }
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}",
    ApplyFormatInEditMode = true)]
public DateTime? DateEnd { get; set; }

public Client Client { get; set; }
public ICollection<ClaimHistory> ClaimHistories { get; set; }
public ICollection<ClaimExpence> ClaimExpences { get; set; }
}

```

Тут ми бачимо зв'язок із сутністю "Client" через властивість ClientID та через створення навігаційної властивості "Client" типу "Client". Повний лістинг коду створення моделей сутностей наведемо в додатку В.

Для імплементції коду в базі даних, видаляємо всі автоматично створені раніше сутності та екземпляр бази даних та в консолі менеджера пакетів NuGet створюємо міграцію. Запускаємо створення бази даних командою update-database та отримуємо новий екземпляр бази даних із усіма залежностями та зв'язками.

Для перевірки вірності виконання коду та коректності створення БД створимо в оточенні Sql Server Management Studio (далі – SSMS) модель-представлення бази даних. Візуально перевіряємо зв'язки, поля та наявність таблиць.

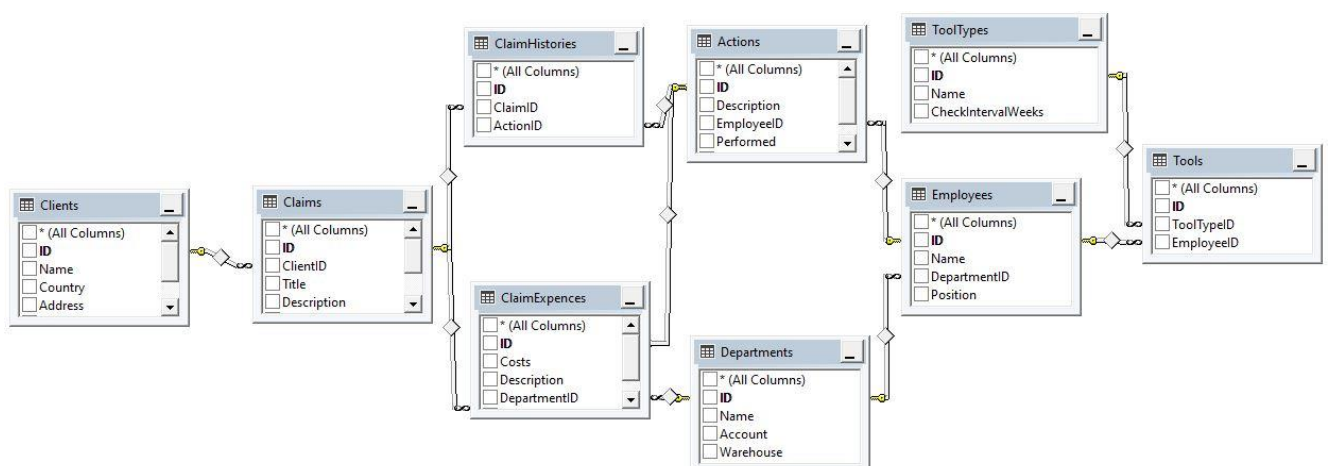


Рисунок 4.3 – Модель автоматично створеної за допомоги EF Core БД

Як бачимо, усі зовнішні ключі присутні, усі зв'язки окреслені саме такими, якими планувалися. Такої структури БД повинно вистачити на початковий опис сутностей "Claim" (рекламації та список пов'язаних завдань) та "Tools" (список виданих інструментів із інтервалами перевірки).

При подальшому можливому розширенні функціоналу треба буде створити нові сутності (що відповідатимуть новим таблицям БД) з навігаційними властивостями, а в старих пов'язаних сутностях створити списки посилань типу ICollection на нову сутність.

Наповнення інформацією первинних таблиць за задумкою повинно відбуватися в окремому розділі зі спеціальними правами доступу. Це має реалізуватись за допомогою нових просторів ("Area"), можливо, через панель адміністратора. На поточному етапі наповнюємо таблицю тестовою інформацією напряму через SSMS, за допомогою T-SQL запитів.

```
USE [Qualify]
GO

INSERT INTO Clients (Name, Country, Email)
VALUES
('MiTip', 'Danmark', 'info@mi.dk'),
('Konekesko', 'Lithuania', 'info@konekesko.lt'),
('Roltex', 'Poland', 'wszystko@roltexagro.pl'),
('Agricasa', 'Hungary', 'info@agricasa.hg'),
('Almex, D.o.o.', 'Serbia', 'info@almex.rs'),
('Agrimasz', 'Poland', 'info@agrimasz.pl'),
('OOO Umega', 'Russia', 'info@umegaagro.ru');
GO

INSERT INTO Departments (Name)
VALUES
('Welding bar'),
('Painting bar'),
('Assembling bar'),
('Preparation bar'),
('Technical dept'),
('Logistic dept'),
('Quality dept');
GO

INSERT INTO ToolTypes (Name, CheckIntervalWeeks)
VALUES
('Caliper', 108),
('Tape-measure', 12),
('Micrometer', 54),
('Try square', 54),
```

