ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської кваліфікаційної роботи

на ступінь вищої освіти бакалавр

на тему: **«Розробка програмного забезпечення для роботи диспетчерських служб мовою Java»**

Виконав: студент 4 курсу, групи ПД-44

спеціальності 121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Меленевський І.В.

(прізвище та ініціали)

Керівник Коба А.Б.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Нормоконтроль

(прізвище та ініціали)

Київ 2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність -121 Інженерія програмного забезпечення

ЗАТВЕРДЖЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____О.В.Негоденко
«____»_____2021 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

МЕЛЕНЕВСЬКОМУ ІГОРЮ ВІТАЛІЙОВИЧУ

1.Тема роботи: «Розробка UI для електронного журналу на мові JavaScript»
Керівник роботи Негоденко Олена Василівна, професор, доктор технічних наук
затверджені наказом вищого навчального закладу від — «12» березня 2021 року №65.

2. Строк подання студентом роботи 01.06.2021

3. Вхідні дані до роботи:

3.1. Середовище розробки IntelliJ IDEA 2021.1.1 (Ultimate edition)

3.2. Алгоритм дії API для диспечерських служб

3.3. Інтерфейс API

3.4. Науково-технічна література, пов’язана з розробкою API

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1. Аналіз повноважень користувача

4.2. збір та аналіз ключових документів

4.3. Дослідження програмних засобів для розробки додатку

4.4. Розробити функціонал створеного додатку

6. Дата видачі завдання 19.04.2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково технічної літератури	19.04.21 – 22.04.21	
2	Аналіз існуючих прототипів	22.04.21 – 23.04.21	
3	Моделювання об’єкту проектування	01.05.21 – 05.05.21	
4	Розробка функціоналу API	05.05.21 – 24.05.21	
5	Вступ, висновки, реферат	24.05.21 – 26.05.21	
6	Розробка презентації застосунку	26.05.21 – 31.05.21	
7	Попередній захист роботи	01.06.21	

РЕФЕРАТ

Текстова частина бакалаврської роботи.

Ключеві слова: IntelliJ IDEA, API, Spring, додаток.

Об'єкт дослідження – поліплення функціонування диспечерських служб, та автоматизація рутинних процесів.

Предмет дослідження – програмне забезпечення для поліпшення функціонування диспечерський служб.

Мета роботи – розробка програмного забезпечення для поліпшення функціонування диспечерський служб.

Методи дослідження – методи теорії інформації, методи структурного аналізу і проектування, методи розробки програмного забезпечення, методи тестування, валідації та верифікації програмного забезпечення.

В роботі виконано аналіз існуючих електонних додатків для збору документів на транспорт. Встановлено переваги та недоліки існуючих додатків. Проаналізовано можливості середовища розробки IntelliJ IDEA. Розроблено логіку практичних завдань та загальну концепцію представлення інформації для користувачів.

Галузь використання – Програма поліпшить функціонування диспечерський служб а саме збір і контроль документів на транспорт.

ЗМІСТ

ВСТУП

В сучасному світі, в якому кожного дня люди створюють терабайти різних даних, щосекундно постає необхідність зберігати, сортувати, редагувати та переглядати ці дані. Важко уявити собі підприємство, громадську або державну установу, яка не має необхідності у зберіганні великої кількості документації, інформації щодо клієнтів, часу та місць заходів тощо.

В такий важкий, з інформаційної точки зору, час на допомогу людині приходить поняття баз даних. Бази даних покликані спростити роботу з великою кількістю інформації, дати можливість швидко і зрозуміло отримувати з великих архівів даних потрібну інформацію за критеріями.

Виходячи з усього вищесказаного — важко недооцінити актуальність розробки різноманітних інформаційних систем на основі баз даних в різних предметних областях.

Об’єктом дослідження є принципи і методи побудови інформаційних систем.

Предмет дослідження, в свою чергу, представлений аналізом методів і інструментів побудови інформаційних систем і їх використання для розроблення інформаційної системи для роботи диспетчерських служб.

Основна мета роботи — проектування і розроблення інформаційної системи для роботи диспетчерських служб.

Для досягнення поставленої мети слід виконати наступні завдання:

- Дати характеристику предметної галузі
    - Проаналізувати поняття диспетчерської служби
    - Проаналізувати поняття десктопних додатків
    - Проаналізувати методи і засоби розробки десктопних додатків
- Обрати засоби реалізації

- Провести огляд мови програмування
- Провести огляд системи управління базами даних
- Провести огляд середовища розробки

- Реалізувати інформаційну систему
  - Створити діаграму варіантів використання
  - Створити структурну діаграму
  - Створити діаграму класів
  - Створити графічний інтерфейс користувача

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.2. Диспетчерська служба

Диспетчер - це працівник зв’язку, який отримує та передає інформацію для координації операцій іншого персоналу та транспортних засобів, що здійснюють обслуговування. [1] Ряд організацій, включаючи поліцію та пожежні служби, службу екстреної медичної допомоги, кур'єрів на мотоциклах, постачальників таксі, автотранспортні компанії, залізниці та комунальні компанії використовують диспетчерів для передачі інформації, направлення персоналу та координації своїх операцій. [2]

Аварійні диспетчери

Екстрений диспетчер, також відомий як диспетчер громадської безпеки або диспетчер 101, отримує дзвінки від осіб, які потребують екстрених служб, включаючи служби міліції, пожежної та екстреної медичної допомоги. Отримавши інформацію від абонента, диспетчер активує відповідні служби, необхідні для реагування на характер виклику про допомогу. Диспетчер також отримує та передає відповідну інформацію польовим підрозділам, щоб допомогти забезпечити адекватність та безпеку реагування.

Екстрені диспетчери можуть також використовувати заздалегідь затверджені протоколи, щоб розмовляти з абонентом або сторонніми особами через рятувальні медичні процедури, такі як серцево-легенева реанімація, пологи та перша допомога. Вони також можуть вимагати спеціальної сертифікації.

У США близько 10% усіх диспетчерів, зайнятих у 2004 році, були диспетчерами громадської безпеки. [3]

Транспортно-сервісні диспетчери

Ряд інших організацій використовують диспетчерів для реагування на дзвінки у службу, узгодження графіків перевезень та організації доставки матеріалів:

- Диспетчер вантажівок найнятий вантажною компанією для моніторингу доставки вантажу на великі відстані та узгодження графіків прийому та вивезення вантажів.
- Диспетчер автобусів стежить за розкладом руху свого автобусного парку та вирішує будь-які проблеми, що виникають під час їх експлуатації.
- Диспетчер евакуатора відповідає на виклики екстреної допомоги на дорогах.
- Диспетчер газо- та водогосподарських служб стежить за їхніми комунальними послугами та отримує виклики на екстрену допомогу, що стосується газопроводів та магістралей.

У США близько 26% усіх диспетчерів, зайнятих у 2004 році, працювали на транспортній та складській промисловості. [3]

Залізничні диспетчери

Поїзний диспетчер наймається на залізниці для керування та полегшення руху поїздів над визначеною територією, яка зазвичай є частиною чи всією залізничної експлуатуючої частини. Диспетчер також відповідає за економічно ефективний рух поїздів та іншого залізничного обладнання на колії для оптимізації фізичних (поїздів) та людських ресурсів (екіпажів) [4].

На залізниці також працює диспетчер екіпажу для відстеження роботи поїзних бригад та їх доручень. Диспетчер екіпажу відповідає за розподіл складу поїздів до поїздів на основі запланованих списків, а також за необхідність вносити корективи в режимі реального часу на основі умов залізничного руху та затримок. Диспетчеру екіпажу, як правило, допомагає той, хто викликає екіпаж, відповідальність якого полягає в телефонуванні екіпажів поїздів та машин, щоб повідомити їх про час для явки на чергування. Диспетчер екіпажу також відповідає за перевірку того, чи кожен поїзд та

екіпаж машин мають належну кваліфікацію для виконання своїх завдань та відпочили відповідно до трудових норм.

Диспетчери відповідають за моніторинг усіх комунікацій у певній географічній зоні. Диспетчери громадської безпеки несуть відповідальність за всі аварійні комунікації, що відбуваються в межах юрисдикції їхнього відділу. Ці працівники приймають та документують вхідні дзвінки, передають повідомлення відповідному персоналу та ведуть журнали щоденної діяльності свого персоналу. Диспетчери громадської безпеки, як правило, працюють у поліцейському відділенні, пожежному відділенні або лікарні. [3] Інші диспетчери працюють у централізованих центрах зв'язку, пов'язаних із конкретною компанією чи службою.

Диспетчери всіх видів працюють із різними системами зв'язку залежно від їх функцій. Ці системи можуть включати, але не обмежуючись цим, телефони, радіостанції, комп'ютери та програми автоматизованої диспетчеризації, камери відеоспостереження та системи обміну повідомленнями наземного літака, такі як ACARS. В результаті тривалого сидіння та використання такого обладнання у диспетчерів можуть виникнути перенапруження очей та проблеми зі спиною. Багато диспетчерів також повинні працювати ненормовано, щоб забезпечити цілодобове обслуговування, яке включає нічний, вихідний та святковий час. [3]

Диспетчери громадської безпеки, як правило, є першим контактом між аварійними службами та громадськістю. Отримуючи вхідні дзвінки на допомогу, ці диспетчери повинні з’ясувати характер, місце розташування та масштаби надзвичайної ситуації. Телефонуючі, які звертаються за екстреною допомогою, часто перебувають у стані підвищеного емоційного переживання, що може ускладнити отримання інформації, необхідної для належної обробки дзвінка [5]. Умови роботи диспетчера громадської безпеки можуть бути особливо напруженими порівняно з іншими, оскільки неправильна обробка дзвінка може затримати або неправильно спрямувати аварійний персонал, що може призвести до серйозних травм або навіть смерті. [3]

Людська помилка може також дати смертельні результати для інших типів диспетчерів. Диспетчера поїздів в Іспанії було визнано винним у вбивстві з необережності через лобове зіткнення поїзда, яке сталося в червні 2003 р. [6] Дев'ятнадцять людей загинули, а сорок вісім отримали поранення внаслідок аварії, коли диспетчер дозволив пасажирському поїзду покинути станцію, коли товарний поїзд наближався до станції на тій самій лінії [6].

Для працевлаштування диспетчером зазвичай не потрібен рівень освіти вищий, ніж диплом середньої школи, але багато хто, хто працює в цій галузі, мають ступінь гуманітарних наук. Роботодавці віддають перевагу кандидатам, які володіють комп'ютерними та канцелярськими навичками, комунікативними навичками та здатністю швидко працювати під тиском. [3]

Від кандидатів на роботу в якості диспетчерів громадської безпеки може знадобитися скласти письмові, усні випробування або перевірку ефективності та регулюються державними або місцевими нормами. Диспетчерам громадської безпеки може також знадобитися отримати сертифікати та пройти додаткове навчання до або після того, як вони будуть працевлаштовані державними чи місцевими органами влади для відправлення до поліції, пожежних служб або служб екстреної медичної допомоги. Рівень підготовки, необхідний для цих диспетчерів, як правило, найширший у порівнянні з іншими диспетчерськими посадами. [3]

Стандартною вимогою сертифікації для диспетчерів громадської безпеки є сертифікація оператора терміналу на доступ до системи баз даних Національного інформаційного центру про злочинність Федерального бюро розслідувань США (ФБР). Доступ до цієї системи баз даних часто дозволяє отримати додатковий доступ до системи державного рівня, порівнянної з NCIC, яка дозволяє диспетчерам громадської безпеки отримати доступ до інформації про реєстрацію автотранспортних засобів та водійських прав, а також до запитів чи доручень різних правоохоронних органів як загальнодержавних, так і національних.

На додаток до сертифікацій, спеціалістам також потрібно або призначити диспетчерів громадської безпеки. Оскільки диспетчери громадської безпеки є першим контактом між громадськістю та аварійними службами, диспетчери громадської безпеки повинні мати можливість витягати широкий спектр інформації із абонента. Таке спеціалізоване навчання для 911 диспетчерів може включати: втручання в суїциди, ведення переговорів із заручниками, погрози бомб, тактичну диспетчеризацію (для спецназівців), домашнє насильство та протидії тероризму в країні та в країні. Багато з них також проходять навчання в якості екстрених медичних диспетчерів, здатних давати вказівки щодо надання першої медичної допомоги постраждалим або їхнім сім'ям до прибуття EMS.

За даними Бюро статистики праці США, у 2004 році було зайнято 266 000 осіб диспетчерами [3]. Крім того, очікується, що низка діючих диспетчерів або перейдуть на інші професії, або покинуть робочу силу, що призведе до збільшення кількості відкритих дверей. [3]

Основним інструментом диспетчера є диспетчерська консоль. Диспетчерська консоль - це система, яка взаємодіє з приватною або державною радіосистемою, дозволяючи диспетчеру безпосередньо спілкуватися з усіма виїзними працівниками, поліцейськими, персоналом СУО та іншими для координації своєї діяльності. Диспетчери використовують різне апаратне та програмне забезпечення для створення диспетчеризації.

1.2. Десктопні застосунки

Прикладне програмне забезпечення (коротше додаток) - це обчислювальне програмне забезпечення, призначене для виконання конкретного завдання, відмінного від того, що стосується роботи самого комп'ютера, [1] як правило, для використання кінцевими користувачами. Прикладами програми є текстовий процесор та медіаплеєр. Прикладне програмне забезпечення колективного іменника відноситься до всіх додатків

у сукупності. [2] Інші основні класифікації програмного забезпечення - це системне програмне забезпечення, що стосується роботи комп'ютера, та утиліта ("утиліти").

Програми можуть поставлятися в комплекті з комп'ютером та його системним програмним забезпеченням або публікуватися окремо і можуть кодуватися як власні, відкриті джерела чи проекти. [3] Програми, створені для мобільних платформ, називаються мобільними програмами.

В інформаційних технологіях додаток (додаток), прикладна програма або прикладне програмне забезпечення - це комп’ютерна програма, призначена для допомоги людям у здійсненні тієї чи іншої діяльності. Залежно від діяльності, для якої він був розроблений, програма може маніпулювати текстом, цифрами, аудіо, графікою та комбінацією цих елементів. Деякі пакети програм зосереджені на одному завданні, наприклад, на обробці текстів; інші, що називаються інтегрованим програмним забезпеченням, включають кілька програм.

Написане користувачем програмне забезпечення адаптує системи для задоволення конкретних потреб користувача. Написане користувачем програмне забезпечення включає шаблони електронних таблиць, макроси текстового процесора, наукове моделювання, аудіо, графіку та сценарії анімації. Навіть фільтри електронної пошти є своєрідним програмним забезпеченням для користувачів. Користувачі самі створюють це програмне забезпечення і часто не помічають, наскільки воно важливо.

