Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

# АТЕСТАЦІЙНА РОБОТА
# Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження методів NLP для розробки сервісу голосового управління
(тема)

Виконав: студент 2 курсу, групи ІПЗм-17-2
спеціальності 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Освітньо-наукової програми Інженерія програмного забезпечення
(повна назва спеціалізації)

Іванов М.Є.
(прізвище, ініціали)
Керівник к.т.н. доц. Голян В.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Дудар З.В.
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)

освітньо-наукова програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

# ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Іванову Миколі Євгеновичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження методів NLP для розробки сервісу голосового управління*
затверджена наказом по університету від ___ _____2019 р. № _____
2. Термін подання студентом роботи до екзаменаційної комісії ___ ____ 2019 р.
3. Вихідні дані до роботи *провести аналіз методів обробки природної мови для розробки сервісу голосового управління. Провести дослідження використання рекурентних мереж для вирішення задачі розпізнавання намірів користувача, використовуючи технологію python як мову програмування, keras як технологію будування нейронних мереж. Проаналізувати сучасні технології розробки сервісів та веб-застосувань Розробити сервіс голосового управління, використовуючи технологію Spring Boot у якості акселератору, MySql у якості бази даних, JQuery у якості UI бібліотеки, Apache Tomcat у якості веб серверу*
4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз преметної галузі, аналіз методів обробки природної мови, дослідження використання RNN для класифікації намірів, огляд технологій розпізнавання сутностей, огляд технологій розпізнавання голосу, аналіз засобів розробки, проектування сервісу голосового управління, архітектура програмної системи, опис функцій кінцевого користувача, тестування системи*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *зображення структури LSTM нейрону, зображення функції помилки на тренувальному та тестовому наборі даних, діаграма архітектури системи, діаграма прецедентів, діаграма класів, діаграма послідовностей, діаграма станів, діаграма розгортки, слайди презентації*

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1 )

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| Спецчастина | доц. Афанасьева І.В. | | |

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Аналіз проблемної галузі | 11.02.2019 – 25.02.2019 | |
| 2 | Дослідження методів предметної галузі | 26.02.2019 – 25.03.2019 | |
| 3 | Проектування сервісної частини даних | 26.03.2019 – 01.04.2019 | |
| 4 | Проектування клієнтської частини | 02.04.2019 – 04.04.2019 | |
| 5 | Створення коду програми | 05.04.2019 – 16.04.2019 | |
| 6 | Тестування і налагодження програми | 17.04.2019 – 19.04.2019 | |
| 7 | Підготовка пояснювальної записки | 20.04.2019 – 31.05.2019 | |
| 8 | Підготовка презентації та доповіді | 03.06.2019 | |
| 9 | Попередній захист | 05.06.2019 | |
| 10 | Нормоконтроль, рецензування | 06.06.2019 | |
| 11 | Занесення диплома в електронний архів | 10.06.2019 | |
| 12 | Допуск до захисту у зав. кафедри | 10.06.2019 | |

Дата видачі завдання ___ _____2019 р.

Студент _____
(підпис)

Керівник роботи _____ _____
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 71 с., 13 рис., 10 джер.

ГОЛОСОВЕ УПРАВЛІННЯ, ОБРОБКА ПРИРОДНОЇ МОВИ, ПРОГРАМНА СИСТЕМА, JAVA, JQUERY, MAVEN, MYSQL, PYTHON, SPRINGMVC, TOMCAT.

Об’єктом дослідження роботи є ефективність застосувань рекурентних нейронних мереж у задачі класифікації намірів користувача.

Метою роботи є дослідження методів NLP для розробки сервісу голосового управління.

Основними технологіями та засобами розробки є програмні мови Java та Python, фреймворки Spring, Hibernate.

Результатом роботи є використання методів NLP для розробки сервісу голосового управління, що складається з серверу, клієнтської бібліотеки, що дозволяє інтегруватися з сервером, та адміністративної сторінки, що відображає кількість запитів, дозволену для користувача.

JAVA, JQUERY, MAVEN, MYSQL, NATURAL LANGUAGE PROCESSING, PYTHON, SOFTWARE SYSTEM, SPRINGMVC, TOMCAT, VOICE CONTROL.

The object of research is the effectiveness of recurrent neural network usage in user intent classification task.

The aim of work is the research of NLP methods for voice control service development.

The core development technologies and tools are Java and Python programming languages, Spring framework, Hibernate.

The result of work is the usage of NLP methods for voice control service development that consists of server, client library that can integrate with server part, and administration page which shows amount of remaining client requests.

**ЗМІСТ**

# ВСТУП

Комунікація – невід’ємна частина нашого життя. Люди використовують її кожен день у найрізноманітніших ситуаціях з метою обміну інформацією, досягнення своїх цілей та вираження своїх почуттів. З давніх часів люди звикли використовувати голосову комунікацію у якості основної, тому мова здається найбільш простим та звичним підходом для досягнення інформаційного обміну.

На початку ери інформаційних технологій, що зумовила появлення різноманітних комп’ютерних систем, голосова комунікація не могла бути застосована у роботі з програмами. Системи були орієнтовані на роботу зі спеціальними пристроями вводу, що використовували кнопки управління, вводу інформації та маніпулювання курсором у разі необхідності конкретизувати свої наміри. Але сьогоднішній розвиток комп’ютерної сфери дозволяє створювати сервіси, здатні оброблювати та аналізувати великі об’єми даних, що відкриває нові можливості для комунікації між людиною та програмою.

Основою проекту є дослідження методів обробки природної мови та проектування прототипу сервісу, що дозволить розуміти голосові команди через природну мову та впроваджувати команди відповіді на такі запити. Він буде складатися з двох частин. Клієнтська буде відповідати за перетворення голосу у текстові дані та відправку їх на серверну частину. Серверна частина буде використовувати засоби обробки природної мови для формування належної команди відповіді на запит користувача.

Обробка природної мови – загальний напрямок інформатики, штучного інтелекту та математичної лінгвістики. Він вивчає проблеми комп'ютерного аналізу та синтезу природної мови. Стосовно штучного інтелекту аналіз означає розуміння мови, а синтез – генерацію розумного тексту. Розв'язок цих проблем буде означати створення зручної форми взаємодії комп'ютера та людини.

Наразі нові можливості зручної взаємодії користувача з системою є одним із основних напрямків розвитку програмних додатків. Багатокористувацькі сервіси

намагаються створити чудовий досвід користування, застосовуючи персоналізацію та приділяючи увагу до кожного клієнта. Надання кінцевому користувачу нових здібностей управління може зіграти вирішальну роль у тенденції розвитку платформи та підвищити її конкурентоздатність у порівнянні з альтернативними продуктами.

Важливим моментом використання даних систем є взаємодія з пристроєм користувача. Сучасні пристрої здатні виконувати багато корисних функцій, таких як запис з мікрофону або виконання дрібної частини функціоналу системи на стороні клієнта. Використання даних можливостей є одним з ключових показників ефективності системи.

Для забезпечення зручної роботи з такими сервісами також важливу роль грають адміністративні інструменти. Можливість дізнатися про стан користувача у системі, доступний функціонал, впровадити контактні дані та отримати підтримку забезпечують якісне користування системою та надають можливості створення персональних налаштувань для індивідуальних потреб.

При створенні інструменту, що інтегрується з різноманітними середовищами, зручність користування та поєднання має велике значення. Продукти, що є незалежними від мови програмування, пристрою користувача, географічних умов стають набагато привабливіше, адже вони надають менше обмежень, що позитивно впливає на різноманіття сегментів потенційних користувачів.

Отже предметна галузь та засоби для виконання роботи є актуальними. Метою атестаційної роботи є дослідження методів обробки природної мови та проектування прототипу сервісу, який дозволить здійснювати голосове управління веб-системами, виконувати швидку інтеграцію сервісу з різноманітними веб-системами.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1Аналіз предметної галузі

Предметна галузь роботи наразі набуває актуальності у сучасній IT сфері, що зумовлено стрімким поширенням систем, орієнтованих на роботу з природною мовою. Далі буде виділено основні сутності та розділи предметної галузі.

Обробка природної мови набуває велике значення через недавній бум так званих чатботів або розмовних агентів у різноманітних галузях. Але, незважаючи на те, що популярність даний напрям здобув лише кілька років тому, вивчення природної мови має давнє минуле, і сучасні чатботи та електронні консультанти є не першим практичним застосуванням даної галузі. Дуже важливо мати глибоке розуміння мови, оскільки воно використовується щодня в різних сценаріях та контекстах. На сьогоднішній день існує багато програм, що дозволяють користувачу спілкуватися з спеціальними ботами, які відповідають на його питання та дають рекомендації. Але важливим обмеженням таких систем є рекомендаційний напрямок. Вони виступають у якості консультантів, що можуть направити користувача до потрібної сторінки, але не надають можливості функціонального управління.

Важливою складовою голосового управління системою також є здатність працювати в умовах контексту певного середовища. Сервіси електронної комерції будуть очікувати голосові команди пошуку продуктів та привабливих акцій, у той час як портали, направлені на реєстрування запитів, квитків та довідок повинні фокусуватись на вводі певних даних та підтвердження оформлення заяви. Тому гнучкий сервіс управління, здатний адаптуватися до умов зовнішнього середовища та надавати інструменти налаштування, зміни та розширення функціоналу буде мати великий попит.

Обробка природної мови визначається як застосування обчислювальної техніки до аналізу та синтезу конструкцій, основаних на людській мові

спілкування. Використання різних методик з інформатики (алгоритмів) та штучного інтелекту забезпечує розуміння і маніпулювання людською мовою.

Іноді обробка природної мови (NLP) плутається з машинним навчанням (ML), але це відбувається лише тому, що деякі інструменти від ML застосовуються до NLP і поліпшують якість рішень. Необхідно розглядати машинне навчання як набір інструментів, деякі з яких будуть дуже корисними для вирішення різноманітних задач обробки природної мови.

Важливим фактором є те, що глибинне навчання є гілкою машинного навчання, яка використовує певний тип архітектур і моделей, названих нейронними мережами, для вирішення навчальних завдань. Моделі глибинного навчання є підмножиною інструментів у наборі моделей ML. Це не єдина гілка, що існує(є й інші, такі як статичні алгоритми), але саме глибинне навчання набуває великого значення в спільноті ML з двох важливих причин.

По перше, продуктивність моделей на великій кількості даних помітно краще за класичні підходи машинного навчання, що є актуальним для сучасних умов, де дослідження та компанії оперують великими обсягами даних.

Також, останні досягнення в галузі обчислювальних можливостей з випуском кращого обладнання (графічні процесори), а іноді й спеціалізовані процесори роблять ці моделі достатньо продуктивними, дозволяючи отримувати результати обробки великої кількості даних у доволі невеликі проміжки часу.

Розвиток NLP систем проходить через багато складностей через певні специфічні проблеми і явища, які надходять при вивченні природної мови. У більшості випадків ці проблеми є унікальними у порівнянні з проблемами, що виникають в інших галузях інформатики або інженерії, і це частково те, що робить NLP такою цікавою та інноваційною галуззю.

Головним завданням NLP є розуміння та моделювання елементів у змінному контексті. У мові слова є унікальними, але можуть мати різні значення в залежності від контексту, в якому вони оцінюються. Це призводить до дуже поширеного мовного явища – неоднозначності. Ми можемо мати слова (або навіть речення) з різними значеннями в одному реченні залежно від того, як люди

інтерпретують ці слова. Це відбувається через різницю між означенням (спосіб, яким люди представляють інформацію, слово) і контекстом (відповідність цієї інформації до середовища).

Іншим ключовим явищем природної мови є те, що можна висловити одну і ту ж ідею з різними термінами. Це відбувається через синонімію, яка також залежить від конкретного контексту [1]. Таким чином, аналіз контексту інформації має одну з ключових цінностей обробки природної мови, адже допомагає ефективно застосовувати прикладні додатки, що будуть функціонувати у співпраці з користувачами.

При обробці природної мови необхідно її нормалізувати, щоб мати змогу керувати нею більш загальним шляхом. Це означає, що в залежності від завдання необхідно, щоб усі слова були без великих літер або множинні терміни перетворювались в форму однини, використовуючи спеціальні інструменти. У інших випадках виникає проблема з різними формами одного і того ж дієслова в документі, та є потреба розглянути саме це дієслово замість того, щоб робити відмінність між кожною формою. Всі ці процеси певним чином нормалізують природну мову і створюються методи, які використовуються для досягнення певної канонічної форми. Ключова ідея полягає в тому, що коли відбувається нормалізація, втрачається частина інформації в обмін на можливість кращого узагальнення. Нормалізація та обмін інформацією є поширеним у аналізі даних явищем але також є дуже важливим питанням для обробки природної мови.

Мова складається з символів, які можуть бути дискретними значеннями, тому що вони можуть приймати лише певні значення: 'a', 'b', 'c' та інші. Натомість поток даних, може приймати будь-яке значення в межах діапазону, наприклад, 0 або 0.43, оскільки оперує числовими представленнями даних. Необхідно обробляти дані, для того щоб отримувати числове представлення яке може бути проаналізовано алгоритмом для визначення залежностей між їх числовими представленнями. Один із способів подолання цієї проблеми і відношення різних термінів полягає у перетворенні слів у відповідні їм спеціальні вектори.

Машинне навчання для NLP і текстової аналітики включає в себе набір статистичних методів для виявлення частин мови, сутностей, емоцій та інших аспектів тексту. Методика може бути виражена як модель, яка потім застосовується до іншого тексту, використовуючи підхід, відомий як кероване машинне навчання. Це також може бути набір алгоритмів, який працює на великих наборах даних без позначень даних, та відомий як машинне навчання без вчителя. Важливо розуміти різницю між навчанням з розміченими даними і нерозміченими даними, і як можна отримати найкраще з обох в одній системі.

## 1.2 Постановка задачі

Сервіси, орієнтовані на обробку природної мови мають особливості, що дозволяє їм ефективно виконувати свою задачу. Важливим фактором розробки таких сервісів є оцінка впливу NLP на архітектурну частину системи, що повинно бути поєднано з сучасними практиками проектування та створення сервісів.

Необхідно розробити алгоритм розпізнавання намірів, який буде комбінувати класифікацію речення на основі нейронних мереж, розпізнавання сутностей та ідентифікацію команд на основі заданих шаблонів речень.

Даний алгоритм повинен мати змогу частково виконуватися у разі виникнення проблем комунікації між різноманітними системами, тобто частково бути реалізований на клієнтській частині.

По-перше, для взаємодії з сервісом користувач повинен використовувати веб браузер або інший клієнт, здатний відсилати різноманітні HTTP запити; також обов’язковою умовою є наявність виходу в Інтернет.

Система повинна обробляти дані на англійській мові. Отримані речення передавати на сервер, який розміщено у відповідному середовищі. На серверах середовища повинен бути встановлений та налаштований Tomcat. Потужності даного середовища повинні витримувати одночасну обробку кількох тисяч

запитів. На сервері баз даних зберігаються дані користувачів у базі даних MySQL [2], що також розміщується у відповідному кластері. Дані, оброблені на серверах, передаються власне клієнтам.