```

        ('Level', 54),
        ('Ruler', 108),
        ('Manometer', 32);
GO

INSERT INTO Employees (Name, DepartmentID, Position)
VALUES
    ('Mikolajus Aivazovskas', 2, 'Foreman'),
    ('Egidijus Suvirinevicius', 1, 'Foreman'),
    ('Михаил Кувалдин', 3, 'Foreman'),
    ('Микита Напильненко', 4, 'Foreman'),
    ('Justinas Bimbamskis', 5, 'Engineer'),
    ('Siuntyte Atvaziaite', 6, 'Manager'),
    ('Keturi Ziedai', 7, 'Auditor');
GO

```

Потрійний надпис "7 rows affected" запевнює нас в успішності запитів, але ми все ж перевіримо факт заповнення таблиць тестовими даними вбудованою функцією "показати першу тисячу строк". Результат очікувано позитивний.

Більшість інших таблиць вже повинні заповнюватись користувачем через фронт-енд. Для цього треба створити представлення (Views) та пов'язати їх з моделями через контролери. Для прикладу розглянемо представлення, пов'язані з головною сутністю – рекламацією. Згідно патерну MVC, ми створюємо один контролер для однієї логічно відокремленої сутності, в якому вже вписуємо методи дій (action methods) – згідно до діаграм Use Cases. І вже для кожного метода дії створюємо окреме представлення. Отже, створюємо контролер для сутності реклаमाцій. В нього за допомогою штатної (нарешті) функції ін'єкції залежностей (dependency injection) передаємо клас контексту БД. Тоді робимо асинхронні запити до БД: Index(), ViewClaim(), AddClaim(). Перший метод повертає нам список сутностей для відображення на головній сторінці розділу рекламацій. Решта – відповідно за перегляд однієї рекламації та за введення нової рекламації до системи.

```

public class ClaimController : Controller
{
    private readonly ClaimRepository _claimRepository = null;
    public ClaimController(ClaimRepository claimRepository)
    {
        _claimRepository = claimRepository;
    }

    public async Task<ViewResult> Index()

```

```

    {
        var data = await _claimRepository.GetAllClaims();
        return View(data);
    }

    public async Task<ViewResult> ViewClaim(int id)
    {
        var data = await _claimRepository.GetClaimById(id);
        ViewBag.ClaimId = id;
        return View(data);
    }

    public ViewResult AddClaim(bool isSuccess = false, int claimId = 0)
    {
        ViewBag.Success = isSuccess;
        ViewBag.ClaimId = claimId;
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> AddClaim(Claim claim)
    {
        int id = await _claimRepository.AddClaim(claim);
        if (id > 0)
        {
            return RedirectToAction(nameof(AddClaim), new {isSuccess =
true, bookId = id});
        }
        return View();
    }
}

```

Метод AddClaim тут перевантажений, останній (з запитом HttpPost) – відповідає безпосередньо за передачу заповнених даних до БД, передостанній – за стартове завантаження сторінки з запрошенням до вводу даних. При чому, якщо ця стартова сторінка визивається після введення реклаमाції (передача параметру IsSuccess через переадресування), на ній відображається повідомлення про вдале додавання інформації до БД. Також бачимо, що кожен метод тут є тільки обгорткою для логіки. Це зроблено для остаточного розділення логіки та контролера. Сама ж логіка винесена до окремого класу

```

public class ClaimRepository
{
    private readonly QualifyContext _context = null;
    public ClaimRepository(QualifyContext context)
    {
        _context = context;
    }
}

```

```

public async Task<int> AddClaim(Claim claim)
{
    var newClaim = new Claim()
    {
        ClientID = claim.ClientID,
        Title = claim.Title,
        Description = claim.Description,
        DateStart = claim.DateStart
        Dirpath = claim.ID.ToString()
    };
    await _context.Claims.AddAsync(newClaim);
    await _context.SaveChangesAsync();
    return newClaim.ID;
}

public async Task<List<Claim>> GetAllClaims()
{
    var claims = new List<Claim>();
    var allClaims = await _context.Claims.ToListAsync();
    if (allClaims?.Any() == true)
    {
        foreach (var claim in allClaims)
        {
            claims.Add(new Claim()
            {
                ID = claim.ID,
                ClientID = claim.ClientID,
                Title = claim.Title,
                Description = claim.Description,
                Dirpath = claim.Dirpath,
                DateStart = claim.DateStart,
                DateEnd = claim.DateEnd
            });
        }
    }
    return claims;
}

public async Task<Claim> GetClaimById(int id)
{
    var currentClaim = await _context.Claims.FindAsync(id);
    if (currentClaim != null)
    {
        var claimDetails = new Claim()
        {
            ID = currentClaim.ID,
            ClientID = currentClaim.ClientID,
            Title = currentClaim.Title,
            Description = currentClaim.Description,
            Dirpath = currentClaim.Dirpath,
            DateStart = currentClaim.DateStart,
            DateEnd = currentClaim.DateEnd
        };
        return claimDetails;
    }
    return null;
}

```

```

public List<Claim> FilterClaim(string title)
{
    Return DataSource().Where(x =>
x.Title.Contains(title)).ToList();
}
}

```

Тут наведено логіку взаємодії моделей з контекстом БД. Якщо контекст зміниться, нам потрібно буде змінити логіку саме тут, що спрощує підтримку коду. Створюємо такий репозиторій з логікою кожному контролеру, а результат роботи саме контролеру ми можемо побачити вже на прикладі коду представлень, отже, перейдемо безпосередньо до них.

Представлення створюються, як *.cshtml-файли з кодом функціонального модуля всередині. Як вже можна зрозуміти з розширення назви файлу, це суміш C# коду та html-розмітки. В межах технології Dot Net Core суміш цих синтаксисів називають “Razor syntax”. Для дотримання принципів «DRY» та зручної підтримки, для всіх представлень створюється загальний шаблон _Layout.cshtml, в який вже включаються необхідні в даний момент елементи.