Однак розмежування між системним програмним забезпеченням, таким як операційна система, та прикладним програмним забезпеченням не є точним і іноді є предметом суперечок [5]. Наприклад, одним із ключових запитань у справі «Сполучені Штати проти Microsoft Corp.» про антимонопольний закон було те, чи є веб-браузер Microsoft Internet Explorer частиною його операційної системи Windows або окремим програмним забезпеченням. Як інший приклад, суперечка щодо імен GNU / Linux, зокрема, зумовлена незгодою щодо взаємозв'язку між ядром Linux та операційними системами, побудованими над

цим ядром. У деяких типах вбудованих систем прикладне програмне забезпечення та програмне забезпечення операційної системи можуть не відрізняти користувача, як у випадку програмного забезпечення, що використовується для управління відеомагнітофоном, DVD-програвачем або мікрохвильовою піччю. Наведені вище визначення можуть виключати деякі програми, які можуть існувати на деяких комп'ютерах у великих організаціях.

Слово "додаток", що вживається як прикметник, не обмежується лише значенням "прикладного програмного забезпечення" або що стосується його "[6]. Наприклад, такі поняття, як інтерфейс прикладного програмування (API), сервер додатків, віртуалізація додатків, управління життєвим циклом додатків та портативний додаток стосуються всіх комп'ютерних програм, а не лише прикладного програмного забезпечення.

Деякі програми доступні у версіях для декількох різних платформ; інші працюють лише над одним і тому їх називають, наприклад, географічним додатком для Microsoft Windows, або Android-додатком для навчання, або грою для Linux. Іноді виникає новий і популярний додаток, який працює лише на одній платформі, збільшуючи бажаність цієї платформи. Це називається вбивцею або вбивцею. Наприклад, VisiCalc був першим сучасним програмним забезпеченням для роботи з електронними таблицями для Apple II і допоміг продати нові тоді персональні комп'ютери в офіси. Для Blackberry це було програмне забезпечення для електронної пошти.

В останні роки скорочений термін "додаток" (створений у 1981 році або раніше [7]) став популярним для позначення додатків для мобільних пристроїв, таких як смартфони та планшети, скорочена форма відповідає їх типово меншому обсягу порівняно з програмами на ПК. Ще нещодавно, скорочена версія використовується і для настільного програмного забезпечення.

Існує багато різних та альтернативних способів класифікації прикладного програмного забезпечення.

З юридичної точки зору, прикладне програмне забезпечення в основному класифікується з використанням чорного ящика щодо прав кінцевих кінцевих користувачів або абонентів (з можливим проміжним та багаторівневим рівнями передплати).

Програмні додатки класифікуються також за мовою програмування, на якій пишеться або виконується вихідний код, а також за призначенням та результатами.

За правами власності та користування

Прикладне програмне забезпечення, як правило, виділяють серед двох основних класів: програмне забезпечення із закритим вихідним кодом та програмне забезпечення з відкритим кодом, а також безкоштовне програмне забезпечення або власне програмне забезпечення.

Запатентоване програмне забезпечення підпадає під виключне авторське право, а ліцензія на програмне забезпечення надає обмежені права на використання. Принцип відкрито-закрито стверджує, що програмне забезпечення може бути "відкритим лише для розширення, але не для модифікації". Такі додатки можуть отримувати доповнення лише сторонніми сторонами.

Вільне програмне забезпечення та програмне забезпечення з відкритим кодом слід запускати, розповсюджувати, продавати або розширювати з будь-якою метою, і, будучи відкритим, слід модифікувати або скасовувати таким же чином.

Програмні додатки FOSS, що випускаються за безкоштовною ліцензією, можуть бути безстроковими та також безкоштовно. Можливо, власник, власник або стороння особа, яка виконує будь-які права (авторське право, торгова марка, патент або ius in re aliena), має право додавати винятки, обмеження, затримки чи дати закінчення до ліцензійних умов використання.

Програмне забезпечення публічного домену - це тип FOSS, який є безоплатним і - відкрито чи зарезервовано - може запускатися, розповсюджуватися, модифікуватися, змінюватися, перевидаватися або

створюватися у похідних роботах без будь-якого приписування авторських прав і, отже, відкликання. Його можна навіть продати, але без передачі власності, що перебуває у відкритому доступі, іншим окремим суб'єктам. Загальнодоступне програмне забезпечення може бути випущено відповідно до юридичної заяви про (не) ліцензування, яка застосовує ці умови протягом невизначеного періоду (протягом усього життя або назавжди).

За допомогою мови кодування

З моменту розробки та майже універсального прийняття Інтернету, важливим розрізненням, яке з'явилося, було між веб-додатками - написаними за допомогою HTML, JavaScript та іншими власними веб-технологіями, які, як правило, вимагають наявності в Інтернеті та запуску веб-браузера - і більш традиційні рідні програми, написані будь-якими мовами, доступними для конкретного типу комп'ютерів. У комп'ютерному співтоваристві ведуться суперечки щодо веб-додатків, які замінюють рідні програми для багатьох цілей, особливо на мобільних пристроях, таких як смартфони та планшети. Веб-програми дійсно значно зросли в популярності для деяких цілей, але переваги програм роблять їх навряд чи зникнуть найближчим часом, якщо взагалі коли-небудь. Крім того, вони можуть доповнювати і навіть інтегрувати. [8] [9] [10]

За призначенням і результатом

Прикладне програмне забезпечення також можна розглядати як горизонтальне або вертикальне. [11] [12] Горизонтальні програми є більш популярними та поширеними, оскільки вони є загальним призначенням, наприклад, текстові процесори або бази даних. Вертикальні додатки - це нішеві продукти, призначені для певного виду галузі чи бізнесу або відділу в організації. Інтегровані пакети програмного забезпечення намагатимуться розглянути всі можливі аспекти, наприклад, виробничого або банківського працівника, бухгалтерії чи обслуговування клієнтів.

Існує багато типів прикладного програмного забезпечення: [13]

- Набір програм складається з декількох додатків, що входять до комплекту. Зазвичай вони мають пов’язані функції, функції та користувальницькі інтерфейси і можуть мати можливість взаємодіяти між собою, наприклад відкривати файли один одного. Бізнес-програми часто бувають в люксах, напр. Microsoft Office, LibreOffice та iWork, які об’єднують текстовий процесор, електронну таблицю тощо; але люкси існують для інших цілей, наприклад графіка або музика.
- Корпоративне програмне забезпечення відповідає потребам процесів і потоків даних усієї організації в декількох відділах, часто у великому розподіленому середовищі. Приклади включають системи планування ресурсів підприємства, системи управління взаємовідносинами з клієнтами (CRM) та програмне забезпечення для управління ланцюгами поставок. Департаментське програмне забезпечення - це підвид корпоративного програмного забезпечення, орієнтоване на менші організації чи групи у великій організації. (Приклади включають управління витратами на проїзд та IT-довідкову службу.)
- Корпоративне інфраструктурне програмне забезпечення забезпечує загальні можливості, необхідні для підтримки корпоративних програмних систем. (Приклади включають бази даних, сервери електронної пошти та системи управління мережами та безпекою.)
- Прикладна платформа як послуга (aPaaS) - це служба хмарних обчислень, яка пропонує середовища розробки та розгортання служб додатків.
- Програмне забезпечення інформаційного працівника дозволяє користувачам створювати та керувати інформацією, часто для окремих проектів у відділі, на відміну від управління підприємством. Приклади включають управління часом, управління ресурсами, аналітичні засоби, інструменти спільної роботи та документації. Текстові процесори, електронні таблиці, клієнти електронної пошти та блогу, система

персональної інформації та окремі редактори засобів масової інформації можуть допомогти у виконанні багатьох завдань інформаційного працівника.

- Програмне забезпечення для доступу до вмісту використовується в основному для доступу до вмісту без редагування, але може включати програмне забезпечення, яке дозволяє редагувати вміст. Таке програмне забезпечення відповідає потребам окремих людей та груп у споживанні цифрових розваг та опублікованого цифрового вмісту. (Приклади включають медіаплеєри, веб-браузери та браузери довідки.)
- Навчальне програмне забезпечення пов’язане із програмним забезпеченням доступу до вмісту, але має вміст або функції, пристосовані для використання викладачами чи студентами. Наприклад, він може проводити оцінки (тести), відстежувати прогрес матеріалу або включати можливості спільної роботи.
- Програмне забезпечення для моделювання імітує фізичні або абстрактні системи для дослідницьких, навчальних або розважальних цілей.
- Програмне забезпечення для розробки засобів масової інформації створює друковані та електронні носії для споживання іншими, найчастіше в комерційних або освітніх умовах. Сюди входить програмне забезпечення для графічного мистецтва, програмне забезпечення для настільних видавництв, програмне забезпечення для розробки мультимедіа, редактори HTML, редактори цифрової анімації, цифрові композиції аудіо та відео та багато інших.
- Програмне забезпечення для розробки продуктів використовується при розробці апаратних та програмних продуктів. Це включає автоматизоване проектування (САПР), автоматизоване проектування (CAE), засоби редагування та компіляції комп’ютерної мови, інтегровані середовища розробки та інтерфейси програмістів програм.

- Розважальне програмне забезпечення може стосуватися відеоігор, заставки, програм для відображення кінофільмів або відтворення записаної музики та інших видів розваг, які можна відчути за допомогою використання обчислювального пристрою.

Програми також можна класифікувати за обчислювальними платформами, такими як конкретна операційна система, мережа доставки, наприклад, у хмарних обчисленнях та додатки Web 2.0, або пристрої доставки, такі як мобільні програми для мобільних пристроїв.

Саму операційну систему можна вважати прикладним програмним забезпеченням при виконанні простих завдань обчислення, вимірювання, рендерингу та обробки текстів, які не використовуються для управління апаратним забезпеченням за допомогою інтерфейсу командного рядка або графічного інтерфейсу користувача. Сюди не входить прикладне програмне забезпечення, що входить до складу операційних систем, таких як калькулятор програмного забезпечення або текстовий редактор.

Розробка програмного забезпечення - це процес задуму, уточнення, проектування, програмування, документування, тестування та виправлення помилок, що беруть участь у створенні та обслуговуванні програм, фреймворків чи інших компонентів програмного забезпечення. Розробка програмного забезпечення - це процес написання та підтримання вихідного коду, але в більш широкому розумінні він включає все, що пов’язане від задуму бажаного програмного забезпечення до остаточного прояву програмного забезпечення, іноді у запланованому та структурованому процесі. [1] Тому розробка програмного забезпечення може включати дослідження, нові розробки, створення прототипів, модифікацію, повторне використання, реінжиніринг, технічне обслуговування або будь-яку іншу діяльність, результатом якої є програмні продукти. [2]

Програмне забезпечення може бути розроблено для різних цілей, серед яких три найпоширеніші - для задоволення конкретних потреб конкретного клієнта / бізнесу (у випадку зі спеціальним програмним забезпеченням), для

задоволення сприйнятих потреб певного набору потенційних користувачів (у випадку з комерційними та програмне забезпечення з відкритим кодом), або для особистого користування (наприклад, вчений може написати програмне забезпечення для автоматизації буденного завдання). Розробка вбудованого програмного забезпечення, тобто розробка вбудованого програмного забезпечення, такого як використовується для контролю споживчих товарів, вимагає інтеграції процесу розробки з розробкою контрольованого фізичного продукту. Системне програмне забезпечення лежить в основі програм і самого процесу програмування, і часто розробляється окремо.

Потреба в кращому контролі якості процесу розробки програмного забезпечення породила дисципліну програмної інженерії, яка спрямована на застосування системного підходу, наведеного в інженерній парадигмі, до процесу розробки програмного забезпечення.

Існує багато підходів до управління програмними проектами, відомих як моделі життєвого циклу розробки програмного забезпечення, методології, процеси чи моделі. Модель водоспаду є традиційною версією, на відміну від останніх інновацій у гнучкій розробці програмного забезпечення.

1.3. Розробка десктопних застосунків

Розробка програмного забезпечення - це процес задуму, уточнення, проектування, програмування, документування, тестування та виправлення помилок, що беруть участь у створенні та підтримці програм, фреймворків чи інших програмних компонентів. Розробка програмного забезпечення - це процес написання та підтримання вихідного коду, але в більш широкому розумінні він включає все, що пов'язане від задуму бажаного програмного забезпечення до остаточного прояву програмного забезпечення, іноді в запланованому та структурованому процесі. [1] Отже, розробка програмного забезпечення може включати дослідження, нові розробки, створення прототипів, модифікацію, повторне використання, реінжиніринг, технічне

обслуговування або будь-яку іншу діяльність, результатом якої є програмні продукти. [2]

Програмне забезпечення може бути розроблено для різних цілей, серед яких три найпоширеніші - для задоволення конкретних потреб конкретного клієнта / бізнесу (у випадку зі спеціальним програмним забезпеченням), для задоволення сприйнятих потреб певного набору потенційних користувачів (у випадку з комерційними та програмне забезпечення з відкритим кодом), або для особистого використання (наприклад, вчений може написати програмне забезпечення для автоматизації буденного завдання). Розробка вбудованого програмного забезпечення, тобто розробка вбудованого програмного забезпечення, такого як використовується для контролю споживчих товарів, вимагає інтеграції процесу розробки з розробкою контрольованого фізичного продукту. Системне програмне забезпечення лежить в основі програм і самого процесу програмування, і часто розробляється окремо.

Потреба в кращому контролі якості процесу розробки програмного забезпечення породила дисципліну програмної інженерії, яка спрямована на застосування системного підходу, наведеного в інженерній парадигмі, до процесу розробки програмного забезпечення.

Існує багато підходів до управління програмними проектами, відомих як моделі життєвого циклу розробки програмного забезпечення, методології, процеси чи моделі. Модель водоспаду є традиційною версією, на відміну від останніх інновацій у гнучкій розробці програмного забезпечення.

Процес розробки програмного забезпечення (також відомий як методологія розробки програмного забезпечення, модель або життєвий цикл) - це структура, яка використовується для структурування, планування та управління процесом розробки інформаційних систем. За ці роки розвинулося широке розмаїття таких систем, кожна з яких має свої визнані сильні та слабкі сторони. Існує декілька різних підходів до розробки програмного забезпечення: деякі використовують більш структурований, інженерно-орієнтований підхід до розробки програмного забезпечення, тоді як інші

можуть застосовувати більш поступовий підхід, коли програмне забезпечення розвивається в міру розробки поштучно. Одна методологія розробки системи не обов'язково придатна для використання у всіх проектах. Кожна з доступних методологій найкраще підходить для конкретних видів проектів на основі різних технічних, організаційних, проектних та групових міркувань. [3]

Більшість методологій поділяють деяку комбінацію наступних етапів розробки програмного забезпечення:

- Аналіз проблеми
- Дослідження ринку
- Збір вимог до пропонованого програмного забезпечення
- Розробка плану або дизайну програмного забезпечення
- Впровадження (кодування) програмного забезпечення
- Тестування програмного забезпечення
- Розгортання
- Технічне обслуговування та виправлення помилок

Ці етапи часто називають спільно життєвим циклом розробки програмного забезпечення або SDLC. Різні підходи до розробки програмного забезпечення можуть виконувати ці етапи в різних порядках або приділяти більше або менше часу різним етапам. Рівень деталізації документації, виробленої на кожному етапі розробки програмного забезпечення, також може змінюватися. Ці етапи також можна проводити по черзі (підхід на основі "водоспаду"), або вони можуть повторюватися протягом різних циклів або ітерацій (більш "екстремальний" підхід). Більш екстремальний підхід, як правило, передбачає менше часу, витраченого на планування та документування, і більше часу, витраченого на кодування та розробку автоматизованих тестів. Більш «екстремальні» підходи також сприяють постійному тестуванню протягом усього життєвого циклу розробки, а також постійному наявності працюючого (або без помилок) продукту. Більш структуровані підходи на основі "водоспаду" намагаються оцінити більшість

ризиків та розробити детальний план програмного забезпечення до початку впровадження (кодування), а також уникнути значних змін дизайну та перекодування на пізніх стадіях планування життєвого циклу програмного забезпечення .

