НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.4'24

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____ 2020 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою
«Інженерія програмного забезпечення комп'ютерних
та інформаційно-пошукових систем»

зі спеціальності 121 Інженерія програмного забезпечення
на тему: «Алгоритмічно-програмний комбінований метод генерації коду шаблонів веб-застосунків»

Виконала:
студентка ІІ курсу, групи КП-91мп
Довганюк Ліна Олегівна _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай Микола Володимирович

_____

Рецензент:
Доцент кафедри ММСА ІПСА, к.т.н., доцент
Марина Віталіївна Дідковська _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студентка _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ Іван ДИЧКА
(підпис)
«__»_____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студентці
Довганюк Ліні Олегівні

1. Тема дисертації «Алгоритмічно-програмний комбінований метод генерації коду шаблонів веб-застосунків» науковий керівник дисертації Заболотня Тетяна Миколаївна, к.т.н., доцент, затверджена наказом по університету від «12» листопада 2020 р. №3298-С
2. Термін подання студентом дисертації «14» грудня 2020 р.
3. Об'єкт дослідження: процес генерування коду шаблонів компонентів.
4. Предмет дослідження: методи і алгоритми генерування програмного коду.
5. Перелік завдань, які потрібно розробити:
   - провести аналіз методів та алгоритмів генерування програмного коду;
   - дослідити існуючі інструменти для генерування програмного коду;
   - проаналізувати можливість генерування коду шаблонів компонентів веб-застосунків;
   - розробити алгоритмічно-програмний комбінований метод генерації коду шаблонів компонентів веб-застосунків;
   - обрати критерії оцінки методу та провести тестування роботи методу;
   - порівняти ефективність запропонованого методу з іншими;
   - розробити систему, яка реалізує запропонований метод генерування коду шаблонів компонентів веб-застосунків;
   - провести аналіз отриманих результатів.
6. Перелік ілюстративного матеріалу:
   - зображення роботи системи на основі моделі від дерева до дерева;
   - узагальнена схема методу;
   - графічне представлення послідовності виконання кроків методу;
   - схема архітектури системи;
   - дерево проблем та рішень;
   - графічне представлення перекладу програми у JavaScript.

7. Перелік публікацій:
   - Тези доповіді «Узагальнений метод генерування коду шаблонів програмних компонентів»;
   - Тези доповіді «Узагальнений підхід до генерування коду шаблонів програмних компонентів»

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Онай М.В., доцент кафедри ПЗКС		

9. Дата видачі завдання «11» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю.	09.12.2019	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук.	30.01.2019	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження.	11.03.2020	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення; підготовка матеріалів доповіді ПМК-2020.	08.05.2020	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження.	14.08.2020	
6.	Проведення наукового дослідження; робота над третім розділом магістерської дисертації.	19.09.2020	
7.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу.	09.11.2020	
8.	Оформлення текстової і графічної частини магістерської дисертації.	08.12.2019	

Науковий керівник _____ Тетяна ЗАБОЛОТНЯ

Студент _____ Ліна ДОВГАНЮК

РЕФЕРАТ

Актуальність теми. На сьогоднішній день постійне зростання обсягів розроблюваного програмного забезпечення та попиту на створення нових програм є причиною того, що підвищується рівень вимог до швидкості написання коду та якості отримуваних застосунків.

Для того, щоб код програми було легко читати та модифікувати, від розробників вимагають дотримання у роботі принципу модульності, винесення основної функціональності застосунку в окремі утиліти, використання об’єктно-орієнтованого програмування і т.д.

Як відомо, для підвищення швидкості написання коду у переважній більшості випадків програмісти вдаються до пошуку готових рішень та подальшої модифікації знайденого фрагменту програмного коду з точки зору наявної задачі. Але такий підхід з копіюванням чужого коду містить потенційний ризик того, що в додатку залишаться зайві дії, що не мають відношення до розв’язуваної ним задачі.

Також відомо, що задля пришвидшення створення програмного продукту розробники або тестувальники досить часто використовують генерування структури проєкту із заготовками коду деяких класів, методів та конфігураційних файлів відповідно до обраного типу проєкту. Звісно, така допомога у написанні коду зменшує час на створення програмного забезпечення та дозволяє розробникам зосереджуватися на більш творчих задачах. Але крім того хотілося б мати можливість автоматизації створення шаблонів окремих програмних компонентів (наприклад, кастомізованих React-компонентів для веб-застосунків), а також мати можливість більш тонкого налаштування генератора заготовок коду: щоб брати до уваги не тільки тип проєкту, а і деякі вхідні дані, складені формалізованою природною мовою.

Аналіз існуючих середовищ розроблення програмного забезпечення показав, що на даний момент вони не надають такої функціональності, а, отже, розроблення відповідного методу генерування коду програмних

компонентів та його наступна програмна реалізація є актуальною задачею.

В даній магістерській дисертації описано метод генерації коду, а саме коду шаблонів React-компонентів веб-застосунків.

Об'єктом дослідження є процес генерування програмного коду шаблонів компонентів веб-застосунків.

Предметом дослідження є методи і алгоритми генерації програмного коду.

Метою дослідження є підвищення ефективності автоматизованого створення програмного коду складових веб-застосунків за рахунок розроблення та реалізації методу генерування коду шаблонів веб-компонентів та відповідного програмного забезпечення.

Методи дослідження: в роботі використовуються методи синтаксичного аналізу текстових даних, методи машинного навчання із використанням моделі «від дерева до дерева».

Наукова новизна роботи полягає в наступному:

Вперше запропоновано метод організації процедури автоматизованого створення програмного коду складових веб-застосунків, який полягає у синтаксичному аналізі текстового опису майбутнього компонента та поданні результату аналізу до моделі «tree2tree» з батьківським механізмом уваги. Це дозволяє підвищити ефективність та швидкість написання програмного коду за рахунок зменшення кількості символів для введення.

Практичне значення отриманих результатів визначається тим, що запропонований метод генерування програмного коду складових веб-застосунків дозволив підвищити ефективність та швидкість написання коду та розробити відповідне програмне забезпечення, придатне до використання у вигляді бібліотеки.

Апробація роботи. Основні положення і результати роботи доповідалися та обговорювалися на XIII науковій конференції магістрантів та аспірантів "Прикладна математика та комп'ютинг" ПМК-2020 та IX

Міжнародній науково-практичній конференції «Проблеми інформатики та комп’ютерної техніки» ПІКТ-2020.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п’яти розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень.

У першому розділі проаналізовано існуючі інструменти для генерування програмного коду, а також розглянуто на проаналізовано методи генерування програмного коду.

У другому розділі описано алгоритмічно-комбінований метод генерації коду шаблонів компонентів веб-застосунків. Запропонований метод полягає у використанні нейронної мережі у поєднанні з моделлю «від дерева до дерева» та батьківською увагою для кожного вузла дерева.

У третьому розділі обґрунтовано вибір технологій для розробки системи для генерування коду шаблонів компонентів веб-застосунків, а саме React-компонентів. Описано загальну архітектуру системи та розглянуто модулі програмного забезпечення.

У четвертому розділі було проведено тестування розробленого методу, проаналізовано ефективність роботи методу в залежності від типу генерованого компоненту.

У п’ятому розділі проаналізовано сферу інформаційних технологій, визначені основі користувачі системи та їх основні проблеми. За виконаним аналізом побудовано бізнес-модель кінцевого продукту та розраховані фінансові показники системи.

У висновках проаналізовано отримані результати дослідження.

Робота виконана на 81 аркуші, містить 3 додатки та посилання на список використаних літературних джерел з 30 найменувань. У роботі наведено 38 рисунки та 6 таблиць.

Ключові слова: генерування коду, машинне навчання, батьківська увага у моделі «від дерева до дерева», React-компоненти.

ABSTRACT

Actuality of theme. Today, the constant growth of software development and demand for new programs is the reason that the level of requirements for the speed of writing code and the quality of applications.

In order for the program code to be easy to read and modify, developers are required to adhere to the principle of modularity, to make the main functionality of the application into separate utilities, to use object-oriented programming, etc.

As you know, to increase the speed of writing code in the vast majority of cases, programmers resort to finding ready-made solutions and further modification of the found fragment of program code in terms of the existing task. But this approach of copying someone else's code carries the potential risk that the application will have unnecessary actions that are not relevant to the task it is solving.

It is also known that in order to speed up the creation of a software product, developers or testers often use the generation of project structure with code blanks of some classes, methods and configuration files according to the selected project type. Of course, this help in writing code reduces the time to create software and allows developers to focus on more creative tasks. But I would also like to be able to automate the creation of templates for individual software components (for example, customized React-components for web applications), as well as to be able to fine-tune the code generator: to take into account not only the project type but also some input data compiled in formalized natural language.

The analysis of the existing software development environments showed that at the moment they do not provide such functionality, and, therefore, the development of an appropriate method of generating code of software components and its subsequent software implementation is an urgent task.

This master's thesis describes a method of code generation, namely the

code of templates React-components of web applications.

Object of research is the process of generating the program code of web application component templates.

Subject of research are methods and algorithms for generating program code.

Goal of the work is to develop an effective method of generating code for web application component templates.

Methods of research: the method of data parsing, machine learning methods using the model "from tree to tree" are used in the work.

Scientific novelty of the work is as follows**:**
An algorithmic-software combined method of generating web application component template code is proposed, which is based on the following components: parsing of the description of the future component presented in a formalized format, coded syntax tree with defined parental attention factors for each tree node.

Practical value of the results obtained in the work is that the proposed method allows to increase the speed of writing a developer or tester of program code, and also gives users the opportunity to focus on more creative tasks when writing code.

Approbation. The main provisions and results of the work were reported and discussed at the XIII scientific conference of undergraduates and graduate students "Applied Mathematics and Computing" PMK-2020 and IX International scientific-practical conference "Problems of Informatics and Computer Engineering" PICT-2020.

Structure and scope of work. The master's thesis consists of an introduction, five chapters and conclusions.

In the introduction the general characteristic of work is given, the estimation of a modern condition of a problem is made, the urgency of a direction of researches is proved.

The first section analyzes the existing tools for generating program code,

as well as analyzes the methods of generating program code.

The second section describes an algorithmic-combined method for generating code for web application component templates. The proposed method is to use a neural network in combination with a "tree to tree" model and parental attention for each node of the tree.

The third section substantiates the choice of technologies for the development of a system for generating code for templates of web application components, namely React-components. The general architecture of the system is described and the software modules are considered.

In the fourth section, the developed method was tested, the efficiency of the method depending on the type of generated component was analyzed.

The fifth section analyzes the field of information technology, identifies the basics of system users and their main problems. According to the analysis, the business model of the final product is built and the financial indicators of the system are calculated.

The conclusion contains brief summary of study results.

The work is performed on 81 sheets, contains 3 appendices and links to a list of used literature sources from 30 titles. The paper presents 38 figures and 3 tables.

Keywords: code generation, machine learning, parental attention in the model "tree2tree", React-components.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Веб-застосунок – розподілений застосунок, в якому клієнтом виступає браузер, а сервером — веб-сервер.

Автоматичне програмування – це поняття, згідно з якими програми можуть бути написані для автоматичного створення програмних компонентів.

XSLT – Extensible Stylesheet Language Transformations – сформований XML-документ, який містить набір правил перетворення і який використовується як шаблон для перетворення.

XML – eXtensible Markup Language – стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками.

UML – Unified Modeling Language – уніфікована мова моделювання для визначення, візуалізації, проєктування й документування програмних систем.

T4 – Text Template Transformation Toolkit – шаблонно орієнтований генератор коду від компанії Microsoft.

API – Application programming interface – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

SQL – Structured Query Language – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення і керування реляційними БД, створення схеми бази даних та її модифікації, системи контролю за доступом до бази даних.

JSON – JavaScript Object Notation – це текстовий формат обміну даними між комп'ютерами.

ADO – ActiveX Data Objects – прикладний програмний інтерфейс для доступу до даних.

MVC – Model-View-Controller – архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.

CSV – comma-separated values – файловий формат для представлення табличних даних, у якому поля відокремлюються символом коми та переходу на новий рядок.

IR – intermediate representation – це структура даних або код, що використовуються внутрішньо компілятором або віртуальною машиною для представлення вихідного коду.

HTML – Hypertext Markup Language – це мова тегів, засобами якої здійсюється розмічання веб-сторінок для мережі Інтернет.

ООП – об’єктно-орієнтоване програмування – одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою.

ПЗ – програмне забезпечення – сукупність програм системи обробки інформації і програмних документів, необхідних для експлуатації цих програм.

WPF – Windows Presentation Foundation – високорівневий об'єктно-орієнтований функціональний шар, що дозволяє створювати двовимірні та тривимірні інтерфейси.

RNN – recurrent neural network – вид нейронних мереж, де зв'язки між елементами утворюють спрямовану послідовність.

LSTM – Long short-term memory – це архітектура рекурентних нейронних мереж.

TreeLSTM – tree-structured LSTM – це узагальнення мереж довгострокової короткочасної пам'яті (LSTM) на деревоподібні топології мережі.

React-компонент – об’єкти, які зберігають стан та повертають React-елементи, які відображаються на веб-сторінці.

ВСТУП

Постійне зростання обсягів розроблюваного програмного забезпечення та попиту на створення нових програм є причиною того, що підвищується рівень вимог до швидкості написання коду та якості отримуваних застосунків.

Для того, щоб код програми було легко читати та модифікувати, від розробників вимагають дотримання у роботі принципу модульності, винесення основної функціональності застосунку в окремі утиліти, використання об’єктно-орієнтованого програмування і т.д.

Для підвищення швидкості написання коду у переважній більшості випадків програмісти вдаються до пошуку готових рішень та подальшої модифікації знайденого фрагменту програмного коду з точки зору наявної задачі. Але такий підхід з копіюванням чужого коду містить потенційний ризик того, що в додатку залишаться зайві дії, що не мають відношення до розв’язуваної ним задачі.

Іншим використовуваним підходом до пришвидшення створення програмного продукту є генерування структури проєкту із заготовками коду деяких класів, методів та конфігураційних файлів відповідно до обраного типу проєкту. Звісно, така допомога у написанні коду зменшує час на створення програмного забезпечення та дозволяє розробникам зосереджуватися на більш творчих задачах. Але окрім формування заготовок інфраструктури проєктів в цілому цінною була б автоматизація створення шаблонів окремих програмних компонентів (наприклад, кастомізованих React-компонентів для веб-застосунків). Крім того, доцільним видається більш тонке налаштування генератора заготовок коду: хотілося б брати до уваги не тільки тип проєкту, а і деякі вхідні дані, складені формалізованою природною мовою. Аналіз існуючих середовищ розроблення програмного забезпечення показав, що на даний момент вони не надають такої функціональності, а, отже, розроблення відповідного

методу генерування коду програмних компонентів та його наступна програмна реалізація є актуальною задачею.

1. АНАЛІЗ ІСНУЮЧИХ ІНСТРУМЕНТІВ ДЛЯ ГЕНЕРУВАННЯ ПРОГРАМНОГО КОДУ

1.1. Поняття генеративного програмування

В інформатиці термін автоматичне програмування [1] визначає тип комп’ютерного програмування, при якому якийсь механізм генерує комп’ютерну програму, що дозволяє програмістам писати код на вищому рівні абстракції.

Існує декілька варіантів точного визначення автоматичного програмування, здебільшого тому, що його значення змінювалось з часом. Девід Парнас, простежуючи історію "автоматичного програмування" в опублікованих дослідженнях, зазначив, що в 1940-х роках в ній описувалася автоматизація ручного процесу штампування паперової стрічки. Пізніше мова йшла про переклад мов програмування високого рівня, таких як Fortran та ALGOL. Парнас дійшов висновку, що "автоматичне програмування завжди було скоріше евфемізмом для програмування мовами вищого рівня" [2].

