* 1. Програмна реалізація

2.3.1 Вибір технологій

Для програмної реалізації АІС було обрано мову програмування C# версії 8.0 у поєднанні з технологією WPF для створення графічних інтерфейсів, яка використовує XAML- розмітку.

До переваг **C#** можна віднести її орієнтованість на об’єктно-орієнтований підхід до програмування, непогану оптимізацію в поєднанні з простотою використання тощо. Однак головною особливістю мови є те, що вона **створена** компанією Microsoft спеціально **для розробки застосунків під** ОС **Windows**. Тож для неї створено чимало технологій для розробки графічного інтерфейсу під цю операційну систему.

Багато в чому найкращою з них можна назвати WPF (**Windows Presentation Foundation**). Вона забезпечує максимальну гнучкість при розробці GUI. Розробник може тонко налаштувати будь-який елемент інтерфейсу, чи створити власні шляхом композиції чи успадковування вже існуючих.

При цьому використовується **мова розмітки XAML** (Extendable Application Markup Language), чий синтаксис багато в чому співпадає з таким мовами розмітки як HTML, тож її вивчення не потребує значних зусиль. **Верстання** за допомогою такого засобу **набагато швидший процес** ніж його створення за допомогою окремих програм з графічним інтерфейсом чи шляхом безпосереднього використання C#. Крім того використання окремої мови **дозволяє краще відділити GUI від бізнес-логіки проєкту.**

Оскільки метою Програми є спрощення роботи із **PostgreSQL**-базою даних, то **для роботи** **із цим діалектом** SQL **використано** бібліотеку **Npgsql**. Оскільки вона **має** значне поширення, а отже й **підтримку** (тобто наявність якісної офіційної документації та постійні оновлення для роботи із останніми версіями PostgreSQL). Крім того бібліотека має **простий синтаксис**, який здебільшого зводиться до передачі базі даних PostgreSQL-команд й отримання від неї зворотного зв’язку, тож її опанування, за умови знання відповідного SQL-діалекту, майже не викликає зусиль.

2.3.2 Інтерфейс користувача

Графічний інтерфейс містить 3 основні вікна, не включаючи діалогові. Кожне з цих вікон є окремим файлом, що містить XAML-розмітку та C# файл. Розмітка призначена для вказання елементів вікна та його параметрів. C#-код призначений перш за все для додавання до елементів вікна подій (наприклад натискання кнопок), а також створення зручного інтерфейсу для роботи з вікном через клієнтський код (наприклад відкриття вікна чи отримання даних із елементів вікна, зокрема текстових полів).

Вікно авторизації користувача, яке містить кнопки Під’єднання до сервера, Створення користувачів та Переходу до вікна Вибору запитів. Вікно також містить код, що регулює активність кнопок Створення й Переходу залежно від того чи зареєстрований користувач.

Розмітка: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/Authorization/UserCreatorWindow.xaml>

Код: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/Authorization/UserCreatorWindow.xaml.cs>

Рис 2.3.2.1 Вікно авторизації користувача (користувач не авторизований)

Зображення, що містить текст, знімок екрана, Шрифт, число

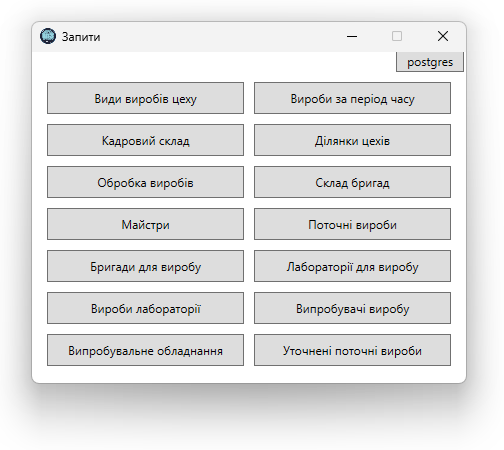
Автоматично згенерований опис

Вікно вибору запитів містить набір кнопок для власне здійснення запитів, які при наведенні показують підказку із детальним їх-запитів описом та кнопку для повернення до вікна Авторизації. Ці кнопки й підказки для них було згенеровано через C# з метою автоматизації. Оскільки це фактично 14 однакових об’єктів, які відрізняються лише текстом й подіями, що відбуваються після їх натискання.