Різні методології мають значні переваги та недоліки, і найкращий підхід до вирішення проблеми за допомогою програмного забезпечення часто залежатиме від типу проблеми. Якщо проблема добре зрозуміла і роботу можна ефективно спланувати заздалегідь, то підхід, що базується на «водоспаді», може працювати найкращим чином. Якщо, з іншого боку, проблема унікальна (принаймні для команди розробників) і структуру програмного забезпечення неможливо передбачити легко, то найкраще може підійти більш "екстремальний" поступовий підхід.

Визначення потреби

Джерел ідей для програмних продуктів безліч. Ці ідеї можуть походити від дослідження ринку, включаючи демографічні показники потенційних нових клієнтів, існуючих клієнтів, перспективи продажів, які відмовились від продукту, інші співробітники внутрішньої розробки програмного забезпечення або творча сторона. Ідеї програмних продуктів зазвичай спочатку оцінюються маркетинговим персоналом з точки зору економічної доцільності, відповідності існуючим каналам розподілу, можливого впливу на існуючі асортименти продукції, необхідних функцій та відповідності маркетинговим цілям компанії. На етапі маркетингової оцінки оцінюються припущення щодо витрат і часу. На початку першого етапу приймається рішення щодо того, чи слід, базуючись на більш детальній інформації, отриманій співробітниками відділу маркетингу та розробки, продовжувати реалізовувати проект [4].

У книзі "Великі дебати щодо програмного забезпечення" Алан М. Девіс стверджує в главі "Вимоги", підрозділі "Відсутній шматок розробки програмного забезпечення"

«Студенти інженерних спеціальностей вивчають техніку і рідко піддаються фінансам або маркетингу. Студенти маркетингу вивчають маркетинг і рідко піддаються фінансам або інженерії. Більшість з нас стають фахівцями лише в одній галузі. Щоб ускладнити ситуацію, мало хто з нас зустрічає міждисциплінарних людей у робочій силі, тому є мало можливостей імітувати. Проте планування програмного продукту має вирішальне значення для успіху розробки і абсолютно вимагає знання багатьох дисциплін. [5]»

Оскільки розробка програмного забезпечення може передбачати компрометацію або виходити за межі того, що вимагає клієнт, проект розробки програмного забезпечення може збитися з менш технічних проблем, таких як людські ресурси, управління ризиками, інтелектуальна власність, складання бюджету, антикризове управління тощо. Ці процеси можуть також спричинити роль розвитку бізнесу в перекритті з розробкою програмного забезпечення.

Процес планування

Планування - це мета кожної діяльності, де ми хочемо виявити речі, які належать до проекту. Важливим завданням у створенні програмного забезпечення є вилучення вимог або аналіз вимог. [6] Клієнти зазвичай мають абстрактне уявлення про те, чого вони хочуть як кінцевий результат, але не знають, що слід робити програмному забезпеченню. Кваліфіковані та досвідчені інженери-програмісти визнають неповні, неоднозначні або навіть суперечливі вимоги на даний момент. Часта демонстрація коду в реальному часі може допомогти зменшити ризик неправильних вимог.

"Незважаючи на те, що на етапі вимог докладається багато зусиль, щоб забезпечити їх повне та узгоджене, рідко це трапляється; залишаючи фазу проектування програмного забезпечення як найбільш впливову, коли йдеться про мінімізацію наслідків нових або зміни вимог. Волатильність вимог є складним завданням, оскільки вони впливають на майбутні або вже спрямовані зусилля з розвитку "[7].

Після того, як загальні вимоги будуть зібрані з клієнта, слід визначити і чітко сформулювати аналіз обсягу розробки. Це часто називають документом сфери дії.

Проектування

Після встановлення вимог дизайн програмного забезпечення може бути встановлений у проектному документі програмного забезпечення. Це передбачає попереднє або високорівневе проектування основних модулів із загальним зображенням (наприклад, блок-схемою) того, як деталі збігаються. Наразі мова, операційна система та апаратні компоненти повинні бути відомі. Потім створюється детальний або низькорівневий дизайн, можливо, з прототипуванням як доказом концепції або для підтвердження вимог.

Впровадження, тестування та документування

Впровадження - це частина процесу, коли інженери програмного забезпечення насправді програмують код проекту.

Тестування програмного забезпечення є невід'ємною та важливою фазою процесу розробки програмного забезпечення. Ця частина процесу забезпечує якнайшвидше розпізнавання дефектів. У деяких процесах, загально відомих як тестова розробка, тести можуть розроблятися безпосередньо перед впровадженням і слугувати орієнтиром для правильності реалізації.

Документування внутрішнього дизайну програмного забезпечення з метою подальшого обслуговування та вдосконалення здійснюється протягом усієї розробки. Це може також включати написання API, будь то зовнішній чи внутрішній. Процес розробки програмного забезпечення, обраний командою розробників, визначатиме, наскільки необхідна внутрішня документація (якщо така є). Планові моделі (наприклад, Водоспад), як правило, надають більше документації, ніж моделі Agile.

Розгортання та технічне обслуговування

Розгортання починається безпосередньо після того, як код буде належним чином протестований, затверджений до випуску та проданий або розподілений іншим чином у виробничому середовищі. Це може включати

встановлення, налаштування (наприклад, встановлення параметрів для значень замовника), тестування та, можливо, тривалий період оцінки.

Навчання та підтримка програмного забезпечення є важливими, оскільки програмне забезпечення ефективно лише за умови його правильного використання.

Технічне обслуговування та вдосконалення програмного забезпечення для вирішення нещодавно виявлених несправностей або вимог може зайняти значний час та зусилля, оскільки пропущені вимоги можуть змусити перепроектувати програмне забезпечення. У більшості випадків необхідне регулярне технічне обслуговування, щоб виправити повідомлення про проблеми та підтримувати роботу програмного забезпечення.

Переглянути модель

Модель представлення - це структура, яка забезпечує точки зору на систему та її середовище, які будуть використовуватися в процесі розробки програмного забезпечення. Це графічне представлення основної семантики подання.

Мета точок зору та поглядів полягає в тому, щоб дати можливість інженерам-людям зрозуміти дуже складні системи та організувати елементи проблеми навколо областей знань. При розробці фізично інтенсивних систем точки зору часто відповідають можливостям та відповідальності інженерної організації. [8]

Більшість складних специфікацій системи настільки великі, що ніхто не може повністю зрозуміти всі аспекти специфікацій. Крім того, ми всі маємо різні інтереси в тій чи іншій системі та різні причини для вивчення специфікацій системи. Керівник бізнесу буде задавати різні питання щодо системного складу, ніж той, хто впроваджує систему. Отже, концепція рамки точок зору полягає у наданні окремих точок зору у специфікації даної складної системи. Кожна з цих точок зору задовольняє аудиторію інтересом до певного набору аспектів системи. З кожною точкою зору пов’язана мова точки зору, яка оптимізує словниковий запас та презентацію для аудиторії цієї точки зору.

Парадигма програмування

Парадигма програмування - це фундаментальний стиль комп’ютерного програмування, який, як правило, не продиктований методологією управління проектами (наприклад, водоспад чи спритність). Парадигми відрізняються між собою концепціями та абстракціями, що використовуються для представлення елементів програми (таких як об'єкти, функції, змінні, обмеження) та етапами, що містять обчислення (наприклад, призначення, оцінка, продовження, потоки даних). Іноді концепції, які стверджує парадигма, використовуються спільно для проектування архітектури систем високого рівня; в інших випадках обсяг парадигми програмування обмежується внутрішньою структурою певної програми або модуля.

Мова програмування може підтримувати кілька парадигм. Наприклад, програми, написані на C ++ або Object Pascal, можуть бути суто процедурними, або суто об’єктно-орієнтованими, або містити елементи обох парадигм. Розробники програмного забезпечення та програмісти вирішують, як використовувати ці елементи парадигми. В об'єктно-орієнтованому програмуванні програмісти можуть розглядати програму як сукупність взаємодіючих об'єктів, тоді як у функціональному програмуванні програму можна розглядати як послідовність оцінок функцій без стану. При програмуванні комп’ютерів або систем із багатьма процесорами, орієнтоване на процеси програмування дозволяє програмістам думати про програми як про сукупності одночасних процесів, що діють на логічно спільні структури даних.

Подібно до того, як різні групи програмної інженерії виступають за різні методології, різні мови програмування виступають за різні парадигми програмування. Деякі мови призначені для підтримки однієї парадигми (Smalltalk підтримує об'єктно-орієнтоване програмування, Haskell підтримує функціональне програмування), тоді як інші мови програмування підтримують кілька парадигм (наприклад, Object Pascal, C ++, C #, Visual Basic, Common Lisp, Scheme, Python, Ruby і Oz).

Багато парадигм програмування так добре відомі тим, які методи вони забороняють, як і тим, що вони вмикають. Наприклад, чисто функціональне програмування забороняє використовувати побічні ефекти; структуроване програмування забороняє використовувати оператори goto. Частково з цієї причини нові парадигми часто розглядаються як доктринерські або надто жорсткі для тих, хто звик до більш ранніх стилів. Уникнення певних методів може полегшити доведення теорем про правильність програми або просто зрозуміти її поведінку.

Графічне представлення поточного стану інформації забезпечує дуже ефективний засіб представлення інформації як користувачам, так і розробникам системи.

- Бізнес-модель ілюструє функції, пов’язані з модельованим бізнес-процесом, та організації, які виконують ці функції. Зображуючи діяльність та інформаційні потоки, створюється фундамент для візуалізації, визначення, розуміння та перевірки природи процесу.
- Модель даних надає деталі інформації, яка повинна зберігатися, і є першочерговою для використання, коли кінцевим продуктом є генерація програмного програмного коду для програми або підготовка функціональної специфікації, що допомагає прийняти рішення про покупку або придбання програмного забезпечення. Див. Малюнок праворуч для прикладу взаємодії між бізнес-процесом та моделями даних. [9]

Зазвичай модель створюється після проведення співбесіди, що називається бізнес-аналізом. Співбесіда складається з ведучого, який задає низку запитань, призначених для отримання необхідної інформації, яка описує процес. Інтерв’юера називають ведучим, щоб підкреслити, що саме учасники надають інформацію. Ведучий повинен мати певні знання процесу, що цікавить, але це не настільки важливо, як наявність структурованої методології, за якою запитання задаються експерту процесу. Методологія

важлива, оскільки зазвичай команда фасилітаторів збирає інформацію по об'єкту, і результати інформації від усіх інтерв'юерів повинні збігатися після завершення [9].

Моделі розробляються як такі, що визначають або поточний стан процесу, і в цьому випадку кінцевий продукт називається моделлю моментального знімка "як є", або набір ідей щодо того, що повинен містити процес, що призводить до "що може" -бути "моделлю. Генерація моделей процесів та даних може бути використана для того, щоб визначити, чи є існуючі процеси та інформаційні системи надійними і потребують лише незначних модифікацій чи вдосконалень, або якщо в якості коригувальної дії потрібно повторне проектування. Створення бізнес-моделей - це більше, ніж спосіб перегляду або автоматизації вашого інформаційного процесу. Аналіз може бути використаний для кардинальної зміни способу ведення операцій вашим бізнесом або організацією. [9]

Комп’ютерна автоматизована програмна інженерія (CASE) в галузі програмної інженерії - це наукове застосування набору програмних засобів та методів для розробки програмного забезпечення, що призводить до отримання високоякісних програмних продуктів без дефектів та ремонтопридатності.[10] Це також стосується методів розробки інформаційних систем разом з автоматизованими засобами, які можуть бути використані в процесі розробки програмного забезпечення. [11] Термін "автоматизована інженерія програмного забезпечення" (CASE) може позначати програмне забезпечення, що використовується для автоматизованої розробки системного програмного забезпечення, тобто комп'ютерний код. Функції CASE включають аналіз, проектування та програмування. Інструменти CASE автоматизують методи проектування, документування та створення структурованого комп’ютерного коду на потрібній мові програмування. [12]

Дві ключові ідеї автоматизованого програмного забезпечення (CASE):

- Сприяти комп’ютерній допомозі в розробці програмного забезпечення та процесах обслуговування програмного забезпечення, та

- Інженерний підхід до розробки та обслуговування програмного забезпечення.

Типові інструменти CASE існують для управління конфігурацією, моделювання даних, перетворення моделі, рефакторингу, генерації вихідного коду.

Інтегроване середовище розробки

Інтегроване середовище розробки (IDE), також відоме як інтегроване середовище проектування або інтегроване середовище налагодження, є програмним додатком, що надає комплексні засоби для програмістів для розробки програмного забезпечення. Зазвичай IDE складається з:

- Редактор вихідного коду,
- Укладач або перекладач,
- Побудувати засоби автоматизації
- Налагоджувач (зазвичай).

IDE розроблені для максимізації продуктивності програміста, забезпечуючи щільні компоненти з подібними інтерфейсами користувача. Зазвичай IDE присвячується певній мові програмування, щоб забезпечити набір функцій, який найбільш відповідає парадигмам програмування мови.

Визначення повторного використання програмного забезпечення - це процес створення програмного забезпечення із заздалегідь визначених програмних компонентів. Підхід повторного використання програмного забезпечення прагне збільшити або максимально використати наявні артефакти програмного забезпечення у життєвому циклі розробки програмного забезпечення.

Нижче наведено деякі загальні методи повторного використання програмного забезпечення:

- Програмне забезпечення - це багаторазове проектування або реалізація програмної системи або підсистеми.

- Інженерія програмного забезпечення на основі компонентів передбачає інтеграцію разом існуючих компонентів для створення програми.
- Сервісно-орієнтовані архітектури або сервісно-орієнтоване програмування спираються на концепцію компонентів для надання мережевих послуг, таких як веб-послуги.
- Лінійки програмних продуктів прагнуть розробити програмне забезпечення, засноване на загальному наборі основних активів та процесів, з тим щоб виробляти цілий ряд продуктів (або "додатків") для певного ринку.
- API (Інтерфейс прикладного програмування, встановлює набір "визначень підпрограм, протоколів та інструментів для побудови прикладного програмного забезпечення", які можна використовувати в майбутніх збірках.
- Документація з відкритим кодом за допомогою таких бібліотек, як GitHub, забезпечує безкоштовний код для розробників програмного забезпечення для повторного використання та впровадження в нові програми або конструкції.