У той же час Мілдред Косс, раніше відомий як програміст UNIVAC, писав наступне: "Написання машинного коду включало кілька нудних етапів – розбиття процесу на дискретні інструкції, присвоєння певного розташування пам'яті всім командам та управління буферами вводу-виводу. Після виконання цих кроків для реалізації математичних підпрограм, бібліотек підпрограм і програм сортування, нашим завданням було розглянути сам процес програмування. Нам потрібно було зрозуміти, як ми можемо повторно використовувати перевірений код і мати машинну допомогу в програмуванні. Під час програмування ми вивчали процес і намагався придумати способи абстрагувати ці кроки, щоб включити їх у мову вищого рівня. Це призвело до розробки інтерпретаторів, асемблерів, компіляторів і генераторів – програм, призначених для роботи з іншими програмами, тобто автоматичного програмування." [3].

Одним з прикладів автоматичного програмування є генеративне програмування або як його можна ще називати – перетворення програм. Генеративне програмування та відповідний термін метапрограмування [4] – це поняття, згідно з якими програми можуть бути написані "для автоматичного створення програмних компонентів" [5]. Тобто він є аналогією автоматизації виробництва традиційних товарів, таких як швейні вироби, автомобілі, хімікати, та електроніки, але вже в контексті створення програмного забезпечення [6].

Метою такого підходу є підвищення продуктивності розробників [7]. Іншими словами – заміна ручного пошуку, адаптації та створення компонентів на автоматичну генерацію необхідних компонентів за вимогами. Також для підвищення продуктивності, якості та часу виходу на ринок при розробленні програмного забезпечення завдяки створенню як стандартних комплектуючих, так і автоматизації виробництва в загальному. Одне з важливих змін парадигми, яке тут мається на увазі, полягає в тому, щоб будувати програмні системи із стандартних компонентів, а не "винаходити колесо" щоразу.

Генеративне програмування означає для розробника додатків те, що він може абстрактно викласти, що хоче отримати в результаті, а генератор створює потрібну систему або компонент. Це працює лише за умови, якщо:

1. Формалізований опис компонента відповідає необхідній структурі.
2. Певні конфігурації визначають, як перевести абстрактні вимоги в конкретні компоненти.
3. Впровадити знання про конфігурацію за допомогою генераторів.

1.2. Аналіз існуючих інструментів генерування коду

Генерація коду дозволяє розробнику зосередитися на високому рівні абстракції під час розробки. Він перетворює код високого рівня, написаний

людиною, у мову низького рівня. Іншими словами, він генерує вихідний код на основі опису або моделі проєкту. Це дозволяє економити час під час розробки додатків та захищає код від людських помилок, які неминучі навіть для найбільш досвідчених розробників.

Генерування коду під час налаштування збірки дозволяє використовувати результати генерації під час компіляції. Найпопулярнішим інструментом для генерації коду є компілятор. Тобто можна вільно використовувати будь-який компілятор, який вважається ефективним для поставленого завдання. Однак існують спеціальні інструменти для перетворення коду високого рівня в код низького рівня. Ось деякі найкращі інструменти генерації коду під час проєктування:

- розширювані шаблони мовної таблиці стилів (XSLT);
- інструменти на основі UML;
- Razor Generator;
- Metadrone;
- Reegenerator;
- шаблони інструментарію для трансформації текстових шаблонів (Text Template Transformation Toolkit або T4);
- Radzen;
- генератор CodeSmith;
- ASP.Net Zero.

Всі ці інструменти корисні, коли потрібно створити простий повторюваний код або будь-який текст відповідно до шаблону.

Відомо, що вибір засобу генерації коду залежить від:

- використовуваної мови;
- типу програми або компонента, які потрібно створити;
- середовища розробки, що використовується.

Далі пропонується детальніше розглянути різні інструменти, які можна використовувати для генерації коду.

1.2.1. Шаблони трансформації XSL

eXtensible Stylesheet Language (XSL) – мова стилів для XML-документів. Розширювана мова перекладу таблиць стилів (XSLT) є частиною цієї мови, яка відповідає за перетворення файлів XML в інші формати. Це рішення для створення шаблону є частиною стандарту W3C XSL. Ключовою перевагою XSLT у порівнянні з іншими подібними інструментами є його гнучкість [8].

XSLT призначений для перетворення ієрархічної структури XML-документа в HTML, PDF, текст, вихідний код тощо. Мова XSLT є потужним інструментом для обробки даних та інформації в ієрархічній структурі.

XSLT оптимізований для створення правил перетворення. Він складається з набору правил, позначених тегами шаблону. Правило складається із статичного тексту та ряду тегів, що нагадує конструкції логічних мов: значення відображення, цикл, умова тощо.

На відміну від класичних мов програмування, XSLT описує перетворення не як набір дій, а як набір правил, що застосовуються до вузлів вхідного XML. Кожне правило містить логічну функцію (предикат). Визначити, чи функція відповідає поточному вузлу, можна за допомогою обчислення цієї функції.

У XSLT такі функції описуються мовою XPath. Якщо предикат є істинним, правило виконується.

Модель XSLT включає:

1. XML-документ – вхідні дані для перетворення в інші типи документів.
2. Таблиця стилів XSLT – сформований XML-документ, який містить набір правил перетворення і який використовується як шаблон для перетворення.
3. Процесор XSLT – додаток, який отримує XML-документи та стилі XSLT як вхідні дані та виконує перетворення, застосовуючи правила зі стилів XSLT до XML-документів.

4. Вихідний документ – результат перетворення.

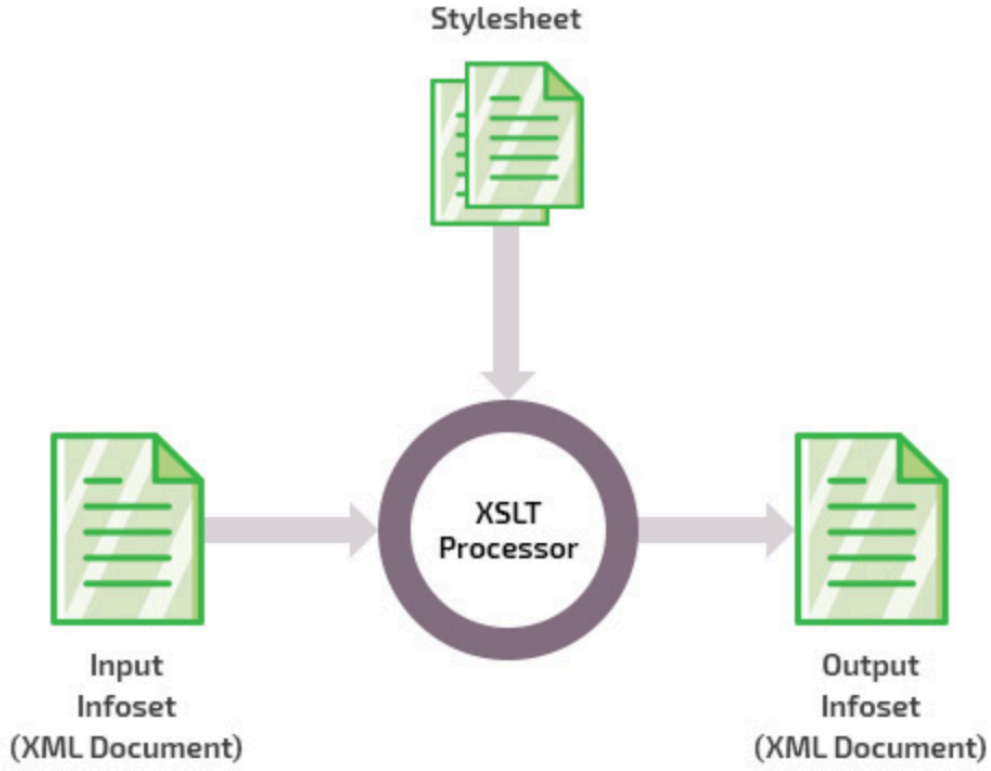


Рис. 1.1. Графічне представлення моделі XSLT

Таким чином метод із використанням XSLT виконує наступні дії:

1. XSLT Processor обробляє таблицю стилів XSLT і застосовує правила перетворення до цільового XML-документу.
2. XSLT Processor генерує відформатований документ у форматі XML, HTML або текстовому форматі.
3. Засіб форматування XSLT генерує фактичне виведення, яке подається на вихід користувачеві.

1.2.2. Інструменти на основі UML

Перетворити код на цільову мову можна за допомогою інструментів на основі моделей уніфікованої мови моделювання (UML) [9]. Вони генерують вихідний код певною мовою з класів UML і дозволяють моделі UML відображати будь-які зміни у вихідному коді.

Двостороння інтеграція допомагає синхронізувати вихідний код та UML. Це означає, що кожного разу, коли ви створюєте фрагмент коду або оновлюєте модель UML, ці зміни об’єднуються.

Генерація коду за допомогою моделей UML вбудована в Microsoft Visual Studio. Він генерує код, написаний на C#, із діаграм класів UML. Це дозволяє зосередитись на бізнес-логіці та архітектурі проєкту, замість того, щоб писати інфраструктурний код низького рівня. Це також допомагає уникнути помилок у коді, які неминуче трапляються під час ручного кодування, і займає багато часу для налагодження.

Для того, щоб генерувати код C# із діаграм класів UML, використовуйте команду «Створити код». За замовчуванням він створює код C# для кожної вибраної діаграми UML. Ви можете змінити або масштабувати цю поведінку, редагуючи або копіюючи текстові шаблони, що генерують код. Ви також можете вибрати будь-яку іншу поведінку для типів, що входять до різних пакетів моделей.

Існують також незалежні інструменти для генерації коду за допомогою моделей UML: UModel, Visual Paradigm, Modeliosoft, Enterprise Architect, erwin Data Modeler тощо. Ці інструменти підтримують генерацію коду на різних мовах програмування, включаючи Java, C++ та Python. Більшість із цих модельних середовищ можна інтегрувати з такими середовищами розробки, як Eclipse, NetBeans, IntelliJ IDEA, Visual Studio та Android Studio.

1.2.3. Razor Generator

Razor Generator (слід не плутати з механізмом Razor) – це інструмент із відкритим кодом, написаний на C#, який підтримує Visual Studio 2019 [10]. Він дозволяє розробнику обробляти файли Razor під час проєктування, а не під час виконання. Завдяки цьому можна інтегрувати файли Razor у свою збірку, щоб полегшити їх повторне використання та розповсюдження.

Цей інструмент надає розробнику більше часу для запуску та дозволяє тестувати подання Razor.

1.2.4. Metadrone

Metadrone – це безкоштовний інструмент, який використовує простий синтаксис шаблону для виведення тексту на основі схеми бази даних [11]. Це дозволяє розробнику зменшити ручне кодування інтерфейсів екранів, зіставлення ORM у фреймворках, збережених процедурах, класах API тощо. Metadrone підтримує бази даних SQL Server, MySQL, Oracle та PostgreSQL.

1.2.5. Reegenerator

Reegenerator – це безкоштовний інструмент генерації коду, інтегрований у Microsoft Visual Studio [12]. Він може використовувати будь-який тип файлу як вхідні дані та генерувати будь-який тип файлу як вихідний файл, наприклад:

- файли даних: XML, JSON;
- файли коду: CS, .VB;
- бази даних.

Reegenerator використовує кілька генераторів в одному файлі. Генератори – це звичайні класи C# / VB.NET у звичайній бібліотеці класів .NET.

1.2.6. Text Template Transformation Toolkit або T4

Набір інструментів для перетворення текстових шаблонів (T4) – це генератор коду, вбудований у Microsoft Visual Studio з 2008 року [13]. Текстовий шаблон – це елемент T4, який генерує вихідні дані у проєкті Visual Studio для кожної збірки. Логіка шаблону може бути записана на C# або VB.NET. Трансформація шаблонів T4 під час проєктування відбувається під час компіляції. Ці шаблони зазвичай використовуються для генерації коду під час компіляції поточного проєкту. Після генерації T4 надає користувачеві шаблони. Ці шаблони можуть бути використані повторно шляхом успадкування або включення.

Шаблони тексту складаються з:

1. Директиви – Елементи, що керують обробкою шаблонів.
2. Текстові блоки – вміст, який потрібно скопіювати у вихідний файл.
3. Блоки управління – програмний код, який вставляє значення змінних у текст та управляє умовними та повторюваними частинами тексту.

На рис. 1.2. представлено приклад роботи T4, де кроки 1 і 2 показують перетворення команди, а крок 3 – кінцевий результат програми.

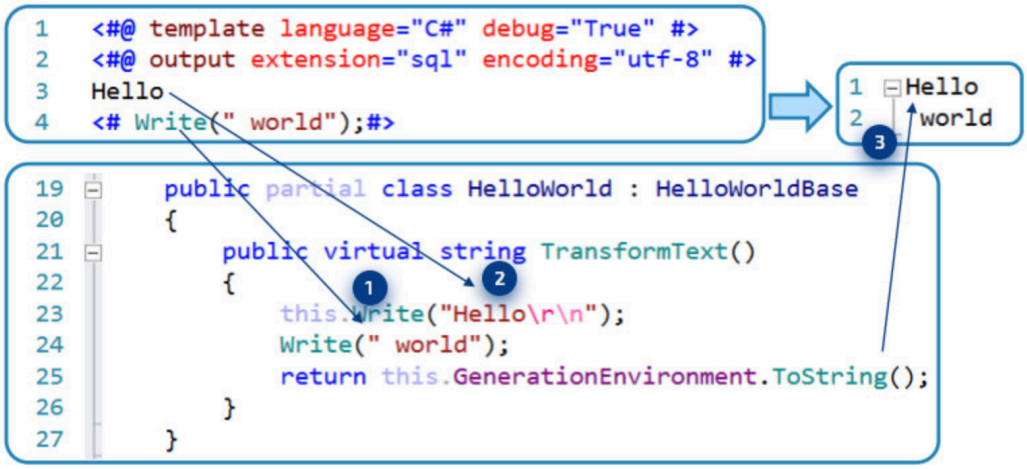


Рис. 1.2. Приклад роботи T4

1.2.7. Radzen

Radzen – це інструмент для розробки веб-додатків, який генерує код для Angular framework [14]. Ви можете працювати з файлами Radzen у Visual Studio Code. Цей інструмент особливо корисний для невеликих проєктів через обмежену функціональність.

Бекенд Radzen написаний на C#. Коли додається джерело даних з Microsoft SQL Server, MySQL або PostgreSQL, Radzen створює серверну програму ASP.NET Core. Її сторінки представлені за зразком дизайну Model – View – Controller. Також можна редагувати такі сторінки, використовуючи часткові класи та методи C#.

Radzen підтримує кілька баз даних, таких як:

- MS SQL Server;
- MySQL;
- PostgreSQL;
- OData;
- Swagger.

1.2.8. Генератор CodeSmith

Генератор CodeSmith є частиною інструментів CodeSmith. Це генератор вихідного коду на основі шаблонів. Він автоматизує генерацію коду для будь-якої мови [15]. Генератор CodeSmith містить набір корисних шаблонів, включаючи шаблони для роботи з перевіреними архітектурами (netTiers, SLA, NHibernate, PLINQO, Entity Framework, Kinetic Framework тощо). Є можливість легко змінити шаблон за замовчуванням або створити власний. Коли генерація завершена, CodeSmith Generator надає результати.

