РЕФЕРАТ

Пояснювальна записка складається із наступних розділів:

* вступ – у цьому розділі описується сутність розробки та її актуальність;
* збір вимог до програмного забезпечення – у цьому розділі міститься аналіз літературних джерел, програмних аналогів, опитування зацікавлених сторін, специфікує вимоги;
* проектування програмного комплексу – у цьому розділі обґрунтовується вибір засобів проектування, наведені розроблені діаграми, описується проектування інтерфейсу користувача та розробка бази даних;
* розробка програми – у цьому розділі описано розробку алгоритму програми та конкретних методів;
* тестування та налагодження – у цьому розділі описуються обрані стратегії тестування та перераховані розроблені тести;
* охорона праці – містить відомості про безпечні умови праці користувача;
* висновки;
* список літератури – містить бібліографічний список використаної літератури.

Кількість таблиць: 16.

Кількість рисунків: 21.

Ключові слова: голосовий запит, мовлення, розпізнання, залізничний квиток, супутня інформація.

ЗМІСТ

[ВСТУП 8](#_Toc421707916)

[1 ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 9](#_Toc421707917)

[1.1 Аналіз літературних джерел 9](#_Toc421707918)

[1.2 Аналіз програмних аналогів 10](#_Toc421707919)

[1.2.1 Огляд Google Now 10](#_Toc421707920)

[1.2.2 Огляд Siri 11](#_Toc421707921)

[1.2.3 Огляд Cortana 12](#_Toc421707922)

[1.2.4 Огляд Laitis 13](#_Toc421707923)

[1.2.5 Огляд RuVC 14](#_Toc421707924)

[1.2.6 Огляд Приват 24 15](#_Toc421707925)

[1.2.7 Огляд замовлення квитків на сайті Укрзалізниці 16](#_Toc421707926)

[1.2.8 Аналіз аналогів та висновки 17](#_Toc421707927)

[1.3 Опитування зацікавлених сторін 18](#_Toc421707928)

[1.3.1 Опитування користувачів 19](#_Toc421707929)

[1.3.2 Опитування адміністраторів 19](#_Toc421707930)

[1.4 Специфікація вимог 20](#_Toc421707931)

[2 ПРОЕКТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ 21](#_Toc421707932)

[2.1 Засоби проектування 21](#_Toc421707933)

[2.2 Діаграма прецедентів 21](#_Toc421707934)

[2.3 Проектування класів за допомогою CRC-карток 23](#_Toc421707935)

[2.4 Діаграма класів 24](#_Toc421707936)

[2.4.1 Клас Entity 25](#_Toc421707937)

[2.4.2 Клас Server 26](#_Toc421707938)

[2.4.3 Клас Client 27](#_Toc421707939)

[2.4.4 Клас Recognizer 28](#_Toc421707940)

[2.5 Діаграми послідовності 28](#_Toc421707941)

[2.5.1 Діаграма послідовності для процесу завантаження сторінки 29](#_Toc421707942)

[2.5.2 Діаграма послідовності для етапів введення назв міст 29](#_Toc421707943)

[2.5.3 Діаграма послідовності для етапу введення дати відправлення 30](#_Toc421707944)

[2.5.4 Діаграма послідовності для етапу пошуку рейсів 30](#_Toc421707945)

[2.6 Структура програми 31](#_Toc421707946)

[2.7 Проектування інтерфейсу користувача 32](#_Toc421707947)

[2.7.1 Розробка структури та сценарію діалогу 33](#_Toc421707948)

[2.7.2 Розробка інтерфейсу 34](#_Toc421707949)

[2.8 Розробка бази даних 36](#_Toc421707950)

[2.8.1 Вибір СУБД 37](#_Toc421707951)

[2.8.2 Проектування бази даних 38](#_Toc421707952)

[3 РОЗРОБКА ПРОГРАМИ 44](#_Toc421707953)

[3.1 Вибір мови програмування 44](#_Toc421707954)

[3.2 Розробка алгоритму 45](#_Toc421707955)

[3.3 Розробка методів 46](#_Toc421707956)

[3.3.1 Розробка методу Server.checkTime() 46](#_Toc421707957)

[3.3.2 Розробка методу Server. findTrains() 49](#_Toc421707958)

[3.3.3 Розробка методу Server. handleMessage() 50](#_Toc421707959)

[3.3.4 Розробка методу processMessage() 53](#_Toc421707960)

[3.3.5 Розробка методу processStates() 56](#_Toc421707961)

[4 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ 59](#_Toc421707962)

[4.1 Тестування методом «білого ящика» 59](#_Toc421707963)

[4.1.1 Тестування методу checkTime() 60](#_Toc421707964)

[4.1.2 Тестування методу handleMessage() 62](#_Toc421707965)

[4.1.3 Тестування методу processMessage() 64](#_Toc421707966)

[4.2 Тестування методом «чорного ящика» 68](#_Toc421707967)

[4.2.1 Тестування методу checkTime() 69](#_Toc421707968)

[4.2.2 Тестування методу findTrains() 71](#_Toc421707969)

[4.3 Вибір методів налагодження програми 72](#_Toc421707970)

[4.4 Налагодження програми 74](#_Toc421707971)

[4.4.1 Налагодження методу checkTime() класу Server 74](#_Toc421707972)

[4.4.2 Налагодження методу start() класу webkitSpeechRecognition 74](#_Toc421707973)

[4.4.3 Налагодження методу processStates() класу Client 75](#_Toc421707974)

[4.4.4 Налагодження методу handleMessage() класу Server 75](#_Toc421707975)

[5 ОХОРОНА ПРАЦІ 76](#_Toc421707976)

[5.1 Аналіз шкідливих та небезпечних виробничих факторів 76](#_Toc421707977)

[5.1.1 Небезпечні чинники, пов’язані з пожежами 77](#_Toc421707978)

[5.1.2 Електронебезпечні чинники 78](#_Toc421707979)

[5.1.3 Фізично шкідливі і небезпечні чинники 79](#_Toc421707980)

[5.1.4 Хімічно шкідливі і небезпечні чинники 79](#_Toc421707981)

[5.2 Заходи з усунення шкідливих та небезпечних чинників 79](#_Toc421707982)

[5.2.1 Вимоги до організації робочого місця 80](#_Toc421707983)

[5.3 Правила безпечного виконання робіт на ЕОМ 81](#_Toc421707984)

[5.3.1 Вимоги безпеки праці перед початком роботи на ЕОМ 81](#_Toc421707985)

[5.3.2 Вимоги безпеки праці під час роботи на ЕОМ 82](#_Toc421707986)

[5.3.3 Вимоги безпеки праці після закінчення роботи на ЕОМ 84](#_Toc421707987)

[5.4 Дії при аварійних ситуаціях 85](#_Toc421707988)

[5.4.1 Допомога при ураженні електричним струмом 85](#_Toc421707989)

[ВИСНОВКИ 87](#_Toc421707990)

[СПИСОК ЛІТЕРАТУРИ 88](#_Toc421707991)

ВСТУП

Залізничний транспорт є однією з найбільш важливих складових транспортно-дорожнього комплексу. Він відіграє значну роль у забезпеченні життєдіяльності багатогалузевої економіки України; на його долю припадає 82% вантажоперевезень та майже 40% пасажирських перевезень, що здійснюються усіма видами транспорту.

Пасажири обирають залізничний транспорт здебільшого через його надійність, високу швидкість та відносно низьку вартість проїзду (порівняно, наприклад, із автомобільним транспортом). Але дуже важливим критерієм також є зручність користування.

У мережі Інтернет уже давно можна знайти розклад руху потягів, дізнатися про вартість квитка та навіть замовити квиток (наприклад, за допомогою сайту Укрзалізниці, сервісу «Приват 24» та ін.). Для подібних цілей існують також мобільні додатки (зокрема, той самий «Приват 24» для Android). Однак і на сайтах, і в додатках для мобільних пристроїв використовуються стандартні на даний момент засоби введення інформації – клавіатура, маніпулятор «миша», сенсорний екран.

Розроблювана система розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації має схожий функціонал та сферу застосування (придбання квитків), однак надає змогу задавати критерії пошуку та вводити основні команди керування програмою за допомогою голосу. Такий спосіб введення інформації є більш природним для людини, а отже – більш зручним. Наразі аналогів (програм з аналогічними функціями та способом введення даних) розроблюваної системи не існує.

Впровадження даного програмного засобу дасть змогу залучити до сервісів Укрзалізниці більше клієнтів (здебільшого за рахунок користувачів мобільних платформ), а також – у перспективі – отримувати прибуток завдяки монетизації додатку.

1 ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Аналіз літературних джерел

У першу чергу необхідно розглянути широкий спектр питань, пов’язаних зі звуком: від основ теорії звуку до детального опису існуючих способів цифрового представлення звуку і базисних методів його обробки і стиснення.

Важливою для розробки є інформація про структуру основних звукових форматів, а також алгоритм розрахунку амплітуди звуку для визначення рівня гучності записуваного звуку (за допомогою цього можна розпізнати, коли користувач закінчив говорити)[1].

Також необхідно розглянути потенційні інструменти розробки та їх можливості. У якості прикладу було розглянуто третю версію фреймворку Spring, зокрема інформацію про мову виразів SpEL, нові анотації для робот из контейнером IoC і підтримку архітектури REST, а також значну кількість прикладів коду, які сприяли розумінню принципів роботи Spring MVC та шаблону проектування Model-View-Controller в цілому[2].

Суттєвою є організація роботи на стороні клієнта. У зв’язку з цим, окрім основ мови Javascript, було проаналізовано взаємодію із протоколом HTTP, застосування технології AJAX, обробку документів XML та JSON, середовище розробки сценаріїв у браузерах, численні приклади коду та обширний довідник із базового та клієнтського JavaScript (класи, методи, властивості, константи і т.д.)[3]

Після реалізації функціоналу необхідно розробити інтерфейс. Було враховано методологію адаптивного веб-дизайну (тобто такого, що враховує можливості відображення даних та взаємодії із користувачем на різних платформах – як уже існуючих, так і тих, що з’являться найближчим часом).

Окрім деяких базових знань про HTML5 та CSS3, було вивчено методи відображення сторінок залежно від розміру екрана, оскільки передбачається, що розроблюваний додаток функціонуватиме не лише на персональних комп’ютерах, а і на мобільних пристроях[4].

Зрештою, не менш важливим є встановлення програми на сервері. Було розглянуто процес встановлення та налаштування Tomcat, його інтегрування з Apache, налаштування та підключення до бази даних за допомогою драйвера JDBC, захист програм. Також обговорено деякі додаткові інструменти, зокрема Ant (для автоматичного компонування веб-додатків)[5].

1.2 Аналіз програмних аналогів

Оскільки програм, повністю ідентичних за функціоналом до розроблюваної системи, немає, натомість були окремо проаналізовані голосові помічники та засоби замовлення квитків. Розглянуті програми описані нижче.

1.2.1 Огляд Google Now

Google Now – персоналізований сервіс пошуку від Google Inc, що використовує обробку природної мови для відповіді на запитання, створення рекомендацій і виконання різноманітних дій. Додаток видає інформацію з урахуванням поточного місцезнаходження користувача, його особистої інформації з календаря, історії пошукових запитів, історії переміщень, історії відвіданих сторінок та ін. Результати оформлені у вигляді так званих «інформаційних карток», кожна з яких відповідає за свій тип інформації (погода, трафік, зустрічі, авіарейси, спорт, місця, переклад, валюта та ін.). Запущений 9 липня 2012 року. Зовнішній вигляд Google Now відображено на рис. 1.1.

Переваги:

* інтерфейс, зручний для постійного оновлення інформації;
* підтримка значної кількості мов, у тому числі російської;
* широкий спектр функцій: управління функціями телефону (набір номерів, відправлення повідомлень, створення нагадувань і т. ін.), функції пошуку (прогноз погоди, встановлення маршрутів), голосове введення тексту;
* інтеграція з іншими сервісами Google (Gmail, Maps, Translate та ін.);
* сумісність із Google Glass.

Недоліки:

* доступний лише у вигляді додатка для пристроїв на Android та iOS;
* не реагує на нестандартні запити.



Рисунок 1.1 – Зовнішній вигляд Google Now

1.2.2 Огляд Siri

Siri – персональний помічник і питально-відповідна система, адаптована для iOS. Цей додаток використовує обробку природної мови, щоб відповідати на питання і давати рекомендації. Siri пристосовується до кожного користувача індивідуально, вивчаючи його переваги протягом довгого часу. Є розробкою Міжнародного Центру Штучного Інтелекту SRI. Запущена 9 серпня 2011 року. Зовнішній вигляд Siri відображено на рис. 1.2.

Переваги:

* розпізнання змісту нетипових запитів, включаючи залежні від контексту;
* підтримка значної кількості мов, у тому числі російської;
* широкий спектр функцій: управління функціями телефону (здійснення дзвінків, зміна налаштувань тощо), взаємодія з іншими додатками (музика, годинник, браузер, карти, пошта та ін.), здійснення операцій в Apple Store, голосове введення тексту;
* якісний голос (у двох варіантах);
* взаємодія із сервісами OpenTable, ReverseTravel, Yahoo, Bing, StubHub, Rotten Tomatoes, The New York Times, Wolfram Alpha та ін.

Недоліки:

* доступна виключно як інтегрована частина iOS;
* підтримка російської мови на даний момент реалізована недостатньо якісно.

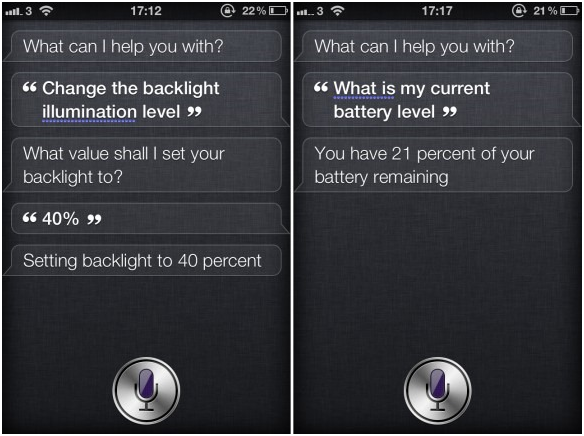


Рисунок 1.2 – Зовнішній вигляд Siri

1.2.3 Огляд Cortana

Cortana – віртуальний голосовий помічник зі штучним інтелектом для пристроїв на Windows Phone. Призначений передбачати потреби користувача. Має вікове обмеження – доступний лише користувачам віком не менше 13 років. Запущений 2 квітня 2014 року. Буде інтегрований у Windows 10. Зовнішній вигляд Cortana відображено на рис. 1.3.

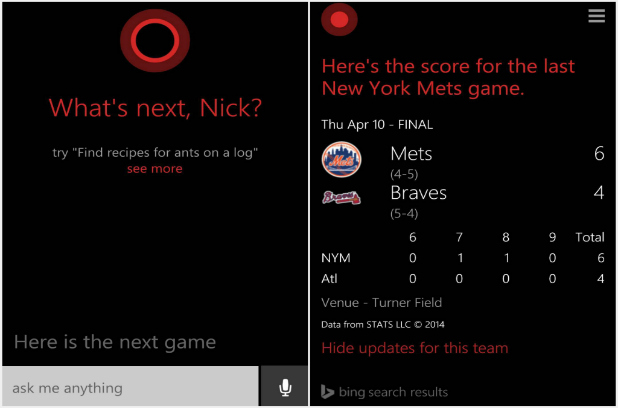


Рисунок 1.3 – Зовнішній вигляд Cortana

Переваги:

* можливість спілкування на вільні теми;
* відстеження інтересів користувача;
* гнучкі налаштування конфіденційності (можливість указати, яку саме інформацію помічникові дозволено збирати);
* широкий спектр функцій: управління функціями телефону (дзвінки, повідомлення, нагадування та ін.), функції пошуку, голосове введення тексту.

Недоліки:

* доступний лише для пристроїв на Windows Phone;
* немає підтримки російської мови.

1.2.4 Огляд Laitis

Laitis – програма для набору тексту та управління комп’ютером за допомогою голосу. Зовнішній вигляд Laitis відображено на рис. 1.4.