```

<!DOCTYPE html>
<html lang="en">
<head>
    @await Html.PartialAsync("MetatagsPartial")
    @await Html.PartialAsync("CssPartial")
</head>
<body>
    <div id="wrapper">
        <div class="my-container" id="working-field">
            @await Html.PartialAsync("HeaderPartial")
            <main id="main">
                @await Html.PartialAsync("AsidePartial")
                <section id="content">
                    @RenderBody()
                </section>
            </main>
            @await Html.PartialAsync("FooterPartial")
        </div>
    </div>
    @await Html.PartialAsync("ScriptsPartial")
</body>
</html>

```

Як ми бачимо, логічні блоки підключаються асинхронним методом PartialAsync до загального шаблону. На підставі цього шаблону, підключаючи

потрібне наразі представлення директивою `@RenderBody()`, створюється кінцева сторінка, яка передається користувачеві. Тепер вже переходимо безпосередньо до наших представлень, створених на підставі методів дій відповідного контролера.

Нижче наведено код представлення для головної сторінки розділу рекламаций (зі списком рекламаций).

```
@model IEnumerable<Qualify.Models.Claim>
@{
    Layout = "_Layout";
}

<div class="minor-navigation">
    <a class="button btn-success"
        asp-controller="claim"
        asp-action="addclaim">Add a claim</a>
</div>

<div class="flex-scroll-container">
    @foreach (var claim in Model)
    {
        <a class="content-item"
            asp-controller="Claim"
            asp-action="ViewClaim"
            asp-route-id="@claim.ID">
            <h3>@claim.Title</h3>
            <p><strong>@claim.ClientID</strong>,
                <em>@claim.DateStart.ToString("yyyy-MM-dd")</em>
            </p><br />
            <p>@claim.Description</p>
        </a>
    }
</div>
```

Як бачимо, код являє собою звичайну розмітку з включенням блоків C# коду. У вищенаведеному прикладі поки що бачимо тільки сторінку з "табличками", що репрезентують екземпляри рекламаций (парсимо список екземплярів та виводимо в циклі). Дуже гарний приклад представлення – це виведення сторінки детального відображення конкретної рекламачії.

```
using Qualify.Models
@model Claim
@{
    ViewData["Title"] = "View claim";
}
<h1 class="form-heading">Claim @ViewBag.ClaimId details</h1>
<div class="content-container">
    <dl>
```

```

<dt>Claim title:</dt>
<dd>@Model.Title</dd>

<dt>Client:</dt>
<dd>@Model.ClientID</dd>

<dt>Description:</dt>
<dd>@Model.Description</dd>

<dt>Registered:</dt>
<dd>@Model.DateStart.ToString("yyyy-MM-dd")</dd>
@if (Model.DateEnd != null)
{
    <dt>Closed:</dt>
    <dd>@Model.DateEnd</dd>
}
else
{
    <dt>Closed:</dt>
    <dd>Not yet...</dd>
}
</dl>
</div>

```

В даному прикладі ми виводимо властивості об'єкту в список визначень. Останню властивість – дату закриття reklamacji – виводимо через умовний оператор. Це зроблено завдяки встановлення властивості "nullable" для властивості типу даних. Розглянемо ще один приклад: представлення для введення нової reklamacji:

```

@using Qualify.Models
@model Claim
@{
    ViewData["Title"] = "Add new claim";
}
<h1 class="form-heading">Add new claim</h1>
@if (ViewBag.Success == true)
{
    <div class="alert alert-success
        alert-dismissible fade show" role="alert">
        <strong>Oh yeah, alright.</strong>
        You added the claim successfully. <br />
        Click <a asp-controller="Claim"
            asp-action="ViewClaim"
            asp-route-id="@ViewBag.ClaimId">here</a>
        to view whar you've declared.
        <button type="button" class="close"
            data-dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
}

```

```

}

<form method="post" asp-action="AddClaim">
  <label asp-for="ClientID" class="form-label">Client</label>
  <input asp-for="ClientID" class="form-control" />

  <label asp-for="Title" class="form-label">Title</label>
  <input asp-for="Title" class="form-control" />

  <label asp-for="Description"
    class="form-label">Description</label>
  <textarea asp-for="Description"
    class="form-control"></textarea>

  <label asp-for="DateStart" class="form-label">
    Registration date</label>
  <input asp-for="DateStart" class="form-control" />