Генератор CodeSmith підтримує мови C#, Java, VB, PHP, ASP.NET, SQL та інші. Шаблони можуть генерувати код будь-якою мовою на основі ASCII.

CodeSmith Generator взаємодіє з базами даних за допомогою SchemaExplorer. Цей інструмент надає генератору типи взаємодій із даними сервера SQL або ADO та засоби проєктування для доступу до цих типів із генератора CodeSmith. Крім того, CodeSmith Generator включає багато шаблонів баз даних.

Окрім генератора, набір інструментів CodeSmith включає:

1. Exceptionless – створює звіти про помилки, функції та журнали в режимі реального часу та працює з програмами, написаними в ASP.NET, Web API, WebForms, WPF, Console та MVC.
2. CodeSmith Frameworks – надає розробнику фреймворк PLINQO, набір шаблонів CodeSmith, які генерують скелети об’єктного реляційного картографування за допомогою відповідних шаблонів дизайну. Фреймворки PLINQO включають

розширений набір функцій для спрощення та оптимізації доступу до даних.

1.2.9. ASP.Net Zero

ASP.NET Zero – це рішення для Visual Studio. Воно базується на багатошаровій архітектурі, тоді як його структура коду базується на SOLID. Воно також оснащено інтерфейсом Metronic, який забезпечує зручний інтерфейс [16].

ASP.NET Zero забезпечує високопродуктивну та масштабовану архітектуру та попередньо створені сторінки. Цей генератор має адміністративну панель з усіма функціональними можливостями для адміністрування системи та користувачів, включаючи додавання записів журналів до бази даних. Динамічна локалізація дає змогу запустити ваш проєкт у кількох країнах, а динамічний інтерфейс дозволяє адаптувати його до різних класів користувачів.

ASP.NET Zero надає різні варіанти фреймворку:

1. Рішення ASP.NET Core 2.x та Single-Page Application (SPA) на Angular 7.x.
2. Рішення ASP.NET Core 2.x та ModalViewController (MVC) архітектура на jQuery.
3. Рішення ASP.NET MVC 5.x, веб-API ASP.NET SPA на основі AngularJS 1.x.
4. ASP.NET MVC 5.x, веб-API ASP.NET та рішення на основі jQuery.
5. Мобільний додаток Xamarin, інтегрований із серверним рішенням (лише для версій ASP.NET Core (MVC або Angular UI); підтримує iOS та Android).
6. Додаток на основі ASP.NET MVC.

Шаблони запуску ASP.NET Zero працюють із SQL Server за замовчуванням. Ви можете адаптувати їх для інших сховищ даних вручну. Крім того, якщо використовується інтегрований EntityFramework, можна

адаптувати шаблони MySQL. В офіційній документації є посібник для цього процесу. Якщо ж використовувати Entity Framework Core, можна інтегрувати його з MySQL, PostgreSQL або SQLite. Також можливо інтегрувати його з іншими типами баз даних, але немає вказівок щодо того, як це зробити.

ASP.NET Zero надає нам вихідний код після придбання ліцензії, але існує два винятки:

1. Інструмент містить багато безкоштовних бібліотек із відкритим кодом як компонентів NuGet. Вони не входять до коду, який ви отримуєте після покупки, оскільки це займе занадто багато місця і буде важко оновити ці бібліотеки.
2. У ASP.NET Zero є компонент NuGet із закритим вихідним кодом. Він захищає правила ліцензування.

Документація до кожної версії ASP.NET Zero доступна на офіційному веб-сайті. Він не є обширним порівняно з документацією для інших інструментів, а також відсутні додаткові вебінари, блоги чи відеоуроки. Але є офіційно підтримана громада. Ці функції особливо корисні для розробників, що працюють для малого та середнього бізнесу, які мають обмежений час та ресурси для створення програмного забезпечення. Вони також корисні для розробки додатків SaaS.

1.3. Методи та алгоритми генерування програмного коду

На сьогоднішній день існує достатньо велика кількість методів для генерування програмного коду. Автором пропонується розглянути деякі з них для більш точного розуміння даного процесу.

Результатом роботи першого методу є лінеаризоване проміжне подання вихідної програми, або intermediate representation (далі IR). IR складається з послідовності виразів префіксів без дужок. У такому поданні за операторами слідують їх операнди. Кожна інструкція комп'ютера

описується префіксом разом із певною "семантичною" інформацією та шаблоном – машинною мовою.

Алгоритм кодування перевіряє збіг шаблонів. Такий процес дуже подібний до синтаксичного аналізу, в якому IR-послідовність виразів префіксів переводиться у послідовність інструкцій, однак існують деякі відмінності від синтаксичного аналізу в деяких аспектах. Оскільки більшість операторів можуть отримати доступ до своїх операндів різними способами, опис цільової машинної програми, як правило, неоднозначний. Дійсно, важливим фактором генерації коду є спосіб вирішення цих неоднозначностей. По-друге, процес генерування коду у такому методі набагато складніший, ніж при синтаксичному аналізі, оскільки він вибирає серед різноманітних інструкцій або послідовностей команд на основі як синтаксичної, так і семантичної інформації.

Таким чином можна представити алгоритм, яким використовує зазначений метод, у такому вигляді:

1. Вхідні дані: функції ACTION та NEXT (представлені у вигляді матриць), опис машинної мови M та лінеаризоване проміжне подання програми P.
2. Вихідні дані: асемблерна програма для мови P машинною мовою M.
3. Метод: виконання синтаксичного розбору Shift-Reduce вхідної IR-послідовності, видаючи цільові інструкції щоразу, коли виконуються обчислення.

Іншим прикладом алгоритму генерування коду є генератор коду, який використовується для створення цільового коду для триадресних операторів. Він використовує регістри для зберігання операндів трьох операторів адреси. Розглянемо триадресне твердження $x := y + z$. Воно може мати наступну послідовність кодів (див. рис. 1.3).

```
MOV x, R0
ADD y, R0
```

Рис. 1.3. Послідовність кодів для триадресного твердження $x := y + z$

При цьому дескриптор реєстру містить вказівник на те, що зараз знаходиться в кожному реєстрі. Дескриптор адреси використовується для зберігання місця, де поточне значення імені може бути знайдене під час виконання. На початку роботи дескриптори реєстру показують, що спочатку всі регістри порожні.

Якщо розглядати алгоритм генерації коду як набір кроків, тоді його можна представити так:

1. Викликати функцію, яка дозволяє дізнатися місце *L*, де слід зберігати результат обчислення b *operarion* c .
2. Знайти y' з опису адреси для *y*. Якщо значення y в даний час знаходиться в пам'яті і зареєстровані обидва значення, тоді обрати реєстр y'. Якщо значення *y* ще не в *L*, тоді сформувати інструкцію *MOV* y', *L*, щоб помістити копію *y* в *L*.
3. Сформувати інструкцію *OPERATION* z', *L*, де z' використовується для відображення поточного розташування *z*. Якщо *z* є в обох, тоді обрати регістр з місцем у пам’яті.
4. Оновити дескриптор адреси *x*, щоб вказати, що *x* знаходиться в певному місці *L*. Якщо *x* знаходиться в *L*, то оновити його дескриптор і видалити *x* з усіх інших дескрипторів.
5. Якщо поточне значення *y* або *z* не має наступного використання, тоді змінити дескриптор реєстру, щоб вказати, що після виконання $x := y$ *operation* z ці регістри більше не будуть містити *y* або *z*.

1.4. Висновки до розділу 1

У даному розділі було розглянуто поняття генерації коду. Під час проєктування дозволяє розробникам економити час та уникати помилок при написанні коду низького рівня. Крім того, дозволяє використовувати результати генерації коду під час розробки різних додатків.

Також у даному розділі було розглянуто та проаналізовано існуючі інструменти для генерування програмного коду, такі як: XSLT, інструменти на основі UML, Razor Generator, Metadrone, Reegenerator, Text Template Transformation Toolkit, Radzen, генератор CodeSmith та ASP.Net Zero. Були досліджені та проаналізовані підходи до реалізації даного процесу у кожній із зазначених системах, їх структуру та особливості. Деякі з них вбудовані в популярні середовища розробки, такі як Microsoft Visual Studio, а інші – незалежні рішення. Вибір між ними залежить від безлічі параметрів, а саме:

1. Яке середовище, мова та структура використовується?
2. Які функції потрібні?
3. Який бюджет проєкту?

Описані системи генерування програмного коду є достатньо повними у наявних функціональних можливостях, проте такі системи зазвичай або коштують великих грошей і є недоступними для простого користувача сервісу, або ж спрямовані на використання у обмежених середовищах розроблення. При цьому більшість вимагає певної структури файлів, які подаються для опису того, що має бути згенеровано. Також якщо розглядати дані системи як спосіб генерування коду шаблонів компонентів у веб-застосунках, то більшість не дозволить отримати необхідний результат.

Також зазначені існуючі інструменти були проаналізовані за критеріями ресурсоємності, можливості налаштування, орієнтованості на веб-застосунки та універсальність щодо мов програмування. Результати даного аналізу зведені у табл. 1.1.

Таблиця 1.1

Аналіз існуючих рішень

Назва рішення	Ресурсоєм-ність	Можливість налашту-вання	Орієнтова-ність на веб-застосунки	Універса-льність
XSLT	+	+	+	+/–
UML	–	+	+/–	+/–
Razor Generator	–	–	–	–
Metadrone	+	+	–	–
Reegenerator	+	+	–	+
T4	+	+	–	–
Radzen	–	+	+	+/–
Генератор CodeSmith	–	+	–	+
ASP.Net Zero	+	–	+/–	+

Бачимо що досить мало рішень спрямовані власне на веб-розробку , а отже це ще раз підтверджує актуальність дисертаційної роботи.

Таким чином, з огляду на вищезазначене, розроблення нового методу генерації коду шаблонів компонентів веб-застосунків з метою пришвидшення створення програмного продукту є актуальною задачею.

2. КОМБІНОВАНИЙ МЕТОД ГЕНЕРАЦІЇ КОДУ ШАБЛОНІВ КОМПОНЕНТІВ ВЕБ-ЗАСТОСУНКІВ

2.1. Аналіз опису компонента у вигляді послідовності токенів

На сьогоднішній день все більше і більше розробників намагаються пришвидшити процес створення програмного коду шляхом пошуку готових рішень. Але оскільки такий підхід до розробки має певні недоліки, тому пропонується розглянути можливість генерування програмного коду на основі формалізованого опису з використанням нейронних мереж.

У більшості випадків генерування коду з формалізованого опису нагадує процес перекладу між мовами, наприклад з англійської на українську і т.д. Тому нещодавні дослідження показали, що використання підходів нейронних мереж, таких як аналіз даних від послідовності до послідовності має достатньо високі показники за критерієм часу та якості отриманих програм. Однак при використанні такого підходу до аналізу тягне за собою деяку проблему: генерування синтаксично правильних програм стає більш складним та довготривалим процесом при збільшенні довжини послідовностей при цьому слід враховувати і те, що нейронній мережі необхідно самостійно виявляти синтаксичну структуру вхідних даних, що є не досить ефективним за часовим критерієм.

Провівши дослідження використання RNN можна зробити такі висновки, що така модель вимагає виконання двох етапів: самостійне вивчення граматики та зіставлення послідовності з граматикою. Але якщо ці два кроки розділити та виконувати окремо, це дозволить підвищити ефективність роботи такого методу.

2.2. Аналіз опису компонента у вигляді дерева

З огляду на вищезазначене визначимо загальний підхід до вирішення питання автоматичної побудови коду шаблонів програмних компонентів застосунків, на основі якого буде розроблено відповідний метод

генерування програмного коду, реалізація якого дозволить підвищити швидкість створення якісного ПЗ.

Як вхідні дані для подальшого генерування коду шаблонів програмних компонентів авторами розглядається стислий формалізований опис майбутнього компонента засобами природної мови. Для вилучення необхідної інформації про розроблюваний компонент пропонується проведення лексичного та синтаксичного аналізу вхідних текстових даних та отримання в результаті дерева розбору, тобто синтаксичного дерева.

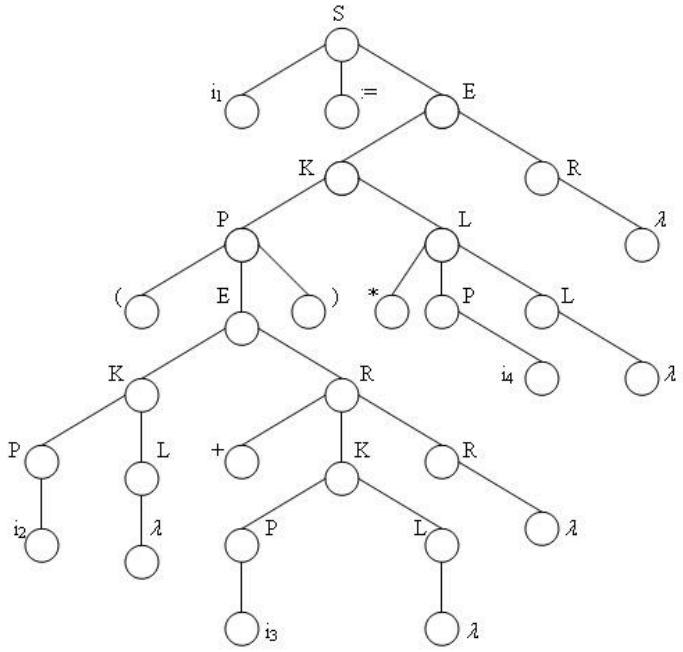


Рис. 2.1. Синтаксичне дерево

Основною гіпотезою, яка повинна стати базисом для нового методу генерування коду шаблонів програмних компонентів, є спроба поєднання синтаксичного аналізу початкового тексту та нейронних мереж, тобто подання на вхід нейронної мережі результатів синтаксичного аналізу тексту з метою подальшого продукування коду, адже однією зі сфер використання нейронних мереж є саме генерування осмислених корисних текстів та створення унікального контенту на задану тематику. При цьому для випадку

створення програмного коду автори вважають за доцільне обрати рекурентні нейронні мережі.

Рекурентні нейронні мережі (РНМ) – це клас штучних нейронних мереж, у якому з'єднання між вузлами утворюють граф, орієнтований у часі [17]. Це створює внутрішній стан мережі, що дозволяє їй проявляти динамічну поведінку в часі. На відміну від нейронних мереж прямого поширення, РНМ можуть використовувати свою внутрішню пам'ять для оброблення довільних послідовностей входів. Це важливо для завдання з оброблення природномовних текстових даних, адже змістовна інформація, пов'язана з поточним словом, може зберігатися не тільки в попередніх словах, а й у наступних словах речення.

Отже, маємо такий загальний підхід до генерування коду шаблонів програмних компонентів (рис. 2.2).