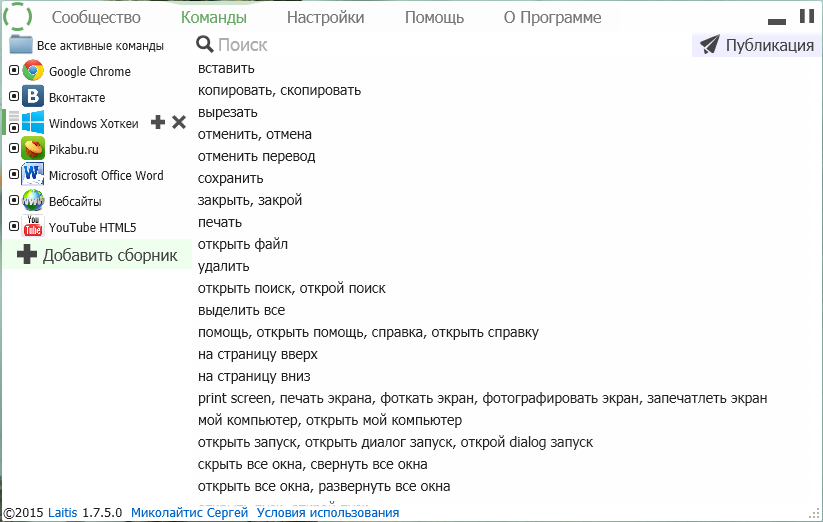


Рисунок 1.4 – Зовнішній вигляд Laitis

Переваги:

* можливість набору тексту голосом зі збереженням пунктуації, коректним відображенням власних назв та іншомовних слів;
* підтримка російської мови;
* можливість запуску програм голосом;
* можливість створення власних шаблонів (можна призначити голосову команду для запуску програми, натиснення комбінації клавіш тощо).

Недоліки:

* програма працює лише на системах, де встановлений .NET Framework.

1.2.5 Огляд RuVC

RuVC (Russian Voice Control) – програма для керування комп’ютером (запуску програм, швидкого пошуку, набору тексту, пошуку в мережі) за допомогою голосу. Зовнішній вигляд RuVC відображено на рис. 1.5.

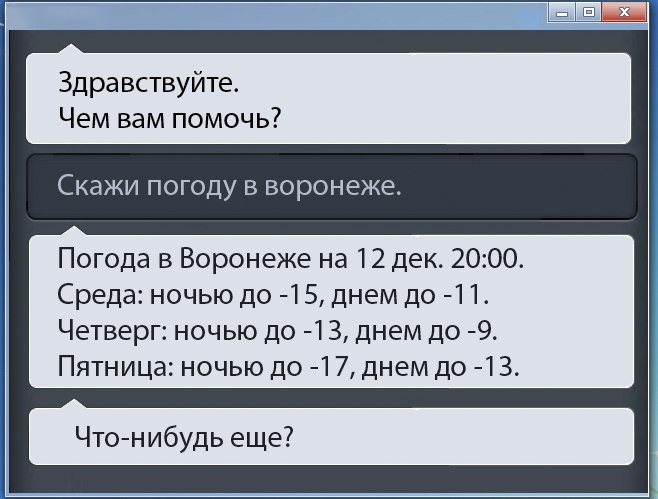


Рисунок 1.5 – Зовнішній вигляд RuVC

Переваги:

* підтримка російської мови;
* можливість запуску програм голосом;
* можливість створення власних шаблонів;
* програма супроводжує виконання дій голосовими коментарями.

Недоліки:

* програма працює лише на системах, де встановлений .NET Framework;
* супровід програми тимчасово призупинений.

1.2.6 Огляд Приват 24

Приват 24 – система інтернет-банкінгу, яка, серед інших функцій, надає можливість замовляти залізничні квитки онлайн. Зовнішній вигляд системи Приват 24 відображено на рис. 1.6.

Переваги:

* значна кількість налаштувань та їх наглядне відображення (можна обрати пункти відправлення та прибуття, дату відправлення, потяг, вагон і місце, додатково можна дізнатися приблизний час у дорозі, тип вагонів, кількість вільних місць та вартість квитка);
* можливість замовити до чотирьох квитків одразу, а також замовити квиток на потяг у зворотній бік;
* надійність (використовується двохетапна аутентифікація, сучасні протоколи шифрування та перевірений сертифікат безпеки);
* зручність оплати замовлених квитків;
* наявність додатку для мобільних пристроїв.

Недоліки:

* для користування послугами системи необхідна реєстрація та авторизація;
* система доступна тільки для клієнтів Приватбанку.

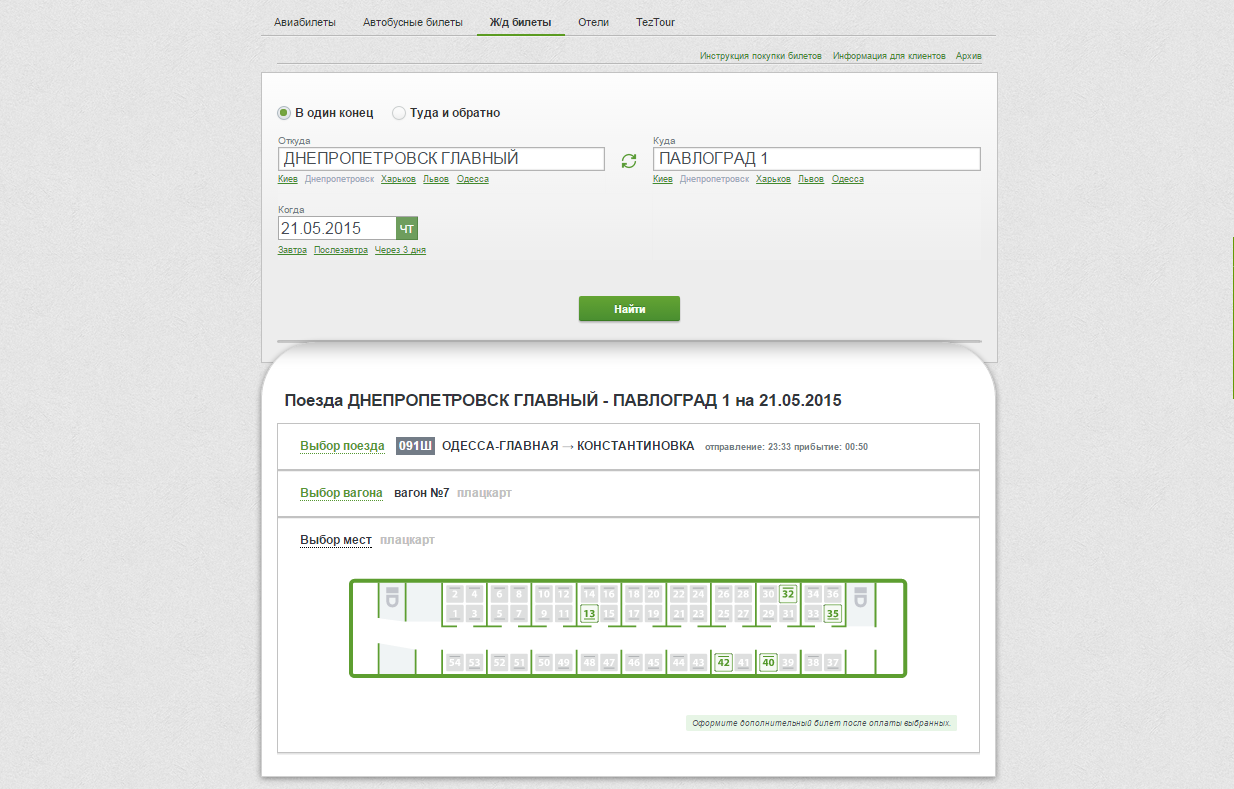


Рисунок 1.6 – Зовнішній вигляд системи Приват 24

1.2.7 Огляд замовлення квитків на сайті Укрзалізниці

На офіційному сайті Укрзалізниці можна замовити квитки онлайн. Зовнішній вигляд сайту відображено на рис. 1.7.

Переваги:

* значна кількість налаштувань та їх наглядне відображення (можна обрати пункти відправлення та прибуття, дату відправлення, потяг, вагон і місце, додатково можна дізнатися приблизний час у дорозі, тип вагонів, кількість вільних місць та вартість квитка);
* можливість замовити квиток на потяг у зворотній бік;
* можливість замовлення квитків без реєстрації.

Недоліки:

* система довгий час знаходиться на етапі публічного тестування та не оновлюється.

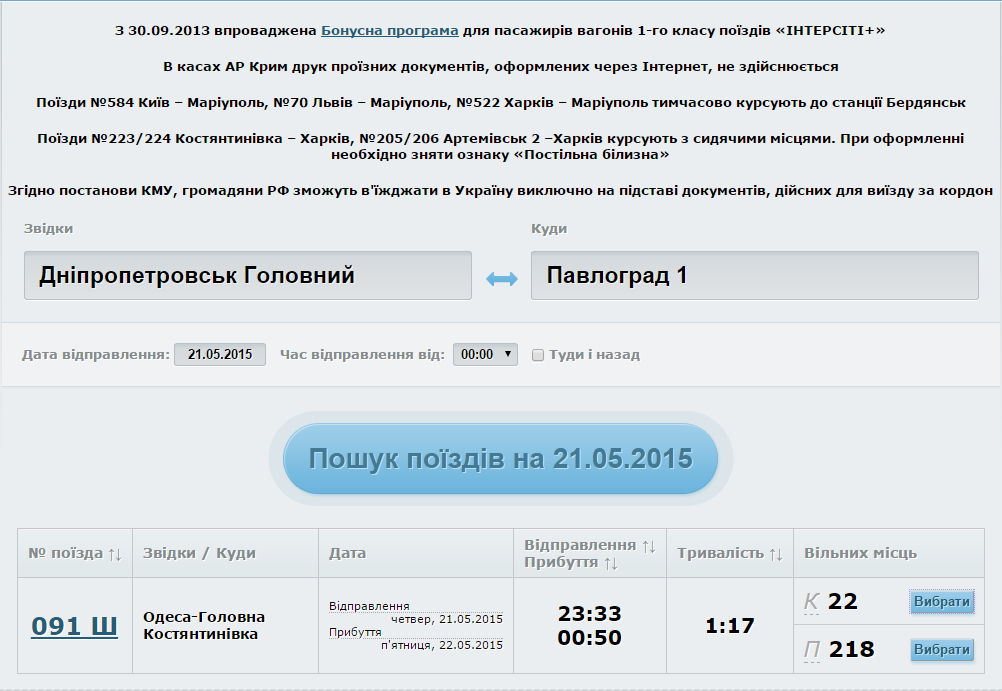


Рисунок 1.7 – Зовнішній вигляд сайту Укрзалізниці

1.2.8 Аналіз аналогів та висновки

На сучасному ринку відсутні програмні засоби із функціоналом, аналогічним до того, який матиме система розпізнання голосових запитів для пошуку залізничних квитків. Тому в цьому розділі був розглянутий більш загальний випадок – додатки, які виконують функції голосових помічників. Перші три з них (Google Now, Siri та Cortana) розроблені для використання на мобільних пристроях, мають широкий спектр можливостей для застосування і достатньо широко розповсюджені.

Наступні два (Laitis та RuVC), розроблені для використання на персональних комп’ютерах, мають досить обмежений функціонал (оскільки створенням кожної з цих програм займалась одна людина) та не користуються значною популярністю.

Також були розглянуті системи, що дозволяють знайти та замовити залізничний квиток. Сайти Укрзалізниці та Приват 24 виконують ті самі функції, що і розроблюваний програмний засіб, однак вони використовують стандартні методи введення (клавіатура та маніпулятор «миша»).

Можна зробити висновок, що розробка системи розпізнання голосових запитів для пошуку залізничних квитків є доцільною, оскільки:

* наразі не існує її повних аналогів;
* формат розробки (веб-додаток) сприятиме набуттю популярності системи, оскільки вона працюватиме на всіх платформах;
* метод голосового введення даних буде більш зручним для користувачів мобільних пристроїв;
* програма розпізнаватиме запити, зроблені російською мовою.

1.3 Опитування зацікавлених сторін

Опитування можна розглядати як один з найпоширеніших методів отримання інформації про суб'єктів – респондентів опитування. Опитування полягає в ставленні людям спеціальних питань, відповіді на які дозволяють досліднику отримати необхідні відомості залежно від завдань дослідження.

Для опитування було обрано метод анкетування.

Анкетування – метод опитування за допомогою анкет (структурно-організованих запитань, кожне з яких пов’язане з метою та завданнями дослідження). Анкета – впорядкований за змістом і формою набір запитань, що має вигляд опитувального листа.

Зацікавлені сторони – це особи чи організації, які мають дійсний інтерес до системи. Вона може впливати на них прямо чи опосередковано. До зацікавлених сторін можна віднести потенційних користувачів системи та адміністраторів. Користувачами системи можуть бути люди, які користуються послугами Укрзалізниці, володіють російською мовою та навичками користування комп’ютером або мобільним пристроєм. Адміністратори – люди, котрі мають спеціальні знання на займатимуться встановленням, налаштуванням і підтримкою серверної частини програмного засобу.

Дані, отримані у процесі опитування, наведені нижче.

1.3.1 Опитування користувачів

Метод опитування користувачів – анкетування.

Анкети містили питання стосовно побажань щодо можливостей, деталей роботи та інтерфейсу програми.

У результаті проведення опитування були виявлені наступні вимоги:

* простий мінімалістичний, але інформативний інтерфейс;
* оперативність роботи (хороша оптимізація на різних платформах);
* адекватні системні вимоги;
* можливість редагувати введені дані, скасовувати останні дії та керувати програмою за допомогою голосу;
* дублювання вихідної інформації (у вигляді тексту на екрані та усних фраз).

1.3.2 Опитування адміністраторів

Метод опитування адміністраторів – анкетування.

Анкети містили питання стосовно побажань щодо технічних характеристик серверної частини програми.

У результаті проведення опитування були виявлені наступні вимоги:

* встановлення на веб-сервер зі статичною IP-адресою (бажано із зареєстрованим доменним іменем);
* встановлення на веб-сервері сертифікату SSL для забезпечення обміну інформацією з користувачами через захищений протокол HTTPS;
* захищеність від SQL-ін’єкцій.

1.4 Специфікація вимог

У ході аналізу літературних джерел, аналогів та опитування зацікавлених осіб були виявлені наступні функціональні вимоги:

* дозволяти голосом обрати критерії для пошуку квитка (такі як місце відправлення та прибуття, дату відправлення та ін.);
* надання вибору конкретного квитка зі списку тих, що задовольняють указаним критеріям (якщо таких квитків декілька);
* виконання переадресації на сторінку придбання знайденого квитка;
* надання можливості у подальшому підтвердити факт придбання (шляхом роздрукування квитка або збереження коду транзакції);
* функціонування на мобільних платформах (також адаптація зовнішнього вигляду для коректного відображення на різних пристроях);
* можливість редагувати введені дані, скасовувати останні дії та керувати програмою за допомогою голосу;
* дублювання вихідної інформації (у вигляді тексту на екрані та усних фраз).

У ході аналізу літературних джерел, аналогів та опитування зацікавлених осіб були виявлені наступні нефункціональні вимоги:

* простий та зручний інтерфейс;
* швидкий відклик на дії користувача;
* захищеність системи.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Засоби проектування

Для проектування системи була обрана мова UML.

UML (англ. Unified Modeling Language) — уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, названої UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація[6].

Діаграми UML дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту. Діаграми підвищують супроводжуваність проекту і полегшують розробку документації[6].

В UML використовується 13 видів діаграм. Серед них:

* структурні (класів, компонент, композитної/складеної структури, кооперації, розгортання, об’єктів, пакетів);
* діаграми поведінки (діяльності, скінченних автоматів (станів), прецедентів);
* діаграми взаємодії (кооперації/комунікації, огляду взаємодії, послідовності, синхронізації).

У якості програмного засобу для створення діаграм UML було обрано пакет Rational Rose.

2.2 Діаграма прецедентів

Діаграма прецедентів (англ. Use Case Diagram, діаграма варіантів використання) відображає відносини між акторами та прецедентами і є складовою частиною моделі прецедентів, що дає змогу описати систему на концептуальному рівні[7].

Актори – ролі користувача (у широкому значенні: людина, зовнішня сутність, клас, інша система), який взаємодіє із іншими сутностями[7].

Прецеденти – можливості модельованої системи (частини її функціональності), завдяки яким користувач може отримати конкретний, вимірний і потрібний йому результат. Прецедент відповідає окремому сервісу системи, визначає один із варіантів її використання та описує типовий спосіб взаємодії користувача із системою[7].

Проектування ПЗ доцільно розпочинати саме зі створення діаграми прецедентів, оскільки вона описує функціональність і поведінку, які мають бути визначені найперше.

Для розроблюваної системи розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації було створено діаграму прецедентів, відображену на рис. 2.1.