  <input type="submit" value="Register the claim"
    class="form-button button btn-success" />
</form>

```

Тут ми бачимо дві конструкції: внизу форма для відправлення даних, а зверху блок умовного оператора, який, у випадку отримання через ViewBag позитивного значення ідентифікатора створеної реклаमाції, показує користувачеві блок інформування про вдале створення запису в БД. Сам блок, як бачимо, імплементований методами Bootstrap.

Виконавши все вищевказане, можемо випробувати ці сторінки в дії. В разі, якщо тести покажуть позитивний результат, інші сутності створюються аналогічно. Перейдемо до тестів.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Задля того, щоб пересвідчитися в працездатності розроблюваного ПЗ, проводять тестування. У випадку клієнт-серверного додатку поперед всього необхідно провести функціональні тести бізнес-логіки та нефункціональне тестування інтерфейсу (клієнтської частини додатку).

Враховуючи те, що значна частина можливостей додатку на даному етапі не буде реалізована, та майже всі задачі є дуже схожими та однорідними, наведемо приклад тестування однієї з функцій. Також, враховуючи, що інтерфейс "тонкого" клієнта було доопрацьовано для сумісності з наступними етапами розробки (маємо деяку кількість елементів-"заглушок" для наступного розширення функціоналу), проведемо нефункціональне тестування інтерфейсу тільки в межах поставленої задачі.

5.1 Функціональне тестування

Функціональне тестування в межах цього проекту проведено, базуючись на діаграмах Use Cases. В ході тестування було перевірено більшість варіантів використання ПЗ в межах роботи, для кожного варіанту використання було створено окремий тестовий сценарій. В цілому, перевірено наступні функції:

- запуск клієнта;
- відображення даних;
- додавання даних;
- видалення даних;
- модифікація та оновлення даних;
- фільтрація даних;

- функція перевірки коректності введених даних;
- обробка помилок.

Приклад функціонального тесту наведено нижче:

Таблиця 1 – Приклад тестового сценарію

Інформація про тестовий випадок			
Ідентифікатор тестового випадку		ТС-F-03	
Власник тесту		Скиданенко Д. М.	
Дата останнього перегляду		02.06.2020	
Мета тесту		Тестування додавання інформації до БД	
Методика тестування			
Налаштування прогону тесту		Не проводиться	
N/A			
Крок	Дія	Очікуваний результат	Відмітка (V/X)
1	Завантажити додаток (на локальному хості)	Завантажується головна сторінка додатку	V
2	Натиснути посилання "Measuring tools"	Завантажується сторінка зі списком приладів	V
3	Натиснути на кнопку додавання елемента "+"	Блокується сторінка, відкривається модульне вікно додавання інструмента	V
4	Заповнити поля форми та натиснути кнопку "Save"	З'являється інформування про вдале збереження; модальне віконце відновлюється для введення інших даних	V