Серверна частина повинна мати наступний функціонал: будь-який користувач може здійснити запит при умові виконання контракту інтеграції. Для забезпечення комфорту кінцевого користувача розробникам систем, що інтегруються з даним сервісом слід виконати інтеграцію, описану у контракті. Користувач буде виконувати запити навігації, пошуку, управління функціоналом та виклику довідки. Користувач має використовувати інтерфейс браузеру для голосового управління. Система, яка інтегрується з сервісом голосового управління, повинна мати можливість перервати виконання команди.

Клієнтська частина повинна складатися з бібліотеки, що буде інтегруватися в веб-системи та повинна мати наступний функціонал: забезпечувати розпізнавання речення, що було введено на англійській мові через мікрофон за умови надання дозволу браузеру на зчитування голосових даних. Окрім того, мати можливість розширення та підтримки інтерфейсу для текстового вводу речення якщо клієнт не забажав використовувати мікрофон. Конфігурація мікрофону та можливість вибору різних пристроїв повинна підтримуватись браузером користувача, забезпечуючи можливість управління бажаним мікрофоном. Отримані дані повинні належним чином відправлятися на серверну частину. Після отримання відповіді від серверу, клієнт повинен передати системі, з якою він сполучений, отриманий результат обробки речення, що містить дані про голосову команду та її параметри.

Додаток має опрацьовувати лише речення, що відповідають вимогам. У іншому разі, користувач отримуватиме повідомлення про неправильно введену інформацію. Інструкція з вимогами буде доступна для кожного користувача системи.

# 2 АНАЛІЗ МЕТОДІВ ОБРОБКИ ПРИРОДНОЇ МОВИ

## 2.1 Існуючі методи використання RNN

Рекурентні нейронні мережі (RNN) є потужним і надійним типом нейронних мереж і належать до найбільш перспективних алгоритмів, які існують на даний момент, оскільки працюють з внутрішньою пам'яттю.

RNN відносно старі, як і багато інших алгоритмів глибокого навчання. Спочатку вони були створені в 1980-х роках, але можуть продемонструвати свій реальний потенціал лише через кілька років через збільшення доступної обчислювальної потужності, величезної кількості даних, які є сьогодні. Мережа короткочасної пам’яті є одним з прикладів таких мереж.

Завдяки їхній внутрішній пам'яті, RNN здатні запам'ятати важливі речі щодо вхідних даних, які вони отримали, що дозволяє їм бути дуже точними у роботі з послідовною інформацією.

Саме тому вони є кращим алгоритмом для послідовних даних, таких як часові ряди, мова, текст, фінансові дані, аудіо, відео, погода та багато іншого, оскільки вони можуть сформувати набагато глибше розуміння послідовності та контексту в порівнянні з іншими алгоритмами.

Навчальна мережа навчається на позначених даних, доки вона не зводить до мінімуму помилки, що виникають у процесі навчання. Завдяки навчанню набор параметрів (або ваги, загальновідомих як моделі), починає змінюватися відповідно до результату. Навчальна мережа, що просувається вперед, може піддаватися будь-якому випадковому збору даних, а перша ітерація, що була подана, не обов'язково змінить класифікацію на другій.

Отже, мережа прямого доступу не має поняття порядку в часі, і єдиним входом, який він вважає, є теперішній приклад. Мережі  прямого поширення є амнезіаками щодо недавнього минулого; вони пам'ятають ностальгічно лише базові моменти навчання.

У випадку прямих мереж, вхідні дані подаються в мережу і перетворюються на вихідні; з навчанням під контролем, вивід буде міткою, іменем, що застосовується до результату. Тобто, вони зіставляють необроблені дані з категоріями, розпізнаючи шаблони, які можуть сигналізувати, наприклад, що вхідне зображення має бути позначене як "кішка" або "слон".

У рекурентних мережах рішення, що зробила мережа, досягнуте на етапі часу t-1, впливає на рішення, яке буде зроблено пізніше в момент часу t. Таким чином, рекурентні мережі мають два джерела внеску, теперішнє і недавнє минуле, які об'єднуються, щоб визначити, як вони реагують на нові дані, так само як і люди в житті.

Рекурентні мережі відрізняються від мереж прямого поширення тим, що зворотний зв'язок пов'язаний з їхніми минулими рішеннями, вводить власні вихідні дані як вхідні дані наступних ітерацій. Часто говорять, що рекурентні мережі мають пам'ять. Додавання пам'яті до нейронних мереж має мету: у самій послідовності є інформація, а повторювані мережі використовують її для виконання завдань на основі досвіду.

Ця послідовна інформація зберігається в прихованому стані рекурентної мережі, якій вдається охопити багато кроків часу, оскільки вона каскадує вперед, впливаючи на обробку кожного нового прикладу. Вона знаходить кореляції між подіями, розділеними багатьма моментами, і ці співвідношення називаються «довгостроковими залежностями», оскільки подія в часі є функцією певної кількості подій, що відбулися раніше.

Подібно до того, як людська пам'ять неявно впливає на нашу поведінку та рішення, інформація циркулює в прихованих станах рекурентних мереж. Приховані стани змінюються на кожній ітерації, для того, щоб впливати на результат на етапі виконання наступного нейрону.

Прихований стан на етапі часу t є h_t. Це функція вхідного одночасно кроку x_t, модифікованого ваговою матрицею W (подібно тій, яку використовують для прямих мереж), доданої до прихованого стану попереднього кроку часу h_t-1, помноженого на його власний прихований стан. Матриця U-прихованого стану U,

інакше відома як матриця переходу  аналогічна ланцюгу Маркова. Матриці ваги – це фільтри, які визначають, наскільки важливо приєднатися як до поточного, так і до минулого прихованого стану. Помилка, яку вони генерують, повернеться через зворотне поширення і буде використана для коригування їх вагових коефіцієнтів, доки помилка не стане допустимою.

Сума введеної ваги і прихованого стану оброблюється функцією φ або логістичною сигмоподібною функцією, або tanh, яка є стандартним інструментом для конденсації дуже великих або дуже малих значень в логістичний простір, а також робить градієнти працездатними для зворотного поширення.

Оскільки цей контур зворотного зв'язку відбувається на кожному кроці в серії, кожен прихований стан містить сліди не тільки попереднього прихованого стану, але і всіх тих, які передували h_t-1 до тих пір, поки пам'ять може зберігатися.

Враховуючи ряд слів, повторювана мережа використовуватиме перше слово, щоб визначити її сприйняття другого слова, при цьому надаючи можливість знаходити зв’язки між словами у реченні, що належить до певного наміру.

Мережі короткочасної пам’яті допомагають зберегти помилку, яку можна розповсюджувати через час і прошарки. Підтримуючи більш стійку помилку, вони дозволяють повторним мережам продовжувати навчання протягом багатьох кроків, відкриваючи тим самим канал для підключення причин, наслідків та ефектів. Це одна з основних проблем машинного навчання та штучного інтелекту, оскільки алгоритми часто стикаються з середовищами, де сигнали про винагороду є розрідженими і затримуються.

Мережі короткочасної пам’яті містять інформацію за межами нормального потоку рекурентної мережі в закритій комірці. Інформацію можна зберігати, записувати або читати з нейрону, подібно до даних у пам'яті комп'ютера. Нейрон приймає рішення про те, що зберігати, і коли дозволити читання, запис і стирання через ворота, які відкриваються і закриваються. На відміну від цифрового зберігання на комп'ютерах, ці гейти є аналоговими, реалізованими з

елементарним множенням на сигмоїди, результати яких знаходяться в діапазоні 0–1. Аналог має перевагу перед цифровим, томущо він є диференційованим, і тому підходить для зворотного поширення. На рисунку 2.1 зображено структуру таких нейронів.

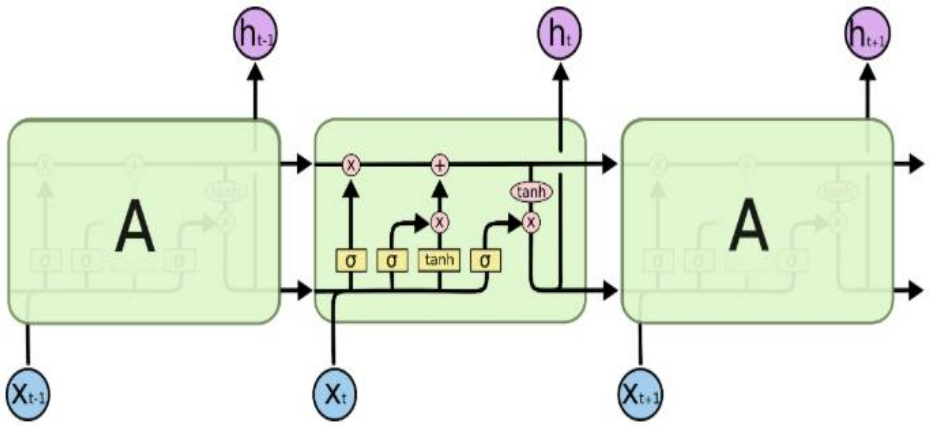


Рисунок 2.1 – Структура LSTM нейрону

Ці ворота діють на сигнали, які вони отримують, і подібно до вузлів нейронної мережі, вони блокують або передають інформацію на основі своєї сили та вводу, що фільтрується своїми власними наборами ваг. Ці ваги, такі як ваги, які модулюють вхідні та приховані стани, регулюються за допомогою процесу повторного навчання мереж. Тобто клітини вивчають коли дозволити вводити дані, залишати їх або видаляти через ітераційний процес внесення припущень, помилок, що поширюються назад, і коригування ваг за допомогою градієнтного спуска. Зворотне поширення в прямих мережах рухається назад від остаточної помилки через виходи, ваги і входи кожного прихованого прошарку, призначаючи на ці вагові коефіцієнти відповідальність за частину помилки, обчислюючи їх часткові похідні або співвідношення між їхніми показниками змін. Ці похідні потім використовуються правилом навчання, градієнтним спуском, для регулювання ваг вгору або вниз, в залежності від того, в якому напрямку зменшується похибка.

Періодичні мережі покладаються на розширення зворотного розповсюдження, яке називається зворотне поширення через час. Час, в даному

випадку, просто виражається чітко визначеною, впорядкованою послідовністю обчислень, що пов'язують один крок з кроком до наступного, що для всіх потребує зворотного поширення.

Нейронні мережі, чи є вони рекурентними чи ні, є просто вкладеними складовими функціями, такими як f (g (h (x))). Додавання елемента часу розширює лише ряд функцій, для яких слід обчислювати похідні з правилом ланцюга. Так само, як пряма лінія виражає зміну в x поряд із зміною y, градієнт виражає зміну всіх ваг щодо зміни помилки. Якщо не знаходиться градієнт, неможливо відрегулювати ваги в напрямку, який зменшить помилку, і мережа не буде навчатися.

Рекурентні мережі прагнуть встановити зв'язки між кінцевим виходом і подіями, але їм дуже важко зрозуміти, наскільки важливо приділяти увагу окремим інпутам. Важливим фактором є те, що інформація, що протікає через нейронні мережі, проходить через багато етапів множення.

Відомо, що будь-яке число, часто помножене на інше, дещо більше, ніж один, може стати незмірно великою (дійсно, ця проста математична істина підкріплює мережеві ефекти). Але його зворотне, множення на число менше одиниці, також вірно. Оскільки прошарки і кроки часу глибоких нейронних мереж пов'язані один з одним через множення, похідні сприйнятливі до зникнення або вибуху.

Вибухові градієнти відносяться до кожної ваги так, що маленька зміна може викликати великий зсув. Ці градієнти ваг стають насиченими на локальних максимумах; вони вважаються занадто потужними. Але вибухові градієнти можуть бути вирішені відносно легко, тому що вони можуть бути усічені або розчавлені. Зникаючі градієнти можуть стати занадто малими для роботи з нейронними мережами – важче вирішити проблему.

Рекурентні нейронні мережі дають прогностичні результати в послідовних даних, які інші алгоритми не можуть виконувати.

Сучасний розвиток процесорів дозволяє ефективно використовувати рекурентні нейронні мережі.

У нейронній мережі прямого поширення ця інформація рухається тільки в одному напрямку, від вхідного шару, через приховані шари, до вихідного шару. Інформація рухається прямо по мережі. Через це інформація ніколи не торкається вузла двічі.

Нейронні мережі, що передають інформацію про потік, не мають пам'яті про вхідні дані, які вони отримали раніше, і тому погано прогнозують, що буде далі. Оскільки мережа прямого доступу розглядає тільки поточний вхід, вона не має поняття порядку в часі. Вони просто не пам'ятають нічого про те, що сталося в минулому, крім їхньої підготовки.

У рекурентних мережах інформація проходить через цикл. Коли нейрон приймає рішення, він приймає до уваги поточний вхід, а також те, що він дізнався з вхідних даних, отриманих раніше.

Щоб вирішити проблему градієнта, що зникає, стандартний підхід рекурентних мереж використовує так звані ворота оновлення і скидання. В основному, це два вектори, які вирішують, яка інформація повинна бути передана на вихід. Особливість полягає в тому, що вони можуть бути навчені зберігати інформацію з давніх-давен, не видаляючи її з часом або видаляючи інформацію, яка не має відношення до прогнозу.

Подібно до звичайних рекурентних мереж, вхід множиться на матрицю ваги і додається до прихованого шару. Проте тут вхід додається до h˜. R – перемикач скидання, який відображає, яку частину раніше прихованого стану використовувати для поточного прогнозування. Існує також мережа для цього перемикача скидання, яка дізнається, що саме з попереднього стану дозволяє передбачити наступний стан. Вона представляє необхідність використання тільки існуючого прихованого стану h або використання його суми з h ~ (new hidden state) для прогнозування вихідного символу.

Мережі короткочасної пам’яті мають важливу мету – ефективне відстеження довгострокових залежностей при одночасному пом'якшенні проблем з зникненням/вибухом. Мережа короткочасної пам’яті робить це через вхідні, вихідні ворота та ворота забування; вхідні ворота регулюють, яка частина нового

стану нейрону буде збережено, ворота забуття регулюють, скільки залишків пам'яті потрібно забути, а вихідні ворота регулюють, що саме з стану нейрону повинно передаватися наступним шарам мережі. Мережа рекурентного блоку працює з використанням воріт скидання і воріт оновлення. Гейт скидання розташований між попередньою активацією і наступною активацією кандидата для того, щоб забути попередній стан, і ворота оновлення визначають, що саме використовувати для оновлення стану нейрона.

Обидві з вищеописаних мереж мають можливість зберігати пам'ять та стан від попередніх активацій, а не замінювати всю активацію, дозволяючи їм запам'ятовувати особливості протягом тривалого часу і дозволяючи зворотному розповсюдженню відбуватися через множинні обмежені нелінійності, що знижує ймовірність отримати зникаючий градієнт.