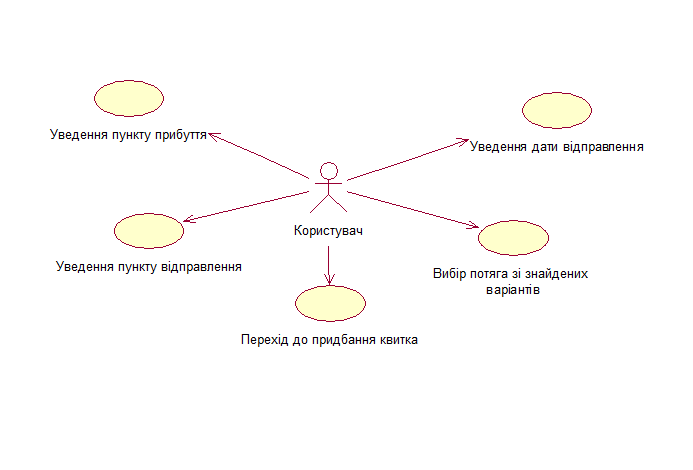


Рисунок 2.1 – Діаграма прецедентів

Передбачається, що із системою взаємодіятиме тільки один актор – користувач. Він матиме можливість обрати пункт відправлення, пункт прибуття, дату відправлення. Після цього користувач зможе вибрати один зі знайдених за введеними параметрами варіантів потягів та перейти безпосередньо до придбання квитка.

Варіанти використання мають виконуватись по черзі (не можна, наприклад, ввести пункт прибуття, якщо не визначено пункт призначення). Однак, якщо користувача не задовольняють результати пошуку (або з інших причин), можна почати введення даних спочатку.

2.3 Проектування класів за допомогою CRC-карток

CRC-картка (англ. Class-responsibility-collaboration card; укр. картка «Клас-відповідальність-кооперація») – метод, призначений для проектування об’єктно-орієнтованого програмного забезпечення. CRC-картки використовуються у тих випадках, коли спочатку у процесі проектування ПЗ визначаються класи і способи їх взаємодії[8].

Зміст CRC-карток:

* назва класу;
* обов’язки класу;
* назви інших класів, з якими пов'язаний даний клас.

Використання CRC-карток при проектуванні ПЗ дає змогу мінімізувати складність дизайну, акцентувати увагу на сутностях (абстрагуватися від деталей), не призначати класу надто багато обов’язків.

У табл. 2.1 – 2.4 наведено розроблені CRC-картки для основних класів.

Таблиця 2.1 – CRC-картка для класу Entity

|  |  |
| --- | --- |
| Entity | |
| Обов’язки | Кооперація |
| Клас описує залізничний рейс | Використовується классом Server |

Таблиця 2.2 – CRC-картка для класу Server

|  |  |
| --- | --- |
| Server | |
| Обов’язки | Кооперація |
| Клас слугує для обробки запитів зі сторони клієнта та роботи із базою даних | Використовує клас Entity, взаємодіє із класом Client |

Таблиця 2.3 – CRC-картка для класу Client

|  |  |
| --- | --- |
| Client | |
| Обов’язки | Кооперація |
| Клас відповідає за взаємодію із сервером та відображення результатів роботи на web-сторінці | Взаємодіє із класом Server |

Таблиця 2.4 – CRC-картка для класу Recognizer

|  |  |
| --- | --- |
| Client | |
| Обов’язки | Кооперація |
| Клас займається розпізнанням мовлення та перетворенням голосових запитів на текст | Взаємодіє із класом Client |

2.4 Діаграма класів

Діаграма класів (англ. Static Structure Diagram) – діаграма, що демонструє класи системи, їх атрибути, методи і взаємозв’язки між ними[9].

Важливою складовою діаграми класів є взаємозв’язки між класами[9]. Взаємозв’язок – це особливий тип логічних відношень між сутностями. В UML представлені наступні види відношень:

* асоціація (об’єкти одного класу (сутності) пов’язані з об’єктами іншої сутності);
* агрегація (об’єкт містить у собі екземпляри інших класів);
* композиція (більш суворий варіант агрегації; є жорстка залежність часу існування екземплярів класу-контейнера та об’єктів, що містяться в них);
* узагальнення, або наслідування (один клас (підтип) є окремою формою іншого (надтип), який називається узагальненням першого);
* реалізація (один елемент реалізує поведінку, задану іншим);
* залежність (зміни у специфікації одного класу ведуть до змін у специфікації іншого).

Для розроблюваної системи розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації було створено діаграму класів, відображену на рис. 2.2.

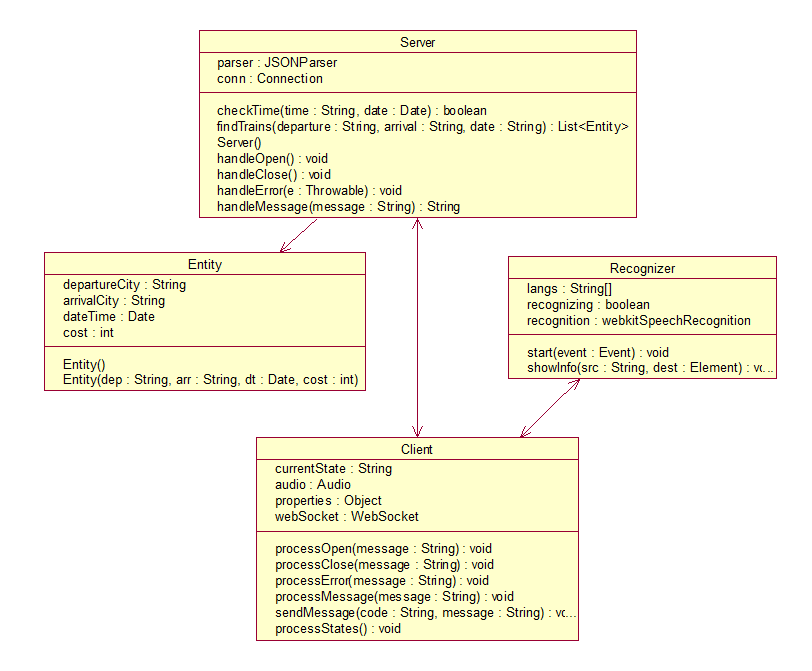


Рисунок 2.2 – Діаграма класів

2.4.1 Клас Entity

Клас описує залізничний рейс.

Поля:

* departureCity – пункт відправлення (у вигляді рядка);
* arrivalCity – пункт призначення (у вигляді рядка);
* dateTime – дата і час відправлення (у вигляді java.util.Date);
* cost – вартість квитка (у вигляді цілого числа).

Методи:

* Entity () – конструктор за замовчуванням;
* Entity (String dep, String arr, Date dt, int cost) –конструктор із параметрами;
* також клас містить методи отримання та встановлення значення для кожного із полів (так звані get- і set-методи).

2.4.2 Клас Server

Клас слугує для обробки запитів зі сторони клієнта та роботи із базою даних.

Поля:

* parser – об’єкт org.json.simple.parser.JSONParser, слугує для роботи з даними у форматі JSON;
* conn – об’єкт java.sql.Connection, необхідний для роботи з базою даних MySQL за допомогою драйвера JDBC.

Методи:

* checkTime (String time, Date date) – перевіряє коректність уведеної дати відправлення (якщо параметр time містить місяць і число, повертає true і змінює поточний стан параметра date, інакше повертає false);
* findTrains (String departure, String arrival, String date) – здійснює пошук рейсів із пункту відправлення departure до пункту призначення з датою відправлення date (повертає знайдені рейси у вигляді списку об’єктів Entity);
* Server () – конструктор за замовчуванням;
* handleOpen () – обробляє відкриття з’єднання із клієнтом;
* handleClose () – обробляє закриття з’єднання із клієнтом;
* handleError (Throwable e) – обробляє помилки при встановленні з’єднання (параметр е типу java.lang.Throwable містить інформацію про подію, що викликала помилку);
* handleMessage (String message) – обробляє повідомлення, що надходять від клієнта (містяться у параметрі message) та повертає власні повідомлення.

2.4.3 Клас Client

Клас відповідає за взаємодію із сервером та відображення результатів роботи на web-сторінці.

Поля:

* currentState – поточний стан програми (етап, на якому вона знаходиться: підтвердження пункту відправлення, уведення дати і т.п.);
* audio – звуковий запис голосової відповіді на запит;
* properties – об’єкт, що описує рейс;
* webSocket – об’єкт, що відповідає за встановлення з’єднання із сервером.

Методи:

* processOpen (String message) – обробляє відкриття з’єднання із сервером;
* processClose (String message) – обробляє закриття з’єднання із сервером;
* processError (String message) – обробляє помилки при встановленні з’єднання;
* processMessage (String message) – обробляє повідомлення, що надходять від сервера;
* sendMessage (String code, String message) – формує JSON-пакет із кодом повідомлення code та текстом message і надсилає його серверу;
* processStates () – обробляє дії користувача залежно від поточного стану.

2.4.4 Клас Recognizer

Клас займається розпізнанням мовлення та перетворенням голосових запитів на текст.

Поля:

* langs – масив строк, у якому перераховано мови, на яких може вестись розпізнання;
* recognizing – показує, чи ведеться розпізнання в даний момент;
* recognition – об’єкт типу webkitSpeechRecognition, частина Google Speech API, відсилає запис фрази, сказаної користувачем, на сервер Google та отримує відповідь – ту ж фразу у вигляді тексту.

Методи:

* start (Event event) – розпочинає розпізнання;
* showInfo (String src, Element dest) – відображає інформацію src в елементі розмітки dest.

2.5 Діаграми послідовності

Діаграма послідовності (англ. sequence diagram) – діаграма, на якій показано взаємодію об’єктів (обмін між ними сигналами та повідомленнями), впорядковану за часом, з відображенням тривалості обробки і послідовності появи[10].

Основними елементами діаграми послідовності є:

* позначення об’єктів;
* вертикальні «лінії життя» (англ. lifeline), що відображають плин часу;
* прямокутники, що відображають діяльність об’єкта чи виконання ним певної функції;
* стрілки, що показують обмін сигналами чи повідомленнями між об’єктами.

Основні діаграми послідовності, розроблені для системи розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації, із коментарями розглянуто у подальших підрозділах.

2.5.1 Діаграма послідовності для процесу завантаження сторінки

Зазначену діаграму відображено на рис. 2.3.

Перш ніж сторінку буде завантажено, програма за допомогою геолокації встановить місцезнаходження користувача, визначить, чи є в базі даних місто, в якому знаходиться користувач, і якщо так, запропонує встановити це місто у якості пункту відправлення.

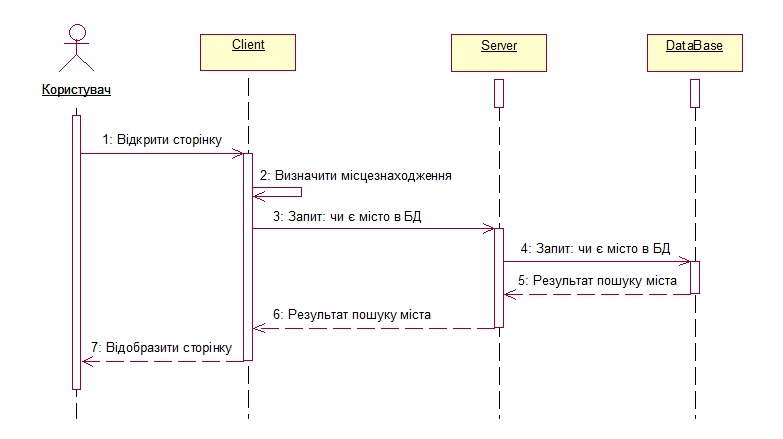


Рисунок 2.3 – Діаграма послідовності для процесу завантаження сторінки

2.5.2 Діаграма послідовності для етапів введення назв міст

Зазначену діаграму відображено на рис. 2.4.

Коли користувач вимовлятиме назви міст (для встановлення у якості пунктів призначення або прибуття), програма перевірятиме, чи є такі міста в базі даних. Якщо так, система зможе перейти до наступного етапу. Якщо такого міста у базі не виявиться, програма повідомить про це і попросить назвати інше місто.

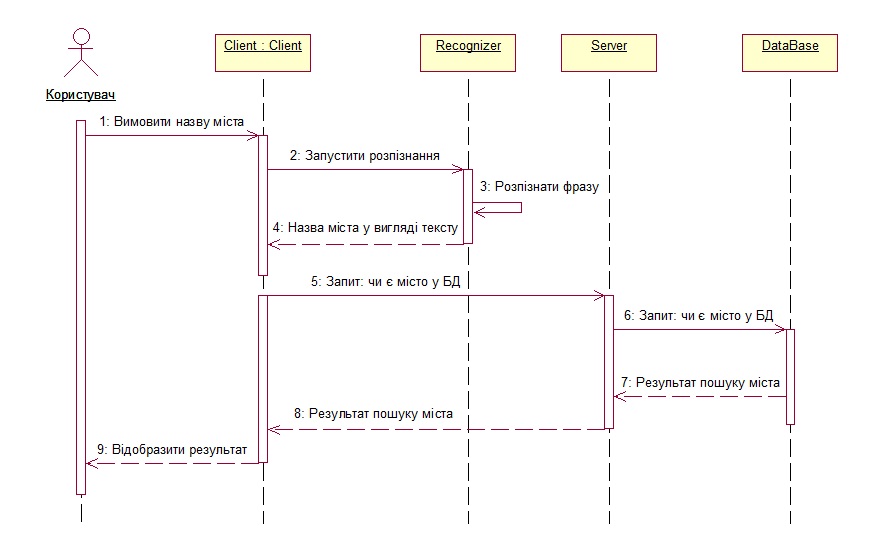


Рисунок 2.4 – Діаграма послідовності для етапів введення назв міст

2.5.3 Діаграма послідовності для етапу введення дати відправлення

Зазначену діаграму відображено на рис. 2.5.

Коли користувач вимовлятиме дату відправлення, програма перевірятиме, чи коректна ця дата (фраза користувача має містити число та місяць). Якщо користувач коректно вказав дату і ця дата настане не більше, ніж через місяць, система зможе перейти до наступного етапу. Якщо дату було вказано некоректно або вона настане більше, ніж через місяць, програма повідомить про помилку і попросить правильно указати дату.

2.5.4 Діаграма послідовності для етапу пошуку рейсів

Зазначену діаграму відображено на рис. 2.6.

Після того, як усі параметри буде введено, система розпочне пошук квитків, що задовольнятимуть указаним вимогам. Результатом буде список знайдених рейсів. Якщо жодного рейсу не буде знайдено, система запропонує вказати інші параметри.

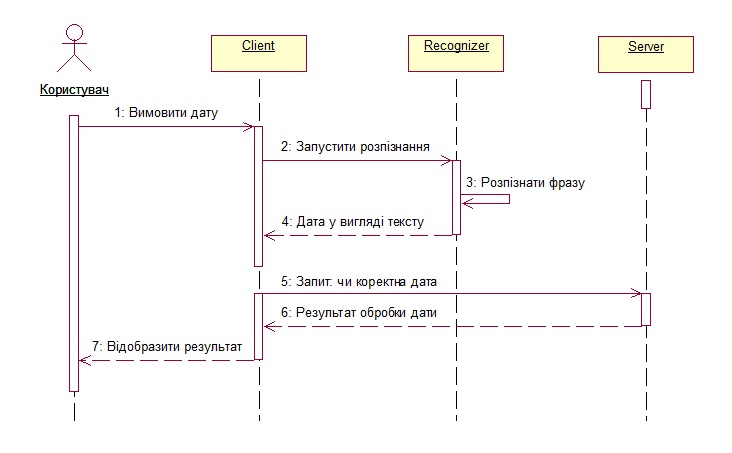


Рисунок 2.5 – Діаграма послідовності для етапу введення дати відправлення

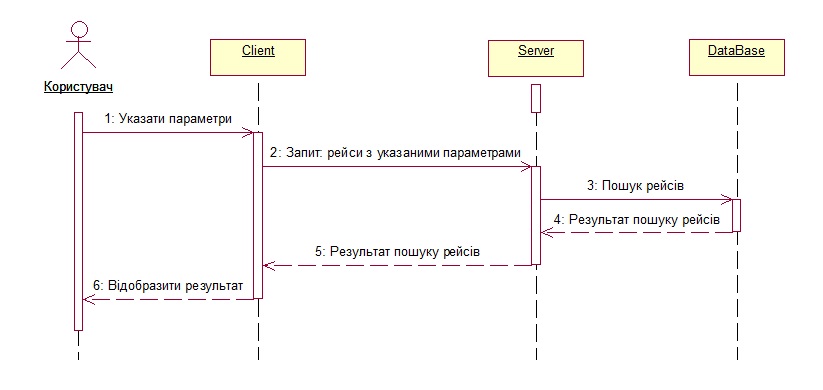


Рисунок 2.6 – Діаграма послідовності для етапу пошуку рейсів

2.6 Структура програми

Структуру програми наведено на рис. 2.7.