5	Закрити форму натисканням кнопки "Close"	Віконце закривається, основне робоче поле додатка відблоковується та список інструментів автоматично оновлюється	V
Результати тесту			
Тестувальник: Скиданенко Д. М.		Дата прогону тесту: 02.06.2020	Результат тесту: (P/F/B): P (пройдено)

В ході тестування було виявлено незначну кількість неточностей та помилок, наприклад, в функціях перевірки коректності введених даних, що реалізовані на стороні клієнта за допомогою мови JS. Зважаючи на те, що ці функції будуть розширюватись (наприклад, буде додаватися функціонал авторизації в додатку) та

будуть дописуватись нові модулі для перевірки даних, ці неточності будуть враховані далі, доопрацьовані та для них буде проведено окреме тестування.

5.2 Нефункціональне тестування

Прикладом нефункціонального тестування може бути тестування інтерфейсу користувача. В ході тестування було перевірено значна кількість варіантів використання ПЗ. Приклад тесту наведений нижче:

Таблиця 2 – Тестування графічного інтерфейсу користувача

Номер	Критерій тестування	Відмітка (V/X)
1	Сторінки додатку завантажуються без очевидних помилок в усіх поширених браузерях (WebKit, Presto, Trident, Gecko, Blink)	V
2	Робоча область додатку автоматично підстраюється до розмірів вікна при їх зміні до мінімально заданих	V
3	Додаток відображається на мобільних платформах (смартфон, планшет)	V
4	Всі посилання працездатні (за виключенням "заглушок") на будь-яких платформах браузерів	V
5	Навігація між елементами можлива за допомогою клавіші "Tab"	V
6	При навігації клавішею "Tab" фокус переміщується по елементах зліва направо та згори вниз	V
7	Контент функціональних блоків не виходить за границі блоків при зменшенні їх розмірів.	V

Вище наведено лише уривок великого чек-листу тестування інтерфейсу, результатом якого було доопрацювання розмітки базової сторінки (_Layout.schtml) та значної кількості декларацій в каскадних таблицях стилів. Метою доопрацювання є не лише покращення показника usability, але й accessibility, що я вважаю за необхідність в разі розробки ПЗ корпоративного сектору.

ВИСНОВКИ

В ході атестаційної роботи було розроблено програмне забезпечення підтримки роботи відділу контролю якості. При розробці та під час обмірковування функціоналу ПЗ стало очевидно, що тема є дуже об'ємною, тому було прийняте рішення розробляти ПЗ за допомогою agile-технологій, а саме, спрінтами.

Такий підхід допоможе сфокусуватися на рішенні конкретних стислих та чітко окреслених завдань, та в коротші строки отримати робочу частину продукту.

Але, незважаючи на реалізацію проекту по частинах, також було прийняте рішення про максимально повну розробку бази даних. Це дозволить зменшити кількість міграцій в подальшому, що значно зменшує ризики втрати даних. Саме тому наприкінці першого спринту розробки, база даних (а точніше, її таблиці) використовуватимуться лише частково (в залежності від імплементованого функціоналу). Також, наявність додаткових таблиць в базі даних дозволить досить просто додавати в роботу нові модулі MVC.