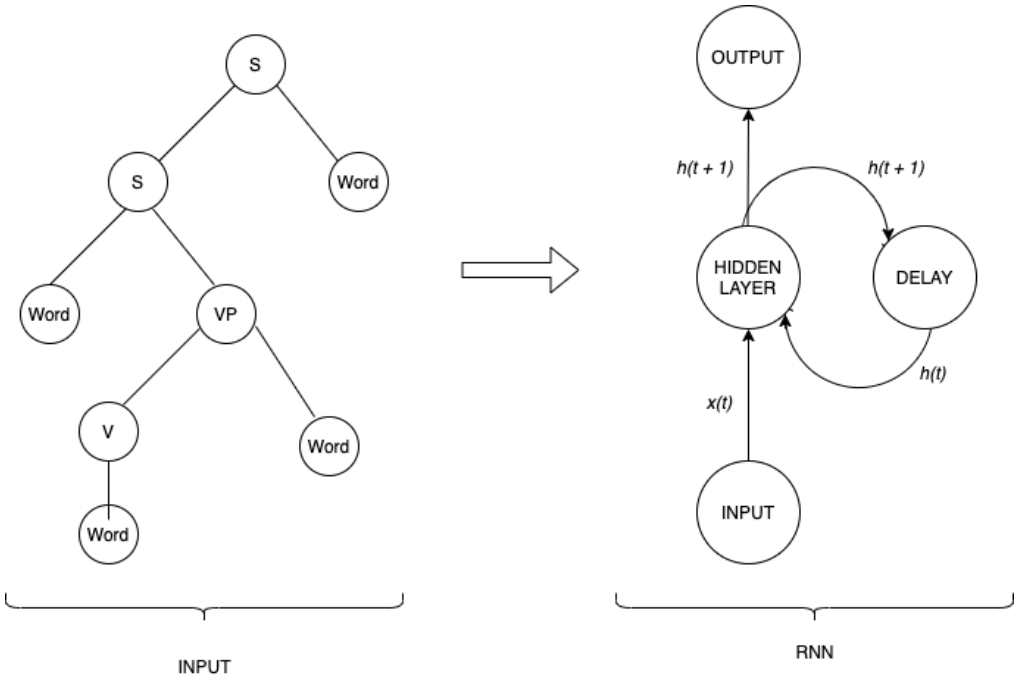


Рис. 2.2. Узагальнена схема методу генерування коду шаблонів програмних компонентів

З урахуванням вищезазначеного пропонується розглянути аналіз вхідних даних, поданих у вигляді дерева. Дерева можна аналізувати за

допомогою рекурсивних нейронних мереж. Вони схожі на рекурентні мережі, але згортають не послідовності, а дерева, тобто до кожного вузла застосовується певна функція $g\left(x, s_1, ..., s_n\right)$, де x – мітка вузла, а s_i – представлення, отримане застосуванням цієї ж нейронної мережі рекурсивно до дочірніх вузлів кожного вузла. При цьому, дана функція часто будується на основі LSTM (така архітектура називається TreeLSTM [18]).

Тобто такий метод матиме наступні кроки: декодувальник розширює нетермінальний символ, він знаходить відповідне піддерево у вхідному дереві за допомогою механізму уваги та використовує інформацію піддерева для розширення нетермінального символу. Зокрема, кодування дерева є корисним у цьому випадку, оскільки воно може агрегувати всю інформацію про піддерева до його кореневого вузла, щоб інформація могла бути доступна для розширення нетермінального символа вже цільового дерева.

Деякі існуючі роботи запропонували архітектури автокодувальників на основі дерева. Однак у цих моделях декодувальник має доступ лише до єдиного прихованого вектора, що представляє вхідне дерево джерела. За результатами дослідження було також встановлено, що при використанні такого підходу без механізму уваги ефективність перекладу у більшості випадків має значення близьке до нуля, а при його використанні – близько 90%.

В іншій роботі пропонується метод на основі принципу кодувальник-декодувальник з використанням уваги, але їх модель працює навіть гірше, ніж модель послідовності до послідовностіз використанням уваги. Одна з головних причин полягає в тому, що їх механізм уваги розраховує ваги уваги кожного вузла окремо. При цьому ієрархічність структури дерева втрачається.

У даній роботі розглядається можливість використання батьківського механізму уваги, який дозволяє отримувати дані про залежності уваги між

різними вузлами. Таким чином можемо стверджувати, що такий підхід може бути більш ефективним при використанні моделі «від дерева до дерева», особливо при збільшенні дерев синтаксичного аналізу.

2.2.1. Проблема перекладу програм

У даній роботі розглядається проблема перекладу програм з формалізованого опису на мову програмування. На даному етапі дослідження пропонується дослідити гіпотези на основі перекладу програми з однієї мови програмування на іншу. Одним із підходів є моделювання проблеми як проблеми машинного перекладу між двома мовами, і таким чином можна застосувати численні підходи до нейронного машинного перекладу.

Слід зауважити, що для кожної програми можна побудувати відповідне дерево синтаксичного аналізу. Таким чином, замість моделювання вхідної програми як послідовності лексем, можемо розглядати проблему перетворення вхідного дерева в цільове дерево. При цьому слід відмітити, що більшість сучасних мов програмування супроводжуються добре розробленим парсером, тому можна припустити, що дерева синтаксичного аналізу як вхідної, так і цільової програм можна легко отримати.

Таким чином основна проблема, яка розглядається, полягає в тому, що крос-компілятор для перекладу програм, як правило, відсутній. Отже, навіть якщо припускати існування синтаксичних аналізаторів як для вхідної, так і для цільової мов, сама проблема перекладу все ще є нетривіальною.

Таким чином сформуємо наступне визначення:

Визначення 1 *(задача перекладу програми). Нехай існують дві мови програмування L_s та L_t, кожна з яких являє собою набір екземплярів (p_k, T_k), де p_k – програма, а T_k – відповідне дерево синтаксичного аналізу. Припускаємо, що існує оракул перекладу π, який відображає екземпляри L_s в екземпляри L_t. Маємо набір пар екземплярів (i_s, i_t) такий, що $i_s \in Ls$, $i_t \in Lt$*

і $\pi(i_s) = i_t$ *. Таким чином наша задача полягає у вивченні функції F, яка визначає кожне* $i_s \in Ls$ *в* $i_t = \pi(i_s)$.

У даній роботі зосередимося на зазначеній вище проблемі, оскільки у нас є набір парних вхідних та цільових програм для навчання перекладача. Слід зауважити, що всі існуючі роботи з перекладу програм [19-21] також вивчають проблему з використання такого припущення. Якщо ж упустити даний етап, проблема перекладу програм стає більш складним процесом. Для вирішення цієї проблеми було запропоновано кілька методів для нейронного машинного перекладу, таких як подвійне навчання [22], які можуть бути розширені і для завдання перекладу програми. Дана модифікація є наступним кроком дослідження.

2.3. Нейронна мережа для моделі «від дерева до дерева»

У даному пункті пропонується розглянути архітектуру нейронної мережі для моделі «від дерева до дерева».

На рис. 2.3 представлений приклад перекладу з CoffeeScript на JavaScript. Слід зауважити, що цікавою властивістю проблеми перекладу програми є те, що процес перекладу може бути модульним. На рисунку виділено підкомпонент у дереві джерела, що відповідає виразу $x = 1$, і його переклад у цільовому дереві, що відповідає виразу $x = 1$. Таке «листування» не залежить від інших частин програми. Таким чином, коли програма стає довшою, і цей вираз може повторюватися кілька разів, для моделі послідовність до послідовності може бути важко знайти відповідність на основі лише послідовностей маркерів без структурної інформації. Таким чином, пропонується пошук піддерева у вхідному дереві при розширенні нетермінального символа в піддереві цільового дерева.

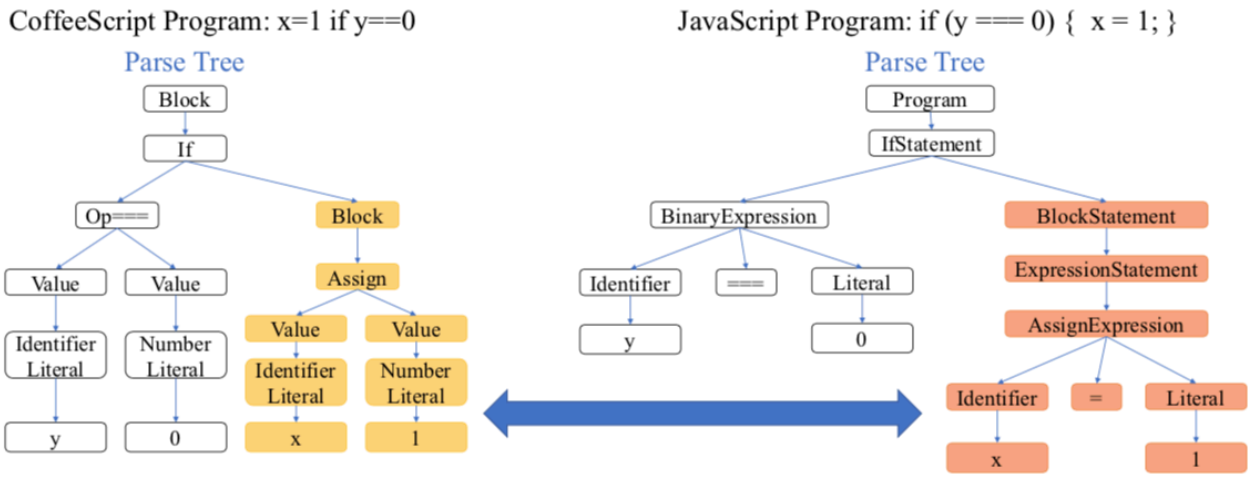


Рис. 2.3. Графічне представлення перекладу програми з CoffeeScript у JavaScript

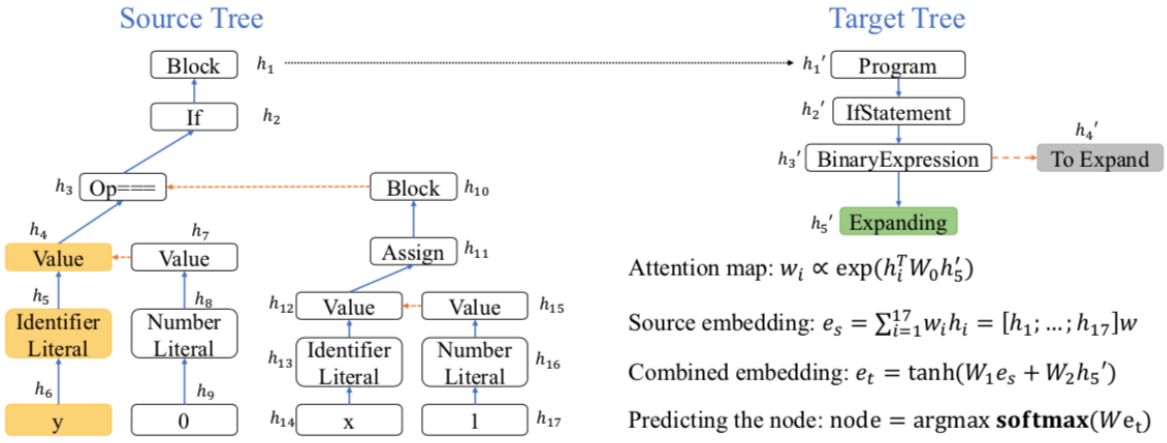


Рис. 2.4. Графічне представлення роботи системи на основі моделі від дерева до дерева

Таким чином, розглянемо нейронну мережу з використанням моделі «від дерева до дерева», яка слідує за принципом роботи кодувальник-декодувальник, щоб кодувати вхідне дерево, подати на вхід декодувальнику та декодувати його для отримання цільового дерева. Для того, щоб зрозуміти процес модульного перекладу, декодувальник використовує механізм уваги для пошуку відповідного піддерева джерела

при розширенні нетермінального символа. Графічне представлення робочого процесу моделі від дерева до дерева зображено на рис. 2.4.

2.3.1. Перетворення дерева в двійкове

Зрозуміло, що вхідні та цільові дерева можуть містити кілька гілок. Хоча розробдення кодувальника та декодувальника для обробки дерев з довільною кількістю гілок можливе, але. припустимо, що кодувальник та декодувальник для бінарних дерев можуть бути більш ефективними за часовим критерієм.

Таким чином, першим кроком є перетворення як вхідного дерева, так і цільового дерева у двійкове. З цією метою для цього перетворення пропонується використовувати перетворення такого виду, як Left-Child Right-Sibling.

2.3.2. Кодування двійкового дерева

Кодувальник використовує структуру Tree-LSTM для обчислення як всього вхідного дерева, так і для кожного його піддерева. Зокрема, розглянемо вузол N, який має два дочірні елементи NL та NR, які є його лівою дочірньою та правою дочірньою структурою відповідно. Кодувальник рекурсивно обчислює вектор фіксованої довжини для вузла N знизу вгору.

Припустимо, що лівий дочірній і правий дочірні вузли мають певний стан LSTM (h_L, c_L) та (h_R, c_R) відповідно, а вектор батьківського вузла дорівнює x. Тоді стан LSTM $(h,\ c)$ N обчислюється як:

$$(h,\ c)\ =\ LSTM(([h_L; h_R],[c_L; c_R]),\ x), \qquad (1)$$

де $[a;\ b]$ позначає конкатенацію a і b. При цьому слід враховувати, що у вузлі можуть бути відсутні один або обидва дочірні вузла. У цьому випадку кодувальник встановлює значення LSTM відсутнього дочірнього вузла рівним нулю.

2.3.3. Декодування двійкового дерева

Декодувальник генерує цільове дерево, починаючи з першого кореневого вузла. Він спочатку копіює стан LSTM (h, c) кореня вхідного дерева та приєднує його до кореневого вузла цільового дерева. Потім у порядку черги всіх вузлів, рекурсивно розширюється кожен з них. У кожній ітерації декодувальник обробляє один вузол із черги та розширює його. Спочатку декодувальник передбачає значення розширюваного вузла. З цією метою він обчислює вектор e_t розширюваного вузла N, а потім подає його в регресійну мережу softmax для прогнозування:

$$t_t = argmax\, softmax(W_{e_t}). \tag{2}$$

Тут W є навчальною матрицею розміру $V_t \times d$, де V_t – розмір словника вихідних даних, а d – розмірність вектора. При цьому, e_t обчислюється за допомогою механізму уваги, який розглядається далі.

Отримані значення кожного вузла t_t – це нетермінал, термінал або спеціальний маркер ⟨EOS⟩. Якщо t_t = ⟨EOS⟩, тоді декодер закінчує розширення цього вузла. В іншому випадку декодер генерує один новий вузол як лівий дочірній елемент, а інший новий вузол - як правий дочірній елемент розширюваного.

Припустимо, що $(h', c'), (h'', c'')$ є LSTM-станами його лівої дочірньої та правої дочірньої дочірних систем відповідно, тоді вони обчислюються як:

$$(h', c') = LSTM_L((h, c), B_{t_t}), \tag{3}$$

$$(h'', c'') = LSTM_R((h, c), B_{t_t}), \tag{4}$$

де B є матрицею для вектора слова розміром $d \times V_t$.

Слід зауважити, що генерація лівого дочірнього та правого дочірнього вузла використовують два різні набори параметрів для $LSTM_L$ та $LSTM_R$ відповідно. Ці нові дочірні вузли потрапляють у чергу всіх вузлів, які

потрібно розширити. Коли черга порожня, процес створення цільового дерева завершується.

2.3.4. Механізм уваги для пошуку вхідного піддерева

Розглянемо, як обчислити e_t. Одним із існуючих підходів є обчислення e_t як h, що є прихованим станом, прикріпленим до розширюваного вузла. Однак, при такому використанні, вектор з часом втратить інформацію про вхідне дерево при генерації глибоких вузлів у цільовому дереві, і як висновок модель може дати дуже низьку продуктивність.

Для кращого використання інформації вхідного дерева розроблювана модель «від дерева до дерева» використовує механізм уваги для пошуку вхідного піддерева, що відповідає піддереву, з розширюваним вузлом у якості кореневого. Зокрема, обчислюється така ймовірність:

$$P\left(N_s - \textit{піддерево джерела, що відповідає } N_t \mid N_t\right), \tag{5}$$

де $\boldsymbol{N_t}$ – вузол, що розширюється. Позначимо цю ймовірність, як $P\left(Ns \mid Nt\right)$ і обчислимо наступним чином:

$$P\left(N_s \mid N_t\right) \propto exp\left(h_s^T W_0 ht\right), \tag{6}$$

де W_0 – навчальна матриця розміром $d \times d$.