Користувач працює із веб-сторінкою index.htm, на якій відображається уся необхідна інформація. Стилі, що використовуються для оформлення цієї сторінки, зберігаються у таблиці стилів styles.css. Скрипти знаходяться у файлах scripts.js та recognition.js. Також використовуються зображення та аудіо файли. Клієнт сполучається із сервером за допомогою технології WebSocket. База даних обробляється у СУБД MySQL, до якої сервер звертається через драйвер JDBC.

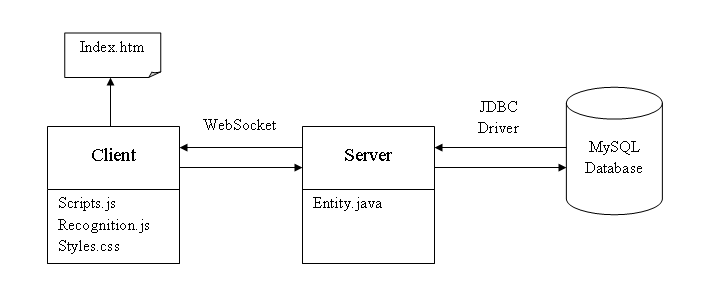


Рисунок 2.7 – Структура програми

2.7 Проектування інтерфейсу користувача

Інтерфейс користувача, або користувацький інтерфейс (англ. user interface, UI) – різновид інтерфейсів, в якому одна сторона представлена людиною (користувачем), а інша – машиною/пристроєм. Являє собою сукупність засобів і методів, за допомогою яких користувач взаємодіє із різноманітними машинами, пристроями, апаратурою[11].

Графічний інтерфейс користувача, або графічний користувацький інтерфейс (англ. graphical user interface, GUI) – різновид користувацького інтерфейсу, елементи якого (меню, кнопки, списки і т.п.), представлені користувачу на екрані, виконані у вигляді графічних зображень. На відміну від інтерфейсу командної строки, у графічному інтерфейсі користувач має довільний доступ до всіх видимих екранних об’єктів (елементів інтерфейсу) і здійснює безпосереднє маніпулювання ними. Графічний інтерфейс є частиною користувацького інтерфейсу і визначає взаємодію з користувачем на рівні візуалізованої інформації[11].

Інтелектуальний інтерфейс – такий різновид інтерфейсу, що збільшує діапазон способів введення та виведення інформації, збагачує граматики введення та виведення та намагається кооперуватися с користувачем для досягнення цілей. Містить модель предметної області для порівняння із контекстними умовами[12].

Інтерфейс розроблюваного програмного засобу є графічним та містить елементи інтелектуального (зокрема, голосове введення та виведення даних).

2.7.1 Розробка структури та сценарію діалогу

Діалог – це послідовність взаємодій між користувачем і програмою, в результаті яких розв’язуються певні задачі. Складність діалогу визначається функціональними можливостями додатка та можливими діями користувачів. Розвиток діалогу з часом можна розглядати як послідовність переходів системи з одного стану в інший. Жоден із цих станів не повинен бути «тупиковим», тобто користувач повинен мати можливість перейти з будь-якого поточного стану діалогу в необхідний (за один чи кілька кроків). Для цього необхідно визначити усі можливі стани діалогу та шляхи переходу з одного стану в інший[13].

Існують наступні варіанти структури діалогу:

* діалог типу «питання-відповідь»;
* діалог на основі меню;
* діалог на основі екранних форм;
* діалог на основі командної мови.

Кожний із цих варіантів має свої особливості і є найбільш зручним для певного класу задач. Для системи розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації було обрано структуру діалогу «питання-відповідь».

Структура діалогу типу «питання-відповідь» заснована на аналогії зі звичайним інтерв’ю. Система бере на себе роль опитувача та отримує інформацію від користувача у вигляді відповідей на питання. У кожній точці діалогу система виводить у якості підказки одне питання, на яке користувач дає одну відповідь. Залежно від отриманої відповіді система може вирішувати, яке питання поставити далі. Дана структура діалогу надає природний механізм введення як команд, так і даних. Особливо доцільна така структура в експертних системах, де діалог має багато розгалужень і для кожного питання передбачена велика кількість відповідей, кожна з яких впливає на те, яке питання буде поставлено наступним.

Розроблений сценарій діалогу відображено на рис. 2.8.

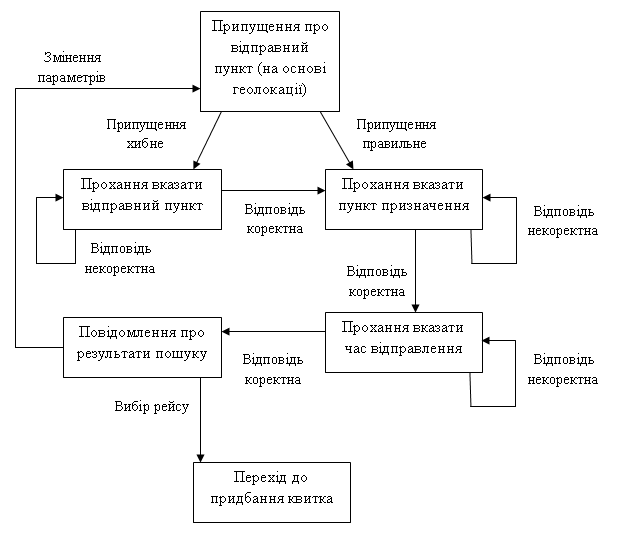


Рисунок 2.8 – Схема сценарію діалогу

2.7.2 Розробка інтерфейсу

На кожному з етапів роботи програми на екрані відображаються:

* текст запитання (складається із двох частин: безпосередньо запитання і його пояснення; програма також програє аудіо запис із текстом запитання);
* текст відповіді (текст, вимовлений користувачем; поки аналіз мовлення не завершено, цей текст буде сірого кольору, а коли завершено – чорного);
* підказки (щодо того, що треба зробити, щоб почати говорити, та що саме говорити).

Щоб почати говорити, треба натиснути і утримувати кнопку з піктограмою мікрофона на екрані або «space» (пробіл) на клавіатурі. При першому запуску програми необхідно дати їй дозвіл на використання мікрофона.

Приклади форматування екрану відображені на рисунках 2.9 – 2.11.

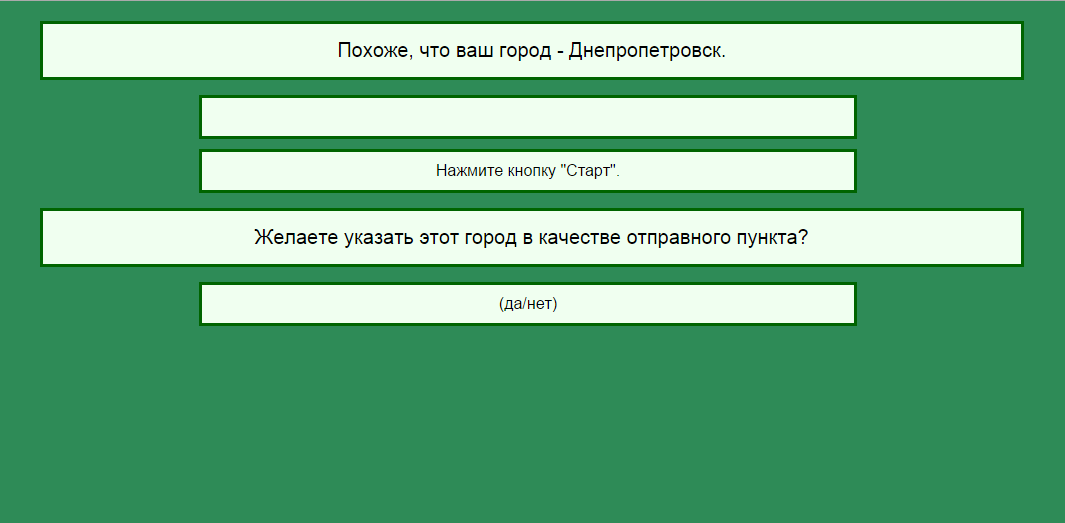


Рисунок 2.9 – Приклад форматування (етап підтвердження пункту відправлення)

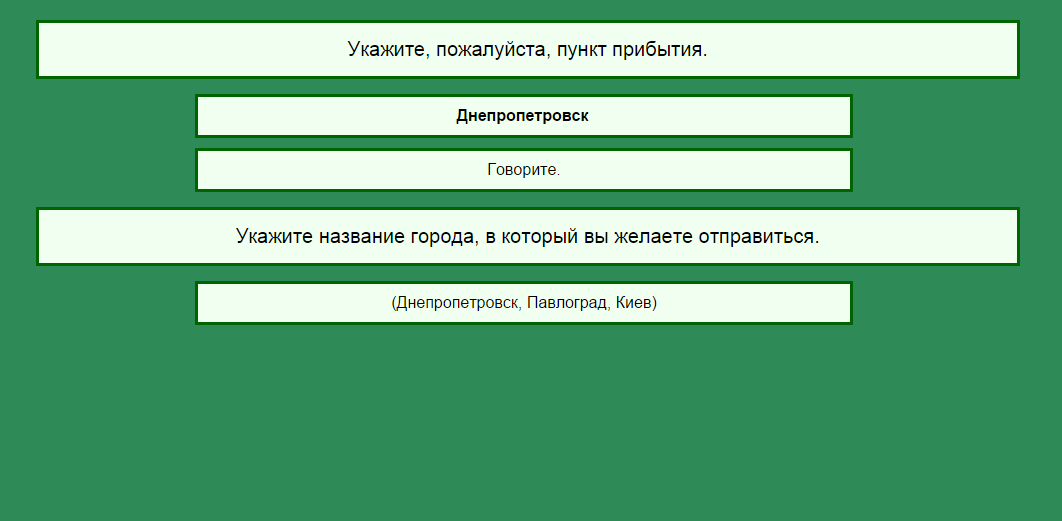


Рисунок 2.10 – Приклад форматування (етап визначення пункту прибуття)

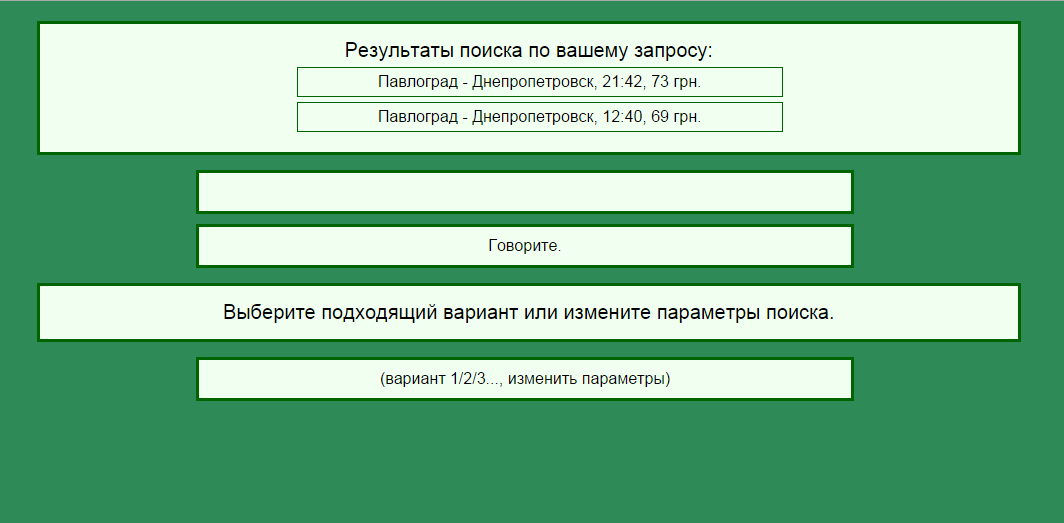


Рисунок 2.11 – Приклад форматування (етап вибору рейсу)

2.8 Розробка бази даних

База даних – сукупність даних, організованих відповідно до концептуальної структури, що описує характеристики цих даних та відносини між ними. База даних є інформаційною моделлю певної предметної області[14].

Предметна область – частина реального світу, множина усіх предметів, властивості яких і відносини між якими розглядаються у межах даного контексту[14].

Відомі три різновиди структури даних: ієрархічна, мережева і таблична. Відповідно, за ознакою структури бази даних розподіляються на ієрархічні, мережеві і табличні (реляційні). Найбільш поширеним типом є реляційні бази даних. Відомо, що будь-яку структуру даних можна звести до табличної форми[14]. Для розроблюваної системи також буде використовуватись реляційна база даних.

Крім цього, класифікація баз даних можлива за характером інформації (фактографічні та документальні БД) та за способом зберігання даних (централізовані та розподілені БД).

Основні поняття організації даних у реляційних БД:

* таблиця – об’єкт бази даних, який слугує для зберігання інформації про певну сутність;
* запис – рядок таблиці, що містить набір значень властивостей, розташованих у полях бази даних;
* поле – стовпець таблиці, що містить значення певної властивості;
* тип поля – визначається типом даних, які в ньому містяться (основні типи даних: лічильник, текстовий, числовий, дата/час, грошовий, логічний, об’єкт OLE, гіперпосилання);
* головний ключ таблиці – поле, значення якого є унікальними та дають змогу однозначно визначити запис у таблиці.

СУБД (система управління базами даних) – програмне забезпечення для роботи з базами даних.

2.8.1 Вибір СУБД

Більш конкретно, система управління базами даних (СУБД) – це сукупність програмних та лінгвістичних засобів загального чи спеціального призначення, що забезпечують управління створенням та використанням баз даних[15].

Основні характеристики СУБД:

* контроль за надлишковістю даних;
* несуперечливість даних;
* підтримка цілісності бази даних (коректність та несуперечливість);
* цілісність описується за допомогою обмежень;
* незалежність прикладних програм від даних;
* спільне використання даних;
* підвищений рівень безпеки.

Можливості СУБД:

* дозволяється створювати БД (здійснюється за допомогою мови визначення даних DDL (Data Definition Language));
* дозволяється додавання, оновлення, видалення та читання інформації з БД (за допомогою мови маніпулювання даними DML, яку часто називають мовою запитів);
* можна надавати контрольований доступ до БД за допомогою: системи забезпечення захисту, яка запобігає несанкціонованому доступу до БД, системи керування паралельною роботою прикладних програм, яка контролює процеси спільного доступу до БД, системи відновлення, яка дозволяє відновлювати БД до попереднього несуперечливого стану, що був порушений в результаті збою апаратного або програмного забезпечення.

У якості СУБД для системи розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації біло обрано MySQL Server 5.5. Такий вибір зроблено через наступні особливості цієї СУБД:

* MySQL є вільною системою управління базами даних;
* MySQL портовано на більшість сучасних платформ;
* MySQL має API для мови Java;
* висока швидкодія (завдяки внутрішньому механізму багатопотоковості) та невибагливість до ресурсів;
* безпека (завдяки системі із п’яти службових таблиць завжди можна описати, який користувач з якого домену з якою таблицею може працювати і які команди може здійснювати);
* надійність та стабільність пакету.

2.8.2 Проектування бази даних

Основні задачі проектування:

* забезпечення зберігання в БД усієї необхідної інформації;
* забезпечення можливості отримання даних за всіма необхідними запитами;
* скорочення надмірності та дублювання даних;
* забезпечення цілісності бази даних.

2.8.2.1 Концептуальне проектування

Концептуальне (інфологічне) проектування – побудова семантичної моделі найбільш високого рівня абстракції. Така модель створюється без орієнтації на якусь конкретну СУБД чи модель даних. Вид і зміст концептуальної моделі бази даних визначається обраним формальним апаратом (зазвичай це графічні нотації, подібні до ER-діаграм)[16].

Найчастіше концептуальна модель бази даних включає:

* опис інформаційних об’єктів чи понять предметної області і зв’язків між ними;
* опис обмежень цілісності (вимог до допустимих значень даних та зв’язків між ними).

Виходячи з прийнятих при проектуванні (зокрема в розділі 2.5.1) рішень, звертання до бази даних відбуватиметься у двох випадках:

* щоб перевірити наявність певного міста;
* щоб знайти рейс із указаними параметрами (пункт відправлення, пункт прибуття, час відправлення, вартість квитка).

Таким чином, достатньо, щоб у базі були описані дві сутності: місто та рейс. Модель, що описує їх, зображено на рис. 2.12.

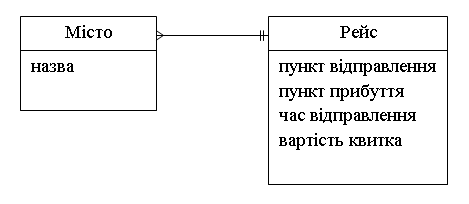


Рисунок 2.12 – Концептуальна модель бази даних

2.8.2.2 Логічне проектування

Логічне (даталогічне) проектування – створення схеми бази даних на основі конкретної моделі даних (реляційної, мережевої або ієрархічної)[16].