Проблема зникнення градієнта – це складність, виявлена в навчанні штучних нейронних мереж з методами градієнтного навчання і зворотного поширення. У таких методах кожна з ваг нейронної мережі отримує оновлення, пропорційне частковій похідній функції помилки відносно поточної ваги в кожній ітерації навчання. Проблема полягає в тому, що в деяких випадках градієнт буде малим, що ефективно запобігає зміні ваги. У найгіршому випадку це може повністю зупинити нейронну мережу від подальшого навчання. Як один із прикладів проблеми, традиційні функції активації, такі як гіперболічна дотична функція, мають градієнти в діапазоні 0–1, а зворотне поширення обчислює градієнти за правилом ланцюга. Це має наслідком множення n цих малих чисел на обчислення градієнтів "передніх" шарів в n-шарі мережі, що означає, що градієнт (сигнал помилки) падає експоненціально n, а передні шари дуже повільно.

Вхідний шар містить вхідні нейрони, які посилають інформацію до прихованого шару. Прихований шар посилає дані на вихідний шар. Кожен нейрон має зважені входи (синапси), функцію активації (визначає вихідний сигнал) і один вихід. Синапси є регульованими параметрами, які перетворюють нейронну мережу в параметризовану систему.

Навчання – це процес оптимізації ваг, в якому мінімізується похибка прогнозів і мережа досягає заданого рівня точності. Метод, який в основному використовується для визначення внеску помилок кожного нейрона, називається зворотним поширенням, який обчислює градієнт функції втрати.

Можна зробити систему більш гнучкою і потужною за допомогою додаткових прихованих шарів. Штучні нейронні мережі з декількома прихованими шарами між вхідним і вихідним шарами називаються глибокими нейронними мережами, і вони можуть моделювати різноманітні складні нелінійні відносини.

Існує дуже багато типів мереж, на вибір яких обираються нові методи, які щодня публікуються і обговорюються.

Важливим фактором є те, що більшість нейронних мереж достатньо гнучкі, щоб вони працювали (робили прогноз), навіть якщо вони використовуються з неправильним типом даних або проблемою прогнозування.

Глибинне навчання – це застосування штучних нейронних мереж з використанням сучасного обладнання.

Воно дозволяє розвивати, навчати і використовувати нейронні мережі, які набагато більші (більше прошарків), ніж вважалося раніше можливим.

## 2.2 Дослідження використання RNN для класифікації намірів

Було проаналізовано вимоги до алгоритму класифікації намірів та визначено, що основні характеристики, які демонструють якість роботи окремого методу – це метрика середньої квадратичної помилки та точності. Вони допомагають зрозуміти відношення успішно розпізнаних прикладів до помилок. В залежності від того, на якому наборі даних помилка є більшою, вона ідентифікує такі явища як недонавчання та перенавчання. Оптимальність системи визначається за результатами тренування та перевіряється відношенням помилки

навчання до помилки тестування. Як правило, дані функції спадають з часом, адже з ростом кількості епох покращується якість розпізнавання намірів. Отже, спостереження за цими метриками дозволить зробити висновки про якість роботи методу.

У якості вхідних даних використовувався файл з близько 1000 реченнями та відповідними намірами. 80% даних було виділено для тренування та 20% для валідації. Було визначено 100 епох виконання. На рисунку 2.2 можна побачити показники помилки на тренувальному та тестовому наборі даних з ростом кількості епох.

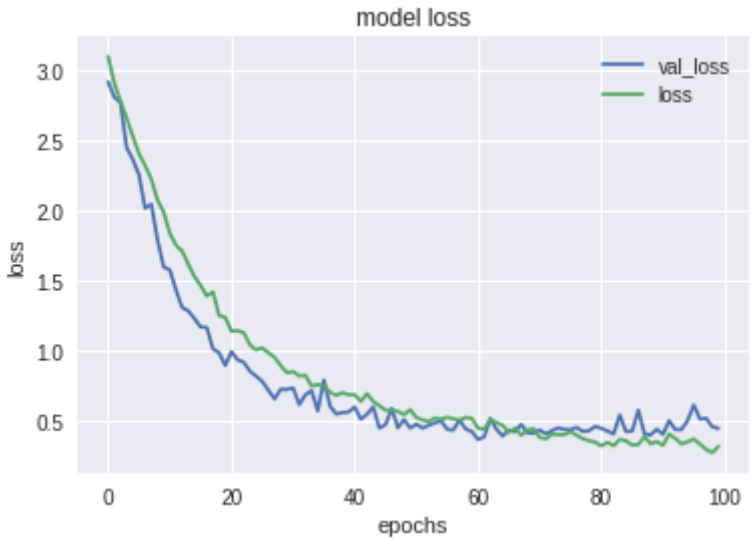


Рисунок 2.2 – Помилка на тренувальному та тестовому наборі даних

Даний підхід продемонстрував точність у 88% на тренувальному наборі даних, та 86% на тестовому. Таким чином, було визначено ефективність використання рекурентних нейронних мереж у задачі класифікації намірів.

Існують тисячі типів специфічних нейронних мереж, запропонованих дослідниками як модифікації або налаштування існуючих моделей. Іноді це цілком нові підходи.

У якості перевірки було використано рекурентні нейронні мережі, адже вони здатні ефективно передавати стани попередніх обчислень до наступних нейронів, а саме мережа короткочасної пам’яті.

Ці мережі забезпечують велику гнучкість і протягом десятиліть довели свою корисність і надійність у широкому спектрі проблем. Вони також мають безліч підтипів, які допомагають спеціалізувати їх у примхах різних фреймів задач прогнозування та різних наборів даних.

Моделі нейронних мереж рідко використовуються окремо. Зазвичай вони використовуються як прошарки в більш широкій моделі, яка також має один або більше прошарків. Технічно це гібридний тип архітектури нейронної мережі. Від організації прошарків залежить ефективність та точність комплексної системи.

Наприклад, розглянемо модель, яка використовує стек прошарків на вході, рекурентну мережу в середині, і багатошаровий перцептрон на виході. Така модель може читати послідовність вхідних зображень, наприклад відео, і генерувати прогноз. Це називається гібридною архітектурою.

Типи мереж можуть бути також інтегровані в певні архітектури, щоб розблокувати нові можливості, такі як багаторазові моделі розпізнавання зображень, які використовують дуже глибокі мережі, які можна додати до нової моделі мереж короткострокової пам’яті і використовувати для фотографування. Крім того, є мережі декодерів, які можуть використовуватися для вхідних і вихідних послідовностей різної довжини.

Глибинне навчання, підмножина машинного навчання, відноситься до систем, які можуть навчатися без розмічених даних. Воно передбачає навчання даних комп'ютеру для розпізнавання закономірностей, а не програмування з конкретними правилами. Тренувальний процес включає в себе подачу великих обсягів даних до алгоритму і дозволяє визначити закономірності у цих даних. Як результат, ці системи продовжують з часом ставати ефективними без втручання людини.

Отже, для розпізнавання намірів користувача ефективним засобом є використання рекурентних мереж, які допомагають ефективно знаходити зв’язки у тексту та мають високі показники точності на тренувальних та тестових наборах даних за результатом дослідження.

## 2.3 Технології розпізнавання сутностей

Розпізнавання сутностей є технікою вилучення інформації, яка відноситься до процесу ідентифікації та класифікації ключових елементів з тексту в попередньо визначені категорії. Таким чином, воно допомагає перетворити неструктуровані дані в структуровані дані, а тому зрозумілі і доступні для стандартної обробки, які можуть бути застосовані для отримання інформації, вилучення фактів і відповіді на запитання. Процес визначення сутностей є комплексним та складається з декількох кроків.

У деяких форматах, таких як файли документів, електронні таблиці, веб-сторінки та соціальні медіа, текст подається у якості неструктурованих даних. Можливість ідентифікувати сутності – людей, місця, організації та концепції, числові вирази (суми валют, телефонні номери тощо), а також тимчасові вирази (дати, час, тривалість, частота тощо) – необхідна складова аналізу намірів, адже вона допомагає отримати параметричні ознаки речення. Незалежно від того, чи є аналітик, який має сотні або більше документів для розгляду, або журналіст, який тільки що отримав дамп даних в декількох гігабайтах, вони не можуть спочатку знати, ні ключові ознаки, що містить інформація, ні того, що вони повинні шукати.

Вилучення об'єктної сутності може забезпечити корисний перегляд невідомих наборів даних шляхом негайного виявлення бажаної контекстної частини інформації. Як результат, аналітик зможе побачити структуровану репрезентацію всіх назв людей, компаній, брендів, міст або країн, навіть телефонних номерів у корпусі, який може служити відправною точкою для подальшого аналізу.

Завдання ідентифікації намірів, параметрів запиту та ключових сутностей – дана задача вилучення інформації є важливою частиною аналізу природної мови. Існує багато методів і методологій, які в даний час вивчаються для вирішення цієї складної задачі. Виявлено та досліджено декілька підходів до машинного

навчання, а також процесу придбання знань, що мають відношення до ідентифікації сутностей. Програми, які вимагають розпізнавання названих об'єктів, найкраще будуть обслуговуватися певною комбінацією підходів, що ґрунтуються на знаннях, та недетермінованих підходів.

Визначення сутностей та об’єктів є не тільки задачею добування інформації, воно також відіграє важливу роль у репрезентативності в інших програмах обробки природної мови. Семантичні парсери, частина маркерів мовлення та подання тематичних значень могли б бути розширені за допомогою такого типу міток, що допоможуть забезпечити кращі результати розпізнавання. Інші, специфічні для розпізнання сутностей додатки використовують чимало технік, включаючи системи запитань і відповідей, автоматичне пересилання, текстові втілення, пошук документів і новин. Навіть на поверхневому рівні розуміння суб'єктів, що беруть участь у документі, дає набагато багатші аналітичні рамки та перехресні посилання.

Іменовані об'єкти мають три категоризації верхнього рівня: імена об'єктів, тимчасові вирази та числові вирази. Оскільки категорія імен суб'єктів описує унікальні ідентифікатори людей, місцеположень, геополітичних тіл, подій та організацій, вони зазвичай називаються названими об'єктами, саме тому більша частина додатків зосереджується виключно на цій категорії, хоча не важко уявити, як розширити пропоновані системи для покриття повного завдання ідентифікації об’єктів. Крім того, задача, на якій базується стандарт оцінки для таких систем, оцінює лише категоризацію організацій, осіб, місцеположень і різних назв. Висновок відноситься до здатності системи визначити, що частина потоку інформації є фактично іменованим об'єктом, або, іноді, більш важливо, визначити класифікацію імені об'єкта, особливо в місцях, де є неоднозначність. Наприклад, «Вашингтон» може посилатися на ім'я або місце розташування. «Галактика» може посилатися на загальний іменник або професійну футбольну команду великої ліги. Моделі максимальної ентропії, приховані марковські моделі та інші статистичні методи використовуються для виконання цього аналізу,

зазвичай реалізованого у якості комплексної системи, що складається з алгоритмів машинного навчання.

Нелокальні моделі залежностей відносяться до можливості ідентифікації декількох маркерів, які повинні мати однакове призначення міток або перехресне посилання. Важливо відзначити, що задача ідентифікації має складності, пов’язані з представленням даних, наприклад. Bengfort, bengfort і BENGFORT повинні бути ідентифіковані як одна й та сама сутність, і вони створюють конфлікти у системах, засновані на правилах на рівні слів (наприклад, знайдіть всі слова, які капіталізуються). Але далі, навіть різні сутності повинні бути ідентифіковані аналогічно – UMBC у якості Університету штату Меріленд Балтімор або визначення комплексної частини назви в одному місці, що відсутні в іншому, як у випадку іменування «президента Барака Обами» та «Обами». Нелокальність цих моделей відноситься до використання цих термінів поза рамками послідовності токенів, які аналізуються разом (зазвичай речення), а скоріше у всьому документі або корпусі.

Імена, оскільки вони однозначно ідентифікують сутності, є галуззю, яка не легко захоплюється навіть найпоширенішими лексиконами. Наприклад, просто створення списку назв усіх компаній, що утворилися в Сполучених Штатах, буде різко розширюватися кожен рік. Проте, для багатьох підходів і рішень, а не тільки для нелокальних моделей залежності, потрібні зовнішні знання та лексики.

Для забезпечення найкращих результатів бібліотеки, що впроваджують розпізнавання сутностей повинні бути спеціальним чином натреновані. Це дозволить їм використовувати знання про структуру речень та загальні обороти для виконання ефективного аналізу речення та вилучення різноманітних сутностей.

Отже, технології розпізнавання сутностей є невід’ємною частиною при роботі з задачею класифікації намірів та допомагають знаходити основні сутності, за якими визначаються параметри запитів.

## 2.4 Огляд технологій розпізнавання голосу

Проблеми з технологією розпізнавання голосових даних численні, але вони звужуються. Прикладом таких проблем є подолання поганого записуючого обладнання, фоновий шум, складні акценти і діалекти, а також різноманітні голоси людей.

Навчання з метою навчитися розуміти розмовну мову, як це роблять люди, – це не вдосконалена галузь знань. Слухання і розуміння системою мови людини є комплексною задачею, що складається з таких етапів як розпізнавання звуків окремих літер, переведення голосових даних у текстові, попередня обробка текстових даних та аналіз окремих слів або речень. Різноманітність акцентів, різні інтонації, тембр голосу є факторами, які ускладнюють задачу розпізнавання голосу, створюючи труднощі для алгоритмів машинного навчання.

Реакція в режимі реального часу залежить від обчислювальних можливостей мережі, підключення до мережі та мікрофона пристрою. Коли користувач надає голосову команду, програма повинна взаємодіяти з сервером, щоб перетворити мовні дані в текст. Після того, як текст перетвориться і надсилається назад на додаток, він аналізується. Процес дослідження надсилання та отримання таких даних називається аналізом реакції в реальному часі. Якщо визначено дію для пошуку, пристрій надсилає на сервер інший запит для отримання результатів. У таких випадках затримка мережі може бути найскладнішою справою. Щоб подолати це, необхідно переконатися, що вихідний код програми належним чином оптимізований. Крім того, архітектурні підходи можуть переміщати функції розпізнавання голосу та пошуку на різні сервера.

Таким чином, сучасні технології розпізнавання голосу знаходяться на стадії стрімкого розвитку та у майбутньому будуть здатні виконувати розпізнавання запису будь-якої якості.

# 3 АНАЛІЗ ЗАСОБІВ РОЗРОБКИ

Для вибору веб-фреймворку слід проаналізувати відмінності між найбільш функціональними засобами – Spring та Struts. Spring як і багато інших веб-фреймворків, розроблено навколо шаблону фронтального контролера, де центральний сервлет, надає спільний алгоритм обробки запитів, а фактична робота виконується конфігурованими компонентами делегатів.

Диспечер, як і будь-який сервлет, повинен бути оголошений і відображений відповідно до специфікації сервлетів за допомогою конфігурації. У свою чергу, диспечер використовує конфігурацію Spring, щоб виявити необхідні компоненти для відображення запитів, роздільної здатності перегляду, обробки винятків та багато іншого.

Технологія конвертації об’єктів використовує ключові бібліотеки Jackson, щоб відобразити вміст відповіді як об’єктну нотацію. Завдяки цьому весь вміст мапи моделі (за винятком класів, специфічних для фреймворку) кодується у форматі, що підтримується сучасними браузерами. Для випадків, коли вміст мапи необхідно відфільтрувати, можна вказати певний набір атрибутів моделі для кодування за допомогою властивості modelKeys. Також є можливість використовувати властивість вилучення специфічного параметру для того, щоб значення в моделі з одним ключем витягувалося і серіалізувалося безпосередньо, а не як карта атрибутів моделі.