Щоб використати інформацію з вхідного дерева, обчислюється ймовірне значення прихованого стану для всіх N, тобто:

$$e_s = E[h_{N_s} \mid N_T] = \sum_{N_s} h_{N_s} * P(N_s \mid N_t). \tag{7}$$

Потім цей вектор можна комбінувати з h, прихованим станом розширюваного вузла, для обчислення e_t наступним чином:

$$e_t = tanh(W_1 e_s + W_2 h), \tag{8}$$

де W_1, W_2 – це матриці розміром $d \times d$ відповідно.

2.3.5. Батьківський механізм уваги

У наведеному вище підході вектори уваги e_t обчислюються незалежно один від одного, оскільки всього один раз e_t використовується для

прогнозування значення вузла t_t та не використовується для подальших прогнозів.

Однак рішення щодо уваги для прогнозування кожного вузла повинні бути пов'язані між собою. Наприклад, для нетермінального вузла N_t у цільовому дереві. Припустимо, що він пов'язаний з N_s у вхідному дереві, тоді дуже ймовірно, що ваги уваги його дітей повинні зосередитися на нащадках N_s. Тому, передбачаючи вектор уваги вузла, модель повинна також використовувати інформацію уваги свого батька.

Дотримуючись цього принципу, запропоновано батьківський механізм подачі уваги, щоб вектор уваги розширюваного вузла враховувався при прогнозуванні векторів уваги його дітей. Формально, окрім векторного значення вузла t_t, модифікуються входи до $LSTM_L$ та $LSTM_R$ декодувальника у рівняннях:

$$(h', c') = LSTM_L((h, c), [B_{t_t}; e_t]), \quad (9)$$

$$(h'', c'') = LSTM_R((h, c), [B_{t_t}; e_t]). \quad (10)$$

При цьому слід зауважити, що зазначені формули у своїх форматах збігаються з методом подачі вхідних даних для послідовних нейронних мереж [23], але їх значення відрізняються. Для послідовних моделей вхідний вектор уваги належить попередньому маркеру, тоді як тут він належить батьківському вузлу.

2.4. Метод генерації коду шаблонів компонентів веб-застосунків

З огляду на вищезазначене пропонується розглянути загальну структуру методу генерації коду шаблонів компонентів веб застосунків. На сьогоднішній день розробники використовують безліч технологій та підходів для реалізації веб-застосунків, тому вдосконалення процесу розроблення таких проєктів у загальному пропонується залишити на майбутнє. Для даного дослідження зосередимося на реалізації методу для генерування програмних React-компонентів.

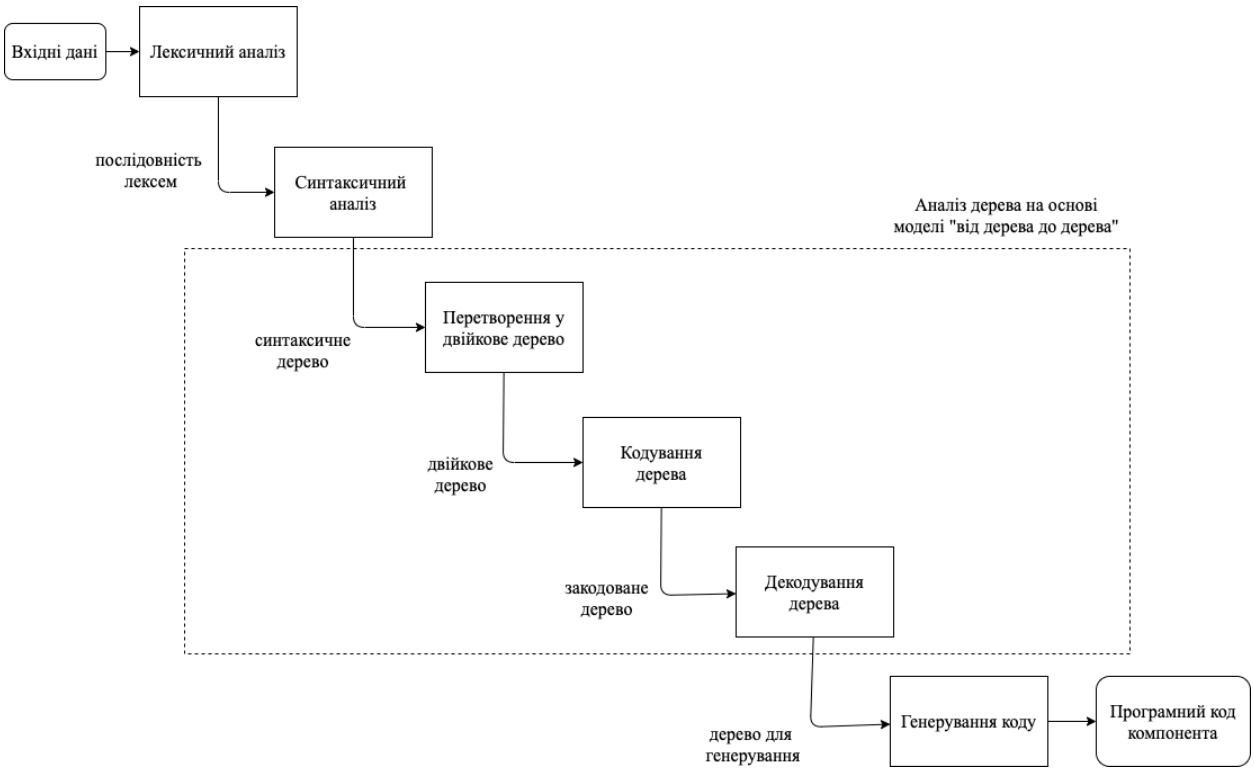


Рис. 2.5. Графічне представлення послідовності виконання кроків методу

Розглянемо загальну структуру методу генерації коду шаблонів компонентів веб-застосунку, який має наступні етапи (рис. 2.5):

1. Лексичний аналіз вхідних даних, поданих у формалізованому форматі. На даному етапі пропонується перетворити вхідні дані у послідовність лексем.
2. Синтаксичний аналіз. Отриману послідовність лексем подаємо на вхід до синтаксичного аналізу в результаті якого отримаємо синтаксичне дерево.
3. Аналіз дерева. Отримане синтаксичне дерево аналізуємо на основі моделі «від дерева до дерева» зазначеної вище:
   - перетворення вхідного дерева у двійкове;
   - кодування отриманого двійкового дерева з використанням батьківської уваги;
   - перетворення вхідного дерева у двійкове.
4. Генерація програмного коду. З отриманого дерева генеруємо програмний код компонента.

2.5. Висновки до розділу 2

У результаті дослідження, що було зроблене у даному розділі було детально вивчено та описано процес аналізу вхідних даних, поданих у вигляді дерева. Розглянуто існуючі моделі для побудови нейронних мереж для розроблюваного методу генерування програмного коду, визначено їх головні недоліки. На основі проаналізованих характеристик існуючих рішень та наявних в них недоліків було сформовано модель нейронної мережі, яка б показала найкращі результати за часовим критерієм. Також було виділено область дослідження, а саме генерування React-компонентів, оскільки розроблення універсального методу вимагає додаткового дослідження. Як результат – було сформовано загальну структуру методу для генерації програмного коду шаблонів компонентів. Також було виділено етапи методу генерації програмного коду компонента, графічне представлення якого зображено на рис. 2.5.

3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ГЕНЕРУВАННЯ КОДУ ШАБЛОНІВ КОМПОНЕНТІВ ВЕБ-ЗАСТОСУНКІВ

3.1. Аналіз засобів розроблення програмного забезпечення

Оскільки область дослідження була звужена до генерування програмного коду React-компонентів, тому в якості засобів розроблення було вирішено обрати дві мови програмування, а саме: Python та JavaScript, де Python використовуватиметься у якості засобу для аналізу вхідних даних та їх перетворення у дерево, а JavaScript – для генерування програмного коду шаблона компонента.

Для реалізації аналізу вхідних даних та перетворення їх у дерево автором пропонується використовувати машинне навчання. Відомо, що машинне навчання – це одне напрямків з найшвидшим показником розвитку. Воно включає в себе різні алгоритми, які продовжують бути популярними для компаній у сфері розроблення програмних забезпечень. Звичайно, ці алгоритми можуть бути реалізовані з використанням різних мов програмування, але іноді розробники можуть стикатися з проблемами при впровадженні цих алгоритмів, і тому потрібна проста мова програмування, яка допомагає ефективно розвивати свій проєкт. Зазвичай, можна зустріти алгоритми машинного навчання реалізовані з використанням різних мов, таких як C, C++, JavaScript, R та Python. Але слід зауважити, що мова програмування Python на сьогоднішній день є вибором кожного розробника завдяки його різноманітним особливостям.

Python є високорівневою крос-платформною мовою програмування, яка підтримує декілька парадигм програмування, в тому числі об’єктно-орієнтоване програмування [24]. Вона є досить універсальною, оскільки використовується для різних типів задач: веб- або десктоп- додатки, машинне навчання, мережеві сервери та т.п. Зазвичай вона використовується при розробленні невеликих проєктів, але при цьому такі

компанії як Google, Facebook, Microsoft та Netflix, також використовують Python для реалізації своїх проєктів.

Мова програмування Python також має кілька переваг, які дозволяють розробникам використовувати її для створення своїх проєктів на основі машинного навчання.

Python має достатньо велику колекцію вбудованих бібліотек: для інтелектуального аналізу даних, маніпулювання даними та машинного навчання:

1. Бібліотека NumPy є абревіатурою "Numerical Python" і використовується для чисельних обчислень у Python. Він забезпечує структури даних, алгоритми та методи, необхідні більшості наукових додатків, включаючи чисельну обробку даних [25]. Тому NumPy має достатньо функцій для виконання основних обчислень із масивами або математичних операцій між масивами. Інструменти для читання та запису масивів даних на основі даних на диск; операції лінійної алгебри, перетворення Фур'є та генерація випадкових чисел.
2. Пакет програм Pandas – забезпечує вдосконалені структури даних, засоби аналізу даних та функції, призначені для швидкого та простого використання структурованих або табличних даних [26].
3. Також слід відмітити бібліотеку Scikit-learn, яка має інструменти для аналізу даних, які оптимізують використання машинного навчання в Python. Вона також має декілька підмодулів для наступних моделей [27]:
    - SVM, найближчі сусіди, випадковий ліс, логістична регресія, які використовуються для завдання класифікації;
    - k-means, спектральна кластеризація – для кластеризації;
    - нормалізація – попередньої обробки тощо.

Також SciPy використовується для вдосконалених обчислень, а Pybrain – для машинного навчання**.**

Ще однією перевагою Python є те, що ця мова програмування є універсальною та добре структурованою мовою програмування, доступною та простою для вивчення та використання. У цьому випадку Python можна легко інтегрувати з іншими мовами або мовами нижчого рівня (такими як C, C++ або Java). Також слід відмітити портативність даної мови програмування, тобто можливість запуску коду на іншій платформі. Це називається Write Once Run Anywhere, або скорочено WORA.

У якості засобу розроблення для генерування програмного коду шаблонів компонента веб-застосунку було обрано JavaScript, оскільки з часу випуску даної мови програмування вона є найпопулярнішою для розроблення веб-застосунків (наразі близько 94% розробників використовують її для створення веб-сайтів). JavaScript – динамічна, об'єктно-орієнтована прототипна мова програмування [28]. Відома як реалізація стандарту ECMAScript [29]. Зазвичай її використовують для створення сценаріїв веб-сторінок, оскільки вона надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки. При цьому на веб-ринку доступно багато платформ JavaScript, таких як AngularJS, ReactJS, NodeJS тощо. Використовуючи ці фреймворки, ви можете зменшити кількість часу та зусиль, необхідних для розробки сайтів та програм на основі JS. JavaScript дозволяє програмістам легко створювати масштабні веб-програми. Це спрощує весь процес розробки масштабних веб-додатків.

3.2. Архітектура розробленого програмного забезпечення

Розроблену систему можна розділити на п’ять модулів, зображених на рис. 3.1.

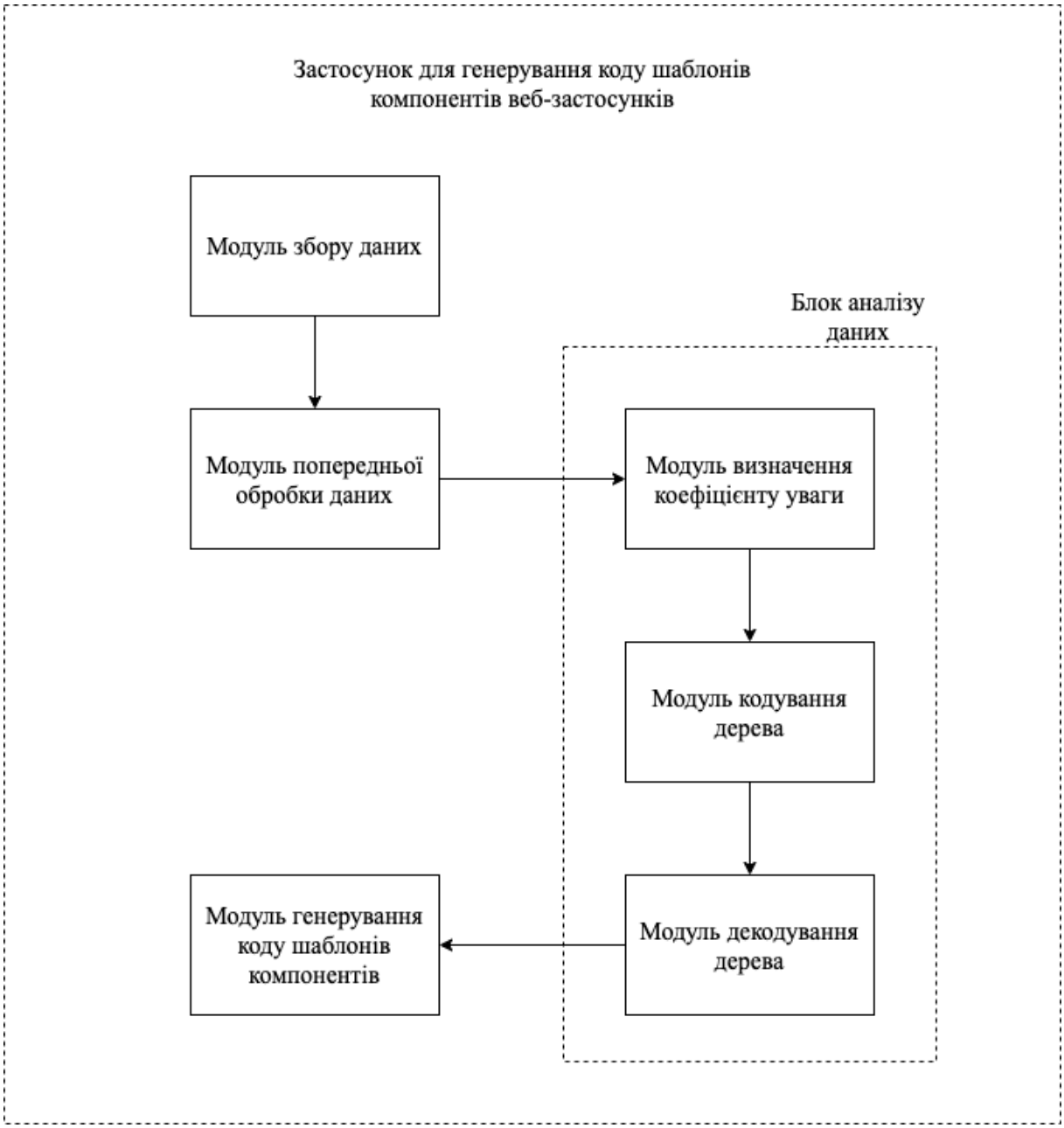


Рис. 3.1. Схема архітектури системи

1. Модуль збору даних – відповідає за взаємодію з користувачем та збір даних, на основі яких проводитиметься генерування програмного коду компонента.