Перетворення концептуальної моделі на логічну здійснюється за формальними правилами.

На етапі логічного проектування враховується специфіка конкретної моделі даних, але може не враховуватися специфіка конкретної СУБД.

Оскільки, як зазначалося, було прийнято рішення використовувати реляційну модель, дані у базі являтимуть собою набір відношень. База міститиме наступні таблиці:

* таблиця «Міста» (атрибути: ідентифікатор, назва);
* таблиця «Рейси» (атрибути: ідентифікатор, пункт відправлення, пункт прибуття, час відправлення, вартість квитка).

Оскільки пункти відправлення та прибуття є містами, у таблиці «Рейси» доцільно зберігати не їх назви, а відповідні ідентифікатори з таблиці «Міста». Ключовими в обох таблицях є поля «ідентифікатор».

Використання реляційної моделі передбачає проведення нормалізації відношень (перевірки на відповідність нормальним формам).

Нормальна форма – властивість відношення у реляційній моделі даних, що характеризує його з точки зору надмірності, яка потенційно призводить до логічно помилкових результатів вибірки чи зміни даних. Нормальна форма визначається як сукупність вимог, яким повинне задовольняти відношення.

Були розглянуті перша, друга та третя нормальні форми.

Перша нормальна форма утворює ґрунт для структурованої схеми баз даних:

* кожна таблиця повинна мати основний ключ: мінімальний набір колонок, які ідентифікують запис;
* уникнення повторень груп (категорії даних, що можуть зустрічатись різну кількість разів в різних записах) правильно визначаючи неключові атрибути;
* атомарність: кожен міста атрибут повинен мати лише одне значення, а не множину значень.

Спроектована модель відповідає вимогам першої нормальної форми.

Друга нормальна форма вимагає, аби дані, що зберігаються в таблицях із композитним ключем не залежали лише від частини ключа:

* схема бази даних повинна відповідати вимогам першої нормальної форми;
* дані, що повторно з'являються в декількох рядках виносяться в окремі таблиці.

Спроектована модель відповідає вимогам другої нормальної форми.

Третя нормальна форма вимагає, аби дані в таблиці залежали винятково від основного ключа:

* схема бази даних повинна відповідати всім вимогам другої нормальної форми;
* будь-яке поле, що залежить від основного ключа та від будь-якого іншого поля, має виноситись в окрему таблицю.

Спроектована модель відповідає вимогам третьої нормальної форми.

2.8.2.3 Фізичне проектування

Фізичне проектування – створення схеми бази даних для конкретної СУБД. Специфіка конкретної СУБД може включати в себе обмеження на іменування об’єктів бази даних, обмеження на типи даних, що підтримуються, і т.п. Крім того, специфіка конкретної СУБД при фізичному проектуванні може включати вибір рішень, пов’язаних із фізичним середовищем зберігання даних (вибір методів управління дисковою пам’яттю, розподіл бази даних за файлами та пристроями, методи доступу до даних), створення індексів і т.д[16].

Схема бази даних, розроблена для СУБД MySQL, відображена на рис. 2.13.

Таблиця «cities» описує міста. Таблиця містить наступні атрибути:

* ключове поле id;
* строкове поле name – назву міста.

Скрипт для створення таблиці на мові MySQL:

DROP TABLE IF EXISTS `cities`;

CREATE TABLE `cities` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

PRIMARY KEY (`id`))

ENGINE=InnoDB AUTO\_INCREMENT=28 DEFAULT CHARSET=cp1251;

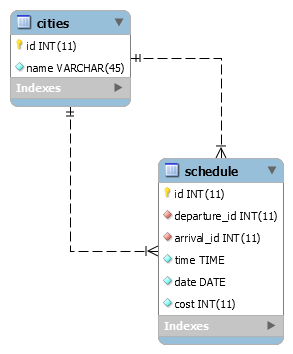


Рисунок 2.13 – Схема бази даних (результат фізичного проектування)

Таблиця «schedule» описує рейси. Таблиця містить наступні атрибути:

* ключове поле id;
* поля departure\_id (пункт відправлення) та arrival\_id (пункт призначення), які пов’язані із ідентифікатором таблиці «cities»;
* поля time та date, що описують час відправлення (час і дата зберігаються окремо, оскільки використовуються також окремо);
* цілочислове поле cost (вартість квитка).

Скрипт для створення таблиці на мові MySQL:

DROP TABLE IF EXISTS `schedule`;

CREATE TABLE `schedule` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`departure\_id` int(11) NOT NULL,

`arrival\_id` int(11) NOT NULL,

`time` time NOT NULL,

`date` date NOT NULL,

`cost` int(11) NOT NULL,

PRIMARY KEY (`id`),

KEY `departure\_idx` (`departure\_id`),

KEY `arrival\_idx` (`arrival\_id`),

CONSTRAINT `arrival` FOREIGN KEY (`arrival\_id`) REFERENCES `cities` (`id`) ON DELETE RESTRICT ON UPDATE RESTRICT,

CONSTRAINT `departure` FOREIGN KEY (`departure\_id`) REFERENCES `cities` (`id`) ON DELETE RESTRICT ON UPDATE RESTRICT)

ENGINE=InnoDB AUTO\_INCREMENT=27379 DEFAULT CHARSET=cp1251;

3 РОЗРОБКА ПРОГРАМИ

3.1 Вибір мови програмування

Серверну частину програми написано з використанням мови Java. Такий вибір зроблено через такі особливості цієї мови:

* незалежність від операційної системи та обладнання;
* автоматичне управління пам’яттю;
* розширені можливості обробки виключних ситуацій;
* наявність стандартних колекцій (масив, список, стек і т.д.);
* наявність класів, що дозволяють виконувати HTTP-запити і обробляти відповіді;
* вбудовані засоби створення багатопоточних програм;
* уніфікований доступ до баз даних (на рівнях окремих SQL-запитів та концепції об’єктів, що володіють здатністю до збереження в базі даних);
* наявність зручних вільних середовищ розробки зі значним функціоналом (NetBeans, Eclipse, IntelliJ IDEA та ін.);
* наявність значної кількості сторонніх бібліотек, фреймворків (ORM, MVC та ін.), що суттєво розширює можливості та полегшує розробку.

Клієнтську частину програми написано з використанням мови JavaScript. Такий вибір зроблено через такі особливості цієї мови:

* підтримка найбільш популярними браузерами «за замовчуванням»;
* висока швидкість роботи;
* можливість роботи з DOM-моделлю веб-сторінки (як модифікувати сторінку, так і реагувати на події, які на ній відбуваються);
* скрипти не потребують компіляції, а підключаються до коду HTML і при завантаженні одразу виконуються;
* можливість виконувати асинхронні запити до сервера і отримувати відповідь без перезавантаження сторінки;
* наявність значної кількості сторонніх бібліотек та фреймворків.

Програмний засіб розроблено у середовищі NetBeans IDE 8.0.2. Такий вибір зроблено через наступні особливості цього засобу розробки:

* підтримка JDK 8 та Java EE 7;
* широкі можливості рефакторингу та профілювання;
* управління системами контролю версій;
* підтримка модульного тестування;
* встановлені фреймворки Spring MVC та Hibernate;
* встановлений драйвер JDBC;
* встановлений HTTP-сервер GlassFish.

3.2 Розробка алгоритму

Узагальнена схема алгоритму роботи програми наведена на рис. 3.1.

Подальша розробка проводилася із використанням методу покрокової деталізації. Даний метод є частиною технології спадного проектування. Розробка алгоритму методом покрокової деталізації полягає в тому, що алгоритм спочатку представляється у вигляді одного розпорядження (постановки задачі), яке потім можна представити у вигляді сукупності більш простих розпоряджень, кожне з яких також можна розподілити (і так доти, доки кожне із розпоряджень не буде елементарним)[17].

Переваги методу покрокової деталізації:

* зберігається концептуальна цілісність програми (рух від складного до простого);
* проектування програми, кодування, перевірку і документування можна проводити паралельно;
* у кожний момент часу в наявності є робочий варіант програми;
* позначення рівнів є хорошими коментарями до програми.

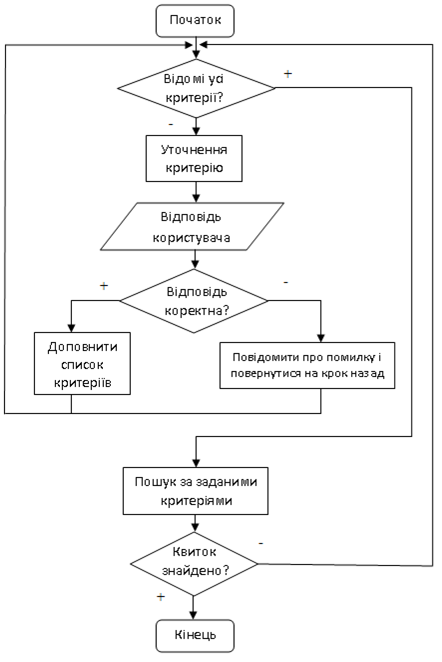


Рисунок 3.1 – Схема загального алгоритму роботи програми

Схема розробки програми, створена за допомогою методу покрокової деталізації, наведена у табл. 3.1.

3.3 Розробка методів

Усі методи, спроектовані на попередньому етапі, також розроблялись с використанням методу покрокової деталізації.

3.3.1 Розробка методу Server.checkTime()

Функціональність: перевіряє коректність уведеної дати відправлення (якщо параметр time містить місяць і число, повертає true і змінює поточний стан параметра date, інакше повертає false). Схема методу наведена в таблиці 3.2.

Таблиця 3.1 – Покрокова деталізація розробки програми

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Р0 | Система розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації | Р1 | Розробка серверної частини програми | Р1.1 | Розробка механізму прийняття та відправлення повідомлень |
| Р1.2 | Розробка механізму взаємодії із базою даних |
| Р1.3 | Розробка функцій для перевірки коректності вхідних даних |
| Р1.4 | Розробка функцій для пошуку квитків із заданими параметрами |
| Р2 | Розробка клієнтської частини програми | Р2.1 | Розробка механізму прийняття та відправлення повідомлень |
| Р2.2 | Розробка функцій для перетворення мовлення на текст |
| Р2.3. | Розробка механізму переходів між етапами роботи |
| Р2.4 | Розробка візуального представлення додатку |

Таблиця 3.2 – Покрокова деталізація перевірки уведеної дати

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Р0 | Перевірка коректності уведеної дати | Р1 | Перевірка отриманого тексту на відповідність вимогам | Р1.1 | Розподіл тексту на слова |
| Р1.2 | Пошук серед отриманих слів назви місяця |
| Р1.3 | Пошук серед отриманих слів числа |
| Р1.4 | Розробка функцій для пошуку квитків із заданими параметрами |
| Р2 | Конвертація тексту в дату | | |

Отриманий текст методу:

public static boolean checkTime(String time, Date date) {

//Р0. Перевірка коректності уведеної дати

boolean res = false;

int number = -1;

int month = -1;

int maxDays = -1;

//Р1. Перевірка отриманого тексту на відповідність вимогам

String[] words = time.split(" ");

for (String i : words) {

for (Month m : months) {

if (m.isSuitable(i)) {

month = m.getID();

maxDays = m.getMaxDays();

}

}

}

for (String i : words) {

try {

number = Integer.parseInt(i);

} catch (NumberFormatException e) {} }

//Р2. Конвертація тексту в дату

if ((month != -1) && (number > 0) && (number < maxDays)) {

res = true;

Calendar calendar = Calendar.getInstance();

calendar.set(calendar.get(Calendar.YEAR), month, number);

date.setTime((calendar.getTime()).getTime());

}

return res;

}

3.3.2 Розробка методу Server. findTrains()

Функціональність: здійснює пошук рейсів із пункту відправлення departure до пункту призначення з датою відправлення date (повертає знайдені рейси у вигляді списку об’єктів Entity). Схема методу наведена в таблиці 3.3.

Таблиця 3.3 – Покрокова деталізація пошуку квитків

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Р0 | Пошук квитків із заданими параметрами | Р1 | Пошук квитків у базі даних | Р1.1 | Підключення до бази |
| Р1.2 | Створення запиту |
| Р1.3 | Виконання запиту |
| Р1.4 | Обробка результатів |
| Р2 | Обробка виключних ситуацій | | |

Отриманий текст методу:

public List<Entity> findTrains(String departure, String arrival, String date) {

//Р0. Пошук квитків із заданими параметрами

List<Entity> res = new ArrayList<>();

//Р1. Пошук квитків у базі даних

try {

Statement statement = conn.createStatement();

Date hlp = new Date();

checkTime(date, hlp);

DateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");

String select = "select \* from schedule where ("

+ "departure\_id = (select id from cities where name='" + departure.toLowerCase() + "') and "

+ "arrival\_id = (select id from cities where name='" + arrival.toLowerCase() + "') and "

+ "date = '" + formatter.format(hlp) + "')";

ResultSet rs = statement.executeQuery(select);

while (rs.next()) {

res.add(new Entity(departure, arrival, rs.getTimestamp("time"), Integer.parseInt(rs.getString("cost"))));

}

}

//Р2. Обробка виключних ситуацій

catch (SQLException ex) {

System.out.println("Trains searching went wrong");

ex.printStackTrace();

}

return res;

}

3.3.3 Розробка методу Server. handleMessage()

Функціональність: обробляє повідомлення, що надходять від клієнта (містяться у параметрі message) та повертає власні повідомлення. Схема методу наведена в таблиці 3.4.

Таблиця 3.4 – Покрокова деталізація обробки повідомлень від клієнта

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Р0 | Обробка повідомлень | Р1 | Прийняття повідомлень | Р1.1 | Обробка запитів на перевірку наявності міста в БД |
| Р1.2 | Обробка запитів на перевірку коректності дати |
| Р1.3 | Обробка запитів на пошук рейсів |
| Р2 | Обробка виключних ситуацій | | |
| Р3 | Відправка відповіді | | |

Отриманий текст методу:

public String handleMessage(String message) {

//Р0. Обробка повідомлень

String res = "";

//Р1. Прийняття повідомлень

try {

JSONObject json = (JSONObject) parser.parse(message);

String type = (String) json.get("type");

JSONObject answer = new JSONObject();

if (type.equals("check\_city")) {

String city = (String) json.get("text");

String hlp = "no";

try {

Statement statement = conn.createStatement();

ResultSet rs = statement.executeQuery("select \* from cities where name='" + city.toLowerCase() + "'");

if (rs.next()) {

hlp = "yes";

}

}

//Р2. Обробка виключних ситуацій

catch (SQLException ex) {

System.out.println("City searching went wrong");

ex.printStackTrace();

}

answer.put("type", "check\_city\_response");

answer.put("text", hlp);

res = answer.toJSONString();

}

if (type.equals("check\_time")) {

String hlp = (String) json.get("text");

answer.put("type", "check\_time\_response");

answer.put("text", (checkTime(hlp,new Date())) ? "yes" : "no");

res = answer.toJSONString();

}

if (type.equals("find\_trains")) {

answer.put("type", "find\_trains\_response");

JSONObject params = (JSONObject) new JSONParser().parse((String)json.get("text"));

List<Entity> list = findTrains((String)params.get("departure"),(String)params.get("arrival"), (String)params.get("date"));

JSONArray arr = new JSONArray();

for (Entity i : list) {

JSONObject obj = new JSONObject();

obj.put("departure",i.getDepartureCity());

obj.put("arrival", i.getArrivalCity());

Calendar calendar = Calendar.getInstance();

calendar.setTime(i.getDateTime());

obj.put("time",String.valueOf(calendar.get(Calendar.HOUR\_OF\_DAY)) + ":" + String.valueOf(calendar.get(Calendar.MINUTE)));

obj.put("cost", String.valueOf(i.getCost()));

arr.add(obj);

}

answer.put("text",arr);

res = answer.toJSONString();

}

//Р2. Обробка виключних ситуацій

} catch (ParseException ex) {

System.out.println("Parsing failed");

ex.printStackTrace();

}

//Р3. Відправка відповіді

return res;

}

3.3.4 Розробка методу processMessage()

Функціональність: обробляє повідомлення, що надходять від сервера. Схема методу наведена в таблиці 3.5.