За необхідності можна налаштувати відображення нотації об’єктів, використовуючи надані бібліотекою анотації. Якщо потрібно додаткове керування, є можливість ввести користувальницький маппер через властивість для випадків, коли потрібно надати користувальницькі серіализатори об’єктної нотації і десериализатори для конкретних типів.

У деяких випадках може знадобитися замінити контролер проксі-класом аспекту під час виконання. Одним з прикладів є ситуація, у якій вирішено мати анотації @Transactional безпосередньо на контролері. Якщо це так, то для

контролерів спеціально рекомендується використовувати проксі-модель на основі класів. Це – типовий вибір для контролерів. Однак, якщо контролер повинен реалізувати інтерфейс, який не є зворотним викликом контексту, можливо, потрібно явно налаштувати проксі на основі інших бібліотек аспектного програмування.

Spring framework є технологією, яка використовується для платформи Java або використовується на мові Java. Вона також називається фреймворком додатків, який використовується Java-додатком для обробки інфраструктури та реалізації інверсії контролю та ін'єкції залежностей. Функції Spring framework використовуються для розробки веб-додатків.

Основи структури Spring були розроблені Pivotal Software. Спочатку фреймворк Spring був випущений в 2002 році. Він був написаний на Java. Він підтримує лише платформи Java Virtual Machine (JVM). Він підтримує крос-платформні операційні системи. Даний фреймворк надає багато переваг для розробки програми, головними з яких є виконання методу в транзакціях баз даних і віддаленій процедурі, без необхідності роботи з інтерфейсом транзакцій і локальною операцією управління методом Java або обробником повідомлень без використання інтерфейсів.

Struts фреймворк – платформа з відкритим вихідним кодом для розробки веб-додатків на основі Java. Він розширює інтерфейси сервлетів і використовує багатошарову архітектуру. Це дозволяє створювати розширювальні, підтримувані та гнучкі веб-додатки на стандартних технологіях, таких як сторінки веб-представлення та біни.

Функції Spring та Struts відрізняються, і обидва вони широко використовуються для розробки веб-додатків. Spring каркас є більш ефективним, ніж struts, іноді Spring засоби збільшують розмір системи, але за рахунок цього постачаються ключові можливості веб розробки. Spring framework має більше функціональних можливостей, ніж Struts. Даний фреймворк використовується в основному тому, що він є більш безпечним і більш ефективним. Його версії наповнені ефективними компонентами та модулями. Через це його каркас дуже

гнучкий і забезпечує кращу продуктивність, використовується на різних платформах. Spring framework успішно працює у веб-додатках у порівнянні з Struts завдяки своїй незалежній архітектурі та чіткій відмінності між моделлю, переглядом та контролером, але в Struts дані особливості не спостерігаються. Spring Security є одним з найкращих способів захисту програми. Spring використовується для розробки веб-інтерфейсів, моделі клієнт-сервер, сервісно-орієнтованої архітектури і для обробки баз даних. Його каркас є більш ефективним для обробки запиту, ніж аналог від Struts.

При виборі серверу слід звертати увагу на гнучку конфігурацію і сумісність з новітніми бібліотеками. Apache Tomcat є сервером веб-додатків який виконує дані вимоги.

Використовуючи свою реалізацію інтерфейсів сервлетів, Tomcat може приймати запити від клієнта, динамічно компілювати клас керований контейнером для обробки запиту, як зазначено у відповідному контексті програми, і повертати результат клієнту. Цей спосіб генерування динамічного контенту забезпечує надзвичайно швидку обробку запитів на платформі.

Крім того, оскільки специфікація сервлетів призначена для взаємодії з усіма іншими основними веб-технологіями Java, сервлет, розміщений на сервері Tomcat, здатний використовувати будь-який ресурс, який Tomcat надає йому. Вкладені файли конфігурації в Tomcat дозволяють здійснювати надзвичайно тонкий контроль доступу до ресурсів, зберігаючи вільну зв'язку, простоту розгортання та логічні описи архітектури, що читаються людиною.

Важливою частиною екосистеми Java є інверсія контролю. У найширшому сенсі це означає, що фреймворк виконує обов'язки від імені компонентів, які містяться в його контексті. Самі компоненти спрощуються, оскільки вони звільняються від цих обов'язків. Наприклад, ін'єкція залежностей звільняє компоненти від відповідальності локалізації або створення їх залежностей. Аналогічно, аспектно-орієнтоване програмування звільняє бізнес-компоненти від загальних перехресних проблем, модулюючи їх у багаторазові аспекти. У

кожному випадку кінцевим результатом є система, яка легше перевіряти, розуміти, підтримувати і розширювати.

Крім того, можливості Spring забезпечують комплексну модель програмування для створення корпоративних додатків. Розробники отримують вигоду від узгодженості цієї моделі і, особливо, від того, що вона ґрунтується на добре встановлених передових практиках, таких як програмування на інтерфейси та надання переваги композиції над успадкуванням. Спрощені абстракції та потужні бібліотеки підтримки підвищують продуктивність розробників, одночасно підвищуючи рівень тестування та портативності.

Модуль інтеграції розроблено за принципами незалежності компонентів. Він поширює модель програмування Spring на домен обміну повідомленнями та будується на існуючій підтримці корпоративної інтеграції Spring, що забезпечує ще більш високий рівень абстракції. Він підтримує архітектури, керовані повідомленнями, де інверсія керування застосовується до проблем, пов'язаних з виконанням, наприклад, коли слід запускати певну бізнес-логіку і де слід відправляти відповідь. Даний модуль підтримує маршрутизацію та трансформацію повідомлень, щоб різні транспортні засоби та різні формати даних могли бути інтегровані без впливу на зміст повідомлення. Іншими словами, питання обміну повідомленнями та інтеграції розглядаються окремо. Компоненти бізнесу далі відокремлюються від інфраструктури, і розробники звільняються від складних функцій інтеграції.

Як розширення моделі програмування Spring, інтеграція надає широкий спектр параметрів конфігурації, включаючи анотації, формат з підтримкою простору імен, засіб для керування елементами і пряме використання базового інтерфейсу. Цей інтерфейс заснований на чітко визначених стратегічних інтерфейсах і адаптерах.

Ін'єкція залежностей – це процес, за допомогою якого об'єкти визначають свої залежності (тобто інші об'єкти, з якими вони працюють) тільки через аргументи конструктора, аргументи фабричного методу або властивості, які встановлюються на об'єкті після його побудови або повернення з викликаного

методу. Контейнер потім ін'єктує ці залежності, коли він створює компонент. Цей процес є принципово зворотним відносно самого компонента, який не контролює реалізацію або розташування його залежностей, використовуючи лише їх інтерфейси за допомогою безпосередньої побудови класів або механізму пошуку сервісів.

Код є більш чистим за принципом інверсії залежностей, і розв'язка є більш ефективною, коли об'єктам надаються їх залежності. Об'єкт не шукає своїх залежностей і не знає розташування або клас залежностей. Як наслідок, ваші класи стають простішими для перевірки, особливо коли залежності знаходяться на інтерфейсах або абстрактних базових класах, які дозволяють використовувати заглушки або макетні реалізації в модульних тестах.

Інверсія залежностей може бути проведена як ін'єкція на основі конструкторів та ін'єкція залежності від сеттера.

Ін'єкція залежностей на основі конструктора є найбільш стандартним підходом. Конструктор, заснований на інверсії залежностей, виконується контейнером, який викликає конструктор з низкою аргументів, кожен з яких представляє залежність. Виклик статичного фабричного методу з конкретними аргументами для побудови біна майже еквівалентний, і цей метод розглядає аргументи конструктора.

Є плюси і мінуси для розгляду модулю валідації як бізнес-логіки, і Spring пропонує дизайн для перевірки (і прив'язки даних), що не виключає жодного з них. Зокрема, перевірка не повинна бути прив'язана до веб-рівня і повинна бути легко локалізована, та повинна бути можливість підключити будь-який доступний валідатор. Беручи до уваги ці проблеми, Spring має інтерфейс контролюючого класу, який є як базовим, так і надзвичайно корисним у кожному шарі програми.

Прив'язка даних корисна для того, щоб вхід користувача міг бути динамічно прив'язаний до моделі домену програми (або будь-яких об'єктів, які використовуються для обробки вводу користувача). Фреймворк надає відповідний клас обробки. Контролюючий клас входить до пакету перевірки, який в основному використовується в рамках веб-частини.

Обгортка класу є фундаментальною концепцією Spring Framework і використовується в багатьох місцях. Однак, ймовірно, не потрібно використовувати безпосередньо даний метод, якщо функціонал може бути реалізований більш простішим інструментом.

Модуль сполучення даних використовує реалізації механізму роботи з конфігурацію для аналізу та форматування значень властивостей. Модулі редагування конфігурації відповідають сучасним потребам налаштування системи, а також надають можливість динамічної зміни її поведінки. Пакет перетворення надає можливість конвертації загального типу, а також пакет форматування сутностей високого рівня для форматування значень полів, отриманих на основі інтерфейсу користувача.

Інструмент аспектного програмування реалізований в Java. Немає необхідності в спеціальному процесі компіляції. Модуль аспектів не потребує контролю ієрархії завантажувачів класів і, таким чином, підходить для використання в контейнері сервлетів або серверах додатків.

Модуль аспектів в даний час підтримує тільки спосіб виконання об'єднаних точок (консультування виконання методів на класах). Поле перехоплення не реалізовано, хоча підтримку перехоплення полів можна було б додати, не порушуючи ядро інтерфейсів. Якщо потрібно впроваджувати доступ до полів і оновити точки з'єднання, використовують підсистему розширених аспектів.

Підхід до аспектного програмування відрізняється від підходу більшості інших інструментів. Мета полягає не в тому, щоб забезпечити найбільш повне впровадження зміни поведінки. Навпаки, мета полягає в тому, щоб забезпечити тісну інтеграцію між реалізацією аспектів платформи Java і модулів Spring, щоб допомогти вирішити загальні проблеми в корпоративних додатках.

Таким чином, наприклад, функціональність аспектів Spring зазвичай використовується спільно з контейнером компонентів. Аспекти налаштовуються за допомогою синтаксису визначення звичайних компонентів (хоча це надає потужні можливості автоматичного проксі). Це є суттєвою відмінністю від інших реалізацій аспектів. Деякі функції неможливо виконати легко або ефективно за

допомогою модулю аспектів, наприклад, контролювати дуже дрібні об'єкти (як правило, об'єкти домену). Аспектний підхід Java є найкращим вибором у таких випадках. Однак досвід свідчить про те, що модуль аспектів забезпечує відмінне рішення більшості проблем у корпоративних Java-додатках.

Модуль аспектів не прагне конкурувати з аналогічними розробками аспектного програмування для забезпечення найкращого рішення. Обидва підходи, такі як модуль аспектів та повномасштабні бібліотеки є розширювальними, і вони доповнюють один одного, а не конкурують. Зазвичай система плавно інтегрує модуль аспектів з бібліотекою аспектів, щоб включити всі способи використання аспектного програмування в рамках послідовної архітектури додатків на основі Spring.

Необхідно проаналізувати системи автоматичної зборки проекту. Для автоматизації проекту необхідна система з підтримкою архетипів. Чудовим прикладом такої системи є Maven. Архетип визначається як оригінальний зразок або модель, з якої робляться системи схожої архітектури. Імена відповідають тому, як необхідно організувати систему, яка забезпечує послідовні засоби генерування проектів Maven. Архетип надає користувачам засоби для створення параметризованих версій цих шаблонів проектів.

Використання архетипів забезпечує відмінний інструмент, щоб дозволити розробникам швидко відповідати найкращим практикам, які використовуються проектом або організацією. В рамках проекту Maven використовують архетипи, щоб спробувати наблизити користувачів до роботи якнайшвидше, надавши зразковий проект, який демонструє багато можливостей Maven, впроваджуючи нових користувачів у найкращі практики Maven. За лічені секунди новий користувач може мати робочий проект Maven, який буде використовуватися в якості акселератору для вивчення більшої кількості функцій у Maven. Також адитивний механізм архетипу дозволяє брати участь у проекті архетипу, щоб частини або аспекти проекту могли бути додані до існуючих проектів. Хорошим прикладом цього є архетип веб системи від Maven. Якщо, наприклад, використати архетип швидкого запуску для створення робочого проекту, можна швидко

створити сайт для цього проекту, використовуючи архетип сайту в межах існуючого проекту. Гнучка система архетипів адаптується під окремі потреби.

Робота як з об'єктно-орієнтованим програмним забезпеченням, так і з реляційними базами даних може бути громіздкою і трудомісткою. Витрати на розробку значно вищі завдяки невідповідності парадигми між тим, як дані представлені в об'єктах і реляційних базах даних. Hibernate – це об'єктне / реляційне рішення зіставлення для середовищ Java. Термін «об'єкт / реляційне зіставлення» відноситься до техніки відображення даних від представлення об'єктної моделі до реляційного представлення моделі даних (і навпаки).

Hibernate не тільки піклується про відображення з класів Java до таблиць баз даних (і від типів даних Java до типів бази даних), але й надає засоби запиту даних і пошуку. Це може істотно скоротити час розробки, що інакше витрачається при ручній обробці даних в базах даних. Мета дизайну Hibernate - позбавити розробника від 95% загальних завдань програмування, пов'язаних із збереженням даних, усунувши необхідність ручної обробки ручних даних за допомогою низькорівневих засобів. Однак, на відміну від багатьох інших рішень, що існують, Hibernate не приховує можливості баз даних і гарантує, що функції реляційних технологій збережені.

Hibernate не може бути найкращим рішенням для орієнтованих на дані додатків, які використовують тільки збережені процедури для реалізації бізнес-логіки в базі даних, найбільша користь йде з об'єктно-орієнтованими моделями доменів і бізнес-логікою в сервісному рівні. Тим не менш, Hibernate, безумовно, може допомогти видалити або інкапсулювати специфічний для постачальника код запитів до бази даних і допоможе з загальним завданням перекладу набору результатів з табличного представлення на графік об'єктів.

Однією з ключових переваг Hibernate (і дійсно маппінгу об’єктів на таблиці в цілому) є поняття портативності бази даних. Це може визначити користувач, який мігрує від одного постачальника бази даних до іншого, або це може впливати на бібліотеки або розгортання програми, що споживає Hibernate для одночасної підтримки різних систем управління базами даних. Незалежно від

точного сценарію, основна ідея полягає в тому, що Hibernate допомагає працювати з будь-якою кількістю баз даних без змін у коді, і в ідеалі без будь-яких змін метаданих відображення.

Перший засіб портативності для Hibernate – діалект, що є спеціалізацією відповідного контракту. Діалект інкапсулює всі відмінності в тому, як Hibernate має зв'язуватися з певною базою даних, щоб виконати деяке завдання, наприклад, отримати значення послідовності або структурувати запит отримання даних. Hibernate поєднує широкий діапазон діалектів для багатьох найпопулярніших баз даних. Спочатку Hibernate завжди вимагала, щоб користувачі вказували, який діалект використовувати. У випадку, коли користувачі прагнуть одночасно орієнтуватися на декілька баз даних, їхня збірка була проблематичною. Як правило, це вимагало від користувачів налаштування діалекту Hibernate або визначення власного методу встановлення цього значення.