   Дані повинні мати:

   - певну формалізовану структуру;
   - відповідні ключові слова.

У якості інтерфейса для взаємодії з користувачем можуть виступати:

- термінальна консоль;
- json файл.

Користувач матиме змогу описати характеристики компонента використовуючи JSON – мову. JSON – це спосіб кодування структур даних, що забезпечує їх легкість для читання на машинах. JSON – це основний формат, в якому дані передаються в API, що є дуже зручним для їх подальшої обробки.

2. Модуль попередньої обробки даних – надає можливість автоматичного перетворення вхідних даних у дерево та їх обробку, а саме:
   - пошук словника для навчання моделі;
   - приведення до форми бінарного дерева;
   - створення структури цільового дерева.
3. Модуль визначення коефіцієнту уваги – відповідає за коректне визначення очікуваного значення прихованого стану для всіх вузлів.
4. Модуль кодування вхідного дерева – надає можливість отримати дерево з відповідними станами замість вузлів – векторами фіксованої довжини.
5. Модуль декодування дерева – відповідає за наповнення цільового дерева даними із закодованого дерева, починаючи з кореневого вузла.
6. Модуль генерування програмного коду шаблона компонента – надає можливість отримати файл із програмним кодом шаблона компонента на основі даних з цільового дерева.

3.3. Особливості реалізації обробки даних

Як раніше було зазначено, обробка даних базується на використанні моделі «від дерева до дерева», яка у свою чергу використовує рекурсивні нейронні мережі (рис. 3.2).

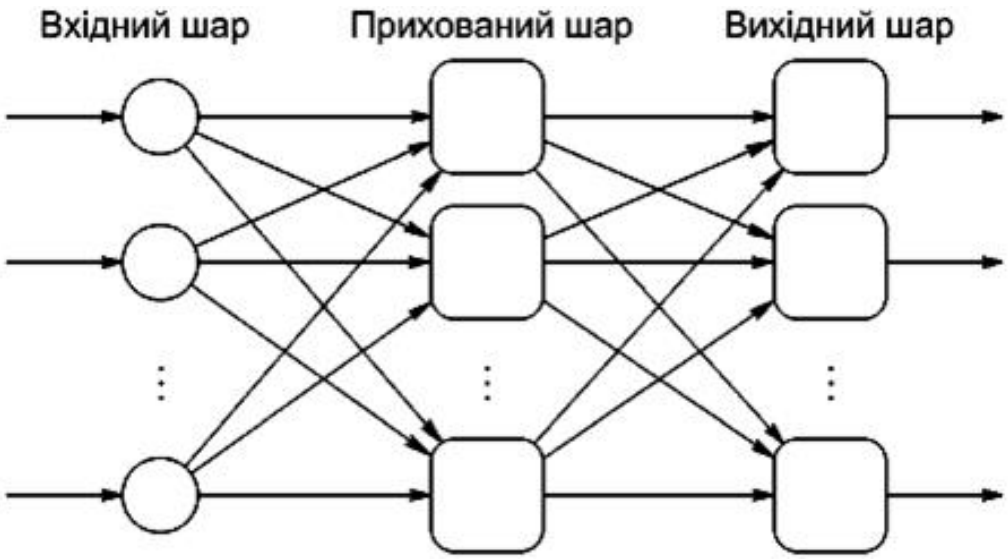


Рис. 3.2. Схема рекурсивної нейронної мережі

Вони схожі на рекурентні мережі, але згортають не послідовності, а дерева, тобто до кожного вузла застосовується певна функція $g(x, s_1, ..., s_n)$, де x – мітка вузла, а s_i – представлення, отримане застосуванням цієї ж нейронної мережі рекурсивно до дочірніх вузлів кожного вузла. При цьому, дана функція часто будується на основі LSTM – Long short-term memory.

Структура TreeLSTM була вперше представлена у статті 2015 року – «Покращені семантичні уявлення з деревоподібних довгострокових мереж короткострокової пам’яті». Основна ідея такої структури полягає у введенні синтаксичної інформації для мовних завдань шляхом розширення ланцюгово-структурованого LSTM до деревоподібного LSTM.

Таким чином для реалізації модулів обробки даних було створено кодувальник та декодувальник на основі LSTM. Оскільки LSTM базується на RNN, тому як і всі повторювані нейронні мережі має форму ланцюга

повторюваних модулів нейронної мережі. Однак у стандартних RNN, цей повторюваний модуль матиме дуже просту структуру, зображену на рис. 3.3.

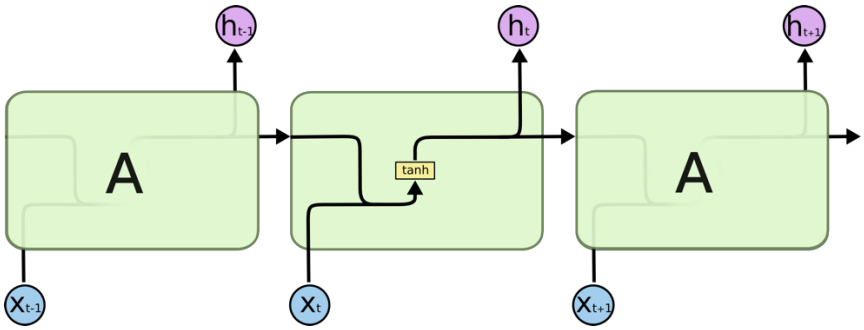


Рис. 3.3. Схема повторюваного модулю стандартних RNN

LSTM у свою чергу також мають цю ланцюгоподібну структуру, але повторюваний модуль має іншу структуру (рис. 3.4). Замість того, щоб мати один шар нейронної мережі, існує чотири, які взаємодіють між собою.

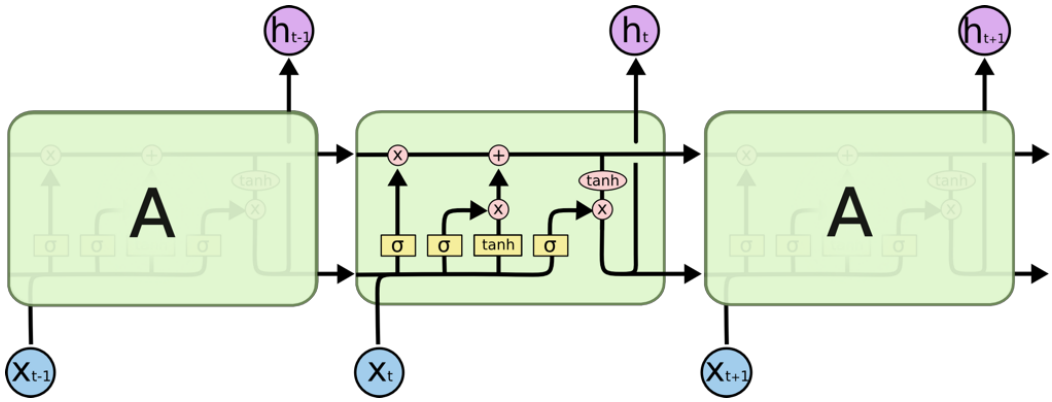


Рис. 3.4. Схема повторюваного модулю LSTM

На наведеній вище схемі кожен рядок представляє собою цілий вектор, від виходу з одного вузла до входів інших. Рожеві кола представляють собою точкові операції, такі як додавання векторів, у той час як жовті прямокутники – шари нейронної мережі.

Ключовими значеннями в LSTM є стани комірок, горизонтальні лінії, зображені на схемі. Стан комірки нагадує собою конвеєр. Він проходить прямо по всьому ланцюжку, лише з деякими незначними лінійними взаємодіями. При цьому інформація передається по ній без змін. Оскільки LSTM має здатність видаляти або додавати інформацію до стану комірки, тому з'являється необхідність у впровадженні додаткових елементів структури – так званих «воріт». «Ворота» – це спосіб додаткового пропуску інформації. Вони складаються із сигмоподібного нейронного шару та операції точкового множення (рис. 3.5).

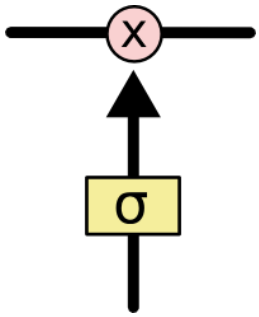


Рис. 3.5. Схема «Воріт» LSTM

Такий шар видає числа від нуля до одиниці, описуючи, яка ймовірність пропуску кожного компонента. Нульове значення означає «нічого не пропускати», тоді як значення одиниця – «пропустити все».

Таким чином реалізація кодувальника та декодувальника зводиться до створення сутностей на основі LSTM, а також використанні додаткового батьківського механізму уваги (див. Додаток 1).

Оскільки процес генерування програмного коду шаблону компонента веб-застосунку на основі формалізованого опису цього компонента дуже схожий на процес перекладу з однієї мови програмування на іншу, навчання нейронної мережі з використанням запропонованої моделі «від дерева до дерева» з батьківським механізмом уваги відбувалося на основі шматків коду різними мовами програмування. Приклад перекладу з мови

програмування CoffeeScript у мову програмування JavaScript зображений на рис. 3.6.

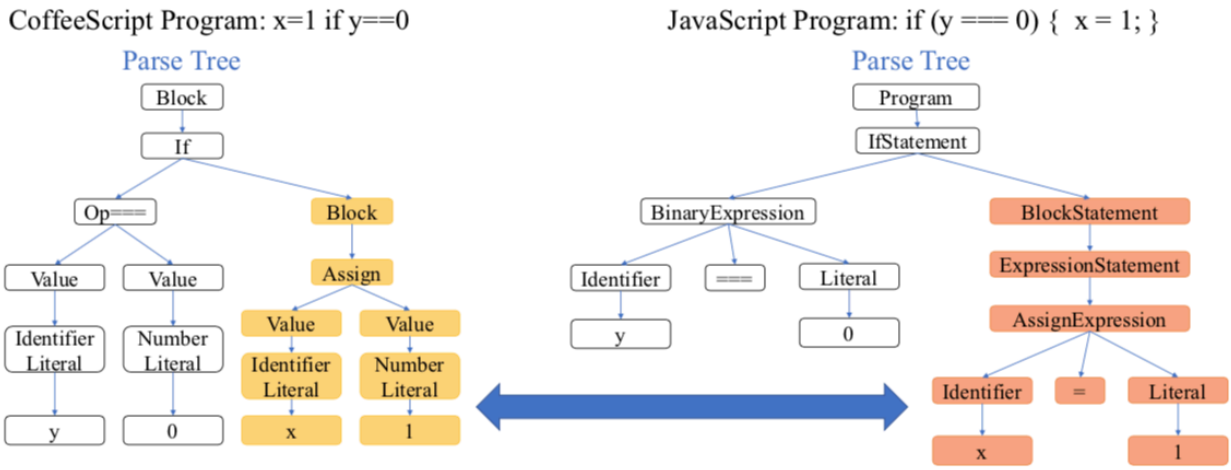


Рис. 3.6. Переклад програми CoffeeScript у JavaScript. Виділено підкомпонент програми CoffeeScript та відповідний переклад у JavaScript

3.4. Особливості реалізації модулю генерування програмного коду

На основі наведених теоретичних даних та розглянутого модулю обробки даних було розроблено програмний засіб автоматизованого генерування коду шаблонів компонентів веб-застосунків. Оскільки дослідження компонентів веб-застосунків у даній дисертації було звужено до React-компонентів, тому розроблюваний програмний засіб генеруватиме шаблони саме React-компонентів.

React-компоненти – це незалежні та багаторазові біти коду. Вони виконують ті самі цілі, що і функції м, але працюють ізольовано і повертають HTML через функцію візуалізації [30]. HyperText Markup Language – мова розмітки гіпертексту – це мова тегів, засобами якої здійснюється розмітка веб-сторінок для мережі Інтернет.

Відомо, що існують два типи React-компонентів, а саме:

- класові компоненти;
- функціональні компоненти.

Прикладом функціонального компонента є звичайна функція у JavaScript, представлена на рис. 3.7. Ключовими словами для завдання генерування програмного коду шаблону функціонального компонента є:

- «function»;
- «return».

Відомо, що на даний момент функцію у JavaScript можна написати різними способами, але наразі зупинимося на такому варіанті з використання ключового слова «function».

```
export default function myFunc() {
    const { name } = props;
    return (
        <div>`Hello,my dear ${name}`</div>
    )
}

const func = () => {
    const { name } = props;
    return (
        <div>`Hello,my dear ${name}`</div>
    )
};
```

Рис. 3.7. Приклад функціонального React-компонента мовою JavaScript

Якщо розглядати можливість експорту даної функції до інших файлів, де вона використовується, то слід врахувати додавання нових ключових слів, а саме:

- «default»;
- «export».

Приклад такої функції представлено на рис. 3.8.

```
export default function myFunc() {
    const { name } = props;
    return (
        <div>`Hello,my dear ${name}`</div>
    )
}
```

Рис. 3.8. Приклад функціонального React-компонента мовою JavaScript, який експортується

React-компонент написаний на основі класів ES-6 є прикладом класового компонента (рис. 3.9). Ключовими словами для завдання генерування програмного коду шаблону класового компонента є:

- «class»;
- «extends»;
- «React.Component»;
- «render»;
- «return».

Також слід розглянути інший вид компонентів, таких як Router (рис. 3.10). Кожен Router створює об'єкт «history», який зберігає шлях до поточного місцезнаходження – «location» і перемальовує інтерфейс сайту, коли відбуваються якісь зміни шляху [34]. Решта функцій, які є надаються React Router покладаються на доступність об'єкта «history» через «context», тому вони мають бути всередині компонента Router.

При цьому компоненти React Router, які не мають в якості предка компонент Router не працюватимуть, оскільки їм не буде доступний об’єкт «context». Також такі компоненти дозволяють використовувати лише один дочірній елемент.

```
class ScrollingList extends React.Component {
    constructor(props) {
        super(props);
        this.listRef = React.createRef();
    }

    getSnapshotBeforeUpdate(prevProps, prevState) {
            const list = this.listRef.current;
            return list.scrollHeight - list.scrollTop;
        }
        return null;
    }

    componentDidUpdate(prevProps, prevState, snapshot) {
        if (snapshot !== null) {
            const list = this.listRef.current;
            list.scrollTop = list.scrollHeight - snapshot;
        }
    }

    render() {
        return (
            <div ref={this.listRef}>{/* ...вміст... */}</div>
        );
    }
}
```

Рис. 3.9. Приклад класового React-компонента мовою JavaScript

```
import React, { Component } from 'react';
import Profile from './routes/profile';
import './App.css';
import { BrowserRouter as Router, Route } from "react-router-dom";

class App extends Component {
    render() {
        return (
            <Router>
                <div className="App">
                    <Route path="/profile" exact component={Profile} />
                </div>
            </Router>
        );
    }
}

export default App;
```

Рис. 3.10. Приклад React Router компонента мовою JavaScript

Таким чином генерація зазначених вище компонентів відбуватиметься за допомогою методів «createComponent», «createRoute» та «createFunction», реалізація яких зазначена у додатках (див. Додаток 1).