Таблиця 3.5 – Покрокова деталізація обробки повідомлень від сервера

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Р0 | Обробка повідомлень | Р1 | Прийняття повідомлень | Р1.1 | Обробка відповіді на запит на перевірку наявності міста в БД |
| Р1.2 | Обробка відповіді на запит на перевірку коректності дати |
| Р1.3 | Обробка відповіді на запит на пошук рейсів |
| Р2 | Обробка виключних ситуацій | | |

Отриманий текст методу:

function processMessage (message) {

//Р0. Обробка повідомлень

try {

//Р1. Прийняття повідомлень

var answer = JSON.parse(message.data);

switch(answer.type) {

case "check\_city\_response":

switch (currentState) {

case "city\_confirm":

if (answer.text == "yes") {

showInfo("city\_confirm\_before",div\_before);

showInfo("city\_confirm\_after",div\_after);

showInfo("city\_confirm",hints);

audio.src = "sounds/city\_confirm.mp3";

audio.autoplay = true;

}

break;

case "city\_input":

if (answer.text == "no") {

showInfo("wrong\_city",div\_before);

audio.src = "sounds/wrong\_city.mp3";

audio.autoplay = true;

} else {

currentState = "arrival\_input";

final\_span.innerHTML = "";

showInfo("arrival\_input\_before",div\_before);

showInfo("arrival\_input\_after",div\_after);

showInfo("arrival\_input",hints);

audio.src = "sounds/arrival\_input.mp3";

audio.autoplay = true;

}

break;

case "arrival\_input":

if (answer.text == "no") {

showInfo("wrong\_city",div\_before);

audio.src = "sounds/wrong\_city.mp3";

audio.autoplay = true;

} else {

currentState = "date\_input";

final\_span.innerHTML = "";

showInfo("date\_input\_before",div\_before);

showInfo("date\_input\_after",div\_after);

showInfo("date\_input",hints);

audio.src = "sounds/date\_input.mp3";

audio.autoplay = true;

}

break;

}

break;

case "check\_time\_response":

if (answer.text == "yes") {

sendMessage("find\_trains",JSON.stringify(properties));

currentState = "find\_trains";

}

else {

showInfo("wrong\_date",div\_before);

audio.src = "sounds/wrong\_date.mp3";

audio.autoplay = true;

}

break;

case "find\_trains\_response":

var arr = answer.text;

var str = "";

for (var i = 0; i<arr.length; i++) {

str += "<p class='entity'>";

str += arr[i].departure;

str += " - ";

str += arr[i].arrival;

str += ", ";

str += arr[i].time;

str += ", ";

str += arr[i].cost;

str += " грн.</p>";

}

if (str == "") {

str = "<p>Извините, совпадений не найдено.</p>";

}

final\_span.innerHTML = "";

find\_results\_before.innerHTML = "Результаты поиска по вашему запросу:" + str;

showInfo("find\_results\_before",div\_before);

showInfo("find\_results\_after",div\_after);

showInfo("find\_results",hints);

audio.src = "sounds/find\_results.mp3";

audio.autoplay = true;

break;

default:

alert(answer.type + ": " + answer.text);

}

}

//Р2. Обробка виключних ситуацій

catch(ex) {

alert(ex);

}

}

3.3.5 Розробка методу processStates()

Функціональність: обробляє дії користувача залежно від поточного стану. Схема методу наведена в таблиці 3.6.

Таблиця 3.6 – Покрокова деталізація реагування на дії користувача

|  |  |  |  |
| --- | --- | --- | --- |
| Р0 | Реагування на дії користувача | Р1 | Обробка фраз управління |
| Р2 | Обробка етапу припущення про відправний пункт |
| Р3 | Обробка етапу введення пункту відправлення |
| Р4 | Обробка етапу введення пункту призначення |
| Р5 | Обробка етапу введення дати відправлення |
| Р6 | Обробка етапу пошуку квитків |

Отриманий текст методу:

function processStates() {

//Р0. Реагування на дії користувача

if (final\_span.innerHTML != "") {

//Р1. Обробка фраз управління

if (final\_span.innerHTML.toLowerCase().indexOf("очистить") > -1) {

final\_span.innerHTML = "";

return;

}

switch (currentState) {

//Р2. Обробка етапу припущення про відправний пункт

case "city\_confirm":

if (final\_span.innerHTML.toLowerCase().indexOf("да") > -1) {

currentState = "arrival\_input";

final\_span.innerHTML = "";

showInfo("arrival\_input\_before",div\_before);

showInfo("arrival\_input\_after",div\_after);

showInfo("arrival\_input",hints);

audio.src = "sounds/arrival\_input.mp3";

audio.autoplay = true;

}

if (final\_span.innerHTML.toLowerCase().indexOf("нет") > -1) {

showInfo("city\_input\_before",div\_before);

showInfo("city\_input\_after",div\_after);

showInfo("city\_input",hints);

currentState = "city\_input";

final\_span.innerHTML = "";

audio.src = "sounds/city\_input.mp3";

audio.autoplay = true;

}

break;

//Р3. Обробка етапу введення пункту відправлення

case "city\_input":

properties.departure = final\_span.innerHTML;

sendMessage("check\_city",final\_span.innerHTML);

final\_span.innerHTML = "";

break;

//Р4. Обробка етапу введення пункту призначення

case "arrival\_input":

properties.arrival = final\_span.innerHTML;

sendMessage("check\_city",final\_span.innerHTML);

final\_span.innerHTML = "";

break;

//Р5. Обробка етапу введення дати відправлення

case "date\_input":

properties.date = final\_span.innerHTML;

sendMessage("check\_time",final\_span.innerHTML);

final\_span.innerHTML = "";

break;

//Р6. Обробка етапу пошуку квитків

case "find\_trains":

if (final\_span.innerHTML.toLowerCase().indexOf("параметр") > -1) {

location.reload();

}

if (final\_span.innerHTML.toLowerCase().indexOf("вариант") > -1) {

alert("Вы выбрали " + final\_span.innerHTML);

final\_span.innerHTML = "";

}

break;

} } }

4 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

Тестування програми – це процес виконання програми з метою виявлення помилок. Тестом називається набор вхідних і вихідних даних, які відповідають специфікації.

До найбільш ефективних методів тестування можна віднести комбінацію стратегій чорного ящика і білого ящика. Стратегія «чорного ящика» передбачає вивчення і тестування зовнішньої поведінки функціонуючої програми; стратегія «білого ящика» передбачає вивчення і тестування внутрішньої архітектури і логіки програми на основі її вихідних кодів.

4.1 Тестування методом «білого ящика»

«Білий ящик» ­– тестування коду на предмет логіки роботи програми та коректності її роботи з точки зору компілятора тієї мови, на якій її написано.

Техніка тестування за принципом «білого ящика» (техніка, якою керує логіка програми) дозволяє перевірити внутрішню структуру програми. Виходячи з цієї стратегії тестувальник отримує тестові дані шляхом аналізу логіки роботи програми[18].

Найбільш поширеними методами тестування за технікою «білого ящика» є наступні:

* покриття операторів;
* покриття рішень;
* покриття умов;
* покриття рішень та умов;
* комбінаторне покриття умов.

Було вирішено, що найбільш доцільним буде проведення тестування розроблених методів за допомогою покриття умов. Метод покриття умов полягає в такому підборі тестів, коли кожна умова (елементарне судження в умовних операторах) приймає як істинне так і хибне значення.

Вхідними даними для тестування програми методами білого ящика є специфікації функції та текст програми.

Таким чином далі буде надаватися:

* текст методу, який тестуємо;
* специфікація;
* тестування методом покриття умов.

4.1.1 Тестування методу checkTime()

Специфікація: метод перевіряє коректність уведеної дати відправлення (якщо параметр time містить місяць і число, повертає true і змінює поточний стан параметра date, інакше повертає false);

Вхідні дані:

* time – дата для перевірки (строкове значення);
* date – об’єкт класу Date, до якого буде занесено розпізнану дату.

Вихідні дані:

* ознака коректності отриманої дати (логічне значення).

Текст методу:

public static boolean checkTime(String time, Date date) {

boolean res = false;

int number = -1;

int month = -1;

int maxDays = -1;

String[] words = time.split(" ");

for (String i : words) {

for (Month m : months) {

if (m.isSuitable(i)) {

month = m.getID();

maxDays = m.getMaxDays();

}

}

}

for (String i : words) {

try {

number = Integer.parseInt(i);

} catch (NumberFormatException e) {}

}

if ((month != -1) && (number > 0) && (number < maxDays)) {

res = true;

Calendar calendar = Calendar.getInstance();

calendar.set(calendar.get(Calendar.YEAR), month, number);

date.setTime((calendar.getTime()).getTime());

}

return res;

}

Розроблені тести:

Тест 1.

Вхід: time = “26 июня”, date = new Date().

Вихід: true.

Тест 2.

Вхід: time = “32 чактобря”, date = new Date().

Вихід: false.

Тест 3.

Вхід: time = “май -1”, date = new Date().

Вихід: false.

Результати тестування за методом покриття умов наведено у табл. 4.1.

Таблиця 4.1 – Результати тестування методу checkTime()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Тест № | month != -1 | | number > 0 | | number < maxDays | |
| + | - | + | - | + | - |
| 1 | + |  | + |  | + |  |
| 2 |  | + | + |  |  | + |
| 3 | + |  |  | + | + |  |

4.1.2 Тестування методу handleMessage()

Специфікація: обробляє повідомлення, що надходять від клієнта (містяться у параметрі message) та повертає власні повідомлення.

Вхідні дані:

* message – прийняте повідомлення у форматі JSON (строкове значення).

Вихідні дані:

* власне повідомлення у форматі JSON (строкове значення).

Текст методу:

public String handleMessage(String message) {

String res = "";

try {

JSONObject json = (JSONObject) parser.parse(message);

String type = (String) json.get("type");

JSONObject answer = new JSONObject();

if (type.equals("check\_city")) {

String city = (String) json.get("text");

String hlp = "no";

try {

Statement statement = conn.createStatement();

ResultSet rs = statement.executeQuery("select \* from cities where name='" + city.toLowerCase() + "'");

if (rs.next()) {

hlp = "yes";

}

}

catch (SQLException ex) {

System.out.println("City searching went wrong");

ex.printStackTrace();

}

answer.put("type", "check\_city\_response");

answer.put("text", hlp);

res = answer.toJSONString(); }

if (type.equals("check\_time")) {

String hlp = (String) json.get("text");

answer.put("type", "check\_time\_response");

answer.put("text", (checkTime(hlp,new Date())) ? "yes" : "no");

res = answer.toJSONString();

}

if (type.equals("find\_trains")) {

answer.put("type", "find\_trains\_response");

JSONObject params = (JSONObject) new JSONParser().parse((String)json.get("text"));

List<Entity> list = findTrains((String)params.get("departure"), (String)params.get("arrival"), (String)params.get("date"));

JSONArray arr = new JSONArray();

for (Entity i : list) {

JSONObject obj = new JSONObject();

obj.put("departure",i.getDepartureCity());

obj.put("arrival", i.getArrivalCity());

Calendar calendar = Calendar.getInstance();

calendar.setTime(i.getDateTime());

obj.put("time",String.valueOf(calendar.get(Calendar.HOUR\_OF\_DAY)) + ":" + String.valueOf(calendar.get(Calendar.MINUTE)));

obj.put("cost", String.valueOf(i.getCost()));

arr.add(obj);

}

answer.put("text",arr);

res = answer.toJSONString();

}

} catch (ParseException ex) {

System.out.println("Parsing failed");

ex.printStackTrace(); } return res; }

Розроблені тести:

Тест 1.

Вхід: message = “{‘type’: ‘check\_city’, ‘text’: ‘Павлоград’}”.

Вихід: “{‘type’: ‘check\_city\_response’, ‘text’: ‘yes’}”.

Тест 2.

Вхід: message = “{‘type’: ‘check\_time’, ‘text’: ‘’}”.

Вихід: “{‘type’: ‘check\_time\_response’, ‘text’: ‘no’}”.

Тест 3.

Вхід: message = “{‘type’: ‘find\_trains’, ‘text’: ‘’}”.

Вихід: “{‘type’: ‘find\_trains\_response’, ‘text’: ‘’}”.

Результати тестування за методом покриття умов наведено у табл. 4.2.

Таблиця 4.2 – Результати тестування методу handleMessage()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тест № | type.equals  (“check\_city”) | | type.equals  (“check\_time”) | | type.equals  (“find\_trains”) | | rs.next() | |
| + | - | + | - | + | - | + | - |
| 1 | + |  |  | + |  | + | + |  |
| 2 |  | + | + |  |  | + |  | + |
| 3 |  | + |  | + | + |  |  | + |

4.1.3 Тестування методу processMessage()

Специфікація: обробляє повідомлення, що надходять від сервера.

Вхідні дані:

* message – прийняте повідомлення у форматі JSON (строкове значення).

Метод не повертає значень, результатом роботи є перехід між етапами та відображення змін на екрані.

Текст методу:

function processMessage (message) {

try {

var answer = JSON.parse(message.data);

switch(answer.type) {

case "check\_city\_response":

switch (currentState) {

case "city\_confirm":

if (answer.text == "yes") {

showInfo("city\_confirm\_before",div\_before);

showInfo("city\_confirm\_after",div\_after);

showInfo("city\_confirm",hints);

audio.src = "sounds/city\_confirm.mp3";

audio.autoplay = true;

}

break;

case "city\_input":

if (answer.text == "no") {

showInfo("wrong\_city",div\_before);

audio.src = "sounds/wrong\_city.mp3";

audio.autoplay = true;

} else {

currentState = "arrival\_input";

final\_span.innerHTML = "";

showInfo("arrival\_input\_before",div\_before);

showInfo("arrival\_input\_after",div\_after);

showInfo("arrival\_input",hints);

audio.src = "sounds/arrival\_input.mp3";

audio.autoplay = true;

}

break;

case "arrival\_input":

if (answer.text == "no") {

showInfo("wrong\_city",div\_before);

audio.src = "sounds/wrong\_city.mp3";

audio.autoplay = true;

} else {

currentState = "date\_input";

final\_span.innerHTML = "";

showInfo("date\_input\_before",div\_before);

showInfo("date\_input\_after",div\_after);

showInfo("date\_input",hints);

audio.src = "sounds/date\_input.mp3";

audio.autoplay = true;

}

break;

}

break;

case "check\_time\_response":

if (answer.text == "yes") {

sendMessage("find\_trains",JSON.stringify(properties));

currentState = "find\_trains";

}

else {

showInfo("wrong\_date",div\_before);

audio.src = "sounds/wrong\_date.mp3";

audio.autoplay = true;

}

break;

case "find\_trains\_response":

var arr = answer.text;

var str = "";

for (var i = 0; i<arr.length; i++) {

str += "<p class='entity'>" + arr[i].departure + " - " + arr[i].arrival + ", " + arr[i].time + ", " + arr[i].cost + " грн.</p>";

}

if (str == "") {

str = "<p>Извините, совпадений не найдено.</p>";

}

final\_span.innerHTML = "";

find\_results\_before.innerHTML = "Результаты поиска по вашему запросу:" + str;

showInfo("find\_results\_before",div\_before);

showInfo("find\_results\_after",div\_after);

showInfo("find\_results",hints);

audio.src = "sounds/find\_results.mp3";

audio.autoplay = true;

break;

default:

alert(answer.type + ": " + answer.text);

}

}

catch(ex) {

alert(ex);

}

}

Розроблені тести:

Тест 1.

Вхід: message = “{‘type’: ‘check\_city\_response’, ‘text’: ‘yes’}”, currentState = “city\_confirm”.

Вихід: перехід до етапу підтвердження відправного пункту.

Тест 2.

Вхід: message = “{‘type’: ‘check\_city\_response’, ‘text’: ‘no’}”, currentState = “city\_input”.

Вихід: повідомлення про помилку (місто з указаною назвою відсутнє в базі даних).

Тест 3.

Вхід: message = “{‘type’: ‘check\_city\_response’, ‘text’: ‘yes’}”, currentState = “arrival\_input”.

Вихід: перехід до етапу введення дати відправлення.

Тест 4.

Вхід: message = “{‘type’: ‘check\_time\_response’, ‘text’: ‘yes’}”.

Вихід: перехід до етапу пошуку квитків.

Тест 5.

Вхід: message = “{‘type’: ‘find\_trains\_response’, ‘text’: ‘{}’}”.

Вихід: повідомлення про результат пошуку (рейсів не знайдено).

Тест 6.

Вхід: message = “{‘type’: ‘find\_trains\_response’, ‘text’: ‘{‘departure’: ‘’, ‘arrival’: ‘’, ‘time’: ‘’, ‘cost’: ‘’}’}”.

Вихід: повідомлення про результат пошуку (з переліком знайдених рейсів).

Результати тестування за методом покриття умов наведено у табл. 4.3.