Видобуток, по суті, є процесом отримання даних з бази даних і наданням доступу до програми. Налаштування вибірки програми є одним з найбільших факторів у визначенні того, як буде виконуватися програма. Збирання занадто великої кількості даних, з точки зору ширини (значення / стовпці) та / або глибини (результати / рядки), додає непотрібні накладні витрати як у зв'язку з драйвером та об’єктом результату запиту. Налаштування того, як програма отримує дані є чудовою можливістю впливати на загальну продуктивність програми.

Термін «доменна модель» походить з галузі моделювання даних. Це модель, яка врешті-решт описує проблематичну галузь системи. Іноді також використовується термін постійних класів.

У підсумку модель домену прикладної програми є центральним концептом в Hibernate. Вона складається з сутностей, які потрібно відобразити. Hibernate найкраще працює, якщо ці класи дотримуються моделі програмування бінів. Однак жодне з цих правил не є жорсткими вимогами. Насправді, Hibernate припускає дуже мало про природу постійних об'єктів. Модель домену можна виразити іншими способами (наприклад, за допомогою дерев).

Історично, додатки, що використовують Hibernate, впроваджували власний формат файлів відображення. У нових версіях більшість цієї інформації тепер визначається таким чином, що переноситься через провайдерів, використовуючи анотації (та / або стандартизований формат). У контексті, де це можливо, буде зроблено акцент на відображення класів. Для функцій відображення Hibernate, які не підтримуються стандартними конфігураціями, слід надати перевагу анотаціям для розширення Hibernate.

Hibernate розуміє різноманітні представлення даних програми. Можливість читання/запису даних з/до бази даних є функцією типу Hibernate. Тип, у цьому використанні, є реалізацією спеціального інтерфейсу. Цей тип контракту Hibernate також описує різні поведінкові аспекти моделі, такі як перевірка рівності та клонування значень.

Аналогом Hibernate є MyBatis - це відкритий, легкий, та зручний фреймворк. Він виступає як альтернатива Hibernate. Завдяки своїм можливостям він автоматизує відображення між базами даних і об'єктами в різноманітних мовах. Схеми відображення відокремлені від логіки програми, упаковуючи оператори запиту до бази даних у файли конфігурації.

Він абстрагує майже весь код запиту до бази даних і знижує тягар налаштування параметрів вручну завдяки автоматичному прошарку між клієнтом та базою даних. Він має простий інтерфейс для взаємодії а також надає підтримку користувацьких запитів, збережених процедур і розширених відображень.

Значна відмінність між MyBatis та іншими системами зберігання полягає в тому, що MyBatis підкреслює використання запитів на мові бази даних, в той час як інші фреймворки, такі як Hibernate, зазвичай використовують спеціальну власну мову запитів, що є спеціальним прошарком між сервісною частиною та базою даних.

MyBatis постачається з наступними філософіями дизайну: простота – MyBatis є однією з найпростіших систем роботи з базами даних, доступних сьогодні; швидкий розвиток – MyBatis робить усе можливе для полегшення розширюваності функціоналу додатків; портативність – MyBatis може бути

впроваджено практично на будь-якій мові або платформі, такі як Java, Ruby і C # для Microsoft .NET; незалежні інтерфейси – MyBatis надає незалежні від бази даних інтерфейси, які допомагають решті програми залишатися незалежними від будь-яких ресурсів, пов'язаних із збереженням.

По суті, MyBatis – це дуже просте у використанні рішення, яке дає користувачам більшу частину семантики інструментарію роботи з базою даних, без додаткової конфігурації. Іншими словами, MyBatis прагне полегшити розробку додатків шляхом абстрагування деталей низького рівня, що беруть участь у комунікації з базою даних (завантаження драйвера бази даних, отримання і керування з'єднаннями, керування семантикою транзакцій і т.д.), а також забезпечення більш високого рівня абстракції (автоматизовані та конфігуровані відображення об'єктів на виклики бази даних, керування перетворенням типів даних, підтримка статичних запитів, а також динамічні запити на основі стану об'єкта, відображення складних об'єднань у складні графіки об'єктів тощо). MyBatis перетворює компоненти на оператори бази даних, використовуючи гнучкий інтерфейс. Простота є ключовою перевагою MyBatis перед іншими інструментами відображення об'єктних зв’язків, але дана простота створює обмеження конфігурації та складності створення комплексних запитів, що робить його непривабливим інструментом.

Кожен веб-інтерфейс має асоційований контекст програми, який прив'язаний до його життєвого циклу: контекст кореневої веб-програми.

Це стара функція, яка існувала ще до популяризації веб-розробки, тому вона не пов'язана спеціально з будь-якою технологією веб-фреймворків.

Контекст починає роботу, коли програма запускається, і руйнується, коли вона зупиняється, завдяки функції реагуванню на події життєвого циклу програми. Найбільш поширені типи контекстів також можуть бути оновлені під час виконання, хоча не всі реалізації контексту мають цю можливість.

Контекст у веб-застосуванні завжди має підтримку веб-компонентів, хоча може містити лише частину з них. Даний контекст розширює класичну реалізацію з контрактом на доступ до сервлету.

У будь-якому випадку, програми зазвичай не повинні турбуватися про ці деталі реалізації: контекст кореневого веб-додатку – це просто централізоване місце для визначення спільних компонентів.

Важливим аспектом аналізу технологій розробки є вибір архітектури системи. Для визначення найбільш привабливої моделі архітектури слід порівняти мікросервіси та моноліт. На відміну від мікросервісів, монолітне застосування будується як єдина автономна одиниця. Це призводить до уповільнення змін до програми, оскільки вони впливають на всю систему. Зміни, внесені до невеликого розділу коду, можуть потребувати створення та розгортання абсолютно нової версії програмного забезпечення. Масштабування конкретних функцій програми також означає, що потрібно масштабувати всю програму.

Мікросервіси вирішують ці проблеми монолітних систем за рахунок розподілу компонентів. Вони допомагають побудувати додаток як набір невеликих послуг, кожен з яких працює у власному процесі і незалежно розгортається. Ці послуги можуть бути написані на різних мовах програмування і можуть використовувати різні методи зберігання даних. Хоча це призводить до розробки систем, які є багатофункціональними та гнучкими, існує потреба динамічного перетворення. Мікросервіси часто з'єднуються через інтерфейс, і можуть використовувати багато тих же інструментів і рішень, які знаходяться в екосистемі веб-застосувань. Тестування інтерфейсів може допомогти перевірити потік даних та інформації протягом розгортання мікросервісу, а також визначити ефективність інтеграції між довільною кількістю мікросервісів.

Програмне забезпечення, побудоване у вигляді мікросервісів, за визначенням може бути розділено на декілька компонентних послуг. Таким чином, кожна з цих служб може бути розгорнута, налаштована, а потім ініціалізовано самостійно без негативних наслідків щодо цілісності програми. Як наслідок, може знадобитися лише внесення змін до однієї або кількох служб, замість того, щоб повторно розгортати цілі програми. Але цей підхід має свої недоліки, включаючи дорогі віддалені виклики (замість викликів у моноліті),

більш віддалені інтерфейси, а також підвищену складність при перерозподілі відповідальності між компонентами.

Мікросервіси подібні до класичної структури операційної системи: вони отримують запити, обробляють їх і відповідно генерують відповідь. Це протилежно тому, як працюють багато інших продуктів, заснованих на повідомленнях, де використовуються високотехнологічні системи для маршрутизації повідомлень, організації архітектури системи. Можна сказати, що мікросервіси мають розумні кінцеві точки, які обробляють інформацію та застосовують логіку.

Архітектура мікросервісів є еволюційним дизайном та ідеально підходить для еволюційних систем, адже неможливо повністю передбачити типи клієнтів, які можуть одного дня отримати доступ до програми. Багато додатків починаються на основі монолітної архітектури, яку можна повільно переробити на мікросервіси, які взаємодіють над старою монолітною архітектурою через спеціальний інтерфейс.

Мікросервіси дозволяють організувати модульну роботу з будь-якою кількістю репозиторіїв. Таким чином різні частини застосунку будуть керуватися незалежно.

Також даний підхід дозволить використовувати різні мови для написання різних частин системи. Таким чином мікросервісна архітектура має значно менше обмежень у спектрі технологій при порівнянні з монолітним підходом. Цей підхід робить архітектуру мікросервісів привабливим вибором для сервісу голосового управління.

У результаті аналізу засобів розробки було обрано найбільш сучасні та функціональні технології розробки сучасних систем, а також визначено ефективну організацію архітектури та взаємодії сервісу.

# 4 ОПИС РЕАЛІЗАЦІЇ СЕРВІСУ ГОЛОСОВОГО УПРАВЛІННЯ

## 4.1 Вибір засобів розробки

Аналіз та моделювання визначили три основні складові проекту. Першою частиною є сервіс, який обробляє команди користувача та взаємодіє з клієнтами за допомогою REST API [3]. З метою забезпечення якості інтеграції з серверною частиною буде створено UI бібліотеку, що буде готувати дані для відправки та обробляти результат. Третя частина – адміністративна сторінка, що буде містити інформацію про профіль поточного клієнта.

Сервер буде розроблено за допомогою мови програмування Java та фреймворку Spring. Java є розповсюдженою та основною мовою для розробки масштабних застосувань. Серед переваг мови слід виділити високу швидкість роботи, підтримку об’єктного програмування, наявність великої кількості бібліотек та інструментів, що спрощують розробку типових функцій.

Spring є найбільш популярним засобом розробки веб застосувань на мові Java. Модульний підхід до проектування та можливість впроваджувати власні налаштування без пошкодження архітектури роблять даний інструмент дуже гнучким та простим для використання. Постійна підтримка та оновлення технології також грають ключову роль при виборі фреймворку, і Spring постійно оновлюється та надає нові засоби для розробки веб частини застосування.

Концепція взаємодії моделі, вигляду та контролеру була створена з метою розділення логіки системи від логіки зміни інтерфейсу користувача [4]. Основними поняттями є модель даних, що виступає у якості джерела даних, вигляд, який є програмним кодом, відповідальним за візуальне розміщення даних, а також їх оновлення та зміну. Контроллер повинен заповнити модель необхідними даними та викликати код відповідного відображення, таким чином змінивши його як реакцію на певну дію користувача.

MySQL – реляційна система керування базами даних. Наявність інтеграції з фреймворками Spring Data та Hibernate, різноманітність засобів профілювання роблять її чудовим вибором для розробки реляційних систем [5].

Hibernate забезпечує автоматичну інтеграцію з базою даних. Даний фреймворк здатний створювати таблиці автоматично на основі сутностей сервісу. Також він забезпечує автоматичну конвертацію даних таблиці в Java об’єкти, що звільняє програміста від шаблонної роботи.

JQuery є класичною бібліотекою для створення візуальної частини застосунку. В дану бібліотеку входить набор інструментів, що дозволяє швидко відправляти запити до серверної частини, обробляти помилки, реагувати на певні зміни сторінки [6]. Функціонал селекторів забезпечує можливість швидко та прозору змінювати контент сторінки, таким чином впроваджуючи динамічність у поведінку сайту.

REST – архітектурний стиль, що подає архітектуру вхідних точок системи як сукупність ресурсів, на яких проводяться різноманітні операції. Зазвичай даний підхід реалізується з використанням протоколу передачі гіпертексту та об’єктної нотації [7].

Слід зауважити, що у деяких випадках REST не є найкращим вибором для будування точки інтеграції. Оскільки він орієнтований на роботу з ресурсами, його важко використовувати у разі, якщо система містить операцію, яку необхідно інтегрувати. У даному випадку створення штучного ресурсу не є найкращою практикою, тому слід використати інший підхід для проектування точки інтеграції.

JSON – стандарт подання даних, який оперує об’єктами Javascript. Простота відображення та генерації таких даних також зумовлює автоматичну інтеграцію з веб клієнтами, адже Javascript є основною мовою програмування у веб додатках. При використанні цього формату на клієнті не потрібна адаптація даних, що безумовно є великою перевагою даного формату у поточному рішенні.

HTTP – протокол передачі даних, який є основним способом зв’язку веб-систем. Даний протокол добре підходить під сучасні потреби веб додатків, адже

здатний передавати інформацію про тип запиту, метадані про пристрій клієнта. Сьогодні саме він використовується у більшості інтеграцій клієнт-сервер, та становиться платформою для інших стилів та протоколів. До недоліків можна віднести відсутність двосторонньої комунікації з клієнтом, але в рамках поточного проекту даний функціонал не потрібен.

При розробці сервісу також було використано Spring Boot – платформа, що надає можливості акселератора та постачає найважливіші компоненти системи у якості шаблону. Це дозволяє створювати нові системи ефективно, не витрачаючи час на написання стандартних налаштувань та компонентів.

Серед основних компонентів які постачаються у Spring Boot є веб-модуль, модуль безпеки та модуль роботи з базою даних. Також у додатках Spring достатньо легко запровадити тестування на рівні інтеграційних тестів та модульних тестів. Одним з підходів до розділення даних та функцій на тестові і реальні є профілювання. За допомогою інструменту профілювання можна змінювати реалізації інтерфейсів та налаштування не тільки для тестування, але і для конкретного середовища. Дана можливість дозволяє спростити процедури релізу та тестування, адже здатна адаптуватися під потреби конкретного середовища. Слід зазначити, що даний інструмент не може змінити профіль під час роботи програми, тому слід заздалегідь подбати про встановлення відповідного профілю.

Spring [8] складається з багатьох фреймворків та компонентів. Окрім зазначених вище модулів, він також містить розробку, що дозволяє виконувати генерацію коду, відповідального за роботу з базою даних. Базові методи читання, зміни, створення та видалення даних будуть доступні автоматично, та використовуватися у вигляді інтерфейсів [9].

У якості студії розробки буде використано Intellij IDEA Community edition. Дана версія програми є безкоштовною, але має поширений функціонал, який здатний генерувати необхідний код, забезпечувати пошук за проектом, інтегрується з системою зборки та має чудові інструменти для підтримки мови Java. Також дана система здатна виконувати запуск серверу та проводити debug.

Це допомагає при стабілізації рішення, та є зручним засобом пошуку проблеми у коді або переконні у правильності виконання. У якості редактору візуальної частини сервісу буде використано редактор Atom. Завдяки великій колекції додатків, даний інструмент ефективно допомагає створювати клієнтський код, вказує на потенційні проблеми та підтримує форматування.

У якості системи будування проекту обрано Apache Maven. Дана система є декларативною, що надає великі переваги у порівнянні з імперативними аналогами. До переваг також відноситься великий репозиторій бібліотек, які даний інструмент може впровадити у рішення за рахунок простої конфігурації, можливість змінювати будування проекту через додавання своїх модулів. Також дане рішення має набор так званих скелетів програм, які містять необхідний набір папок та файлів, що зазвичай використовуються у певному рішенні.