3.5. Висновки до розділу 3

У даному розділі наведено архітектуру розроблюваного програмного забезпечення, а також особливості реалізації оброки даних та генерування програмного коду.

Розроблене програмне забезпечення призначене для генерування коду шаблонів компонентів веб-застосунків на основі запропонованого в роботі алгоритмічно-програмного комбінованого методу.

Розроблена система надає можливості автоматично отримати файли з шаблоном React-компонента, опис якого користувач описує у термінальному вікні або json-файлі.

Завдяки використанню моделі нейронної мережі даний метод можна використовувати для генерації різних шаблонів компонентів різними мовами програмування.

4. АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Сфера застосування

Оскільки на сьогоднішній день попит на розробників програмного забезпечення зростає, тому з’являються нові критерії для прийняття того чи іншого спеціаліста на роботу. Серед таких критеріїв є:

1. Швидкість написання програмного коду відповідно до поставленого завдання.
2. Якість написаного коду, тобто його структурованість, отпимізація та подібне.

Звичайно, більшість скаже, що пошук готових рішень на різних платформах є досить поширеною практикою, але слід враховувати, що для розробників з малим багажем досвіду, оптимізувати знайдений шматок коду під власні задачі є досить не тривіальною задачею. Готове програмне рішення може мати зайві дії, а отже є не отпимізованим.

Щодо швидкості написання програмного коду можна зазначити, що набагато легше модифікувати певний шаблон і описувати лише алгоритмічні кроки, ніж писати все з нуля.

Іншим прикладом використання такого рішення може слугувати тенденція на автоматизоване покриття проєктів тестами. На сьогоднішній день такі відомі компанії як Microsoft, Google та інші намагаються відмовитися від мануального тестування проєктів. У свою чергу таке рішення призводить до того, що звичні дії тестувальника мають бути перенесені у програмний код. Тобто іншими словами, виникає потреба у написанні програмного коду людиною без великого досвіду власне програмування.

4.2. Результат роботи програми

Як було зазначено, нині від розробників програмного забезпечення вимагають дотримання принципу модульності. Відповідно до цього, реалізація розроблюваного методу включає в себе можливість імпортування та екпортування файлів. На сьогоднішній день у сфері веб-додатків присутнє негласне правило зі створення певної архітектури, а саме винесення стилізуючих компонентів до окремих файлів, а також використання саме файлу «index.js» для взаємодії з необхідним модулем. Приклад такої архітектури зображено на рис. 4.1.

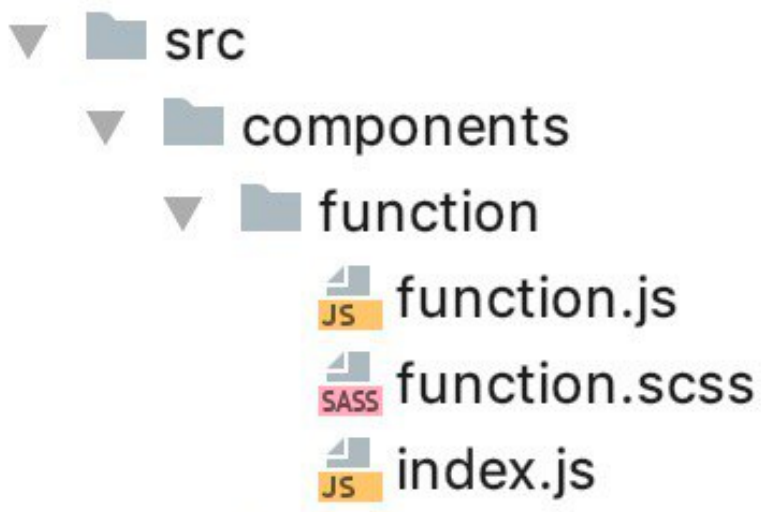


Рис. 4.1. Архітектура згенерованого компонента

Також за допомогою програмної реалізації присутня можливість окрім лише назви компонента вказати певні характеристики для процесу генерування програмного коду шаблона компонента, а саме:

- «–d», «—destination <destination>», що означає заміну місця розташування компонента;
- «–p», «—template-path <template-path>» – дозволяє замінити шлях до генерованого шаблону;

- «–t», «—template <template>», відповідає за заміниу типу шаблона (за замовчуванням генератор використовує папку «components»);
- «–f», «—force» – примусове створення компонента (за замовчуванням неможливо створити компонент, якщо шлях призначення відсутній, однак цей параметр змушує створювати компонент і генерує цільові папки, якщо вони не існують).

Зазначені вище параметри передбачають собою використання термінальної консолі.

Якщо розглядати типи компонентів, то за допомогою описаної в попередньому розділі програмної реалізації розроблюваного методу генерування коду шаблонів компонента веб-застосунків користувач даної системи має можливість створювати шаблони наступних компонентів:

- класовий React-компонент;
- React-компонент на основі функції;
- Router-компонент;
- Redux-компоненти;
- Redux-module-компонети.

На початку взаємодії з розробленою бібліотекою, яка є програмною реалізацією методу, користувачеві будуть доступні питання (рис. 4.2), які дозволяють більш детально ознайомитись із принципом роботи з системою.

```
> rg "functional-component" "func"

? What name do you want to use? func
? Where do you want to create the functional-component? ./src/components/
```

Рис. 4.2. Питання при першому використанні бібліотеки

Для подальшої взаємодії з системою для генерування класового компонента користувач повинен ввести команду, представлену на рис.4.3.

```
npm run rg component myClassComponent
```

Рис. 4.3. Команда для генерування класового компонента

Результатом виконання такої команди є наступний шаблон компонента:

```
import React, {Component} from 'react';
import './myClassComponent.scss'
import { connect } from "react-redux";
import { bindActionCreators } from "redux";
import * as myClassComponentActions from "../../store/myClassComponent/actions";
export default class myClassComponent extends Component {
    constructor(props) {
        super(props);
        this.state = {};
    }
    render() {
      return <div className="component-my-class-component">Hello! component myClassComponent</div>;
    }
  }
export default connect(
    ({ myClassComponent }) => ({ ...myClassComponent }),
    dispatch => bindActionCreators({ ...myClassComponentActions }, dispatch)
  )( myClassComponent );
```

Рис. 4.4. Згенерований програмним забезпеченням класовий компонент

При цьому архітектура класового компонента виглядатиме наступним чином:

Рис. 4.5. Архітектура згенерованого класового компонента

Для генерування функціонального компоненту користувач повинен ввести наступну команду у терміналі:

```
npm run rg functional-component myFuncComponent
```

Рис. 4.6. Команда для генерування функціонального компонента

Результатом виконання такої команди є наступний шаблон компонента:

```
import React from 'react';
import './myFuncComponent.scss'

export default myFuncComponent = (props) => {
  return <div className="component-my-func-component">
    Hello! component myFuncComponent
  </div>;
}
```

Рис. 4.7. Згенерований програмним забезпеченням функціональний компонент

Його архітектура має наступний вигляд:

Рис. 4.8. Архітектура згенерованого функціонального компонента

Для генерування Router-компоненту користувач повинен ввести наступну команду у терміналі:

```
> rg "route"

? What name do you want to use? myRouter
? Where do you want to create the route? ./src/
```

Рис. 4.9. Команда для генерування Router-компонента

Результатом виконання зазначеної команди є наступний програмний код, представлений на рис. 4.10.

```
export default class myRouter extends Component {
  render() {
    const routeComponents = routes.map( callbackfn: ({ path, component, exact, type }, key) => {
      if(type === "public") {
        return exact ? (
          <PublicRoute exact path={path} component={component} key={key} />
        ) : (
          <PublicRoute path={path} component={component} key={key} />
        );
      }
      else {
        return exact ? (
          <PrivateRoute exact path={path} component={component} key={key} />
        ) : (
          <PrivateRoute path={path} component={component} key={key} />
        );
      }
    });
    return (
      <div>
        <Suspense fallback={<div className="loader-div"></div>}>
          <Switch>
            <Redirect exact from="/" to={isAuthenticated() === true ? "/home" : "/login"} />
            {routeComponents}
          </Switch>
        </Suspense>
      </div>
    );
  }
}
```

Рис. 4.10. Згенерований програмним забезпеченням Router-компонент

Його архітектура зазначена на рис. 4.11.

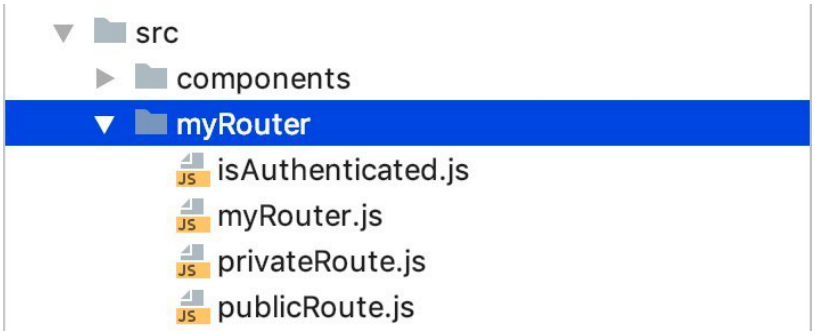


Рис. 4.11. Архітектура згенерованого Router-компонента

Оскільки використання Redux тягне за собою досить складну архітектуру, тому така генерація дещо відрізняється від результатів, зазначених вище. Для генерування Redux-компонентів користувач повинен ввести наступну команду у терміналі:

```
npm run rg redux myRedux
```

Рис. 4.12. Команда для генерування Redux-компонентів

Результатом виконання зазначеної команди є наступний програмний код, представлений на рис. 4.13.

```
import { initialState } from "./states";

export const user = (state = initialState, action) => {
  switch (action.type) {
    case "SET_USER_DETAIL": {
      return Object.assign( target: {}, state, source2: {
        userDetail: action.data
      });
    }
    default:
      return state;
  }
};
```

Рис. 4.13. Згенерований програмним забезпеченням основний компонент Redux

Загальна архітектура Redux-компонентів зазначена на рис. 4.14.

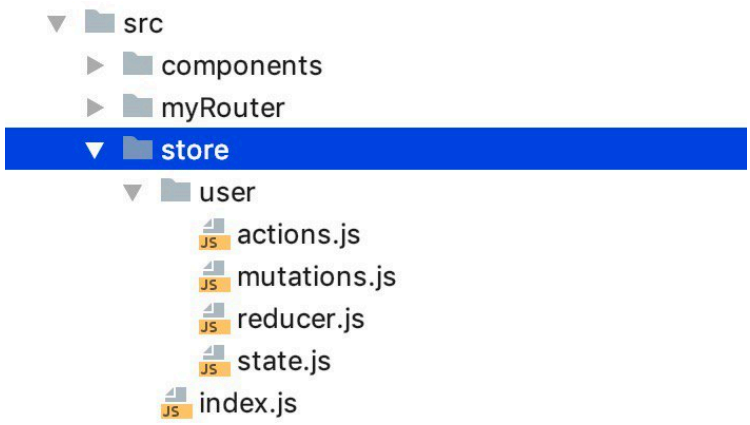


Рис. 4.14. Загальна архітектура згенерованих Redux-компонентів

Також розроблене програмне забезпечення дозволяє генерувати цілі модулі Redux-компонентів. Для цього користувачеві необхідно ввести наступну команду до термільного рядка:

```
npm run rg redux-module testModule
```

Рис. 4.15. Команда для генерування Redux-module-компонетів

Результатом виконання зазначеної команди є наступний програмний код, представлений на рис. 4.16.

Загальна архітектура Redux-module-компонентів представлена на рис. 4.17.

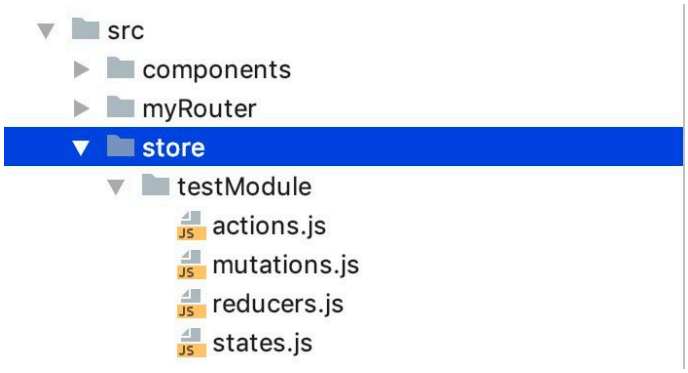


Рис. 4.17. Загальна архітектура згенерованих Redux-module-компонентів

```
import { initialState } from "./states";

export const testModule = (state = initialState, action) => {
  switch (action.type) {
    case "SET_TESTMODULE_LIST": {
      return Object.assign( target: {}, state, source2: {
        testModuleList: action.data
      });
    }
    case "SET_ACTIVE_TESTMODULE": {
      return Object.assign( target: {}, state, source2: {
        activeTestModule: action.data
      });
    }
    case "SET_TESTMODULE_LIST_DATA": {
      const data = [...state.testModuleList];
      data.unshift(action.data);
      return Object.assign( target: {}, state, source2: {
        testModuleList: data
      });
    }
    case "REMOVE_TESTMODULE_FROM_LIST": {
      const data = state.testModuleList.filter(data => data.id !== action.data);
      return Object.assign( target: {}, state, source2: {
        testModuleList: data
      });
    }
    case "SET_UPDATE_TESTMODULE_LIST_DATA": {
      const data = state.testModuleList.map( callbackfn: data => {
        if (data.id === action.data.id) {
          return (data = action.data);
        } else {
          return data;
        }
      });
      return Object.assign( target: {}, state, source2: {
        testModuleList: data
      });
    }
    case "IS_FETCHING_DATA": {
      return Object.assign( target: {}, state, source2: {
        isFetching: action.data
      });
    }
    default:
      return state;
  }
};
```

Рис. 4.16. Згенерований програмним забезпеченням основний компонент Redux-module

4.3. Аналіз ефективності роботи програми

Автором пропонується проаналізувати ефективність роботи розробленого програмного забезпечення за критерієм відношення кількості введених символів до кількості отриманих. Дані для даного аналізу представлені у табл. 4.1.

Таблиця 4.1

Дані для аналізу ефективності роботи програми

Компонент	Кількість введених символів	Кількість отриманих символів
Класовий React-компонент	169	169
Класовий React-компонент згенерований ПЗ	36	169
Функціональний React-компонент	168	168
Функціональний React-компонент згенерований ПЗ	35	168
Router-компонент	455	447
Router-компонент згенерований ПЗ	19	447

Таким чином можемо побудувати графік, який показує ефективність роботи розробленого програмного забезпечення (рис. 4.18).