2 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Вибір мови програмування

Java - це об'єктно-орієнтована мова програмування, що базується на класах, і розроблена так, щоб мати якомога менше залежностей від реалізації. Це мова програмування загального призначення, призначена для того, щоб розробники програм могли писати один раз, запускати їх де завгодно (WORA) [16], що означає, що скомпільований код Java може працювати на всіх платформах, що підтримують Java, без необхідності перекомпіляції. [17] Програми Java зазвичай компілюються в байт-код, який може працювати на будь-якій віртуальній машині Java (JVM), незалежно від базової архітектури комп'ютера. Синтаксис Java подібний до C та C ++, але має менше засобів низького рівня, ніж будь-який з них. Час виконання Java забезпечує динамічні можливості (такі як відображення та модифікація коду середовища виконання), які, як правило, недоступні в традиційних компільованих мовах. Станом на 2019 рік, Java була однією з найпопулярніших мов програмування, що використовується згідно з GitHub, [18] [19], особливо для веб-додатків клієнт-сервер, із 9 мільйонами розробників, про які повідомляється [20].

Спочатку Java була розроблена Джеймсом Гослінгом у Sun Microsystems (яку з тих пір придбала Oracle) і випущена в 1995 році як основний компонент Java-платформи Sun Microsystems. Оригінальні та довідкові реалізатори Java-компіляторів, віртуальних машин та бібліотек класів були спочатку випущені Sun під власні ліцензії. Станом на травень 2007 року, згідно з вимогами Процесу спільноти Java, Sun здійснила ліцензію на більшість своїх технологій Java під загальною публічною ліцензією GNU. Oracle пропонує свою власну віртуальну машину HotSpot Java, однак офіційним посиланням є OpenJDK JVM, яке є безкоштовним програмним забезпеченням з відкритим кодом і

використовується більшістю розробників і є JVM за замовчуванням майже для всіх дистрибутивів Linux.

Станом на березень 2021 року останньою версією є Java 16, з Java 11, яка наразі підтримується довгостроковою підтримкою (LTS), випущена 25 вересня 2018 р. Oracle випустила останнє публічне оновлення із застарілою версією Java 8 LTS у січні 2019 року для комерційного використання, хоча в іншому випадку він все ще підтримуватиме Java 8 із загальнодоступними оновленнями для особистого користування на невизначений час. Інші постачальники почали пропонувати збірки OpenJDK 8 і 11 з низькою вартістю, які все ще отримують покращення безпеки та інші оновлення.

Oracle (та інші) настійно рекомендують видалити застарілі версії Java через серйозні ризики через невирішені проблеми безпеки. [21] Оскільки Java 9, 10, 12, 13, 14 і 15 більше не підтримуються, Oracle радить своїм користувачам негайно перейти на останню версію (на даний момент Java 16) або випуск LTS.

Джеймс Гослінг, Майк Шерідан та Патрік Нотон ініціювали мовний проект Java у червні 1991 р. [22] Спочатку Java була розроблена для інтерактивного телебачення, але на той час вона була надто розвиненою для галузі цифрового кабельного телебачення [23]. Спочатку мову називали Дуб на честь дуба, що стояв біля кабінету Гослінга. Пізніше проект пішов під назвою Зелений і, нарешті, був перейменований на Java, з кави Java, різновиду кави з Індонезії [24]. Гослінг розробив Java із синтаксисом стилю C / C ++, який системні та прикладні програмісти знайдуть добре. [25]

Sun Microsystems випустила першу публічну реалізацію як Java 1.0 у 1996 році. [26] Він обіцяв функцію «Написати один раз, запустити де завгодно» (WORA), забезпечуючи безкоштовний час роботи на популярних платформах. Досить безпечний і з можливістю налаштування безпеки, він дозволяв обмеження доступу до мережі та файлів. Основні веб-браузери незабаром включили можливість запуску аплетів Java на веб-сторінках, і Java швидко стала популярною. Компілятор Java 1.0 був переписаний на Java Артур

ван Гоффом, щоб суворо відповідати специфікації мови Java 1.0. [27] З появою Java 2 (випущений спочатку як J2SE 1.2 у грудні 1998 - 1999), нові версії мали кілька конфігурацій, побудованих для різних типів платформ. J2EE включає технології та API для корпоративних програм, як правило, працюють у серверних середовищах, тоді як J2ME пропонує API, оптимізовані для мобільних додатків. Настільна версія була перейменована в J2SE. У 2006 році для маркетингових цілей Sun перейменував нові версії J2 на Java EE, Java ME та Java SE відповідно.

У 1997 році Sun Microsystems звернувся до органу зі стандартів ISO / IEC JTC 1, а згодом і до Міжнародної організації екзаменаційних технологій (Ecma International), щоб офіційно оформити Java, але незабаром він відмовився від цього процесу [28] [29] [30]. Java залишається фактичним стандартом, який контролюється через процес спільноти Java. [31] Свого часу Sun зробив більшість своїх реалізацій Java доступними безкоштовно, незважаючи на статус власного програмного забезпечення. Sun приносив дохід від Java за рахунок продажу ліцензій на спеціалізовані продукти, такі як Java Enterprise System.

13 листопада 2006 року Sun випустила більшу частину своєї віртуальної машини Java (JVM) як вільне програмне забезпечення з відкритим кодом (FOSS) на умовах Загальної публічної ліцензії GNU (GPL). 8 травня 2007 року Sun завершив процес, зробивши весь основний код свого JVM доступним за умовами вільного програмного забезпечення / розповсюдження з відкритим кодом, окрім невеликої частини коду, на яку Sun не належав авторських прав. [32]

Віце-президент Sun, Річ Грін, сказав, що ідеальна роль Sun у відношенні Java - це євангеліст [33]. Після придбання корпорацією Oracle корпорації Sun Microsystems у 2009–10 рр. Oracle назвала себе розпорядником технологій Java, що невпинно прагне сприяти розвитку спільноти участі та прозорості [34]. Це не завадило Oracle незабаром подати позов проти Google за використання Java всередині Android SDK (див. Розділ Android).

2 квітня 2010 року Джеймс Гослінг звільнився з Oracle [35].

У січні 2016 року Oracle оголосив, що середовища виконання Java, засновані на JDK 9, припинять роботу плагіна браузера. [36]

Програмне забезпечення Java працює на всьому: від ноутбуків до центрів обробки даних, ігрових консолей до наукових суперкомп'ютерів. [37]

Однією з цілей дизайну Java є портативність, що означає, що програми, написані для платформи Java, повинні працювати аналогічно на будь-якій комбінації обладнання та операційної системи з адекватною підтримкою часу роботи. Це досягається компіляцією коду мови Java у проміжне подання, що називається байт-кодом Java, а не безпосередньо в машинний код, специфічний для архітектури. Інструкції байт-коду Java аналогічні машинному коду, але вони призначені для виконання віртуальною машиною (ВМ), написаною спеціально для апаратного забезпечення хоста. Кінцеві користувачі зазвичай використовують середовище виконання Java (JRE), встановлене на їх машині для окремих програм Java або у веб-браузері для аплетів Java.

Стандартні бібліотеки забезпечують загальний спосіб доступу до особливостей хоста, таких як графіка, створення потоків та створення мереж.

Використання універсального байт-коду спрощує перенесення. Однак накладні витрати на інтерпретацію байт-коду в машинних інструкціях робили інтерпретовані програми майже завжди більш повільними, ніж власні виконувані файли. Компілятори JIT, які компілюють байт-коди до машинного коду під час виконання, були введені з ранньої стадії. Сама Java не залежить від платформи і пристосована до конкретної платформи, на якій вона повинна працювати віртуальною машиною Java (JVM) для неї, яка переводить байт-код Java на машинну мову платформи. [46]

Програми, написані на Java, мають репутацію повільніших і вимагають більше пам'яті, ніж програми, написані на C ++. [47] [48] Однак швидкість виконання програм Java значно покращилася завдяки введенню вчасно зведеної компіляції в 1997/1998 для Java 1.1 [49], додавання функцій мови, що

підтримують кращий аналіз коду (наприклад, внутрішні класи, клас StringBuilder, необов’язкові твердження тощо), та оптимізації у віртуальній машині Java, такі як HotSpot, яка стала стандартною JVM Sun у 2000 році. У Java 1.5 продуктивність була покращена додаванням пакета java.util.concurrent, включаючи реалізації ConcurrentMaps та інших багатоядерних колекцій, що не містять блокування, і додатково покращена за допомогою Java 1.6.

Деякі платформи пропонують пряму апаратну підтримку Java; є мікроконтролери, які можуть запускати байт-код Java в апаратному забезпеченні, а не на програмній віртуальній машині Java [50], а деякі процесори на базі ARM можуть мати апаратну підтримку для виконання байт-коду Java за допомогою опції Jazelle, хоча підтримка здебільшого відмовлена в поточних реалізаціях APM.

Java використовує автоматичний збирач сміття для управління пам’яттю в життєвому циклі об’єкта. Програміст визначає, коли створюються об'єкти, а час виконання Java відповідає за відновлення пам'яті, коли об'єкти більше не використовуються. Як тільки посилання на об'єкт не залишаються, недоступна пам'ять стає придатною для автоматичного звільнення збирачем сміття. Щось подібне до витоку пам’яті все одно може статися, якщо код програміста містить посилання на об’єкт, який більше не потрібен, як правило, коли об’єкти, які більше не потрібні, зберігаються в контейнерах, які все ще використовуються. Якщо викликаються методи для неіснуючого об'єкта, викидається нульовий виняток покажчика. [51] [52]

Однією з ідей, що стоять за моделлю автоматичного управління пам'яттю Java, є те, що програмісти можуть бути позбавлені тягаря необхідності виконувати ручне управління пам’яттю. У деяких мовах пам’ять для створення об’єктів неявно виділяється в стеку або явно виділяється та звільняється з купи. В останньому випадку відповідальність за управління пам'яттю покладається на програміста. Якщо програма не вивільняє об'єкт, відбувається витік пам'яті. Якщо програма намагається отримати доступ або

звільнити пам'ять, яка вже була вивільнена, результат невизначений і важко передбачуваний, і програма, ймовірно, стане нестабільною або вийде з ладу. Це можна частково виправити за допомогою розумних покажчиків, але це додає додаткових витрат та складності. Зверніть увагу, що збирання сміття не запобігає витоку логічної пам'яті, тобто тих, де пам’ять все ще посилається, але ніколи не використовується.

Вивіз сміття може відбуватися в будь-який час. В ідеалі це відбувається, коли програма не працює. Це гарантовано спрацьовує, якщо в купі недостатньо вільної пам’яті для виділення нового об’єкта; це може призвести до того, що програма на мить зупиниться. Явне управління пам'яттю неможливе в Java.

Java не підтримує арифметику покажчика стилю C / C ++, де адресами об'єктів можна арифметично маніпулювати (наприклад, додаванням або відніманням зміщення). Це дозволяє збиральнику сміття переміщувати об'єкти, на які посилаються, і забезпечує безпеку та захист типу.

Як і в C ++ та деяких інших об'єктно-орієнтованих мовах, змінні примітивних типів даних Java зберігаються або безпосередньо в полях (для об'єктів), або в стеку (для методів), а не в купі, як це зазвичай стосується непримітивних даних типи (але див. аналіз втечі). Це було свідоме рішення дизайнерів Java з міркувань продуктивності.

Java містить кілька типів збирачів сміття. Починаючи з Java 9, HotSpot використовує Garbage First Garbage Collector (G1GC) за замовчуванням. [53] Однак є також кілька інших збирачів сміття, які можна використовувати для управління купою. Для більшості програм на Java достатньо G1GC. Раніше в Java 8 використовувався паралельний збирач сміття.

Вирішивши проблему управління пам’яттю, не звільняє програміста від тягаря належної обробки інших видів ресурсів, таких як підключення до мережі або бази даних, дескриптори файлів тощо, особливо за наявності винятків.

На синтаксис Java значною мірою впливають C ++ та C. На відміну від C ++, який поєднує синтаксис для структурованого, загального та об’єктно-орієнтованого програмування, Java будувалася майже виключно як об’єктно-орієнтована мова. [17] Весь код пишеться всередині класів, і кожен елемент даних є об'єктом, за винятком примітивних типів даних (тобто цілих чисел, чисел з плаваючою комою, булевих значень та символів), які не є об'єктами з міркувань продуктивності. Java повторно використовує деякі популярні аспекти C ++ (наприклад, метод printf).

На відміну від C ++, Java не підтримує перевантаження оператора [54] або множинне успадкування для класів, хоча для інтерфейсів підтримується багаторазове успадкування. [55]

Java використовує коментарі, подібні до коментарів на C ++. Існує три різних стилі коментарів: стиль одного рядка, позначений двома косими рисками (//), стиль декількох рядків, відкритий / * і закритий * /, і стиль коментування Javadoc відкритий / ** і закритий * / . Стиль коментування Javadoc дозволяє користувачеві запускати виконуваний файл Javadoc для створення документації для програми і може бути прочитаний деякими інтегрованими середовищами розробки (IDE), такими як Eclipse, щоб дозволити розробникам отримати доступ до документації в IDE.

Синтаксис

Усі вихідні файли мають бути названі за загальнодоступним класом, який вони містять, додаючи суфікс .java, наприклад, HelloWorldApp.java. Спочатку його потрібно скомпілювати в байт-код, використовуючи компілятор Java, створюючи файл із суфіксом .class (у цьому випадку HelloWorldApp.class). Тільки тоді його можна виконати або запустити. Вихідний файл Java може містити лише один загальнодоступний клас, але він може містити кілька класів із модифікатором непублічного доступу та будь-яку кількість загальнодоступних внутрішніх класів. Коли вихідний файл містить декілька класів, необхідно зробити один клас (введений ключовим

словом класу) загальнодоступним (перед ним - загальнодоступне ключове слово) і назвати вихідний файл цим загальнодоступним іменем класу.

Клас, який не оголошено загальнодоступним, може зберігатися у будь-якому файлі .java. Компілятор створить файл класу для кожного класу, визначеного у вихідному файлі. Ім'я файлу класу - це ім'я класу з доданим .class. Для генерації файлів класів анонімні класи обробляються так, ніби їх ім'я є об'єднанням імені їхнього класу, що містить, $ та ціле число.

Ключове слово public означає, що метод може бути викликаний з коду в інших класах, або що клас може використовуватися класами поза ієрархією класів. Ієрархія класів пов’язана з іменем каталогу, в якому знаходиться файл .java. Це називається модифікатором рівня доступу. Інші модифікатори рівня доступу включають ключові слова private (метод, до якого можна отримати доступ лише в тому ж класі) та захищений (що дозволяє отримати доступ до коду з того самого пакету). Якщо фрагмент коду намагається отримати доступ до приватних методів або захищених методів, JVM видасть виняток SecurityException

Ключове слово static [18] перед методом вказує на статичний метод, який пов'язаний лише з класом, а не з будь-яким конкретним екземпляром цього класу. Тільки статичні методи можна викликати без посилання на об'єкт. Статичні методи не можуть отримати доступ до будь-яких членів класу, які також не є статичними. Методи, які не позначаються статичними, є методами екземпляра та потребують роботи певного екземпляра класу.

Ключове слово void означає, що основний метод не повертає жодного значення абоненту. Якщо програма Java повинна вийти з кодом помилки, вона повинна явно викликати System.exit ().