Для керування версіями системи буде використано Git. Він є багатофункціональним інструментом роботи з проектом. Дозволяє легко контролювати та впроваджувати зміни. Також він є децентралізованою системою контролю версій, що забезпечує наявність кількох точок відновлення системи у разі несподіваної втрати даних.

## 4.2 Архітектура програмної системи

При проектованні архітектури було враховано аспект взаємодії додатку через веб технології. У якості архітектурного підходу було вирішено використати патерн «клієнт-сервер». Даний підхід є класичним при будуванні веб-систем і забезпечує гнучкість та масштабованість. Ключовою перевагою є контроль над виконанням операцій та навантаження системи.

Окрім того, даний підхід позитивно впливає на якість коду. Таким чином забезпечується контроль над потенційними помилками. Рознесення функціоналу

дозволяє мати невеликі, але змістовні файли. На рисунку 4.1 наведене схематичне зображення архітектури системи.

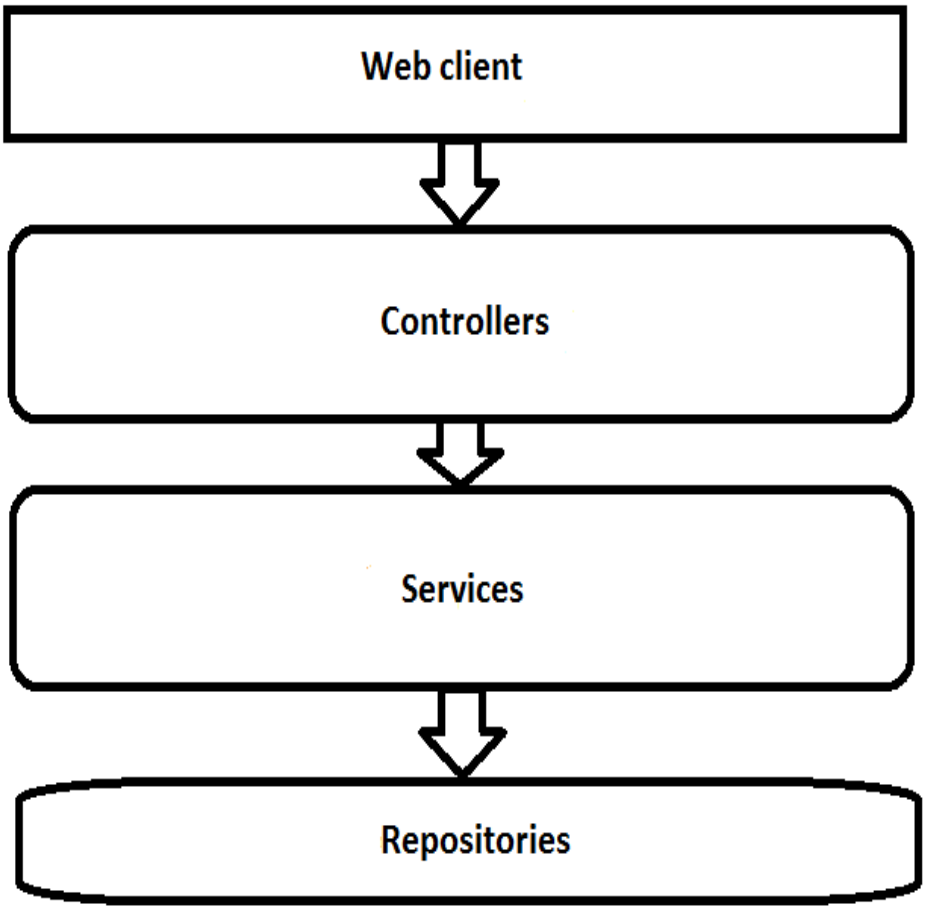


Рисунок 4.1 – Архітектура системи

При даному підході клієнт взаємодіє з сервером завдяки протоколу HTTP. У запиті клієнт впроваджує необхідні дані для виконання операції, а сервер виконує відповідні обчислення та надає відповідь клієнту. Першим кроком є підготування запиту та впровадження інформації у необхідну сутність даних. Далі свою роботу виконує фреймворк, що визначає, до якої частини бізнес-логіки належить цей запит та викликає необхідний контролер.

При виконанні обчислень важливою частиною є організація компонентів системи. Розподілення виклику бази даних, обчислень та підготування відповіді є ключовим аспектом проектування системи, адже саме такий підхід дозволяє легко масштабувати та розширювати її можливості.

Також даний підхід є сумісним з концепцією організації веб застосувань. Таким чином, система здатна розділяти метадані від бізнес-даних, що дозволяє спрощувати структуру рівню сервісів. Використовування фреймворків, що виконують маппінг сутностей на різних рівнях (моделі, об’єкти трансферу даних) позбавляє від необхідності написання шаблонного коду, адже необхідні зв’язки будуть встановлені автоматично.

## 4.3 Моделювання сервісу голосового управління

При моделюванні системи було створено ключові діаграми UML [10], що дозволяють відобразити систему з різноманітних перспектив. Отримані діаграми показують основні сценарії користувача, інфраструктурну частину системи, послідовність дій та основних учасників у процесах, та відношення між основними класами, таким чином візуалізуючи окремі деталі реалізації сервісу.

Користувач використовує команди голосового управління, на які веб-система повинна дати відповідь. Команди можуть бути пошуком товарів на сайті, запит отримання довідкової інформації, підтвердження натиснення форми та інші. Для того, щоб розпізнати значення команди, веб-система буде звертатися до сервісу голосового управління. Отримання даних про команду можливе за умови розпізнавання голосу та його перетворення у текстове речення. Про це буде дбати клієнтська бібліотека, яка, розпізнавши голосове речення, постачатиме на сервер текстове відображення слів користувача, після цього сервіс голосового управління буде здатний віддати необхідну інформацію веб-системі. У якості відповіді буде надано назву команди та необхідні параметри, що допоможуть конкретизувати запит користувача.

Важливим моментом є здатність системи обробити відповідь сервісу. Для цього на клієнтській частині необхідно буде реалізувати механізм реагування на різноманітні команди. Слід приділити увагу випадкам, коли команда, що

підходить під запит користувача не була знайдена. У такому разі клієнтська система повинна відобразити спеціальне повідомлення, яке надасть інформацію про основні типи користування голосовими командами та можливість здійснити новий запит, що їм відповідає. Таким чином буде забезпечено належну якість роботи кінцевого користувача з функціоналом голосового пошуку.

Діаграма прецедентів має наступних акторів: кінцевий користувач, веб система, якою він користується та власне сервіс голосового управління, що представлено на рисунку 4.2.

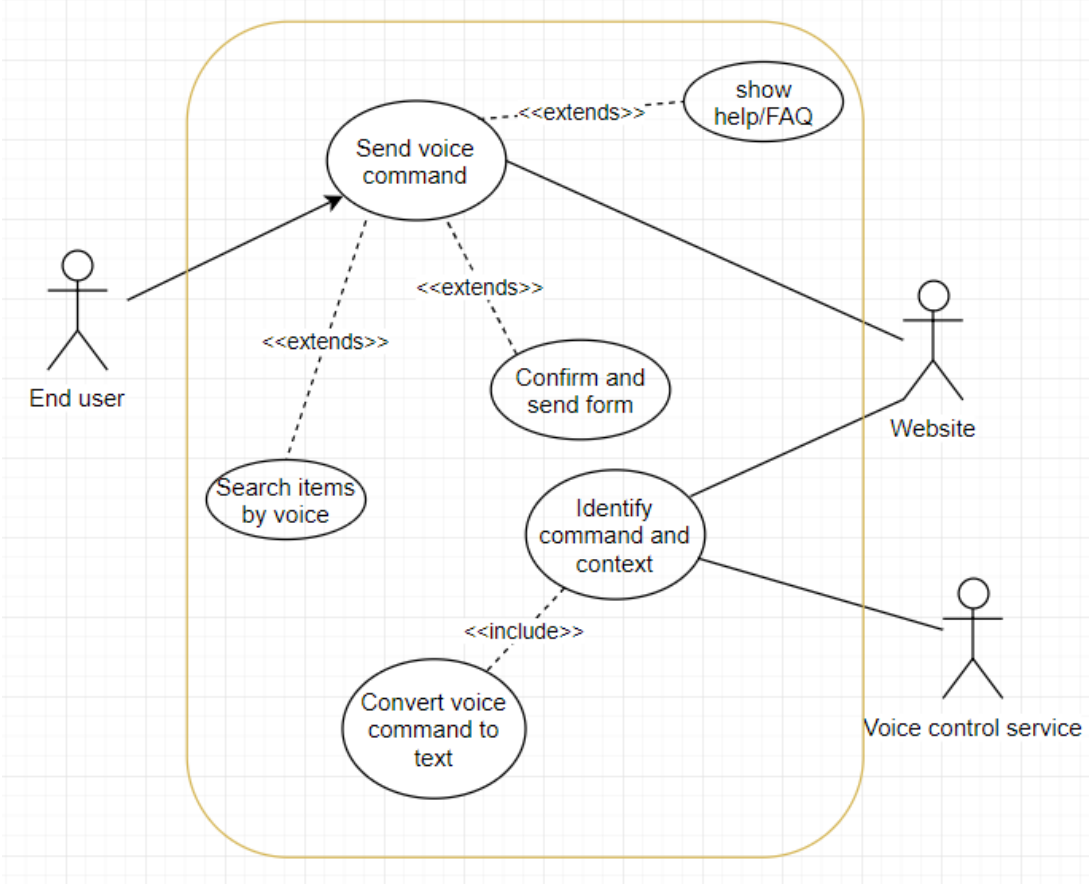


Рисунок 4.2 – Діаграма прецедентів

Основними сутностями будуть користувач, токени та команда. Користувач буде мати ім’я, адресу електронної пошти, пароль та токени, що дозволяють йому використовувати систему. Токен буде містити інформацію про кількість запитів, дозволену користувачу. Команда буде містити результат обробки даних користувача, що складається з дії, яку треба виконати у якості реакції системи на

запит та контексту, що буде містити набор аргументів. Прикладом запиту пошуку товарів буде відповідна команда з даними про операцію пошуку та назвою продукту у якості контексту.

Забезпечувати оновлення даних та генерацію команди будуть відповідні сервіси. Основними сервісами, що контролюють стан системи будуть UserService, TokenService та CommandService. Вони будуть мати відповідні сервіси, що будуть організовувати роботу з джерелом даних. Наявність розділення на інтерфейс та імплементацію організує роботу з різними джерелами даних за необхідності.

На діаграмі класів, яка представлена на рисунку 4.3, зображено моделі та сервіси, які будуть взаємодіяти у системі.

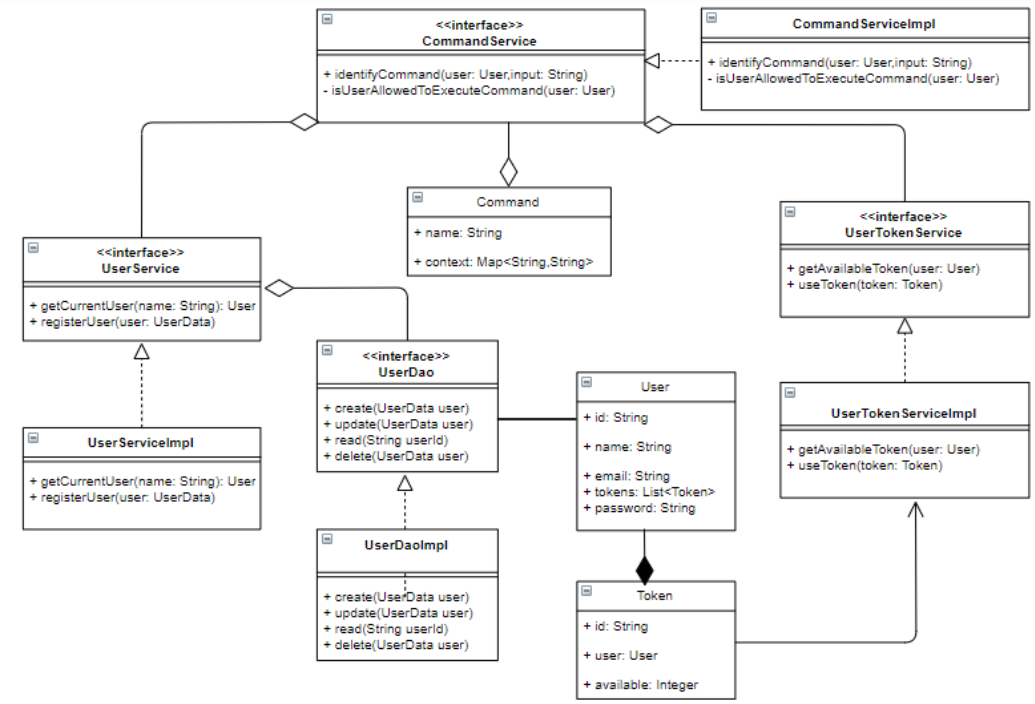


Рисунок 4.3 – Діаграма класів

Важливим моментом є організація отримання залежностей між компонентами. Оскільки явний вибір імплементації може призвести до складності зміни системи, було вирішено виділяти управління залежностями у спеціальний інструмент. Таким чином, основні сутності системи не знають, яка імплементація використовується у залежності.

Для користування системою користувачу необхідно буде записати голосом свою команду, хоча сервіс розпізнавання команд працює з текстом, що означає варіацію з текстовим вводом команди у разі якщо користувач не має мікрофону. Обробка вводу користувача проводиться спеціальним інструментом, який за сценарієм буде створювати умови для обробки команди.

На діаграмі послідовностей, що представлена на рисунку 4.4, зображено послідовність дій, які будуть виконані при використанні сервісом.