4.2 Тестування методом «чорного ящика»

У цьому методі програма розглядається як «чорний ящик», тобто її алгоритм невідомий. Мета тестування – визначення обставин, за яких поведінка програми не відповідає специфікації. Для знаходження усіх помилок у програмі необхідно виконати вичерпне тестування, тобто тестування на всіх можливих наборах даних. Для більшості програм це неможливо, тому застосовують «розумне» тестування, обмежуючись невеликою підмножиною з усіх можливих наборів даних. При цьому необхідно вибирати найбільш підходящі підмножини (ті, в яких імовірність знаходження помилок найбільша) [19].

Таблиця 4.3 – Результати тестування методу handleMessage()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тест № | answer.type == “check\_city\_response” | | answer.type == “check\_time\_response” | | answer.type == “find\_trains\_response” | | currentState == “city\_confirm” | | currentState == “city\_input” | | currentState == “arrival\_input” | | str == “” | | answer.text == “yes” | |
| + | - | + | - | + | - | + | - | + | - | + | - | + | - | + | - |
| 1 | + |  |  | + |  | + | + |  |  | + |  | + |  |  | + |  |
| 2 | + |  |  | + |  | + |  | + | + |  |  | + |  |  |  | + |
| 3 | + |  |  | + |  | + |  | + |  | + | + |  |  |  |  | + |
| 4 |  | + | + |  |  | + |  |  |  |  |  |  |  |  | + |  |
| 5 |  | + |  | + | + |  |  |  |  |  |  |  | + |  |  |  |
| 6 |  | + |  | + | + |  |  |  |  |  |  |  |  | + |  |  |

Найбільш поширеними методами тестування за технікою «чорного ящика» є наступні:

* еквівалентне розбиття;
* аналіз граничних значень;
* аналіз причинно-наслідкових зв’язків;
* припущення про помилку.

Було вирішено, що найбільш доцільним буде проведення тестування розроблених методів методом еквівалентного розбиття. Метод еквівалентного розбиття полягає у розбитті всіх тестів на класи еквівалентності. Якщо один тест виявляє помилку, то і інший виявить, та навпаки.

4.2.1 Тестування методу checkTime()

Специфікація: метод перевіряє коректність уведеної дати відправлення (якщо параметр time містить місяць і число, повертає true і змінює поточний стан параметра date, інакше повертає false).

Вхідні дані:

* time – дата для перевірки (строкове значення);
* date – об’єкт класу Date, до якого буде занесено розпізнану дату.

Вихідні дані:

* ознака коректності отриманої дати (логічне значення).

Класи еквівалентності наведено у табл. 4.4.

Таблиця 4.4 – Класи еквівалентності методу checkTime()

|  |  |  |
| --- | --- | --- |
| Вхідні умови | Правильні класи | Неправильні класи |
| Місяць | Назви місяців російською мовою у називному або родовому відмінках | Будь-які інші слова |
| Число | Від 1 до максимальної к-сті днів у визначеному місяці | Будь-яке інше число |

Розроблені тести:

Тест 1.

Вхід: time = “26 июня”, date = new Date().

Вихід: true.

Тест 2.

Вхід: time = “май 31”, date = new Date().

Вихід: true.

Тест 3.

Вхід: time = “выходной”, date = new Date().

Вихід: false.

Тест 4.

Вхід: time = “-1 июня”, date = new Date().

Вихід: false.

Тест 5.

Вхід: time = “май 32”, date = new Date().

Вихід: false.

4.2.2 Тестування методу findTrains()

Специфікація: здійснює пошук рейсів із пункту відправлення departure до пункту призначення з датою відправлення date (повертає знайдені рейси у вигляді списку об’єктів Entity).

Вхідні дані:

* departure – пункт відправлення (строкове значення);
* arrival – пункт прибуття (строкове значення);
* date – дата відправлення (строкове значення).

Також метод звертається до бази даних. Відношення, використані для тестування методу, описані у табл. 4.5, 4.6.

Таблиця 4.5 – Тестове відношення cities

|  |  |
| --- | --- |
| ID | Name |
| 1 | Павлоград |
| 2 | Днепропетровск |
| 3 | Киев |

Таблиця 4.6 – Тестове відношення schedule

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Departure\_ID | Arrival\_ID | Date | Time | Cost |
| 1 | 2 | 1 | 25.06.2015 | 6:45:00 | 75 |
| 2 | 2 | 1 | 25.06.2015 | 19:20:00 | 125 |
| 3 | 1 | 2 | 25.06.2015 | 7:50:00 | 70 |
| 4 | 1 | 2 | 25.06.2015 | 20:30:00 | 115 |

Вихідні дані:

* знайдені рейси у вигляді списку об’єктів класу Entity.

Класи еквівалентності наведено у табл. 4.7.

Таблиця 4.7 – Класи еквівалентності методу findTrains()

|  |  |  |
| --- | --- | --- |
| Вхідні умови | Правильні класи | Неправильні класи |
| Пункти відправлення та прибуття | Назви міст України | Назви інших населених пунктів, інші назви |
| Дата відправлення | Дати, що настануть упродовж наступного місяця | Інші дати |

Розроблені тести:

Тест 1.

Вхід: departure = “Днепропетровск”, arrival = “Павлоград”, date = “25 июня”, база даних (опис у табл. 4.5, 4.6).

Вихід: {Entity(“Днепропетровск”,” Павлоград ”,”25.06.2015 6:45:00”,75), Entity(“Днепропетровск”,” Павлоград ”,”25.06.2015 19:20:00”,125)}.

Тест 2.

Вхід: departure = “Днепропетровск”, arrival = “Токио”, date = “25 июня”, база даних (опис у табл. 4.5, 4.6).

Вихід: {}.

Тест 3.

Вхід: departure = “Днепропетровск”, arrival = “пригород”, date = “25 июня”, база даних (опис у табл. 4.5, 4.6).

Вихід: {}.

Тест 4.

Вхід: departure = “Днепропетровск”, arrival = “Павлоград”, date = “31 декабря”, база даних (опис у табл. 4.5, 4.6).

Вихід: {}.

4.3 Вибір методів налагодження програми

Налагодження – це процес локалізації та виправлення помилок, знайдених при тестуванні програмного забезпечення. Локалізацією називають процес визначення оператора програми, виконання якого викликало порушення нормального розрахункового процесу. Для виправлення помилки необхідно визначити її причину, тобто знайти оператор чи фрагмент, який містить помилку[20].

Відповідно до етапу розробки, на якому з’являються помилки, розрізняють:

* синтаксичні помилки – помилки, які фіксуються компілятором виконанні синтаксичного і частково семантичного розбору програми;
* помилки компонування – помилки, виявлені редактором зв’язків при об’єднанні модулів програми;
* помилки виконання – помилки, виявлені операційною системою, апаратними засобами або користувачем при виконанні програми.

Синтаксичні помилки та помилки компонування можна знаходити та виправляти за допомогою автоматизованих засобів, вбудованих у середовище розробки, паралельно із написанням коду. Помилки виконання виявити зазвичай важче[20].

Виділяють наступні способи проявлення помилок виконання:

* поява повідомлення про помилку, зафіксовану схемами контролю виконання машинних команд (переповнення розрядної сітки, ділення на нуль, порушення адресації);
* поява повідомлення про помилку, виявлену операційною системою (порушення захисту пам’яті, спроба запису на пристрої, захищені від запису, відсутність файлу із заданим іменем);
* «зависання» комп’ютера (ігнорування дій користувача);
* розбіжність отриманих результатів із очікуваними.

Налагодження програми передбачає обдумування і логічне осмислення усієї наявної інформації про помилку. Більшість помилок виявляються за непрямими ознаками шляхом детального аналізу текстів програм і результатів аналізу. При цьому використовують наступні методи:

* тестування вручну;
* індукції;
* дедукції;
* зворотного відстеження.

Для впровадження в даній розробці було обрано метод тестування вручну. Цей спосіб є найпростішим та найбільш природним. При виявленні помилки необхідно виконати програму, що тестується, вручну, використовуючи тестовий набір, при роботі з яким була знайдена помилка. Метод є ефективним для невеликим програм без складних обчислень (саме такою програмою є розроблювана система розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації).

4.4 Налагодження програми

Опис знайдених помилок, місця їх виникнення та обрані способи усунення зазначені нижче.

4.4.1 Налагодження методу checkTime() класу Server

При введенні дати з некоректним числом (менше першого або більше, ніж кількість днів у обраному місяці), програма не повідомляла про помилку і неправильно розпізнавала дату.

Проблема виникала тому, що робота з датою базується на взаємодії об’єктів класів Date та Calendar. Ці класи не мають контролю вхідних даних та дозволяють задавати некоректне число, додаючи певну кількість днів з наступного місяця або віднімаючи з попереднього (наприклад, 32 травня не буде помилковим значенням, однак зберігатиметься як 1 червня).

Для усунення проблеми було організовано власну перевірку коректності вхідних даних.

4.4.2 Налагодження методу start() класу webkitSpeechRecognition

За несприятливих умов (зокрема, у шумних приміщеннях) програма відображала проміжний результат розпізнання голосу, але не переходила до остаточного результату.

Проблема виникала тому, що розпізнання голосу реалізується за допомогою Google Speech API. Цей інструмент, почавши процес розпізнання, продовжує його доти, доки користувач продовжує говорити. Якщо у приміщенні високий рівень шуму, програма вважає, що користувач говорить, навіть якщо він уже припинив говорити. Тому програма не могла перейти до відображення остаточних результатів.

Для усунення проблеми уведено можливість використовувати проміжні результати розпізнання, якщо остаточні не були отримані.

4.4.3 Налагодження методу processStates() класу Client

При оновленні клієнтської частини програми не працювали додані функції.

Проблема виникала тому, що система тестувалася на сервері GlassFish. У налаштуваннях проекту не було ввімкнено автоматичне розгортання (deployment) на сервері, тому там були відсутні оновлені файли скриптів з доданими функціями.

Для усунення проблеми було увімкнено автоматичне розгортання на сервері при збереженні проекту.

4.4.4 Налагодження методу handleMessage() класу Server

Програма не знаходила квитків з указаними параметрами.

Проблема виникала тому, що неправильно проходило розпакування даних із отриманого JSON-пакету (використовувався об’єкт JSONArray замість потрібного JSONObject). Як наслідок, параметри, що зберігали назви пунктів відправлення і прибуття, не набували необхідних значень.

Для усунення проблеми розпакування JSON-пакету реалізовано із використанням належних засобів.

5 ОХОРОНА ПРАЦІ

5.1 Аналіз шкідливих та небезпечних виробничих факторів

Небезпечні виробничі чинники – це виробничі чинники, вплив яких на працівника в певних умовах призводить до травм, гострого отруєння або іншого раптового різкого погіршення здоров'я або до смерті [21].

До різкого погіршення здоров'я можна віднести отруєння, опромінення, удар електрострумом, тепловий удар та ін.

До небезпечних факторів відносяться електронебезпека та пожежонебезпека.

Шкідливі виробничі фактори – фактори, тривалий вплив яких на працюючого у визначених умовах приведе до захворювання, зниження працездатності i (або) негативного впливу на здоров'я нащадків [21]. У залежності від рівня і тривалості впливу шкідливі фактори можуть класифікуватися і як небезпечні.

Згідно санітарно-гігієнічним вимогам [22] умови праці робітника, який працює з системою розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації на ЕОМ, повинні відповідати I або II класу.

Робота на ЕОМ пов'язана з наступними шкідливими факторами:

* недолік освітлення природним світлом;
* відблиски на екрані монітора;
* іонізуюче випромінювання електронно-променевого монітору;
* електромагнітне поле.

Недостатнє освітлення приводить до швидкої стомленості очей, що у свою чергу, призводить до зниження продуктивності праці і росту кількості прийнятих помилкових рішень.

Відблиски на екрані електронно-променевого монітора, що виникають при неправильному освітленні, приводить до погіршення зору, а у випадку тривалого впливу даного небезпечного фактора, може призвести до повної втрати зору. З метою зниження рівня впливу на робітника даного шкідливого фактора, варто дотримуватись вимог [23] або застосовувати рідкокристалічні монітори, які в силу своєї конструкції і використовуваних матеріалів мають менший коефіцієнт відбиття світла, ніж електронно-променевий монітор, тому відблисків на них практично не буває.

Рівні електромагнітного випромінювання і магнітних полів на робочому місці повинні відповідати вимогам [24].

5.1.1 Небезпечні чинники, пов’язані з пожежами

Пожежі становлять особливу небезпеку для життя людини, і можуть призвести до великих матеріальних втрат. Джерелами загоряння можуть виявитися електронні схеми ЕОМ, елементи, що перегрілися, які здатні викликати загоряння пальних матеріалів.

Для всіх споруджень та приміщень, у яких експлуатуються система розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації на ЕОМ, повинна бути визначена категорія по вибухонебезпечній і пожежній безпеці відповідно до [25], і клас зони згідно ППЕ [26]. Відповідні позначення повинні бути нанесені на вхідні двері приміщення.

Будинки та ті їх частини, у яких розміщуються ЕОМ, повинні мати ступінь вогнестійкості не нижче II. Приміщення для обслуговування, ремонту і налагодження ЕОМ повинні відноситися:

* за пожежонебезпекою до категорії В – пожежонебезпечні приміщення, де розташовуються тверді пальні речовини (ТПР);
* за класом приміщення до категорії П ІІ [26].

Неприпустимим є розташування приміщень категорії А и Б [5], а також виробництв із мокрими технологічними процесами поруч із приміщеннями, де розташовуються ЕОМ, виконується їхнє обслуговування, налагодження і ремонт, а також над такими приміщеннями під ними.

Приміщення з ЕОМ повинні бути оснащені:

* системою автоматичної пожежної сигналізації відповідно до вимог переліку однотипних за значенням об'єктів, що підлягають устаткуванню автоматичними установками пожежогасіння і пожежної сигналізації;
* димовими пожежними оповіщувачами;
* переносними вуглекислотними вогнегасниками з розрахунку 2 шт. на кожні 20 м2 площі приміщення з обліком гранично припустимих концентрацій вогнегасної рідини відповідно до вимог Правил пожежної безпеки в Україні.

Стіни приміщень з ЕОМ виготовляються з не пальних матеріалів. Підходи до засобів пожежогасіння повинні бути вільними.

5.1.2 Електронебезпечні чинники

При роботі на ЕОМ можуть трапитися нещасні випадки, зв'язані з ураженням електричним струмом, викликаним дотиком до оголених місць струмоведучих частин устаткування, або частин, що знаходяться під напругою.

Вплив струму на людину проявляється по-різному, в залежності від його величини:

* струм до 0,6 мА не відчувається людиною;
* струм завбільшки 6 мА призводить до скорочення м'язів тієї частини, тіла, що піддалася його впливу;
* струм 50 мА вражає органи дихання та серцево-судинну систему;
* при досягненні струму порога 100мА настає фібриляція серця і, потім, його зупинка;
* при впливі на тіло людини струму в 3-4 А виникає обвуглювання ділянок тіла.

За способом захисту людини від ураження електричним струмом відеотермінали, ЕОМ, периферійні пристосування ЕОМ і оснащення для обслуговування, ремонту і налагодження ЕОМ повинні відповідати I класу захисту. Вимоги електричної і механічної безпеки і методи іспитів чи повинні бути заземлені.

Лінія електромережі для живлення ЕОМ, периферійних устроїв ЕОМ і устаткування для обслуговування, ремонту і налагодження ЕОМ виконується як окрема групова, трьох провідна мережа, шляхом прокладки фазового, нульового робочого і нульового захисного провідників.

Підключення на розподільному щиті до одного контактного затиску нульового робочого і нульового захисного провідників заборонено.

Площа перетину нульового робочого і нульового захисного провідника в груповій трьох провідній мережі повинна бути не менш площі перетину фазового провідника.

У приміщенні, де одночасно експлуатується чи обслуговується більш п'яти персональних ЕОМ, на видному доступному місці встановлюється аварійний вимикач, за допомогою якого можливо зробити знеструмлення приміщення (за винятком освітлення).

5.1.3 Фізично шкідливі і небезпечні чинники

До фізичних шкідливих і небезпечних чинників відносяться:

* підвищені рівні електромагнітного, рентгенівського, ультрафіолетового і інфрачервоного випромінювання;
* підвищений рівень статичної електрики і запиленої повітря робочої зони;
* підвищений вміст позитивних аеронів і знижений вміст негативних аеронів в повітрі робочої зони.

5.1.4 Хімічно шкідливі і небезпечні чинники

Хімічні шкідливі і небезпечні чинники наступні:

* підвищений вміст в повітрі робочої зони двоокису вуглецю, озону;
* бідність повітря робочої зони на кисень.

5.2 Заходи з усунення шкідливих та небезпечних чинників

Для якісної і зручної роботи з системою розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації на ЕОМ, необхідне проведення проектних заходів: правильна облаштованість, належне дотримання ергономічних характеристик основних елементів робочого місця [27], санітарно-гігієнічних вимог [28, 29] і т.п.