Назва методу main не є ключовим словом у мові Java. Це просто назва методу, який запускає програма запуску Java для передачі управління програмою. Класи Java, що працюють в керованих середовищах, таких як аплети та Enterprise JavaBeans, не використовують або потребують методу main (). Програма Java може містити кілька класів, які мають основні методи,

а це означає, що віртуальній машині потрібно явно сказати, з якого класу запускати.

Основний метод повинен приймати масив об'єктів String. За домовленістю, він згадується як аргументи, хоча може використовуватися будь-яка інша юридична назва ідентифікатора. Починаючи з Java 5, основний метод також може використовувати змінні аргументи у формі відкритого статичного void main (String ... args), що дозволяє викликати основний метод із довільною кількістю аргументів String. Ефект цього альтернативного оголошення семантично ідентичний (параметру args, який все ще є масивом об’єктів String), але він дозволяє альтернативний синтаксис для створення та передачі масиву.

Запуск Java запускає Java шляхом завантаження заданого класу (зазначеного в командному рядку або як атрибут у JAR) та запуску його загальнодоступного методу static void main (String []). Окремі програми повинні чітко декларувати цей метод. Параметр args String [] - це масив об’єктів String, що містить будь-які аргументи, передані в клас. Параметри до main часто передаються за допомогою командного рядка.

Друк є частиною стандартної бібліотеки Java: Клас System визначає загальнодоступне статичне поле, яке викликається. Об'єкт out є екземпляром класу PrintStream і забезпечує безліч методів друку даних до стандартного виводу, включаючи println (String), який також додає новий рядок до переданого рядка.

Java була обрана основною мовою розробки даного проекту з огляду на усе вищеописане, а саме — на функціонал, який підтримує дана мова, операційні системи, які підтримуються та простота вивчення.

Загалом — функціонал Java найкраще підходить під дану розробку, за рахунок простої реалізації основних принципів ООП роботи з аналізом зображень

2.2. Вибір середовища розробки

IntelliJ IDEA - це інтегроване середовище розробки (IDE), написане на Java для розробки комп'ютерного програмного забезпечення. Він розроблений JetBrains (раніше відомий як IntelliJ) і доступний як ліцензована спільнота Apache 2 [6] та у власній комерційній версії. Обидва вони можуть бути використані для комерційного розвитку. [7]

Перша версія IntelliJ IDEA була випущена в січні 2001 року і була однією з перших доступних IDE Java з розширеними можливостями навігації коду та можливостями рефакторингу коду. [8] [9]

У звіті InfoWorld за 2010 рік IntelliJ отримав найвищий бал центру тестування з чотирьох найкращих інструментів програмування Java: Eclipse, IntelliJ IDEA, NetBeans та JDeveloper. [10]

У грудні 2014 року Google оголосив версію 1.0 Android Studio, IDE з відкритим вихідним кодом для програм для Android, засновану на виданні спільноти з відкритим кодом IntelliJ IDEA. [11] Інші середовища розробки, засновані на рамках IntelliJ, включають AppCode, CLion, DataGrip, GoLand, PhpStorm, PyCharm, Rider, RubyMine, WebStorm та MPS. [12]

IDE надає певні функції [15], такі як заповнення коду шляхом аналізу контексту, навігація кодом, яка дозволяє перейти до класу або оголошення в коді безпосередньо, рефакторинг коду, налагодження коду [16], встановлення посилань та можливості виправлення невідповідностей за допомогою пропозицій.

IDE забезпечує [15] інтеграцію з інструментами складання / упаковки, такими як grunt, bower, gradle та SBT. Він підтримує системи контролю версій, такі як Git, Mercurial, Perforce та SVN. Бази даних, такі як Microsoft SQL Server, Oracle, PostgreSQL, SQLite і MySQL, можна отримати безпосередньо з IDE у версії Ultimate через вбудовану версію DataGrip.

Дане середовище розробки було обрано по декількох критеріях:

- Підтримка мови Java

- Інтуїтивно зрозумілий інтерфейс
- Можливості графічного редактору, підходящі до задач

Так як усі критерії задовільнено — середовищем розробки було обрано саме IntelliJ IDEA 2021.1.

2.3. Вибір СКБД

Бази даних використовуються для структурованого зберігання даних, а SQLite - це популярний формат баз даних, що з'являється у багатьох мобільних системах, а також традиційних операційних системах. SQLite - це технологічна бібліотека, яка реалізує самостійний, безсерверний, механізм баз даних SQL з нульовою конфігурацією. Код SQLite знаходиться у відкритому доступі і, отже, безкоштовний для будь-яких цілей, комерційних або приватних.

SQLite - це вбудований механізм баз даних SQL. На відміну від більшості інших баз даних SQL, SQLite не має окремого серверного процесу. SQLite читає та пише безпосередньо на звичайні файли диска. Повна база даних SQL з кількома таблицями, індексами, тригерами та поданнями міститься в одному дисковому файлі. Як правило, SQLite працює швидше, чим більше пам'яті ви йому надаєте. Тим не менше, продуктивність, як правило, досить хороша навіть у середовищах з низьким обсягом пам'яті. Залежно від того, як він використовується, SQLite може бути швидшим, ніж прямий ввід / вивід файлової системи [33].

SQLite дуже ретельно перевіряється перед кожним випуском і має репутацію дуже надійного. Більшість вихідних кодів SQLite присвячені виключно тестуванню та верифікації. Автоматизований набір тестів запускає мільйони і мільйони тестових випадків, що включають сотні мільйонів окремих операторів SQL, і забезпечує 100% покриття тестуванням філій. SQLite витончено реагує на помилки розподілу пам'яті та помилки вводу-

виводу диска. Все це перевіряється автоматизованими тестами за допомогою спеціальних тестових джгутів, які імітують відмови системи. Звичайно, навіть при всьому цьому тестуванні все ще існують помилки. Але на відміну від деяких подібних проектів (особливо комерційних конкурентів), SQLite відкрито та чесно ставиться до всіх помилок і надає списки помилок та щохвилинні хронології змін коду.

Отож, для наглядного огляду переваг взято одного з конкурентів SQLite, а саме SQL Server, і в свою чергу було здійснено їхнє коротке порівняння:

- Модель ціноутворення:
    - SQLite безкоштовний;
    - SQL Server - це рекламний ролик із обмеженою безкоштовною версією;
- Серверні операційні системи:
    - SQLite не потребує сервера;
    - SQL Server працює на Linux та Windows;
- API та інші методи доступу:
        - SQLite підтримує такі драйвери:
            - ADO.NET;
            - JBDC;
            - ODBC;
        - SQL Server підтримує:
            - OLE DB;
            - ADO.NET;
            - JBDC;
            - ODBC;
            - Табличний потік даних (TDS);
- Підтримувані мови програмування:
    - SQL Server підтримує такі мови програмування (C#, C++, Delphi, Go, Java, JavaScript (Node.js), PHP, Python, R, Ruby, Visual Basic);

    - SQLite підтримує майже будь-які мови програмування, про які ви можете подумати (Actionscript, Ada, Basic, C, C#, C++, D, Delphi, Forth, Fortran, Haskell, Java, JavaScript, Lisp, Lua, MatLab, Objective-C, OCaml, Perl, PHP, PL/SQL, Python, Ruby, Scala, Scheme, Smalltalk, Tclx);
- Підтримка збереженої процедури:
    - SQLite не підтримує збережену процедуру;
    - SQL Server має його з мовами Transact SQL та .NET;
- Методи розділення:
    - SQLite не підтримує;
    - У SQL Server таблиці можуть розподілятися між кількома файлами (горизонтальне розділення);
- Методи реплікації:
    - SQLite не підтримує;
    - SQL Server підтримує;
- Контроль доступу користувачів:
    - SQLite не має концепції контролю доступу користувачів;

SQL Server має чіткі права доступу відповідно до стандарту SQL.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1. Діаграма варіантів використання

Діаграма використання найпростіша - це представлення взаємодії користувача із системою, яка показує взаємозв'язок між користувачем та різними випадками використання, в яких користувач бере участь. Діаграма випадків використання може ідентифікувати різні типи користувачів системи та різні випадки використання, і часто вона супроводжується також іншими типами діаграм. Варіанти використання представлені кругами або еліпсами.

Незважаючи на те, що сам випадок використання може детально вивчити кожну можливість, діаграма прикладів використання може допомогти забезпечити огляд системи на більш високому рівні. Раніше вже було сказано, що "схеми використання - це принципи вашої системи".

Через їх спрощений характер, схеми використання можуть бути хорошим інструментом комунікації для зацікавлених сторін. Креслення намагаються імітувати реальний світ і дають зацікавленій стороні уявлення про те, як буде розроблена система. Сіау та Лі провели дослідження, щоб визначити, чи взагалі існувала дійсна ситуація для схем використання або вони були непотрібними. Було виявлено, що діаграми випадків використання передають намір системи більш спрощеним чином зацікавленим сторонам і що вони "інтерпретуються більш повно, ніж діаграми класів".

Метою діаграми використання є відображення динамічного аспекту системи. Додаткові схеми та документація можуть бути використані для забезпечення повного функціонального та технічного уявлення про систему. Вони забезпечують спрощене та графічне представлення того, що система насправді повинна робити.

Елементи:

- рамки системи (англ. system border) - прямоугольник із назвою у верхніх частинах та еліпсами (прецедентами) всередині. Часто може бути опущено без корисної інформації про полезну інформацію,
- актор (англ. актор) - стилізований людський персонаж, обозначаючий набір ролей користувача (розуміється в широкому змісті: людина, внешняя сутність, клас, інша система), взаємодіючого з деякою сутністю (системною, підсистемою, класом). Актори не можуть бути пов'язані між собою з іншим (за вимкнення відносин щодо обробки / дослідження),
- прецедент - еліпс із надписом, що обозначає виконувану систематичну дію (може включати можливі варіанти), що призводить до спостерігаємих акторами результатів. Надпис може бути ім'ям або описом (з точки зору актора) того, "що" робить система (а не "як"). Ім прецедента зв’язано з неперервним (атомарним) сценарієм - конкретною послідовністю дій, ілюстріруючою повіддю. Під час сценарію актори обмінюються із систематичними повідомленнями. Сценарій може бути приведений на діаграмі прецедентів у відео UML-коментарі. З одним прецедентом може бути пов'язано кілька різних сценаріїв

На рисунку 3.1 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

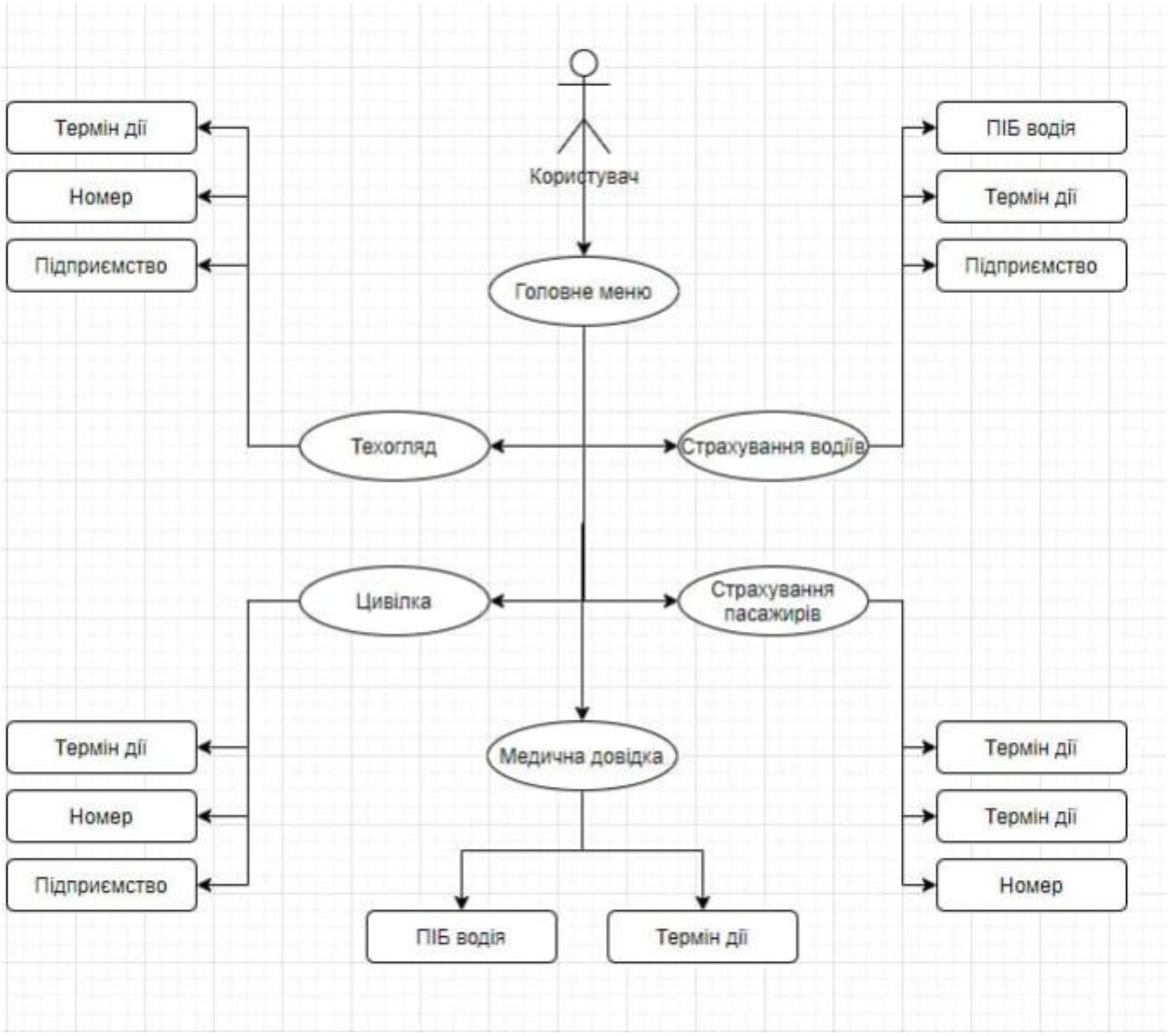


Рис. 3.1 — Діаграма варіантів використання

3.2. Структурна діаграма ІС

Структурна діаграма інформаційної системи, зображена на рисунку 3.2, показує внутрішню будову системи, описуючи взаємодію між внутрішніми модулями.