Розмітка: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/Queries/MainWindow.xaml>

Код: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/Queries/MainWindow.xaml.cs>

Рис 2.3.2.2 Вікно вибору запитів



Вікно виводу результатів запитів містить текстове поле для виводу результату запиту та кнопку для його збереження в текстовий файл. Також вікно містить код для відображення відповідного тексту при відкритті клієнтським кодом.

Розмітка: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/Output/Display.xaml>

Код: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/Output/Display.xaml.cs>

Рис 2.3.2.3 Вікно виводу результатів запитів

Зображення, що містить текст, знімок екрана, програмне забезпечення, комп’ютер

Автоматично згенерований опис

Діалогові вікна можна поділити на 2 системні й користувацькі (вони ж Форми (див. розділ 2.1.). До системних можна віднести вікна для відображення помилок, вибору дати чи шляху для збереження файлу. Всі вони вбудовані в систему й доступні до використання через відповідні функція для їх створення, відкриття, зчитування з них даних тощо.

Розмітка: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/General/Form/Field/UI/DateField.xaml>

Код: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/General/Form/Field/UI/DateField.xaml.cs>

Рис 2.3.2.4 Спливаюче вікно вибору дати

Зображення, що містить текст, знімок екрана, число, програмне забезпечення

Автоматично згенерований опис

Форми – це діалогові вікна призначені для введення користувачем інформації, необхідної для здійснення певного запиту. Вони містять набір полів різних типів та кнопки Надсилання запиту й Скасування запиту. Всі поля є користувацькими елементами інтерфейсу, що складаються з набору вбудованих (наприклад текстове поле для вводу й напис для заголовку). Шляхом використання можливостей WPF створення форм було максимально автоматизоване. Зведене до введення набору полів із вказанням їх параметрів у XAML розмітці.

Розмітка: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/Queries/Second/FormUi.xaml>

Код: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/Queries/Second/FormUi.xaml.cs>

Рис 2.3.2.4 Діалогове вікно для здійснення запиту про виробництво виробів протягом зазначеного періоду.

Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

2.3.3 Бізнес-логіка Проєкту

2.3.3.1 Шаблон Form Observer

Почнімо з основи проєкту – його Архітектурного патерну. Розгляньмо ієрархію класів створених для його реалізації, що знаходяться у відповідних файлах теки Form. (див Рис. 2.3.3.1).

Рис. 2.3.3.1.1 – Тека з кодом, що відповідає за реалізацію Архітектурного Патерну проєкту

Зображення, що містить текст, знімок екрана, дизайн

Автоматично згенерований опис

Посилання: <https://github.com/Denchyk-chyk/cw-ad-2024-code/tree/master/CS/General/Form>

Почнімо з дочірньої директорії Logic. Вона містить основні інтерфейси та класи, що відповідають за бізнес-логіку форми. Зокрема інтерфейси для обробника форми (IFormHandler), її користувача (IFormUser) та графічного інтерфейсу (IFormUi), які реалізують відповідні класи в конкретних розділах проєкту: Авторизації та Здійснені запитів.

Крім того тут знаходяться класи, що відповідають за використання форми (FormSender (статичний)) та обробку помилок (FormErroHadnler), що можуть виникнути під час цього процесу. FormSender здійснює «доставку» даних із форми до її обробника. В разі виникнення помилок, що відстежуються відповідними FormErroHadnler’ами, він інформує користувача про їх деталі й дозволяє повторно заповнити діалогове вікно.

Рис. 2.3.3.1.2 – Діаграма, що показує розділення абстракції та реалізації полів діалогових вікон

Зображення, що містить текст, схема, ряд, Паралель

Автоматично згенерований опис

Папка Field містить класи, що відповідають за поля форми. Вони розділені на абстракцію (бізнес-логіку – тека Logic) та реалізацію (графічний інтерфейс – тека Ui), що є реалізацією патерну Компонувальник (див…). Таке розділення дозволяє взаємодіяти з різними елементами інтерфейсу незалежно від їх реалізації, що схематично показано на Рис 2.3.3.2. Різні типи Field (логіки) взаємодіють з різними видами об’єктів інтерфейсу через їхній спільний інтерфейс IFieldUi.