Облаштованість робочого місця користувача при роботі з підсистемою повинне забезпечувати відповідність всіх елементів робочого місця і їхнього розташування.

Площа, яка виділена для одного робочого місця з ВДТ чи ЕОМ, повинна складати не менш 6 м2 , а обсяг – не менш 20 м3.

Робочі місця з ВДТ щодо світлових прорізів повинні розташовуватися так, щоб природне світло падало з боку, переважно ліворуч.

При розташуванні робочих місць із ВДТ необхідно дотримувати наступних вимог:

* робочі місця з ВДТ розташовуються на відстані не менш 1 м від стін зі світловими прорізами;
* відстань між бічними поверхнями ВДТ повинне бути не менш 1,2 м;
* відстань між тильною поверхнею ВДТ і екраном іншого ВДТ повинне бути не менш 2,5 м;
* прохід між рядами робочих місць повинний бути не менш 1 м.

Висота робочої поверхні столу для ВДТ повинна бути в межах 680 - 800 мм, а ширина - забезпечувати виконання належних операцій.

Розміри столу, що рекомендуються: висота - 725 мм, ширина ~ 600 - 1400 мм, глибина - 800-1000 мм.

Робочий стіл для ВДТ повинний мати простір для ніг висотою не менш 600 мм, шириною не менш 500 мм, глибиною на рівні колін не менш 450 мм, на рівні витягнутої ноги ~ не менш 650 мм,

Можливість обертання екрана ВДТ навколо горизонтальної і вертикальної осі.

5.2.1 Вимоги до організації робочого місця

При роботі з системою управління розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації на ЕОМ необхідно виконувати вимоги до організації робочого місця, що наведені нижче.

До самостійної роботи допускаються особи, які пройшли медичний огляд, навчання за професією, вступний інструктаж з охорони праці на робочому місці.

Конструкція робочого місця робітника при роботі з ЕОМ [30] (при роботі сидячи) повинна забезпечувати підтримку оптимальної робочої пози з наступними ергономічними характеристиками:

* стопи ніг – на підлозі чи на підставці для ніг;
* стегно – у горизонтальній площині;
* передпліччя – вертикально;
* лікті під кутом 70°-90° до вертикальної площини;
* зап'ястя – зігнуті під кутом не більш 20° щодо горизонтальної площини;
* нахил голови 15°-20° відносно вертикальної площини.

Дисплей та клавіатура мають розташовуватися на оптимальній відстані від очей користувача, але не ближче 600 мм, з урахуванням розміру алфавітно-цифрових знаків та символів.

Розташування екрану відео терміналу має забезпечувати зручність зорового спостереження у вертикальній під кутом ±30° о від лінії зору працівника.

Клавіатуру слід розміщувати на поверхні столу або на спеціальній, робочій і такій, яка регулюється за висотою поверхні окремо від столу на відстані 100-300 мм від краю, ближчого до працівника. Кут нахилу клавіатури має бути в межах 5-15°.

5.3 Правила безпечного виконання робіт на ЕОМ

5.3.1 Вимоги безпеки праці перед початком роботи на ЕОМ

Перед початком роботи на ЕОМ робітник інструктується первинним інструктажем. Первинний інструктаж завжди проводиться на робочому місці з безпосереднім показом робіт (стажування 1 місяць). Потім, через кожні 6 місяців проводиться повторний інструктаж. Результати інструктажу заносяться в "Журнал реєстрації інструктажів з питань охорони праці". У журналі після проходження інструктажу повинен бути підпис людини, яка інструктувала та робітника. Далі підготовка робочого місця повинна виконується відповідно до нижче приведених пунктів:

* увімкнути систему кондиціонування повітря в приміщенні;
* оглянути робоче місце і привести його в порядок: переконатися, що на ньому немає сторонніх предметів; все устаткування і блоки ЕОМ з'єднані із системним блоком за допомогою сполучних шнурів;
* перевірити надійність установки апаратури на робочому столі. Монітор повинен стояти не на краю столу. Повернути монітор так, щоб було зручно дивитися на екран – під прямим кутом (а не збоку) – та ледь зверху вниз, при цьому екран повинен бути ледь нахилений – нижній його край ближче до робітника;
* перевірити загальний стан апаратури, або перевірити чи справні електропроводка, сполучні шнури, штепсельні вилки, розетки, перевірити заземлення захисного екрана;
* відрегулювати освітлення робочого місця;
* відрегулювати і зафіксувати висоту крісла, зручний для робітника нахил його спинки;
* включити апаратуру комп'ютера перемикачами на корпусі в послідовності: стабілізатор напруги, монітор, системний блок;
* відрегулювати яскравість світіння екрана до яскравості навколишніх його поверхонь у робочій зоні – і не більше 3:1;
* при виявленні будь-яких несправностей роботу не починати, повідомити про це керівника робіт.

5.3.2 Вимоги безпеки праці під час роботи на ЕОМ

Під час роботи на ЕОМ необхідно дотримуватися наступних правил:

* при виконанні роботи на комп’ютері слід сидіти прямо, не напружуватися;
* не дозволяються сторонні розмови, подразнюючі шуми;
* постійно слідкувати за функціонуванням систем опалення і кондиціювання;
* для зниження напруженості праці на комп’ютері необхідно рівномірно розподілити і чергувати характер робіт відповідно до їх складності.

Для зменшення негативного впливу на здоров'я працюючих виробничих факторів необхідно застосовувати регламентні перерви. У табл. 5.1 наведений час регламентованих перерв робітника залежно від категорії і групи робіт.

Таблиця 5.1 – Час регламентованих перерв операторів (користувачів) залежно від категорії і групи робіт

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категорія  робіт | Група роботи | | | Час перерви при 8-годинній зміні, хв. |
| А, кількість  знаків | Б, кількість  знаків | В, година |
| Оптимальна-І | до 20000 | до 15000 | до 2 | 20 |
| Припустима-ІІ | 20000 – 40000 | 15000 – 30000 | 2 – 4 | 40 |
| Важка-ІІІ | 40000 – 60000 | 30000 – 45000 | 4 – 6 | 60 |

Група А – читання інформації з попередні запитом (діалоговий режим роботи).

Група Б – введення інформації.

Група В – творча робота в режимі діалогу ЕОМ (налагодження програм, переклад і редагування текстів і т, д.).

При восьмигодинній робочій зміні регламентована перерва повинна бути такою:

* для I категорії робіт – через 2 години від початку зміни і через 2 години після обідньої перерви (кожен тривалістю 10 хв.);
* для II категорії робіт – через 2 години від початку зміни (тривалістю 15 хв.), через 1,5 і 2,5 годин після обідньої перерви (15 і 10 хв. відповідно чи тривалістю 5-10 хв. Через щогодини роботи, в залежності від характеру технологічного процесу);
* для Ш категорії робіт – через 2 години від початку зміни (тривалістю 15 хв.), через 1,5 і 2,5 годин після обідньої перерви (тривалістю 20 хв. Кожний тривалістю 5–15 хв. Щогодини роботи, у залежності від характеру технологічного процесу);
* під час нічної зміни, незалежно від групи і категорії робіт, тривалість регламентованих переривів збільшується на 60 хв.

Під час роботи забороняється:

* використовувати рідинні або аерозольні засоби для чищення поверхонь комп’ютера, крім передбачених для цього засобів ( для чищення пластмасових поверхонь використовують чисту бавовняну ганчірку, зволожену мильним розчином; екрани та захисні екрани протирають ватою, змоченою спиртом);
* самостійно ремонтувати апаратуру;
* класти будь-які речі на вузли комп’ютера;
* закривати вентиляційні отвори апаратури, що може привести до її перегрівання і виходу з ладу.

5.3.3 Вимоги безпеки праці після закінчення роботи на ЕОМ

Після закінчення роботи необхідно:

* вимкнути системний блок, принтер, інші периферійні пристрої, монітор;
* вимкнути стабілізатор живлення, якщо комп'ютер підключений до мережі через нього;
* штепсельні вилки витягнути з розетки;
* накрити клавіатуру кришкою для запобігання потрапляння на неї пилу;
* прибрати робоче місце.



5.4 Дії при аварійних ситуаціях

В аварійних ситуаціях працівник зобов’язаний:

* при раптовому припиненні подавання електроенергії вимкнути послідовно периферійні пристрої, ВТП, процесор і витягнути штепсель вилки з розетки;
* при виявленні ознак горіння (дим, запах горіння), вимкнути апаратуру, знайти джерело займання і вжити заходів для його ліквідації, повідомити керівництво чи системного адміністратора;
* у разі виникнення пожежі негайно повідомити пожежну частину, вжити необхідних заходів для евакуації людей згідно з планом евакуації по підприємству і приступити до гасіння первинними засобами пожежогасіння.

5.4.1 Допомога при ураженні електричним струмом

Перша медична допомога при ураженні людини електричним струмом повинна бути спрямована на звільнення її від дії струму та підтримку або відновлення серцевої діяльності і дихання.

Якщо потерпілий при свідомості, але не спроможний розтиснути руки, йому слід запропонувати підскочити або впасти, щоб розірвати електрострумовий ланцюг.

Коли уражений без свідомості, то потрібно негайно знеструмити його шляхом вимикання напруги в мережі рубильником, вимикачем або запобіжником. При неможливості достатньо швидко вимкнути електромережу слід перерізати чи перерубати електропроводи.

При ситуації, коли потерпілого можна відтягти від ушкоджуючи струмоведучих частин, до яких він торкається, рятівник повинен подбати про власну безпеку: по можливості діяти однією рукою, тягнути тільки за суху одежу, уникати доторкання до оголених частин тіла потерпілого та оточуючих металевих деталей. Електроізолюватися від потерпілого можна за допомогою діелектричних рукавиць або обмотаними тканиною руками, ставши на гумовий килимок, суху дошку або інший підручний предмет, що має діелектричні властивості. Відокремивши ураженого від струмопроводу, потрібно ретельно оглянути його.

Якщо в потерпілого наявні ознаки клінічної смерті, то слід невідкладно розпочати серцево-легеневу реанімацію. При "несправжній смерті" єдиним ефективним заходом першої допомоги є негайне проведення штучного дихання (з рота в рот); інколи його доводиться робити протягом кількох годин поспіль. Штучне дихання при працюючому серці, як правило, швидко покращує стан ураженого: шкіра стає нормального кольору, визначаються пульс і артеріальний тиск. Серцево-легеневу реанімацію або тільки штучне дихання по можливості потрібно поєднувати з підшкірним введенням 2 мл кордіаміну або 1 мл 10% розчину кофеїну.

Непритомну людину в подальшому слід утеплити, напоїти великою кількістю води, чаю чи компоту (давати алкоголь і каву шкідливо) та відправити на транспорті до лікарні. При необхідності реанімаційні заходи чи штучне дихання слід безперервно продовжувати і в транспорті.

Під час транспортування до лікарні потрібно особливо уважно стежити за станом потерпілого, тому що в будь-який час у нього може виникнути зупинка дихання і серцевої діяльності. Слід бути напоготові до реанімаційних заходів у дорозі.

ВИСНОВКИ

Наразі дуже швидким є розвиток інформаційних технологій. Однією з його галузей є створення і впровадження інтелектуальних інтерфейсів, що дозволяють користувачам взаємодіяти із комп’ютером за допомогою природного мовлення. Інтерес до технологій розпізнання голосу підвищується завдяки створенню та популяризації великими корпораціями (Google, Apple, Microsoft) голосових помічників у вигляді додатків для мобільних пристроїв. Програми, що використовують голосовий інтерфейс для керування комп’ютером, на даний момент малопоширені через те, що їх розробкою та розповсюдженням займаються не великі компанії, а окремі люди. Однак у майбутньому природно-мовні інтерфейси стануть основним способом взаємодії людини з комп’ютером (що є логічним, оскільки такий спосіб є простим і звичним для людини; крім того, це підтверджується зацікавленістю великих компаній у розвитку цієї сфери).

Тому і виникла ідея створення системи розпізнання голосових запитів для пошуку залізничних квитків та супутньої інформації, яка має на меті полегшити процес пошуку і замовлення квитків, зробити його більш простим та природним.

Розроблений програмний продукт надає змогу обрати пункт відправлення, пункт прибуття та дату відправлення, а потім обрати рейс із знайдених за обраними параметрами та перейти до придбання квитка на нього. Усі дані та команди керування задаються голосом, вихідні повідомлення надаються також голосом, а крім того дублюються у вигляді тексту на екрані. Програму виконано у формі веб-додатку.

СПИСОК ЛІТЕРАТУРИ

1. Радзишевский, А. Ю. Основы аналогового и цифрового звука. – М.: Вильямс, 2006. – 288 с.: ил.

2. Уоллс К. Spring в действии. – М.: ДМК Пресс, 2013. – 752 с.: ил.

3. Флэнаган Д. JavaScript. Подробное руководство. – Пер. с англ. – СПб: СимволПлюс, 2008. – 992 с., ил.

4. Бен Фрейн. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств. – СПб.: Питер, 2014. – 304 с.: ил.

5. Амит Бакор. Apache Tomcat для профессионалов. – СПб.: Кудиц-Образ, 2005. – 544 с.: ил.

6. Грейди Буч. Язык UML. Руководство. – 2-е изд. – М., СПб.: ДМК Пресс, Питер, 2004. – 432 с.

7. Бабич А. В. Введение в UML. Лекция: Диаграммы прецедентов. – Електронні дані. – Режим доступу: http://www.intuit.ru/studies/courses/1007/229/ lecture/5962

8. K. Beck, W. Cunningham. A laboratory for teaching object oriented thinking. – Електронні дані. – Режим доступу: http://dl.acm.org/citation.cfm? doid=74878.74879

9. Грейди Буч. Объектно-ориентированный анализ и проектирование с примерами приложений. – 3-е изд. – М.: ООО «И. Д. Вильямс», 2008. – 720 с.: ил.

10. UML basics: The sequence diagram. – Електронні дані. – Режим доступу: http://www.ibm.com/developerworks/rational/library/3101.html

11. Фисун А. П., Гращенко Л. А. и др. Теоретические и практические основы человеко-компьютерного взаимодействия: базовые понятия человеко-компьютерных систем в информатике и информационной безопасности. — Деп. в ВИНИТИ 15.10. 2004 г. № 1624 – В 2004. — Орел: Орловский государственный университет, 2004. — 169 с. — (Рукопись).

12. Электронное учебное пособие. Интелектуальные интерфейсы. – Електронні дані. – [Бийский технологический институт]. – Режим доступу: http://do.bti.secna.ru/lib/book\_it/intelect\_interfase.html

13. Анатольев А.Г. Этапы разработки пользовательского интерфейса. – Електронні дані. – Режим доступу: http://www.4stud.info/user-interfaces/stages-of-development-user-interface.html

14. Когаловский М. Р. Энциклопедия технологий баз данных. – М.: Финансы и статистика, 2002. – 800 с.

15. Дейт К. Дж. Введение в системы баз – 8-е изд. – М.: Вильямс, 2005. – 1328 с.

16. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика – 3-е изд. – М.: «Вильямс», 2003. – 1436 с.

17. Окулов С. Программирование в алгоритмах. – Бином, 2007. – 384 с.

18. С. Полаженко. Экспертиза исходных текстов как метод тестирования безопасности и защищённости программных продуктов. – Електронні дані. – Режим доступу: http://www.realcoding.net/articles/ekspertiza-iskhodnykh-tekstov-kak - metod - testirovaniya - bezopasnosti - i - zashchishchennosti - programmnykh -produktov.html

19. Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. – Питер, 2004. – 320 с.

20. Стив Магьюир, «Создание надёжного кода» (Steve Maguire. Writing Solid Code. Microsoft Press, 1993).

21. ДСТУ 2293-99 «Охорона праці. Терміни та визначення основних понять»

22. ДЕСТ 12.1.00-88 «ССБТ. Загальні санітарно-гігієнічні вимоги до повітря робочої зони»

23. ДБН В.2.5-28-2006 «Природне і штучне освітлення»

24. НПАОП 0.03-3.30-96 «Державні санітарні норми і правила захисту населення від впливу електромагнітних випромінювань»

25. НАПБ Б.07.005-86 «Визначення категорій приміщень і будинків по вибухонебезпечній і пожежній безпеці»

26. НАПБ В.01.056-2005/11 «Правила побудови електроустановок. Протипожежний захист електроустановок»

27. НПАОП 0.00-1.31-99 «Правила охорони праці під час експлуатації електронно-обчислювальних машин

28. ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку»

29. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень»

30. ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин»

31. ДСТУ ГОСТ 7.1:2006 «Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання (ГОСТ 7.1-2003, IDT)». – К.: Держстандарт України, 2007.