В цілому, проект має можливість трансформуватися в систему керування підприємством, але для цього необхідна правильна та професійна постановка ТЗ компетентними в досить вузьких областях спеціалістами. Також обов'язковим наступним кроком вважаю додавання в проект функціоналу авторизації користувачів та розподілення їх по ролях – це забезпечить значну економію часу завдяки автоматизації догляду за власниками / виконувачами процесів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ISO 9000 системы менеджмента качества // інформаційна сторінка організації ISO. URL: <https://www.iso.org/ru/iso-9001-quality-management.html> (дата звернення: 12.05.2020)

2. ISO 9001:2015 for small enterprises - what to do? // інформаційна сторінка стандарту версії 2015 року. Дата останнього оновлення 2016 р. URL: <https://www.iso.org/ru/publication/PUB100406.html> (дата звернення: 12.05.2020)

3. Eight disciplines problem solving // сторінка Вікіпедії.
Дата останнього оновлення: 04.03.2020. URL: https://en.wikipedia.org/wiki/Eight_disciplines_problem_solving (дата звернення: 12.05.2020)

4. ERP system for manufacturing companies // домашня сторінка продукту. URL: <https://www.monitorerp.com/> (дата звернення: 12.05.2020)

5. Матвеевский В.Р. Надежность технических систем. Учебное пособие – Московский государственный институт электроники и математики. М., 2002 г. – 113 с. ISBN 5–230–22198–4

6. ASP.NET documentation | Microsoft docs // сторінка документації розробника. URL: <https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1> (дата звернення: 12.05.2020)

7. ASP.NET MVC Pattern | .NET // сторінка документації розробника. URL: <https://dotnet.microsoft.com/apps/aspnet/mvc> (дата звернення: 12.05.2020)

8. Сергей Рогачев. Обобщённый Model-View-Controller // rsn.org. Дата останнього оновлення: 10.12.2016. URL: <http://rsdn.org/article/patterns/generic-mvc.xml> (дата звернення: 12.05.2020)

9. Introduction to Razor Pages in ASP.NET Core | Microsoft Docs // сторінка документації розробника. URL: <https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-3.1&tabs=visual-studio> (дата звернення: 12.05.2020)

10. Learn ASP.NET Core 3.1 - Full Course for Beginners [Tutorial] // навчальне відео ресурсу FreeCodeCamp(). Дата останнього оновлення: 05.02.2020. URL: <https://youtu.be/C5cnZ-gZy2I> (дата звернення: 12.05.2020)

11. Bootstrap · The most popular HTML, CSS, and JS library in the world // домашня сторінка продукту. URL: <https://getbootstrap.com/> (дата звернення: 12.05.2020)

12. Can I use... Support tables for HTML5, CSS3, etc // сервіс перевірки підтримки front-end технологій настільними та мобільними браузерами. Дата останнього оновлення: 01.06.2020. URL: <https://caniuse.com/#feat=flexbox> (дата звернення: 02.06.2020)