Таким чином, зокрема, TextField може одночасно використовуватись як класами IntegerField так StringField – вони здійснюють конвертацію даних із текстового рядка, та перевіряють правильність введених даних. При цьому сам TextField лише надає інтерфейс для взаємодії через клієнтський код, будучи повністю незалежним від того, хто його використовує.

Напротивагу логічний клас StringField може використовувати як текстове поле (TextField) так і поле з датою (DateField). Він не залежить від їх фактичного змісту, він просто отримує від них один інтерфейс взаємодії, який надає йому можливість зчитати або записати дані в форматі string.

Крім того в рамках цього рішення допустима ієрархія успадковування класів логіки. Наприклад EnumField (суперклас, що використовує дані із випадаючого списку) має дочірній клас OptionaEnumField, який відрізняється лише тим, що дозволяє користувачу не обирати жоден варіант зі списку.

Крім того директорія Form містить власне клас Form та клас Container (регулює активність полів залежно від вибору користувача (наприклад користувач може обрати якої структурної частини підприємств стосується його запит: цеху, ділянки чи бригади, й залежно від вибору будуть активні лише певні поля)). Обидва класи беруть під контроль певний елемент інтерфейсу, не зосереджуючись на його реалізації.

Container використовує набір полів (Field) і регулює їх активність залежно від умов, виставлених користувачем. Він не залежить від типу полів, їх розташування, елементів, які викликають події для зміни умови їх активності й тд. Значення має лише те, що всі ці поля можна ввімкнути або вимкнути.

Так само Form працює з будь-яким вікном, якщо воно реалізує інтерфейс IFormUi. Для форми важливо лише те, що з вікна обов’язково можна отримати значення полів або записати їх, а також те, що це вікно можна відкривати, закривати, згортати тощо. Конкретну реалізацію всієї описаної тут логіки розглянемо в двох наступних розділах.

2.3.3.2 Авторизація користувача

Авторизація користувача в основі своїй є практичною реалізацією шаблону Form Observer, описаного в попередньому розділі. Файли, що відповідають за цей розділ знаходяться в теці Authorization.

Рис. 2.3.3.2.1 – Тека з кодом, що відповідає за авторизацію користувача

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

Він містить основне вікно – клас MainWindow (див. рис 2.3.3.2.2), який реалізує інтерфейс IFormUser, для того, щоб відкривати відповідні діалогові вікна для введення даних, необхідних для виконання з’єднання з сервером (див. рис 2.3.3.2.2) та додавання нових користувачів до бази даних. (див. рис 2.3.3.2.3).

Рис. 2.3.3.2.2 – Вікно авторизації користувача (користувач авторизований)

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Ці сценарії відповідно мають по одній FormUi та FormUser. Щодо останніх, то це singleton-класи, що містять лише один метод та набір обробників помилок (FormErrorHandler), що можуть виникнути в процесі його-методу виконання.

Для ServerConnector’а цими помилками можуть бути неправильний логін чи пароль користувача. Неможливість підключення до вказаного хоста чи некоректна назва бази даних. В разі виникнення однієї з перелічених помилок, користувач повертається до заповнення форми, при цьому отримуючи діалогове вікно з інформацією про помилку та позначення некоректно заповнених полів. На рис. 2.3.3.2.5 показано як це виглядає на прикладі неправильно введеного пароля.

Рис. 2.3.3.2.3 – Форма під’єднання до сервера (авторизації користувача)

Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

Рис. 2.3.3.2.4 – Форма додавання нового користувача

Зображення, що містить текст, знімок екрана, монітор, Прямокутник

Автоматично згенерований опис

Рис. 2.3.3.2.5 – Повідомлення про помилку при під’єднанні до сервера

Зображення, що містить текст, знімок екрана, монітор, число

Автоматично згенерований опис

2.3.3.3 Здійснення запитів до бази даних

Це основна частина програми, яка так само як і попередня ґрунтується на її Архітектурному Шаблоні. Її код розміщено в 2-х теках: Queries та Output (рис. 2.3.3.3.4). Перша відповідає за власне надсилання запитів до бази даних та отримання від неї їхніх результатів. В той час, як друга забезпечує зчитування отриманих від БД й даних та їхній вивід у зручному для користувача форматі.