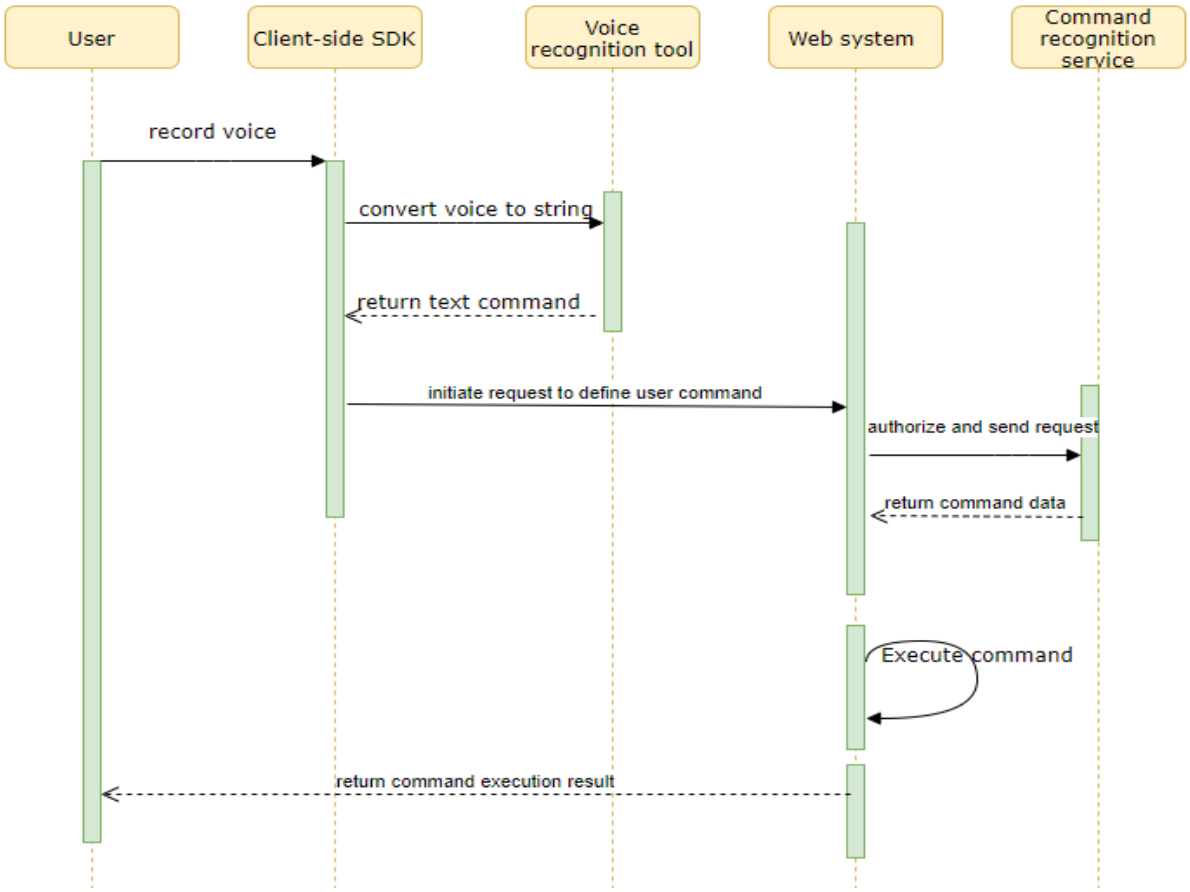


Рисунок 4.4 – Діаграма послідовностей

Після того, як користувач за допомогою мікрофону вводить команду, клієнтська частина сервісу користується механізмами для перетворення голосу в текст. Після цього веб-система, у яку було інтегровано сервіс, виконує запит до сервісу визначення команд. Після отримання відповіді від сервісу, веб-система виконує логіку обробки отриманої команди. У разі, якщо команда була успішно розпізнана, веб система дає відповідь клієнту. У якості різноманітних реакцій

системи можуть бути наступні: натиснення на кнопку, перехід до специфічної сторінки, виклик додаткових запитів до системи через REST API. У підсумку, система залишає за собою право ігнорувати відповідь сервісу, змінювати її, або ж впроваджувати комплексну поведінку на основі контексту відповіді.

На діаграмі станів, що зображена на рисунку 4.5, зображено процес користування системою у контексті контролю запитів користувача.

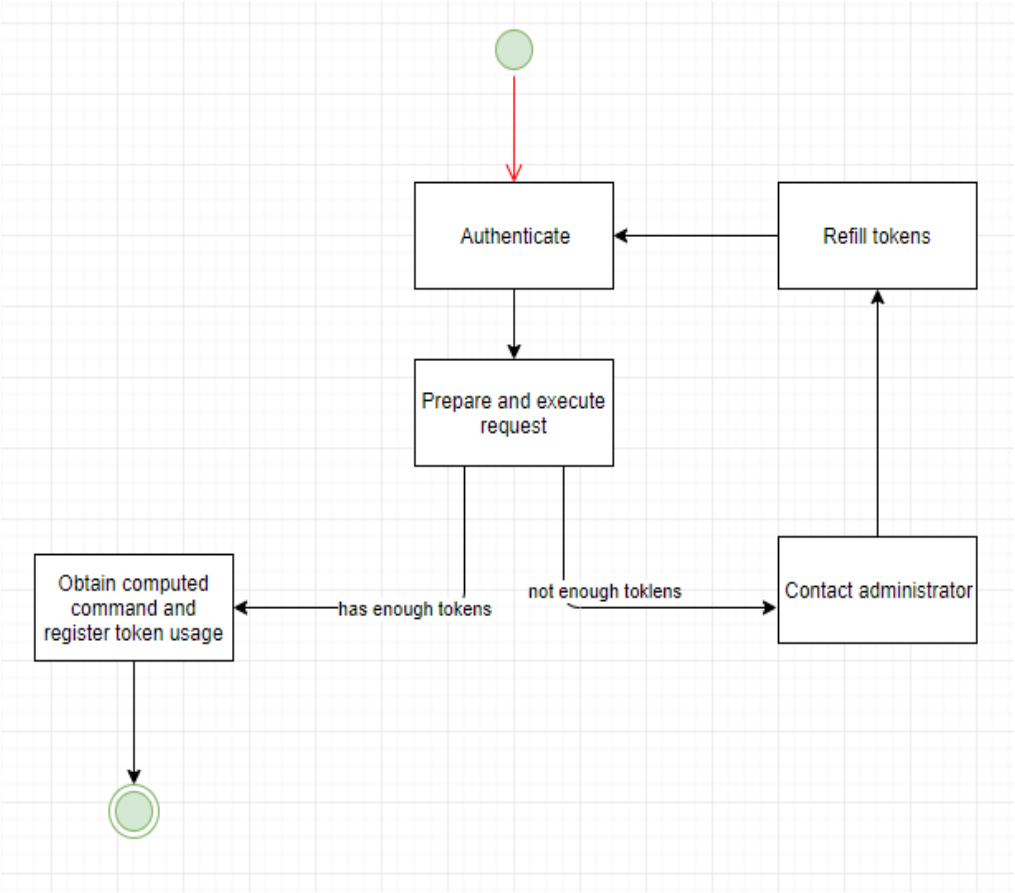


Рисунок 4.5 – Діаграма станів

Після підготовки запиту він направляється на серверну частину. Якщо серверна частина не знаходить токени для нових запитів, користувач повинен замовити собі нові. Для цього йому необхідно зв’язатися з адміністратором. Після отримання нових токенів користувач повинен виконати повторну аутентифікацію. У разі успішного оновлення балансу він буде здатний отримувати результати у формі команд.

Дані про баланс також будуть доступні на спеціальній сторінці, що буде постачатися для перевірки користувачем кількості обмежень. Серед подальших планів можливо введення різноманітних способів (користування системою протягом зазначеного часу та інші умови). Даний підхід дозволяє створити систему контролю за користуванням, яка буде підтримувати розширення та впровадження нового функціоналу. Дані про контакти адміністратора також будуть присутні у цій системі. За допомогою будь-якого зручного виду комунікації клієнт може поновити кількість спроб.

Існують два можливих сценарію взаємодії з сервером. При першому сценарії через браузер клієнта виконується прямий запит до системи розпізнавання команд. Такий підхід є гарним для швидкої інтеграції або створення демо. Другий підхід використовує систему, що інтегрується з сервісом у якості проксі. Таким чином забезпечується безпека взаємодії, адже дані авторизації зберігаються на серверній частині та недоступні для клієнта безпосередньо.

У якості артефакту для клієнта використовується спеціальний модуль, що здатний обробляти дані користувача та ініціювати запит до сервісу. До артефактів сервісу належить система обробки вводу та створення команд, а також база даних. Важливим моментом є сценарій роботи системи у разі виникнення інфраструктурних обмежень. Частина алгоритму, що виконується на клієнті, буде здатна продовжити свою роботу, незважаючи на відмову двох інших частин алгоритму.

Важливим моментом проектування системи є врахування приблизної кількості запитів та навантаження на систему. Система повинна бути готова до несподіваного збільшення кількості запитів та адекватну реагувати на виникнення даних обставин. Для цього слід забезпечити модульну архітектуру, яка може бути динамічно маштабована.

Таким чином забезпечуються різноманітні сценарії інтеграції та збереження необхідних даних про використання системи. Дана архітектура відображена на

рисунку 4.6, та демонструє комунікацію між пристроєм клієнта та серверною частиною, а також взаємодію різноманітних частин інфраструктури сервісу.

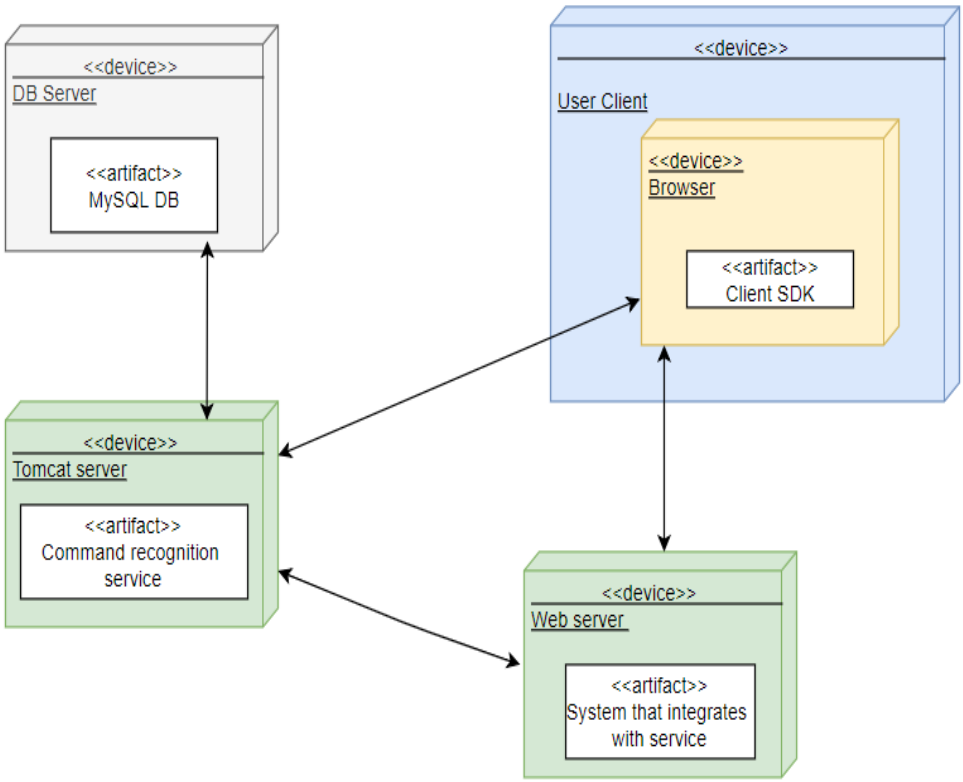


Рисунок 4.6 – Діаграма розгортання

За допомогою даних діаграм було відображено ключові аспекти системи. Було відображено інфраструктурну перспективу, основні процеси сервісу, сценарії використання та послідовність дій різноманітних учасників процесу.

## 4.4 Огляд функцій кінцевого користувача

З точки зору користувача голосове управління інкапсульовано в окремий елемент, завдяки якому користувач здатний вводити команди через мікрофон.

Приклад інтеграції даного клієнтського компоненту з системою магазину та зовнішній вигляд можливої інтеграції для виконання запитів користувача зображено на рисунку 4.7.

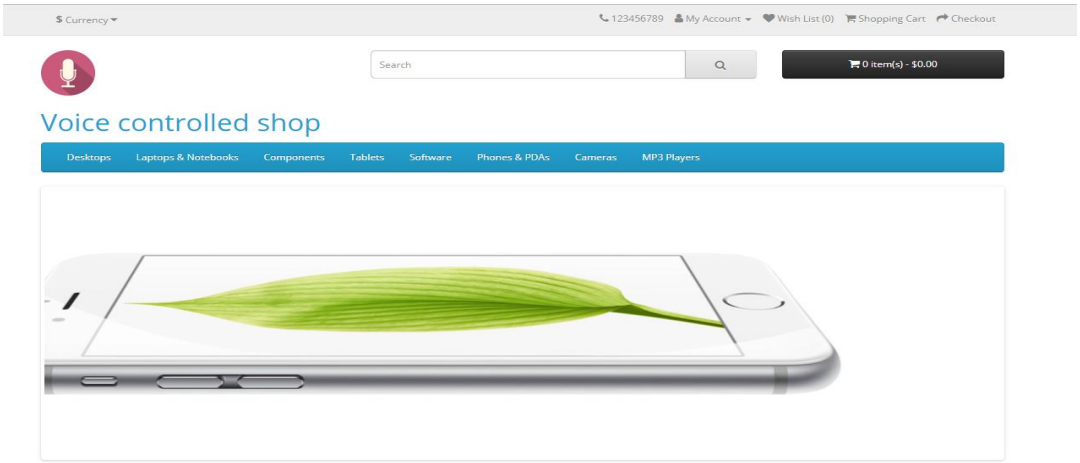


Рисунок 4.7 – Інтеграція компоненту з веб-сторінкою

Для використання голосового управління необхідно дозволити сторінці використовувати мікрофон.. На рисунку 4.8 можна побачити діалогове вікно, що дозволяє контролювати використання мікрофону у браузері.

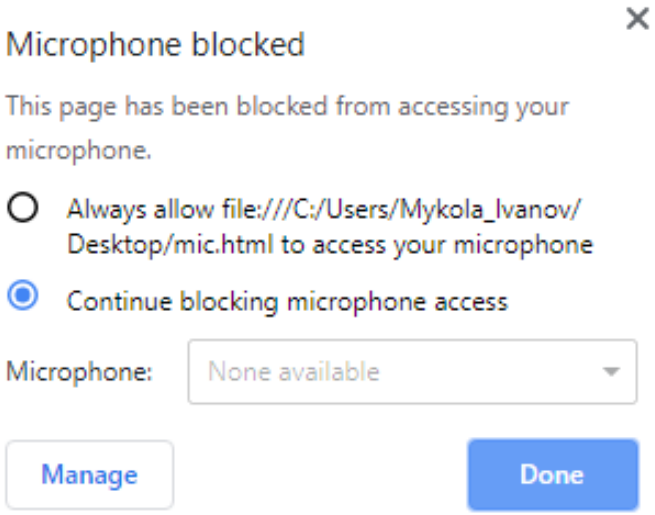


Рисунок 4.8 – Конфігурація мікрофону у браузері

Однією з ключових можливостей користувача є навігація по сайту. Після того, як користувач говорить відповідне речення, наприклад “go to cart”, система

розпізнає команду навігації та віддає дані про команду до веб сайту. У свою чергу, браузер перенаправляє користувача на відповідну сторінку. На рисунку 4.9 зображено стан браузера після запиту переходу до корзини з товарами.