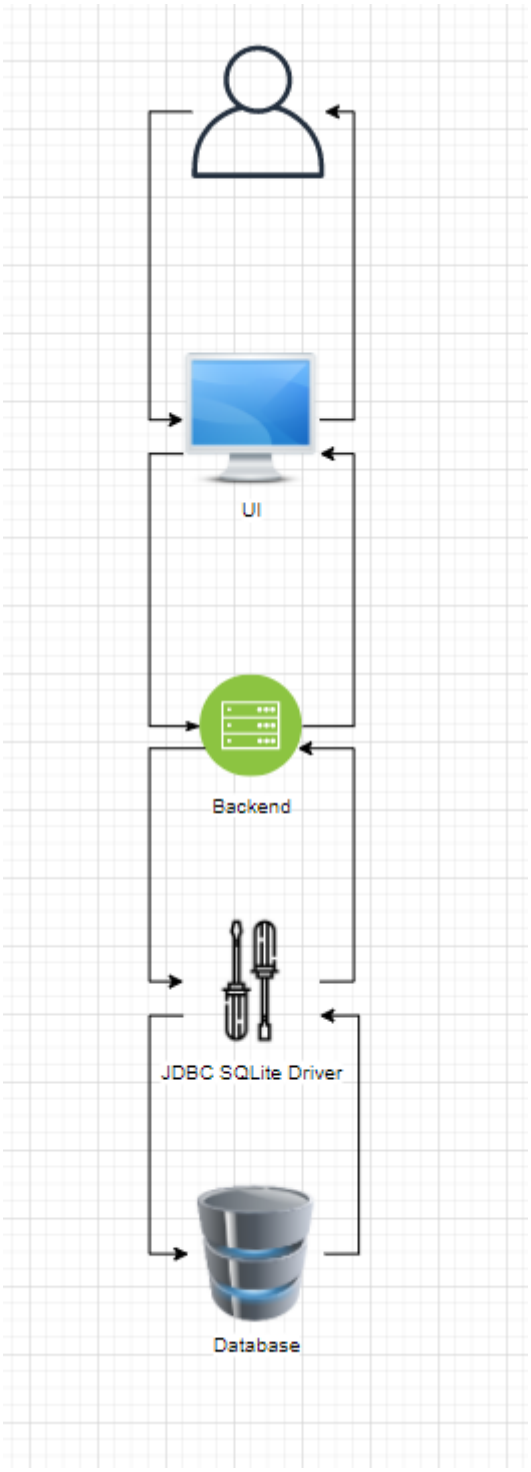


Рис. 3.2 — Структурна діаграма ІС

3.3. Діаграма класів програмного продукту

У програмній інженерії діаграма класів в Уніфікованій мові моделювання (UML) - це тип статичної структурної діаграми, що описує структуру системи, показуючи класи системи, їх атрибути, операції (або методи) та взаємозв'язки між об'єктами.

Діаграма класів є основним будівельним елементом об’єктно-орієнтованого моделювання. Він використовується для загального концептуального моделювання структури програми та для детального моделювання переведення моделей у програмовий код. Діаграми класів також можуть бути використані для моделювання даних. Класи на діаграмі класів представляють як основні елементи, взаємодії в програмі, так і класи, що програмуються.

На схемі класи представлені вікнами, які містять три відділення:

- У верхньому відділенні міститься назва класу. Надруковано жирним шрифтом і відцентровано, а перша літера написана великими літерами.
- Середній відсік містить атрибути класу. Вони вирівняні за лівим краєм, а перша буква мала.
- У нижньому відділенні містяться операції, які може виконувати клас. Вони також вирівняні за лівим краєм, а перша буква - мала.

При проектуванні системи ряд класів ідентифікується та згруповується у схему класів, яка допомагає визначити статичні відносини між ними. При детальному моделюванні класи концептуального проекту часто поділяються на ряд підкласів.

Залежність - це семантичний зв’язок між залежними та незалежними елементами моделі. Він існує між двома елементами, якщо зміни у визначенні одного елемента (сервера або цілі) можуть спричинити зміни для іншого (клієнта або джерела). Ця асоціація є односпрямованою. Залежність

відображається у вигляді штрихової лінії з відкритою стрілкою, яка вказує від клієнта до постачальника.

Для подальшого опису поведінки систем ці діаграми класів можуть бути доповнені діаграмою стану або машиною стану UML.

Асоціація представляє родину посилань. Двійкова асоціація (з двома кінцями) зазвичай представляється у вигляді рядка. Асоціація може пов’язувати будь-яку кількість класів. Асоціація з трьома ланками називається потрійною асоціацією. Асоціацію можна назвати, а кінці асоціації можна прикрасити іменами ролей, показниками власності, кратністю, видимістю та іншими властивостями.

Існує чотири різні типи асоціацій: двонаправлена, односпрямована, агрегаційна (включає агрегацію композиції) та рефлексивна. Двонаправлені та односпрямовані асоціації є найбільш поширеними.

Наприклад, клас польоту асоціюється з класом літака двонаправлено. Асоціація представляє статичне відношення, яке ділиться між об'єктами двох класів.

Агрегація є варіантом взаємозв'язку "має"; агрегація є більш конкретною, ніж асоціація. Це асоціація, яка представляє частково цілі або часткові стосунки. Як показано на зображенні, професор "має" клас для викладання. Як тип асоціації, агрегація може бути названа та мати ті самі прикраси, що і асоціація. Однак агрегація не може включати більше двох класів; це має бути бінарна асоціація. Крім того, навряд чи існує різниця між агрегаціями та асоціаціями під час реалізації, і діаграма може взагалі пропустити відносини агрегування. [7]

Агрегація може відбуватися, коли клас є колекцією або контейнером інших класів, але вміщені класи не мають сильної залежності життєвого циклу від контейнера. Вміст контейнера все ще існує, коли контейнер знищений.

В UML він графічно представлений у вигляді порожнистої форми ромба на вміщуючому класі одним рядком, що зв’язує його із вміщеним класом. Сукупність - це семантично розширений об’єкт, який у багатьох операціях

трактується як одиниця, хоча фізично він складається з декількох менших об’єктів.

Приклад: Бібліотека та студенти. Тут студент може існувати без бібліотеки, зв'язок між студентом і бібліотекою є агрегацією.

Це вказує на те, що один із двох пов’язаних класів (підклас) вважається спеціалізованою формою іншого (супер тип), а суперклас - узагальненням підкласу. На практиці це означає, що будь-який екземпляр підтипу є також екземпляром суперкласу. Зразкове дерево узагальнень цієї форми зустрічається в біологічній класифікації: людина - це підклас маймуни, який є підкласом ссавців тощо. Зв’язок найлегше зрозуміти за допомогою фрази „А - це В” (людина - це ссавець, ссавець - тварина).

Графічне представлення UML узагальнення - це форма порожнистого трикутника на кінці суперкласу рядка (або дерева рядків), що зв’язує його з одним або кількома підтипами.

Відносини узагальнення також відомі як спадщина або відносини "є".

Суперклас (базовий клас) у відносинах узагальнення також відомий як "батьківський", суперклас, базовий клас або базовий тип.

Підтип у відносинах спеціалізації також відомий як "дочірній", підклас, похідний клас, похідний тип, клас успадкування або тип успадкування.

Зверніть увагу, що ці стосунки нічим не схожі на біологічні стосунки батьків та дітей: використання цих термінів надзвичайно поширене, але може ввести в оману.

А - це тип В

Наприклад, "дуб - це тип дерева", "автомобіль - це тип транспортного засобу"

Узагальнення може бути показано лише на діаграмах класів та на діаграмах використання.

При моделюванні UML взаємозв'язок реалізації - це взаємозв'язок між двома елементами моделі, в яких один елемент моделі (клієнт) реалізує

(реалізує або виконує) поведінку, яку вказує інший елемент моделі (постачальник).

Графічне представлення UML реалізації - це порожниста форма трикутника на кінці інтерфейсу штрихової лінії (або дерева рядків), яка з'єднує її з одним або кількома реалізаторами. Проста головка стрілки використовується на кінці інтерфейсу штрихової лінії, що з'єднує її з користувачами. У діаграмах компонентів використовується графічна умова «м'яч і сокет» (реалізатори виставляють кульку або льодяник, тоді як користувачі показують сокет). Реалізації можна показати лише на діаграмах класів або компонентів. Реалізація - це взаємозв'язок між класами, інтерфейсами, компонентами та пакетами, що з'єднує елемент клієнта з елементом постачальника. Зв'язок реалізації між класами / компонентами та інтерфейсами показує, що клас / компонент реалізує операції, пропоновані інтерфейсом.

Залежність - це слабша форма зв'язку, яка вказує на те, що один клас залежить від іншого, оскільки він використовує його в певний момент часу. Один клас залежить від іншого, якщо незалежний клас є змінною параметра або локальною змінною методу залежного класу. Це відрізняється від асоціації, де атрибут залежного класу є екземпляром незалежного класу. Іноді відносини між двома класами дуже слабкі. Вони взагалі не реалізовані зі змінними-членами. Швидше вони можуть бути реалізовані як аргументи функції-члена.

інший. Ці відносини зазвичай описуються як "А має В" (у матері-кота є кошенята, у кошенят - мати-кішка).

Представлення UML асоціації - це лінія, що з'єднує два пов'язані класи. На кожному кінці рядка є додаткові позначення. Наприклад, ми можемо вказати, використовуючи наконечник стрілки, що загострений кінець видно з хвоста стрілки. Ми можемо вказати власність шляхом розміщення кульки, ролі, яку відіграють елементи цього кінця, вказавши ім'я ролі та множинність

екземплярів цієї сутності (діапазон кількості об'єктів, які беруть участь в асоціації з точки зору іншого кінця).

Класи сутності моделюють довгоживучу інформацію, якою обробляє система, а іноді і поведінку, пов'язану з цією інформацією. Їх не слід ідентифікувати як таблиці баз даних чи інших сховищ даних.

Вони намальовані як кола з короткою лінією, прикріпленою до нижньої частини кола. Як варіант, їх можна намалювати як звичайні класи із позначенням стереотипу «сутність» над назвою класу.

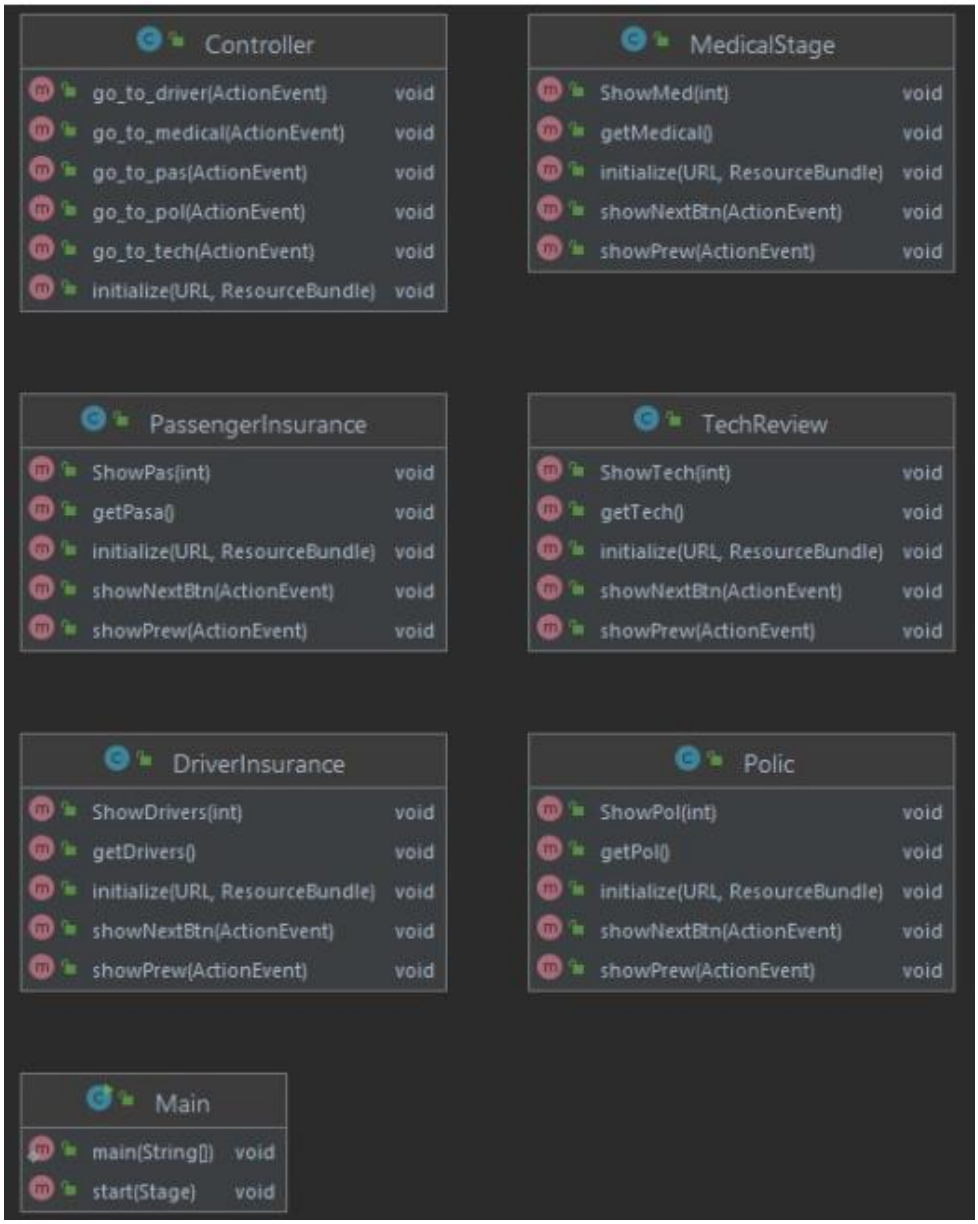


Рис. 3.3 — Діаграма класів

3.4. Графічний інтерфейс

Графічний інтерфейс користувача складається з 6 основних вікон, а саме:

- Форма головного меню (рис. 3.4)
- Форма техолгядів (рис. 3.5)
- Форма цивілки (рис. 3.6)
- Форма страхування водіїв (рис. 3.7)
- Форма страхування пасажирів (рис. 3.8)
- Форма медичних довідок (рис. 3.9)