Рис. 2.3.3.3.4 – Теки з кодом, що відповідає за здійснення запитів до БД та вивід їхнього результату на екран користувача

Зображення, що містить текст, знімок екрана, дизайн

Автоматично згенерований описЗображення, що містить текст, знімок екрана, дизайн

Автоматично згенерований опис

Посилання:

Queries: <https://github.com/Denchyk-chyk/cw-ad-2024-code/tree/master/CS/Queries>

Output: <https://github.com/Denchyk-chyk/cw-ad-2024-code/tree/master/CS/Output>

Почнімо з Queries. Її головний файл, MainWindow, містить IFormUser-вікно, з якого здійснюються всі запити. Цих запитів 14. Для кожного з них створено IFormUi-вікно та IFormHandler-клас.

Детальніше про кожен із них. Принцип роботи інтерфейсу форми вже було детально розглянуто в попередніх розділах, однак варто уточнити кілька моментів. Зокрема стосовно випадаючих списків – ListField та їх логічної обгортки – EnumField. Основною властивістю будь-якого такого меню є його варіанти вибору. В контексті розроблюваної АІС можна видіти два аспекти, що їх стосуються.

По-перше ці варіанти часто повторюються в різних формах (тобто діалогових вікнах). Вводити їх вручну кожен раз доволі не раціонально. Крім того це може призвести до помилок. То ж було створено окремий глобальний словник (щось на кшталт Dictionary<enum, string[]>), що містить всі набори варіантів спадних списків, що зустрічаються в додатку. Ключем в цьому словник є enum-клас, тож помилка при виборі потрібного списку майже виключена.

Таким чином все, що треба для заповнення випадаючого списку – це обрати в EnumField відповідне значення ключа з глобального словника.

Іншою проблемою є те, що деякі варіанти спадних меню беруть свої варіанти з бази даних, наприклад: список виробів чи ділянок підприємства (див рис. 2.3.3.3.1). Можна ввести їх вручну, але це абсурд, оскільки треба буде переписувати програму при внесені змін в базу даних. Рішенням для цієї проблеми стало отримання таких списків за допомогою запитів до бази даних. Всі ці запити здійснюються в раніше згаданому ServerConenctor`і. Тобто при під’єднанні до сервера. Їхні результаті зберігаються в уже згаданому глобальному словники під відповідними ключами й до них можна отримати доступ за наведеним у попередньому абзаці способом.

Розмітка: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/General/Form/Field/UI/ListField.xaml> ()

Код: <https://github.com/Denchyk-chyk/cw-ad-2024-code/blob/master/CS/General/Form/Field/UI/ListField.xaml.cs> ()

Рис. 2.3.3.3.3 – Випадаючий список

Зображення, що містить текст, електроніка, знімок екрана, монітор

Автоматично згенерований опис

Крім того варто розглянути ще один аспект, що стосується IFormUi. Таблиця 2.3.3.3.1 містить всі поля, необхідні для введення параметрів щодо виконання всіх чотирнадцяти запитів до БД. Як не складно помітити частина їх збігається. Форми 5, 9, 10; 6, 7 та 12, 13, 14 – ідентичні. Оскільки інтерфейс достатньо відділений від бізнес-логіки можна просто використати одну форму в кількох запитах. Таким чином достатньо створити всього 9 форм замість 14, оптимізувавши 36% роботи завдяки правильно підібраній архітектурі Проєкту.