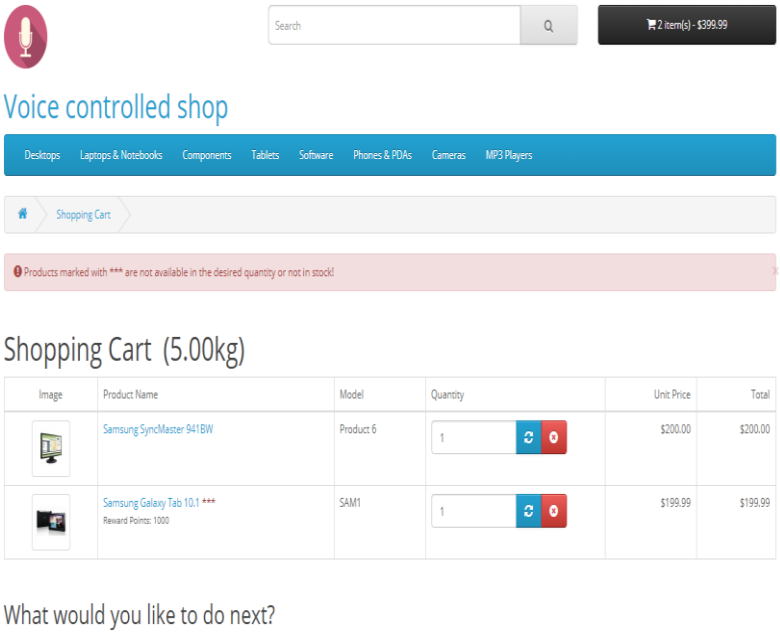


Рисунок 4.9 – Результат виконання команди навігації

Важливим моментом є здатність сервісу розпізнавати певні сутності з речення користувача. Наприклад, у разі, якщо користувач виконує запит пошуку, необхідно зрозуміти, що саме хотів шукати користувач. Сервіс здатний виділити аргумент пошуку та передати його разом з контекстом команди. Прикладом такої команди є фраза “Can you search me an Iphone?”. Таким чином, системи, що містять функціонал пошуку, надають можливість користувачу пошуку за голосовими командами.

Пошук – не єдиний варіант використання голосового управління з розпізнаванням сутностей. Доволі значним сценарієм використання є заповнення форм. Існує доволі багато ситуацій, коли користувачу необхідно ввести багато даних для продовження користування функціоналом системи. Є можливість налаштувати заповнення форм на основі голосового вводу користувача. На рисунку 4.10 зображено стан веб сайту після відправлення команди на пошук продукції Samsung.

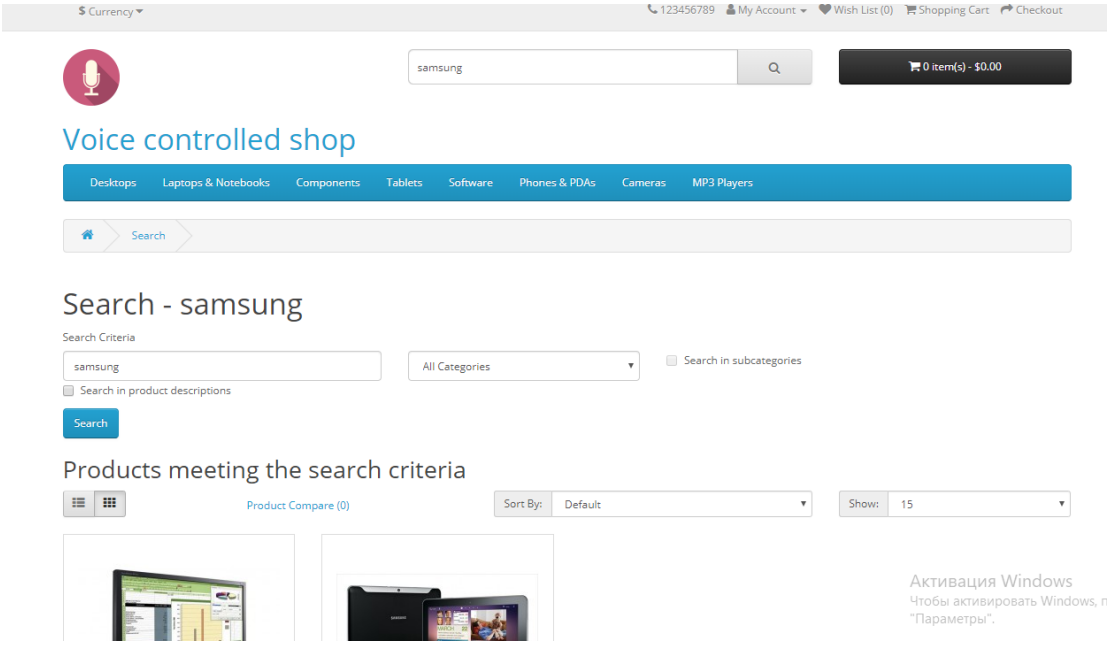


Рисунок 4.10 – Результат виконання команди пошуку

У разі, коли користувач каже речення, що схожі на речення вводу даних, вони класифікуються як команди заповнення форми. По-перше, алгоритм виконує запит до нейромережі, що класифікує тип запиту. Паралельно модуль розпізнавання сутностей вилучає параметр імені поля та введеного значення. Третя частина алгоритму за необхідністю використовує шаблон з указаними ключовими словами, які вилучать дані про запит у разі, якщо два перших крока алгоритму не змогли розпізнати намір. У цьому разі є необхідність розуміння значення, яке користувач хоче ввести, а також сполучити його з відповідним полем на формі. Для цього використовуються налаштування на стороні веб системи завдяки роботі клієнтської частині алгоритму. Саме вона забезпечує виконання одного з ключових компонентів алгоритму. На рисунку 4.11 показано вигляд системи після виконання команди голосового управління, а також зазначено журналювання, яке відображає необхідну інформацію для розробників для спрощення процесу пошуку помилки та тестування з’єднання між сервісом голосового управління, та системою, яка виконує запити розпізнавання намірів користувача.

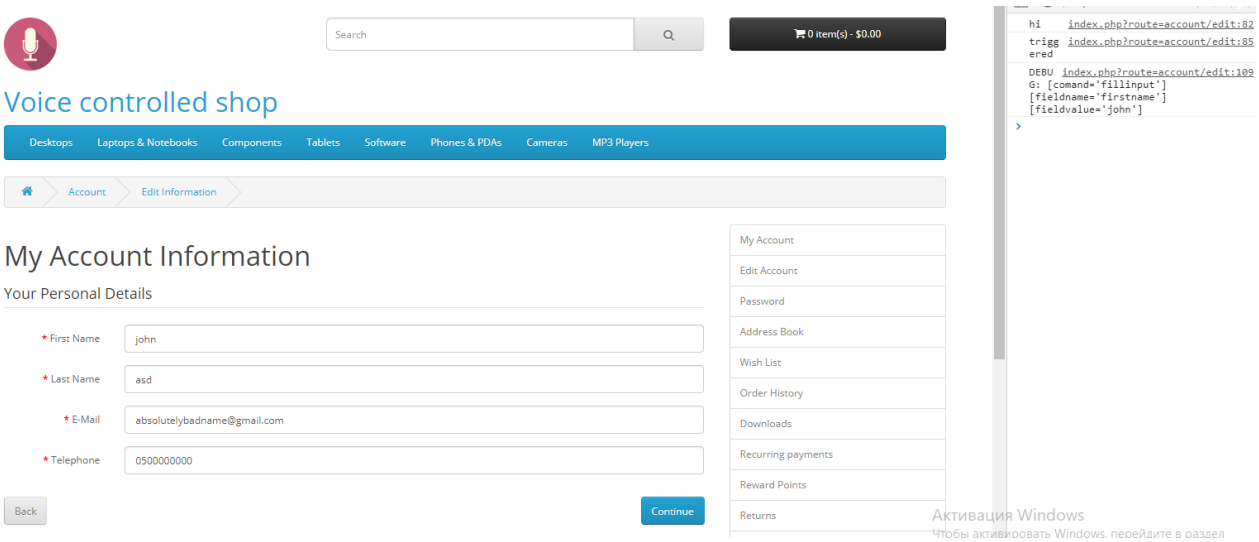


Рисунок 4.11 – Результат вводу команди заповнення даних

Таким чином, веб-системи, що інтегруються з сервісом, здатні надавати нові можливості кінцевим користувачам. Спрощення управління, наявність альтернативного способу вводу інформації надають змогу покращити досвід користування додатками. Інтеграція з інтерфейсами браузерів надає змогу швидко ввімкнути мікрофон та почати розпізнавання голосу. Також, користувач може обрати бажаний мікрофон або заборонити його використання, користуючись можливостями, які надають сучасні браузери. Можливість інтегрувати компонент через асинхронні запити дозволяє уникати перезавантаження сторінки, що позитивно впливає на зручність використання системи. Для виконання нового запиту не обов’язково повністю оновлювати сторінку, адже відповідь сервісу дозволяє забезпечити зміни тільки у тих компонентах, які цього потребують, пропонуючи системі, що інтегрується з сервісом, прийняти рішення самостійно. Додатки здатні створювати відповідності між своїми формами та командами, що контролюють можливість голосового вводу даних у форми або у вікно пошуку.

# 5 ТЕСТУВАННЯ СЕРВІСУ ГОЛОСОВОГО УПРАВЛІННЯ

Тестування програмного забезпечення проводиться з метою перевірки якості створеної системи, відповідності до вимог та готовності до використання кінцевим користувачем.

Модульне тестування є одним з ефективних засобів перевірки програми на ранньому етапі. До переваг цього підходу слід віднести швидкість написання та запуску тестів, адже вони не потребують повної ініціалізації системи. Також вони дозволяють ефективно використовувати методологію розробки через тестування. Завдяки цьому підходу покращується якість покриття позитивних та негативних сценаріїв, адже специфікація тесту створюється перед реалізацією.

Модульне тестування здатне знаходити помилки на ранньому етапі життєвого циклу функціоналу, таким чином зменшуючи затрати на вирішення проблеми.

Основною особливістю такого тестування є використання mock об’єктів, що імітують поведінку інших модулів при взаємодії з компонентом тестування. Саме це дозволяє тестам бути швидкими та не потребувати ресурси, які не входять у даний тестовий план.

Незважаючи на ефективність та швидкість, даний тип тестування не здатний покрити комплексні сценарії, які перевіряють реальну взаємодію об’єктів. У таких випадках створюються інтеграційні тести. Головною ознакою цих тестів є прогін функціоналу при системі з конкретним набором даних. Це дозволяє перевірити, як тестовий функціонал буде взаємодіяти з реальними залежностями та сервісами.

Інтеграційне тестування також дозволяє демонструвати поведінку системи при контакті з базою даних. Таким чином, для функціоналу, що приводить до виконання запитів у базу даних, даний рівень покриття є життєво необхідним, адже демонструє здатність обробляти персистентні об’єкти.

Інтеграційне тестування використовується на сервісному прошарку системи. Але з таким підходом дуже важко протестувати view частину застосунку. Тому для покриття повного циклу функції використовують інші підходи.

Одним із засобів, що перевіряють функціонал, враховуючи вигляд для кінцевого користувача є автоматичні end-to-end тести. Вони здатні імітувати кінцевого користувача та його браузер через спеціальні драйвери, таким чином проходячи шлях користувача з перспективи взаємодії через клієнт та перевіряють функціонал через зміни у контрольованому браузері. При використанні автотестів важливо дотримуватися BDD підходу. Даний інструмент є одним з видів TDD, але крім зосередження на контракті, додає інші можливості для опису специфікації.

Основним аспектом BDD є наявність контекстної мови та доменної моделі. У поєднанні вони здатні створювати опис тесту, який буде зрозумілий різним учасникам розробки програмного забезпечення. Зазвичай така мова дуже схожа на англійську та оперує доменними поняттями на основі словників з документації. Таким чином, опис тесту одночасно зрозумілий інженеру, що його розробив, іншим розробникам його команди та навіть бізнес користувачам, що можуть перевіряти сценарії на відповідність вимогам.

Крім того, було використано мануальне тестування. Для покриття системи автоматичними сценаріями необхідно багато часу, також, для перевірки комплексного функціоналу дані тести можуть бути занадто складними та неефективними. При мануальному сценарії складається тестовий план, що складається з різноманітних видів тестових сценаріїв. Основними є позитивне (happypath) тестування, тестування edge-case сценаріїв та негативне тестування. Комбінування даних сценаріїв та створення стратегії мануального тестування дозволяють переконатися у коректності роботи системи на end-to-end рівні.

# ВИСНОВКИ

У результаті виконання роботи було досліджено засоби NLP для розпізнавання природної мови та розроблено прототип сервісу, що здатний використовувати засоби обробки природної мови для розпізнавання намірів користувача. В основу сервісу покладено сервісно-орієнтовану архітектуру, основними частинами якої є клієнтський додаток, сервер обробки команд та натренована модель. Сервіс працює з голосовим вводом, що зроблено на англійській мові.

Для дослідження засобів природної мови використовувалась мова Python, фреймворк Keras та середовище Google Colab. Сервісна частина була написана на мові Java. У якості фреймворків для серверної частини було обрано Spring MVC, Hibernate. Клієнтська частина використовує бібліотеку JQuery. У якості бази даних використовується MySQL. У якості серверу використовується Apache Tomcat.

В результаті виконання роботи було проведено дослідження використання RNN мереж для виконання задачі класифікації намірів користувача; створено додаток до клієнту, що підтримує різноманітні браузери користувачів та інтегрується з серверною частиною; створено серверну частину, що здатна оброблювати та генерувати команди.

У результаті роботи поставлена задача була виконана. Сервіс має швидку інтеграцію, можливість обробляти різноманітні типи команд та перетворювати голосове повідомлення на текстове за допомогою клієнтського інструменту.

Після виконання усіх поставлених цілей, сервіс має бути дороблено перед початком використовування його у комерційних цілях. Ключовими доробками будуть виступати створення механізму оплати за допомогою інтернету, збільшення бази шаблонів речень, за якими визначаються команди, локалізація голосового вводу.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Abadi M. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems [Електронний ресурс] / M. Abadi, A. Agarwal, P. Barham // Google Research. – 2016. – Режим доступу: https://arxiv.org/abs/1603.04467/ – 20.04.2019. – Загол. з екрану.

2. PostgreSQLdocs [Електронний ресурс] / Postgresql – Режим доступу: www/URL:http://docs.postgresql.org/manual/ – 20.04.2019. – Загол. з екрану.

3. Мартин, Р. Чистый код. Создание, анализ и рефакторинг [Текст] : пер. с англ. Е.: Матвеев/ Р. Мартин. – Питер, 2010. – 464 с.

4. Еккель, Б. Философия Java [Текст] : пер. с англ. Е.: Матвеев/ Б. Еккель. – Спб: Питер, 2001.– 880с.

5. Смирнов, Н. И. JAVA 2 Enterprise. Основы практической разработки распределенных корпоративных приложений[Текст]/ Н.И. Смирнов. – М.: КУДИЦ–ОБРАЗ, 2002. – 240 с.

6. Кржиштоф, Ц.Инфраструктура программных проектов: соглашения, идиомы и шаблоны для многократно используемых библиотек[Текст] : пер. с англ. – М.: ООО “И.Д. Вильямс”, 2011. – 416 с.

7. Нотон, П. JAVA: Справочное руководство [Текст] : пер. с англ. – М.: БИНОМ, 1996. – 447с.

8. Кузнецов, С. Д. Основы баз данных [Текст] / С. Д. Кузнецов. – М.: Интернет–Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. – 484 с.

9. Тителл Е. HTML, XHTML и CSS для чайников [Текст] / Е. Тителл, Дж. Ноубл. – М. : Диалектика, 2011. – 400 с.

10. Фаулер, М. UML. Основы [Текст] : пер. с англ. А.: Петухов/ М. Фаулер,К. Скотт. – СПб.: Символ, 2006. – 184 с.