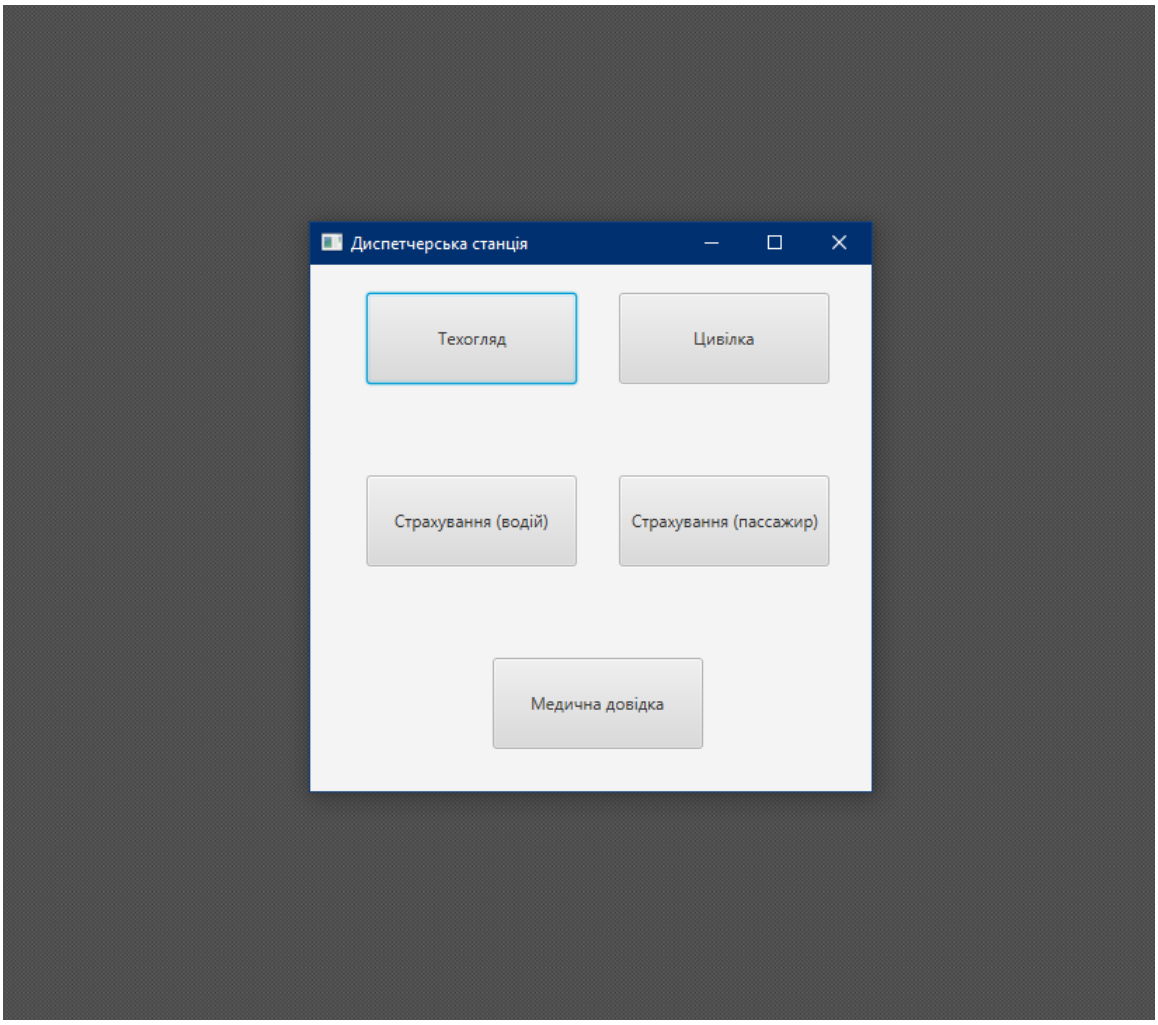


Рис. 3.4 — Форма головного меню

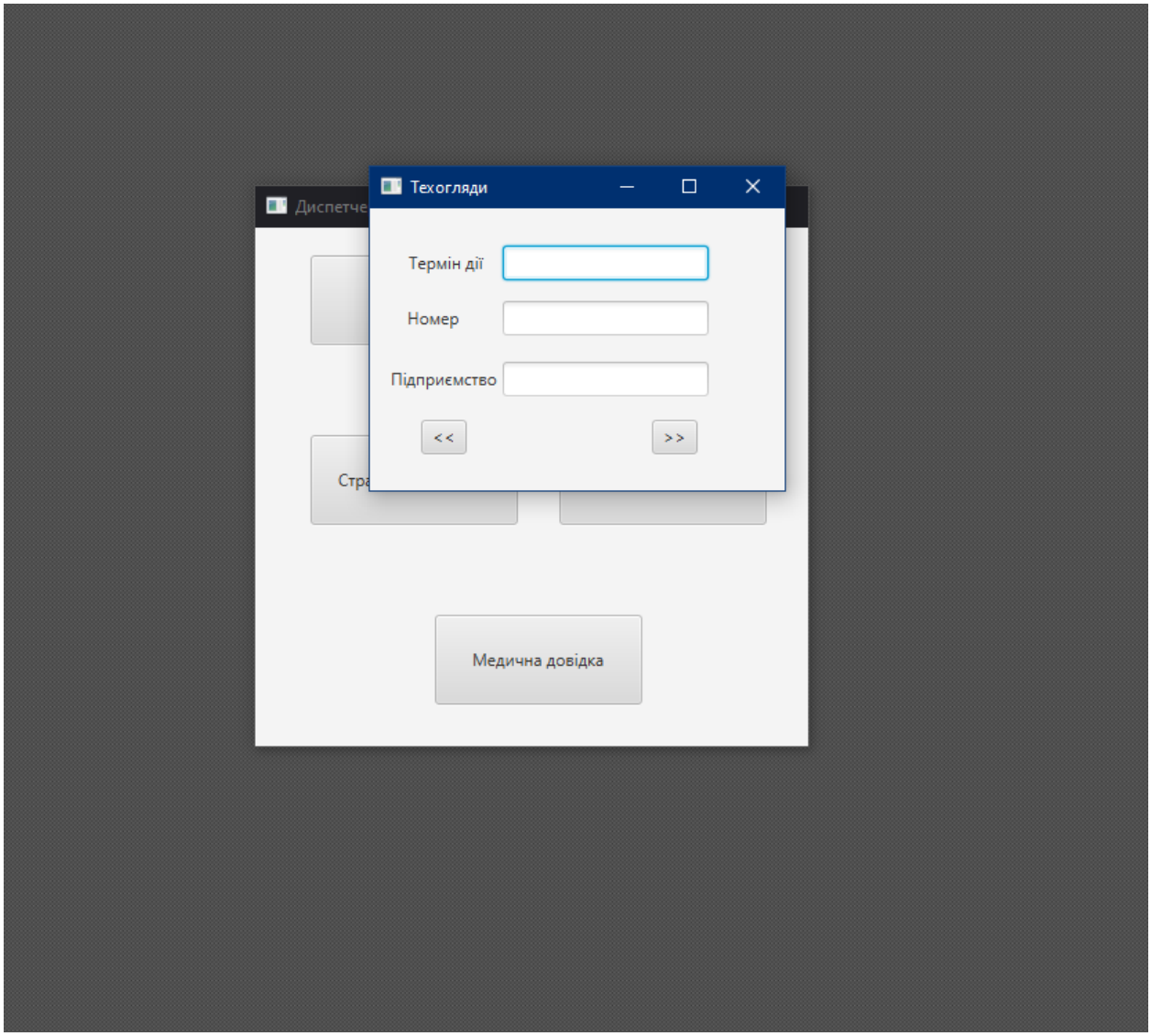


Рис. 3.5 — Форма техогляду

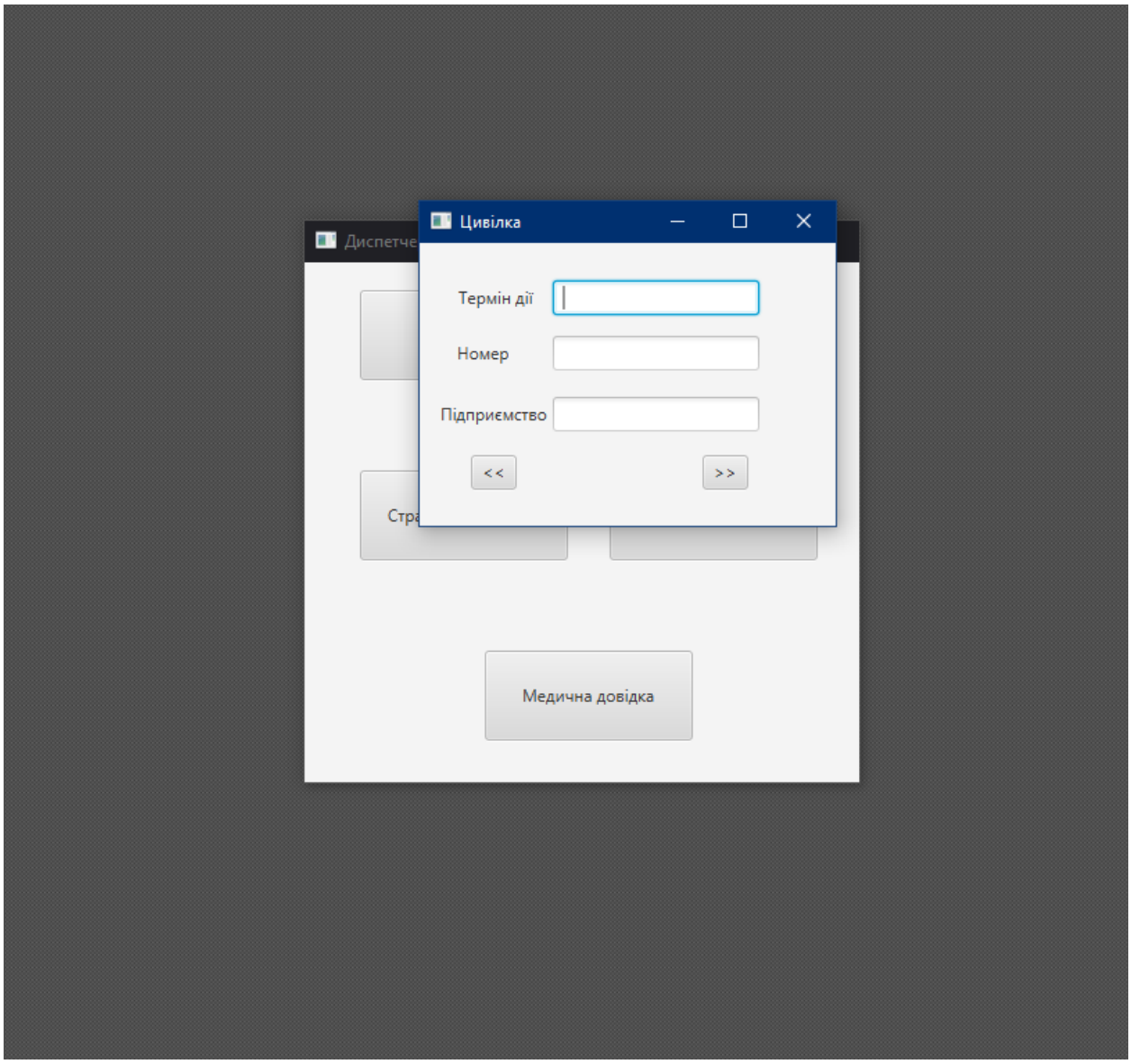


Рис. 3.6 — Форма цивілки

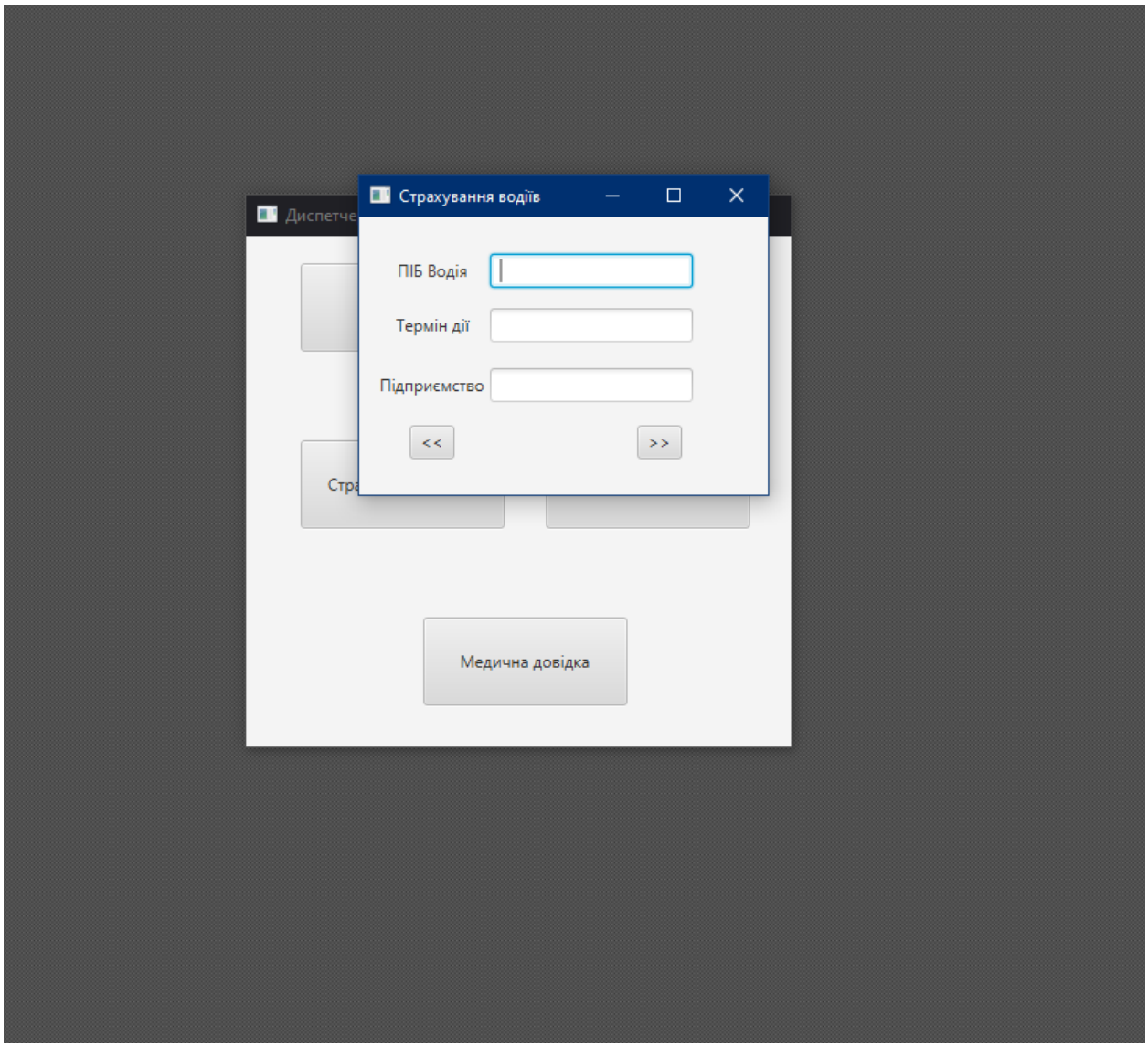


Рис. 3.7 — Форма страхування водія

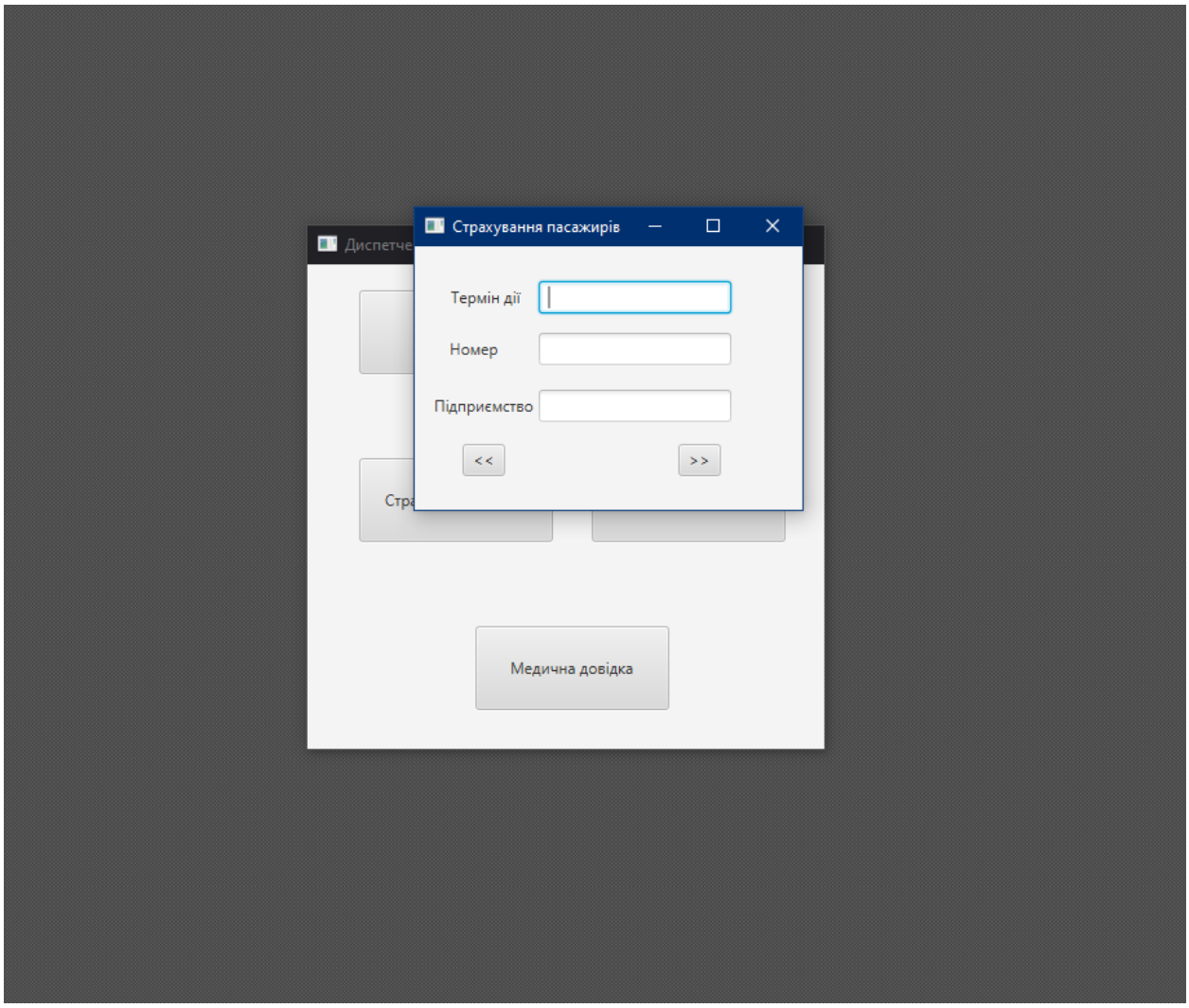


Рис. 3.8 — Форма страхування пасажирів

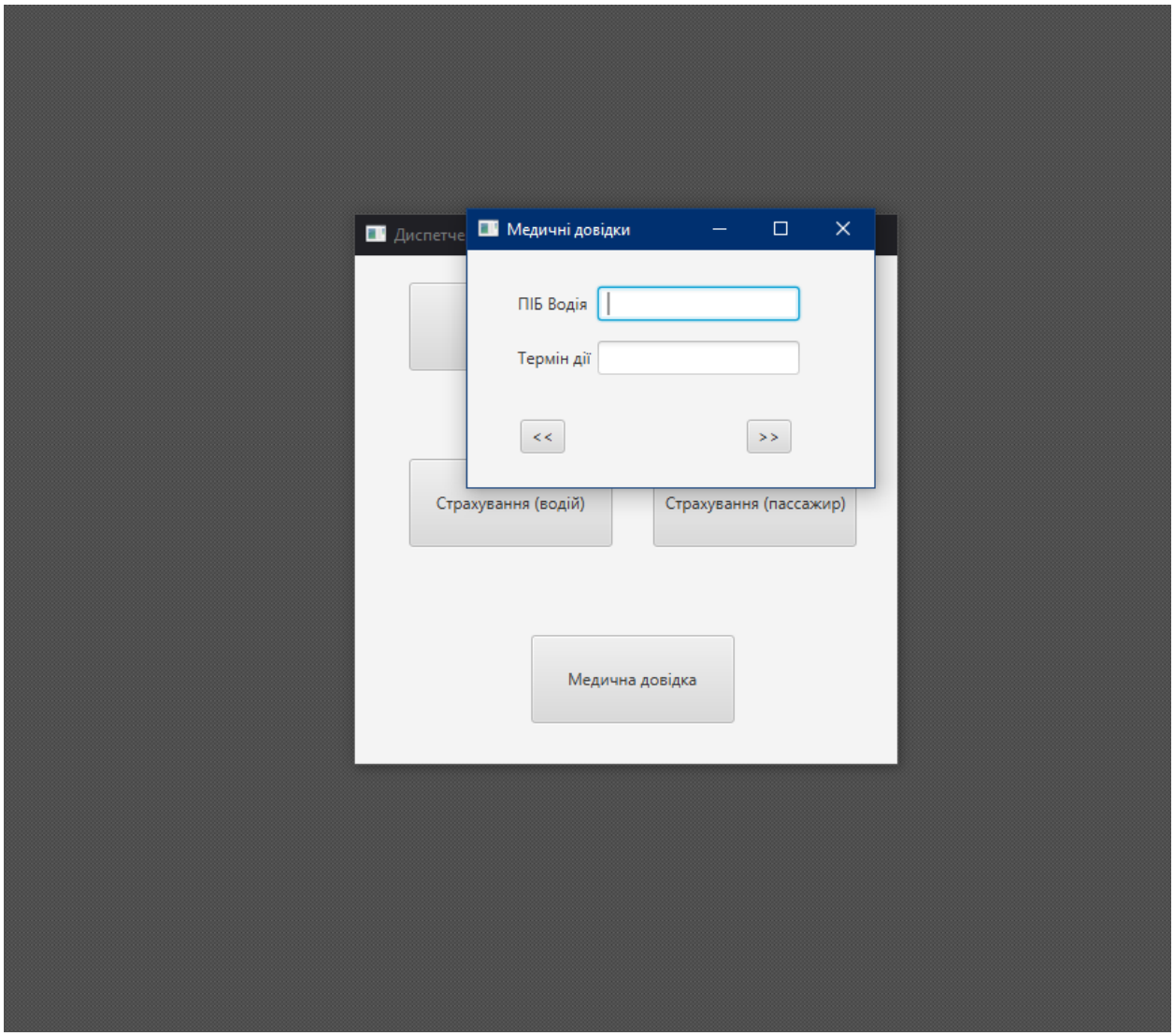


Рис. 3.9 — Медичні довідки

ВИСНОВКИ

В процесі роботи над даним проектом були виконані наступні завдання:

Дані характеристики предметної галузі, а саме проаналізоване поняття диспетчерських служб та можливі застосування даного програмного забезпечення. Проведений аналіз роботи диспетчерських служб та визначені їх потреби. Визначені документи та їх поля потребуючі постійного контролю.

Буди вибрані зручні інструменти розробки. Проведений огляд мови програмування. В ході роботи використана система управління базами даних. Обрано середовище розробки. Також створені діаграми, зокрема діаграма класів та діаграма варіантів використання. Був розроблений зручний інтерфейс додатку.

Завдяки чіткому виконанню завдань, поставлених на початку роботи, в результаті було отримано повноцінну інформаційну систему, яку можна використовувати в роботі диспетчерів авто-перевезень

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Grier, Robin. "Dispatch". Dispatch Solutions. Catalyst Communications Technologies, Inc. Retrieved 28 May 2011.
2. Grier, Robin. "What are Dispatch and Interoperability?". Catalyst. Catalyst CommunicationsTechnologies. Retrieved 24 May 2011.
3. www.bls.gov Occupational Outlook Handbook 2006-07 Edition. URL accessed on April 6, 2006
4. "Career Choices - Rail Traffic Controller". irtcanada.net. Archived from the original on 17 January 2011. Retrieved 18 February 2010.
5. Simpson, Rylan (2020-09-03). "Calling the Police: Dispatchers as Important Interpreters and Manufacturers of Calls for Service Data". Policing: A Journal of Policy and Practice: 4–5. doi:10.1093/police/paaa040. ISSN 1752-4512.
6. "Train dispatcher sentenced to two years for negligent homicide". Train dispatcher sentenced to two years for negligent homicide. Retrieved 2006-06-07.
7. "Application Development (AppDev) Defined and Explained". Bestpricecomputers.co.uk. 2007-08-13. Retrieved 2012-08-05.
8. DRM Associates (2002). "New Product Development Glossary". Retrieved 2006-10-29.
9. System Development Methodologies for Web-Enabled E-Business: A Customization Framework Linda V. Knight (DePaul University, USA), Theresa A. Steinbach (DePaul University, USA) and Vince Kellen (Blue Wolf, USA)
10. Joseph M. Morris (2001). Software Industry Accounting. p.1.10
11. Alan M. Davis. Great Software Debates (October 8, 2004), pp:125-128 Wiley-IEEE Computer Society Press
12. Ralph, P., and Wand, Y. A Proposal for a Formal Definition of the Design Concept. In, Lyytinen, K., Loucopoulos, P., Mylopoulos, J., and Robinson,

W., (eds.), Design Requirements Engineering: A Ten-Year Perspective: Springer-Verlag, 2009, pp. 103-136

13. Otero, Carlos. "Software Design Challenges". IT Performance Improvement. Taylor & Francis LLC. Retrieved 19 October 2017.
14. Edward J. Barkmeyer ea (2003). Concepts for Automating Systems Integration NIST 2003.