13. Entity Framework Core Tutorials // навчальний ресурс з технології EF Core. URL: <https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx> (дата звернення: 02.06.2020)

The diagram illustrates a database schema with the following tables and their attributes:

- PRODUCT**: ID INT, NAME VARCHAR(45)
- QUALITY_CARD**: ID INT, LINK VARCHAR(60)
- ORDER**: ID INT, INT_CODE CHAR(10), EXT_CODE VARCHAR(45), ORDER_DATE DATE, PROD_DATE DATE, PRICE DECIMAL(6,2), COMMENT MEDIUMTEXT, CLIENT_ID INT, ORDER_TYPE_ID TINYINT, PRODUCT_ID INT, QUALITY_CARD_ID INT
- ORDER_TYPE**: ID TINYINT, NAME VARCHAR(45)
- ORDER_EXPENSES**: ID INT, CLAIM_ID INT, DATE DATE, SUM DOUBLE(5,2), LINK VARCHAR(60), COMMENT VARCHAR(45), DEPARTMENT_ID TINYINT
- CLAIM**: ID INT, DATE_START DATE, DATE_END DATE, ORDER_ID INT, CLIENT_ID INT
- CLAIM_HISTORY**: ID INT, CLAIM_ID INT, ACTION_ID INT
- CLAIM_COMMENTS**: ACTION_ID INT, CLAIM_ID INT, COMMENT TINYTEXT
- ACTION**: ID INT, AIM VARCHAR(45), DATE_START DATE, DATE_END DATE, WORKER_ID INT
- COUNTRY**: ID TINYINT, NAME VARCHAR(45), LOCAL CHAR(2), TRADE ONE VARCHAR(45), CURRENCY CHAR(3)
- CLIENT**: ID INT, NAME VARCHAR(45), CONTACT VARCHAR(45), PHONE VARCHAR(16), EMAIL VARCHAR(45), ADDRESS VARCHAR(60), COUNTRY_ID TINYINT
- WORKER**: ID INT, EMPLOYEE_ID INT, POSITION_ID INT, TOOL_LIST_ID INT, TOOL_PROP_ID INT, TOOL_UTILIZED TINYINT
- TOOL_LIST**: ID INT, LAST_CHECK DATE, TOOL_TYPE_ID INT, WORKER_ID INT, TOOL_UTILIZED TINYINT
- TOOL_PROP**: ID INT, NAME VARCHAR(45), CHECK_PERIOD MONTH TINYINT
- COMPETITION_MATRIX**: ID INT, EMPLOYEE_ID INT, LANGUAGES TINYINT, WELDING TINYINT, BRAKES TINYINT, FORKLIFT TINYINT, DRAWING TINYINT, CONSTRUCTING TINYINT, CHEMISTRY TINYINT, ACCOUNTING TINYINT, MANAGEMENT TINYINT, QUALITY TINYINT
- DEPARTMENT**: ID TINYINT, NAME VARCHAR(48), DEPT_CODE CHAR(8), DEPT_HEAD INT, DEPT_WAREHOUSES TINYINT, DEPT_ACCOUNTS TINYINT, DEPT_COMPETITION_PATTERN TINYINT
- COMPETITION_PATTERN**: ID TINYINT, NAME VARCHAR(20), LANGUAGES TINYINT, WELDING TINYINT, BRAKES TINYINT, FORKLIFT TINYINT, DRAWING TINYINT, CONSTRUCTING TINYINT, CHEMISTRY TINYINT, ACCOUNTING TINYINT, MANAGEMENT TINYINT, QUALITY TINYINT
- ACCOUNT**: ID TINYINT, CODE CHAR(8), NAME VARCHAR(45)
- WAREHOUSE**: ID TINYINT, CODE CHAR(8), NAME VARCHAR(45)
- WH_ITEMS**: ID INT, NAME VARCHAR(45), DESOR VARCHAR(45), UNITS ENUM(...), QUANTITY FLOAT(5,2), PRICE DECIMAL(5,4), WAREHOUSE_ID TINYINT

Рисунок А.1 – Схема взаємодії таблиць

ДОДАТОК Б

Скетчі дизайну клієнтської частини програми

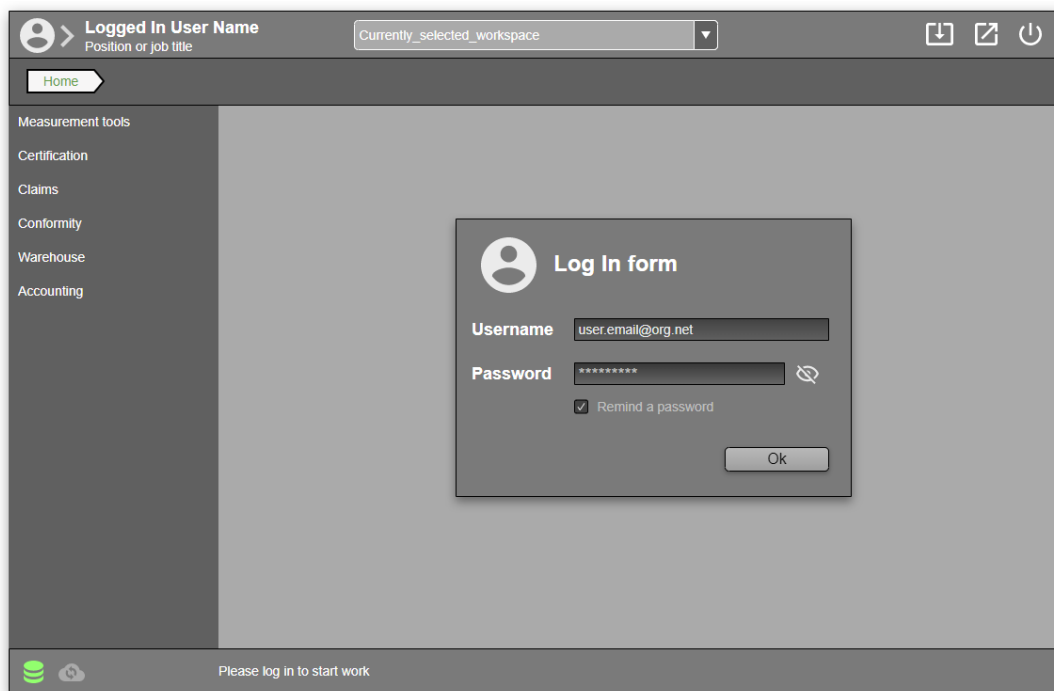


Рисунок Б.1 – Форма авторизації

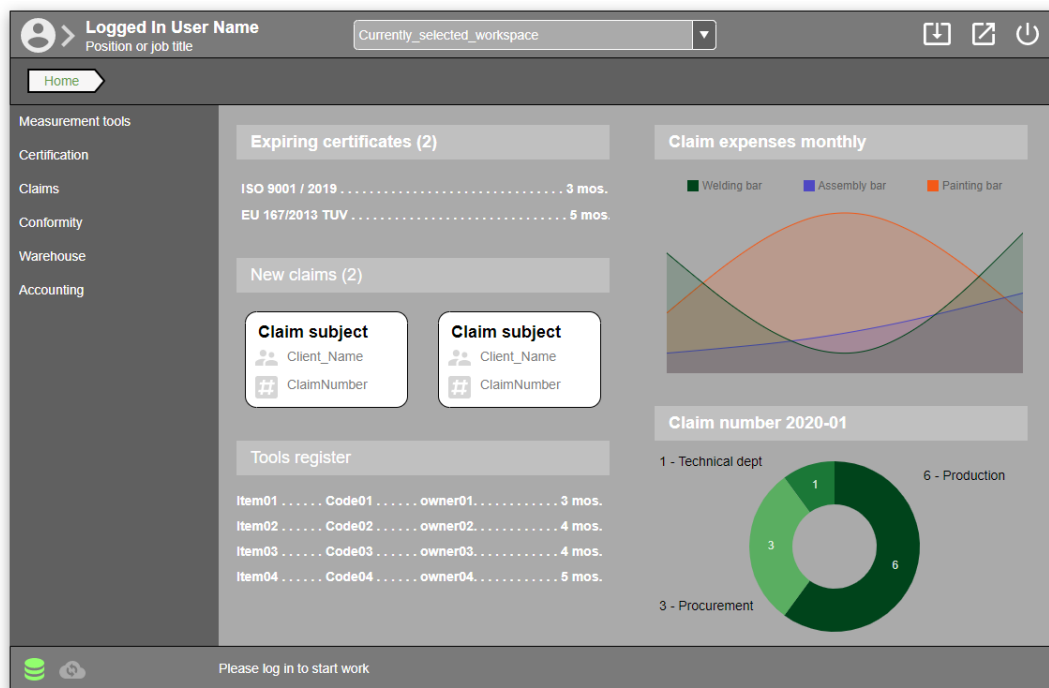


Рисунок Б.2 – Головна сторінка

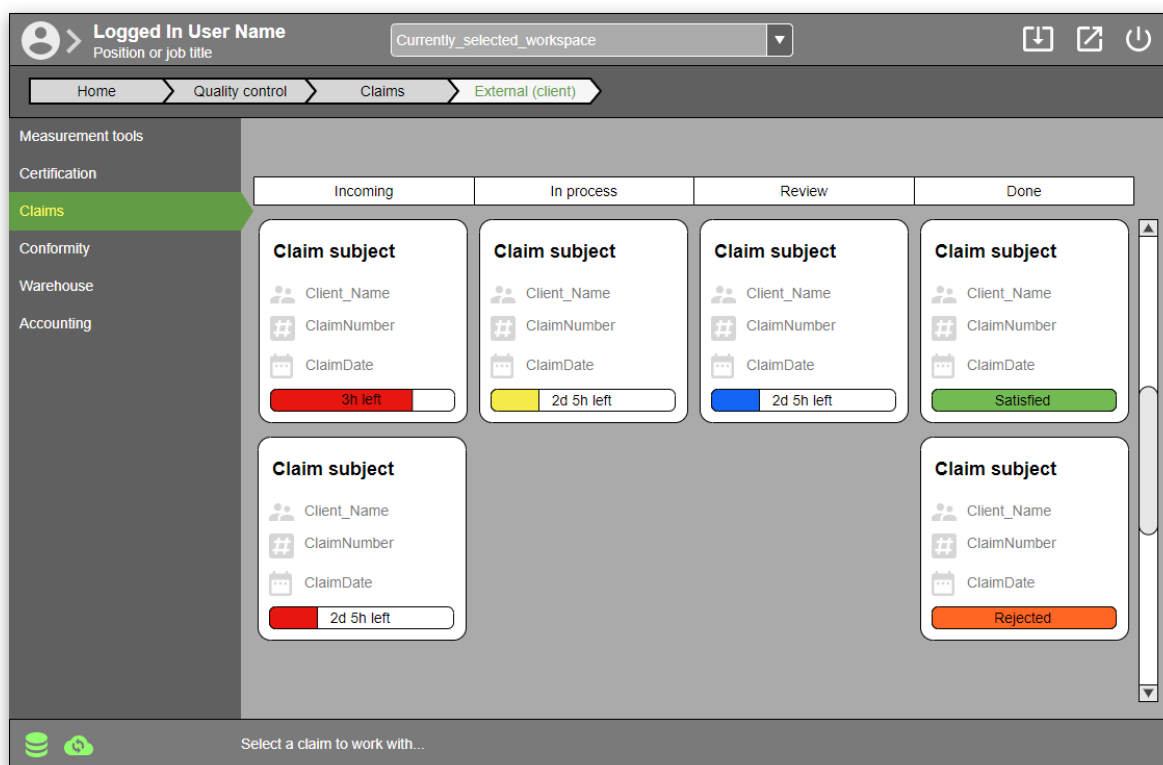


Рисунок Б.3 – Список рекламцій

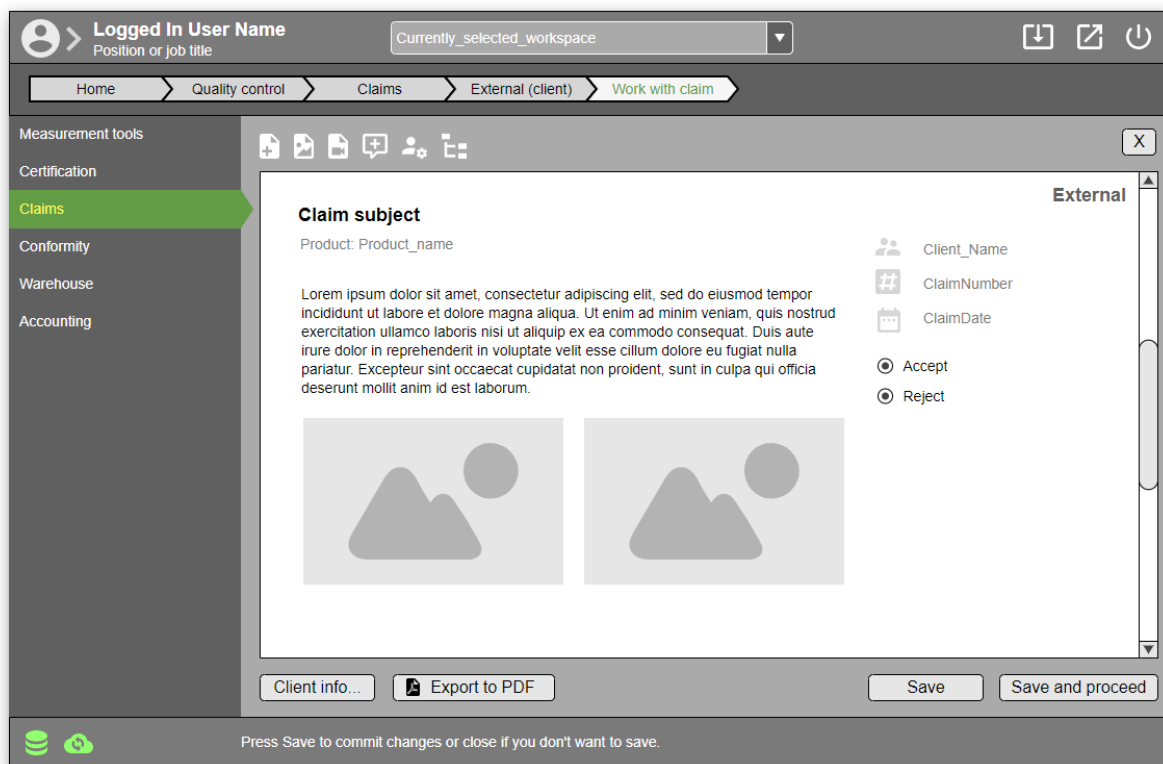


Рисунок Б.4 – Деталі рекламції

ДОДАТОК В

Лістинг коду усіх сутностей (Models) додатку