Таблиця 2.3.3.3.1 – Поля, необхідні для здійснення кожного запиту

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Поле 1 | Поле 2 | Поле 3 | Поле 4 | Поле 5 | Поле 6 |
| 1 | Категорія | Цех |  |  |  |  |
| 2 | Категорія | Підрозділ | Цех | Ділянка | Початкова дата | Кінцева дата |
| 3 | Підрозділ | Цех | Ділянка | Персонал | Категорія інженерів | Категорія робітників |
| 4 | Цех |  |  |  |  |  |
| 5 | Виріб |  |  |  |  |  |
| 6 | Цех | Ділянка |  |  |  |  |
| 7 | Цех | Ділянка |  |  |  |  |
| 8 | Категорія | Підрозділ | Цех | Ділянка |  |  |
| 9 | Виріб |  |  |  |  |  |
| 10 | Виріб |  |  |  |  |  |
| 11 | Категорія | Лабораторія | Початкова дата | Кінцева дата |  |  |
| 12 | Вироби | Категорія | Тип | Лабораторія |  |  |
| 13 | Вироби | Категорія | Тип | Лабораторія |  |  |
| 14 | Категорія | Підрозділ | Цех | Ділянка |  |  |

Що стосується Queries, то всі вони є класами дочірніми до супертипу Query. Цей клас містить один метод для здійснення запиту, однак він поділений на окремі підметоди, реалізуючи тим самим патерн Шаблонний метод. Це зроблено для того, щоб в дочірніх класах можна було перевантажувати лише невелику частину логіки, тим самим сильно спрощуючи процес наслідування.

Варто трохи уточнити принцип роботи самого класу Query. Він поділяється на наступні етапи:

1. Створення текстового рядка із SQL-запитом (select) на основі даних, отриманих із форми. Це ключовий метод, який завжди перевантажується в дочірніх класах.
2. Ініціалізація об’єкту-читача (NpgsqlDataReader) на основі рядка із запитом. Цей читач проходить кожним записом отриманого результату запиту (рядком таблиці)
3. Прочитання кожного рядка. Метод, який завжди перевантажується в дочірніх класах, однак більшість логіки цього процесу передана стороннім об’єктам типу Displayable. Кожен із них створює текстовий рядок на основі запису із таблиці. Підкласу Query треба лиш обрати потрібний тип такого об’єкта.
4. Вивід даних. В процесі прочитання таблиці із виводом утворюється колекція Displayable-об’єктів. Вони передаються класу Display, який виводить їх на екран в окремому вікні.

Розгляньмо особливості Displayable-об’єктів. Їхні конструктори приймають в якості атрибуту запис select-таблиці, отримують з нього значення відповідних полів і зберігають їх в якості параметрів, з яких потім створюють рядок для виводу. Цей процес часто тривіальний. Найпростіший такий об’єкт – Line. Він просто зчитує одне поле й виводить його значення. Назва цього поля передається в якості параметра конструктора разом із записом (хоча зазвичай використовується значення за умовчуванням – «name»). Однак бувають набагато цікавіші випадки.

Деякі із цих таблиць стосуються об’єктів зі складною ієрархією. Наприклад персоналу підприємства. Кожен працівник входить одночасно до кількох таблиць. Наприклад: персонал, робітники, зварювальники чи персонал, інженери, менеджмент. Оскільки в рамках шаблонної реалізації запитів у Проєктів можлива лише одна таблиця із результатами доводиться проводити join всіх таблиць ієрархії. На виході отримуємо результуючу таблицю з десятками полів, більшість записів, яких мають null-значення. Прочитання такої таблиці зводиться до того, що Displayable-об’єкт (в цьому випадку Employee) проходиться всіма полями таблиці й до додає їх значення в свій список параметрів, якщо воно не null.

Бувають і цікавіші випадки. Наприклад запит з відомостями кількість та дату виробництва автомобілів. Отримати цю інформацію про дати не складно. Кількість теж – для цього потрібно лиш згрупувати запит за відповідними полями. Однак для отримання обох відомостей потрібні 2 select’и, що заперечує архітектуру Програми. Тож доводиться проводити «групування» на рівні C#. Відповідні Displayable-об’єкти (Production) працюють наступними чином. По. В них приватні конструктори. Ззовні можливий лише метод розширення, що дозволяє додати об’єкт до списку List<Production>, передавши NpgsqlDataReader (тобто запис таблиці) в якості параметру. Ідентифікатор рядка порівнюється з наявними у списку, якщо він відступній, то створюється новий Production, якщо ні, то дані із нього (дата й кількість виготовлених машин) додаються до існуючого об’єкта з відповідним id. Вся логіка інкапсульована в межах об’єкта. Клієнтському коду лише треба створити список і додати до нього всі записи із select-таблиці.