15. Paul R. Smith & Richard Sarfaty (1993). Creating a strategic plan for configuration management using Computer Aided Software Engineering (CASE) tools. Paper For 1993 National DOE/Contractors and Facilities CAD/CAE User's Group.
16. Kuhn, D.L (1989). "Selecting and effectively using a computer-aided software engineering tool". Annual Westinghouse computer symposium; 6-7 Nov 1989; Pittsburgh, PA (USA); DOE Project.
17. P. Loucopoulos and V. Karakostas (1995). System Requirements Engineering. McGraw-Hill.
18. CASE Archived 2012-02-18 at the Wayback Machine definition In: Telecom Glossary 2000 Archived 2005-11-22 at the Wayback Machine. Retrieved 26 Oct 2008.
19. K. Robinson (1992). Putting the Software Engineering into CASE. New York : John Wiley and Sons Inc.
20. Xiao He (2007). "A metamodel for the notation of graphical modeling languages". In: Computer Software and Applications Conference, 2007. COMPSAC 2007 – Vol. 1. 31st Annual International, Volume 1, Issue, 24–27 July 2007, pp 219-224.
21. Merx, Georges G.; Norman, Ronald J. (2006). Unified Software Engineering with Java. Prentice-Hall, Inc. p. 201. ISBN 0130473766.

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для роботи диспетчерських служб мовою Java

Виконав студент 4 курсу
групи ПД-44
Меленевський Ігор Віталійович
Керівник роботи
Коба Андрій Борисович

Київ – 2020

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - розробка програмне забезпечення для поліпшення функціонування диспечерський служб.
- **Об’єкт дослідження** – поліпшення функціонування диспечерських служб, та усуненя рутинних процесів.
- **Предмет дослідження** - програмне забезпечення для поліпшення роботи диспечерський служб.

ПЕРЕВАГИ

Переваги:

• Більший обсяг використання

• зручний інтерфейс використання

• Доступ до будь яких документів на транспорт

Недоліки:

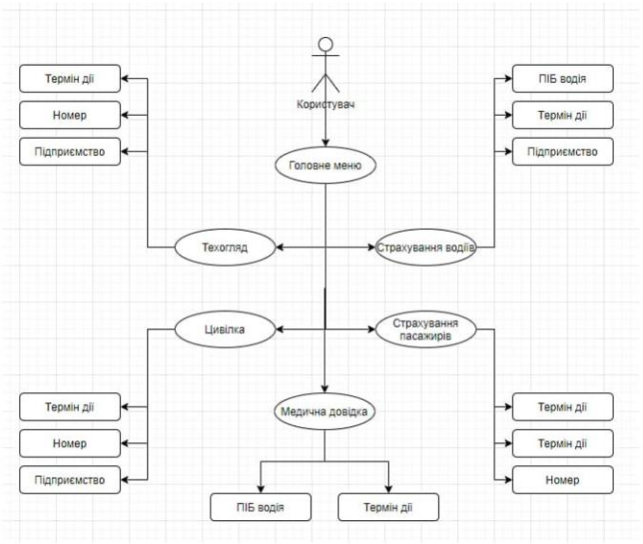
• Неможливість контролювати документи для багатьох транспортних засобів

• Не реалізований спільний доступ до документів

3

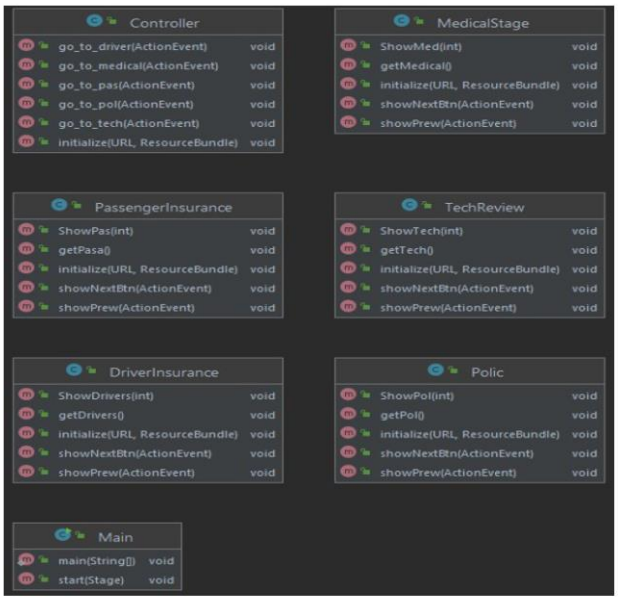
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

4



Діаграма варіантів використання

5



Діаграма класів

6

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

- Науково-технічна конференція «Застосування програмного забезпечення в ІКТ», тема Розробка програмного забезпечення мовою програмування Java для контролю документів на транспорт
- XII Науково-технічна конференція студентів та молодих вчених «Сучасні інфокомунікаційні технології», Вибір інстументів розробки API для диспечерських служб

ВИСНОВКИ

В процесі роботи над даним проектом були виконані наступні завдання:

Дані характеристики предметної галузі, а саме проаналізоване поняття диспетчерських служб та можливі застосування даного програмного забезпечення. Проведений аналіз роботи диспетчерських служб та визначені їх потреби. Визначені документи та їх поля потребуючі постійного контролю.

Буди вибрані зручні інструменти розробки. Проведений огляд мови програмування. В ході роботи використана система управління базами даних. Обрано середовище розробки. Також створені діаграми, зокрема діаграма класів та діаграма варіантів використання. Був розроблений зручний інтерфейс додатку.

Завдяки чіткому виконанню завдань, поставлених на початку роботи, в результаті було отримано повноцінну інформаційну систему, яку можна використовувати в роботі диспетчерів авто-перевезень

ДЯКУЮ ЗА УВАГУ!