```
namespace Qualify.Models
{
    public class Action
    {
        public int ID { get; set; }
        public string Description { get; set; }
        public int EmployeeID { get; set; }
        public bool? Performed { get; set; }
        public bool? Done { get; set; }

        public Employee Employee { get; set; }
        public ICollection<ClaimHistory> ClaimHistories { get; set; }
    }

    public class Claim
    {
        public int ID { get; set; }
        public int ClientID { get; set; }
        public string Title { get; set; }
        public string? Description { get; set; }
        public string? Dirpath { get; set; }
        [DisplayFormat(DataFormatString =
            "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime DateStart { get; set; }
        [DisplayFormat(DataFormatString =
            "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime? DateEnd { get; set; }

        public Client Client { get; set; }
        public ICollection<ClaimHistory> ClaimHistories { get; set; }
        public ICollection<ClaimExpence> ClaimExpences { get; set; }
    }

    public class ClaimExpence
    {
        public int ID { get; set; }
        public double Costs { get; set; }
        public string Description { get; set; }
        public int DepartmentID { get; set; }
        public int ClaimID { get; set; }

        public Claim Claim { get; set; }
        public Department Department { get; set; }
    }

    public class ClaimHistory
    {
        public int ID { get; set; }
        public int ClaimID { get; set; }
        public int ActionID { get; set; }

        public Claim Claim { get; set; }
    }
}
```

```

        public Action Action { get; set; }
    }

public class Client
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string? Country { get; set; }
    public string? Address { get; set; }
    public string? Email { get; set; }
    public string? Phone { get; set; }

    public ICollection<Claim> Claims { get; set; }
}

public class Department
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string? Account { get; set; }
    public string? Warehouse { get; set; }

    public ICollection<Employee> Employees { get; set; }
    public ICollection<ClaimExpense> ClaimExpenses { get; set; }
}

public class Employee
{
    public int ID { get; set; }
    public string Name { get; set; }
    public int? DepartmentID { get; set; }
    public string? Position { get; set; }

    public ICollection<Action> Actions { get; set; }
    public ICollection<Tool> Tools { get; set; }
    public Department Department { get; set; }
}

public class Tool
{
    public int ID { get; set; }
    public int ToolTypeID { get; set; }
    public int EmployeeID { get; set; }

    public Employee Employee { get; set; }
    public ToolType ToolType { get; set; }
}

public class ToolType
{
    public int ID { get; set; }
    public string Name { get; set; }
    public int CheckIntervalWeeks { get; set; }

    public ICollection<Tool> Tools { get; set; }
}
}

```

```

namespace Qualify.Service
{
    public class QualifyContext : DbContext
    {
        public QualifyContext(DbContextOptions<QualifyContext>
                                options) : base(options)
        {}

        public DbSet<Claim> Claims { get; set; }
        public DbSet<Models.Action> Actions { get; set; }
        public DbSet<ClaimExpence> ClaimExpences {get; set;}
        public DbSet<ClaimHistory> ClaimHistories {get; set;}
        public DbSet<Client> Clients {get; set;}
        public DbSet<Department> Departments { get; set; }
        public DbSet<Employee> Employees {get; set;}
        public DbSet<Tool> Tools {get; set;}
        public DbSet<ToolType> ToolTypes {get; set;}
    }
}

```

ДОДАТОК Г

Слайди презентації до проекту