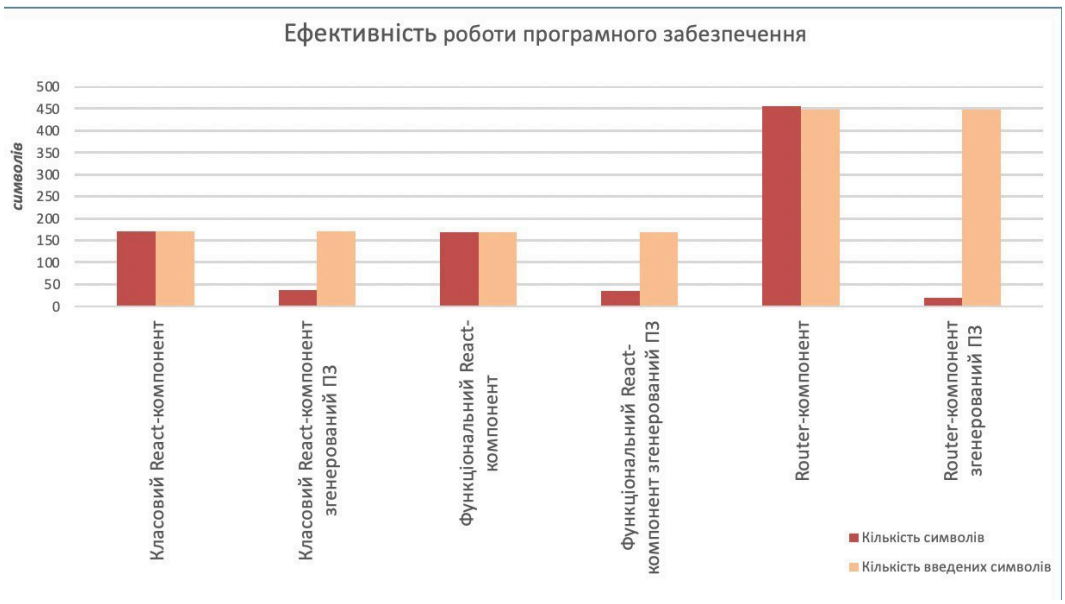


Рис. 4.18. Графічне представлення результатів аналізу ефективності роботи розробленого програмного забезпечення

На представленому графіку помаранчевим кольором зображена кількість введених символів, а зеленим – кількість отриманих.

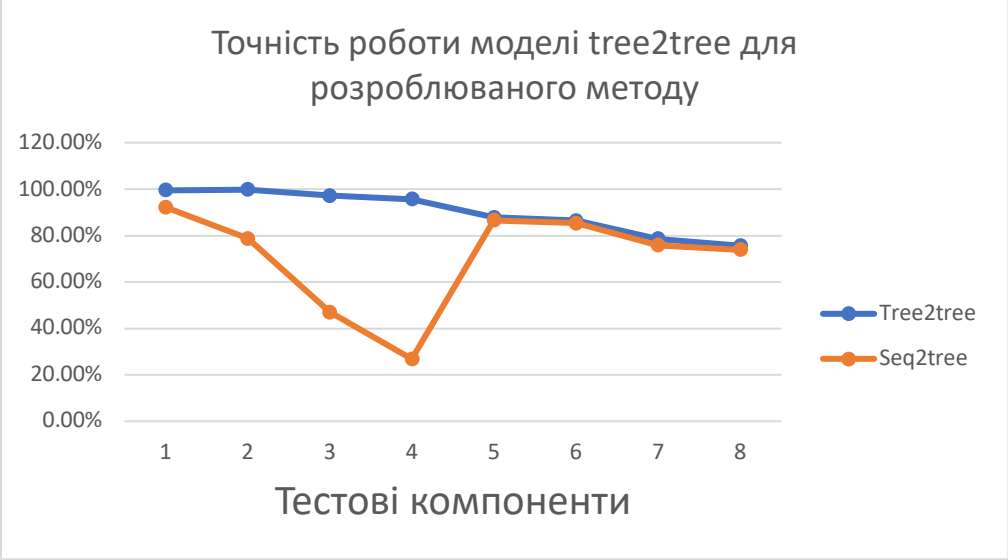


Рис. 4.19. Графічне представлення результатів аналізу точності роботи розробленого програмного забезпечення

Також пропонується проаналізувати точність роботи розроблюваного методу із використанням моделі «від дерева до дерева» з батьківським

механізмом уваги. Для цього використовувалися певні набори даних, а саме код тестових компонентів, які порівнювалися з отриманими деревами.

На рис. 4.19 зображено результати роботи методу із використанням моделі «від дерева до дерева» та «від послідовності до дерева».

4.4. Висновки до розділу 4

У даному розділі було виділено сферу застосування розробленого програмного забезпечення із використанням запропонованого методу генерації коду шаблонів компонентів веб-застосунків.

Також було продемонстровано роботу даного ПЗ, а саме можливіть генерування таких компонентів, як:

- класовий React-компонент;
- функціональний React-компонент;
- Router-компонент.

Ефективність роботи розробленого ПЗ було проаналізовано за критерієм відношення кількості введених символів до кількості отриманих символів. Результат даного аналізу зображено на рис. 4.7.

5. ПОБУДОВА БІЗНЕС-МОДЕЛІ

5.1. Опис проблеми

Розробка програмного забезпечення часто описується як сфера діяльності, яка вимагає достатньо великої кількості знань. Впровадження та обслуговування систем програмного забезпечення підприємства вимагає широкого знання різних мов програмування та застосування інтерфейсів програмування. Якщо у 2002 році для створення веб-сайту розробник повинен був знати такі технології як HTML / CSS, PHP та MySQL, то у 2017 році для цього потрібні знання про всю екосистему інтерфейсу, основи створення та різні мови запитів не тільки MySQL, а вже і NoSQL.

Зазвичай, відповіді на питання стосовно реалізації того, чи іншого функціонального модулю можна знайти у документації, але вона у свою чергу є досить об'ємним джерелом для вирішення простих завдань. Особливо для спеціалістів, які тільки починають свою кар'єру у сфері створенні програмного забезпечення. Тому задля скорочення часу, який витрачається на розроблення, люди почали регулярно використовувати пошукові системи та баз даних Q&A (наприклад, Treude, Barzilay, та Storey). Також розробники часто звертаються до сервісів, таких як StackOverflow, оскільки він дозволяє знайти рішення проблем, які вже могли виникнути у іншого спеціаліста. Іншою причиною також є те, що розробники часто прагнуть знайти існуючі приклади робочого коду для вирішення простих завдань замість того, щоб писати і тестувати його з нуля. А це тягне за собою наступні маніпуляції: щоб знайти відповідний фрагмент коду, розробники програмного забезпечення спочатку формулюють його опис у запиті пошукової системи.

Однак пошук в Інтернеті – це трудомістке завдання, яке викликає переривання процесу кодування. В якості альтернативи, опис коду можна перекласти безпосередньо в код. Такий інструмент перекладу зменшить тягар запам'ятовування деталей певної мови чи API та дозволить

розробнику використовувати свій час для більш творчих аспектів розроблення. Тобто розроблена модель трансляції опису коду дозволяла б перетворювати неформальні вказівки на фактичну реалізацію модулю певною мовою програмування, що б дозволило вирішити основні проблеми, які зображені на рис. 5.1.

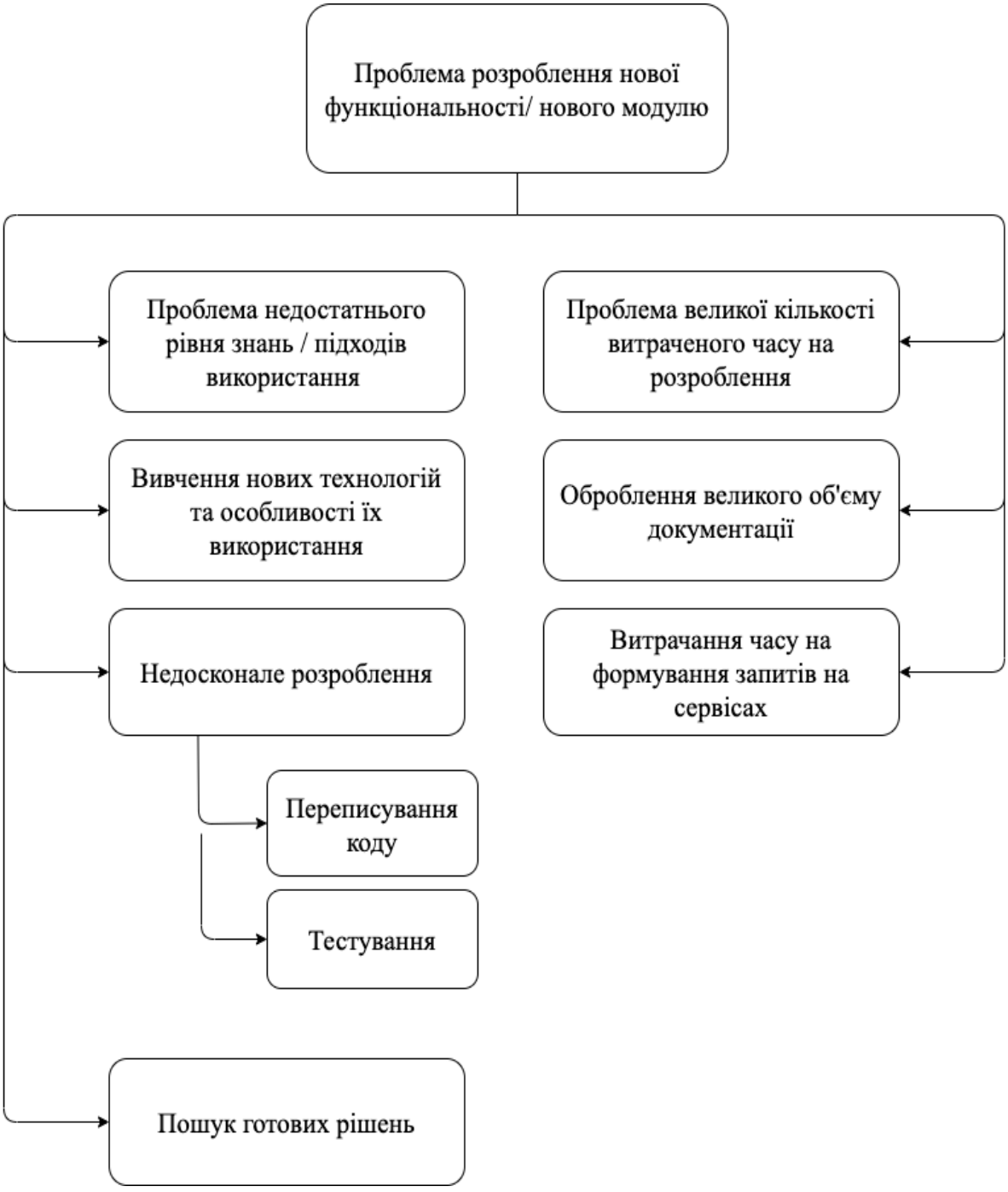


Рис. 5.1. Дерево проблем

5.2. Зацікавлені сторони

У вирішенні описаної проблеми, а саме проблеми розроблення нової функціональності або нового модулю, існує декілька зацікавлених сторін.

Основною групою, якій би було цікаво працювати з розроблюваним програмним продуктом, є спеціалісти зі створення програмного забезпечення. Оскільки написання програмного коду є основною задачею для представника даної групи, тому запропонований метод генерування коду шаблонів компонентів веб-застосунків дозволив би зменшити кількість витрачених годин на створення певного програмного модулю або його тестування.

Також серед розробників є досить велика частина початківців, тобто тих, хто тільки починає свою кар’єру розробника, або ж ті, хто тільки починає знайомитися з певною технологією програмування. Для даного сегменту розроблюваний метод дозволить зменшити час на пошук готових рішень та пошуку необхідної реалізації у документації.

Іншою групою, яка є зацікавленою у даному продукті, є навчальні заклади. Оскільки у навчальних закладах, таких як школа або ліцей, основна увага приділяється процесу мислення та ідеям, тому генерування коду шаблонів компонентів веб-застосунків дозволив би школярам сконцентруватися більше на особливостях реалізації певної функції, а не витрачати час на реалізацію базової структури проєкту чи модулю. Відносно попередніх зацікавлених осіб, вплив даної сторони є опосередкованим, але тим не менше він існує і ним не варто нехтувати.

Наступною групою, яка є зацікавленою у розроблюваному продукті є студенти певних IT академій чи курсів. Оскільки початківцям важко одразу зрозуміти усі особливості процесу розроблення програмного продукту, тому дане програмне рішення дозволить спочатку пояснити матеріал на рівні взаємодії компонентів або модулів, а вже з часом перейти на рівень реалізації даних компонентів програми.

Останньою зацікавленою стороною можна виділити тестувальників. Оскільки розроблення програмного продукту завжди супроводжується тестуванням, тому для полегшення даного процесу можна використати розроблюваний метод. Тестувальники зазвичай концентрують свою увагу на вразливих місцях реалізації, тому генерування базової структури певного компоненту програмного коду дозволив би направити основний потік ідей саме на особливості розроблення компоненту.

Усе описане вище можна звести до однієї таблиці, де вказано вплив певного сегменту на процес розроблення програмного продукту, його інтерес та стратегії приваблення кожної групи зацікавлених сторін (див. табл. 5.1).

Таблиця 5.1

Зацікавлені сторони

Зацікавлена сторона	Інтерес зацікавленої сторони	Вплив зацікавленої сторони	Стратегії приваблення зацікавленої сторони
Розробники програмного забезпечення	Економний та надійний спосіб розроблення базової структури програмного продукту	Високий	Орієнтованість на розроблення базової структури компоненту, що дозволить зменшити час для написання програмного коду

Продовження табл. 5.1

Зацікавлена сторона	Інтерес зацікавленої сторони	Вплив зацікавленої сторони	Стратегії приваблення зацікавленої сторони
Навчальні заклади (школа, ліцей)	Можливість надання практичних завдань для розвитку мислення щодо особливостей реалізації логіки, а не базової структури компоненту	Середній	Орієнтованість на отримання базових практичних навичок
ІТ академії, курси	Можливість надання знань поступово через розуміння принципів роботи та взаємодії компонентів, а не особливостей реалізації	Середній	Орієнтованість на поступовий процес навчання новим технологіям
Тестувальники	Економний спосіб розроблення програмного коду	Високий	Надання можливості самостійно створити програмний компонент без необхідності вивчення особливостей реалізації

5.3. Комерційне рішення. Основні характеристики

На основі вищезазначених проблем можна описати кінцевий продукт, який повинен вирішити ці проблеми. Цей програмний продукт реалізує автоматизований метод формування коду шаблону для компонентів веб-додатків, описаний у попередніх розділах. Даний метод дозволяє ефективно написати базовий шаблон компонента веб-застосунку. Розробнику чи іншому представнику зацікавлених сторін надається зручний інструмент у вигляді розширення можливостей середовища розроблення. Даний програмний продукт покращить процес розробки з точки зору відношення витраченого часу до об’єму програмного коду.

По-перше, зменшиться кількість часу, який витрачається на написання програмного коду. По-друге, зменшиться кількість друкарських помилок при написанні певного компоненту чи модулю.

Даний програмний продукт повинен легко інтегруватися з існуючими середовищами розробки. У випадку написання коду нового компонента чи модулю, дане програмне забезпечення дозволить описати звичною для користувача мовою основні характеристики, які перетворяться на програмний код з базовою структурою необхідного компонента. Тобто, замість того, щоб писати повністю програмний код для компонента, користувач матиме змогу описати основні характеристики природомовним текстом, який згодом перетвориться на базову структуру необхідного модулю.

5.4. Конкурентні переваги рішення

Оскільки розроблення програмного забезпечення є сферою, яка стрімко розвивається та все більше з’являється необхідність у нових програмних рішеннях, тому вдосконалення процесу написання програмного коду дозволить вирішувати поставлені перед розробником задачі швидше. Оскільки наразі існують програмні рішення, які дозволяють лише виправляти ключові слова у коді або використовувати автодоповнення

малого словника користувача, тому перевагами розроблюваного методу генерування програмного коду на основі синтаксичного аналізу природомовних текстових даних є:

- зменшення кількості друкарських помилок при розробленні програмного компонента чи модуля;
- збільшення швидкості розроблення програмного компоненту чи модулю;
- покращення якості програмного коду шляхом зменшення необхідності у тестуванні компоненту;
- унікальний метод визначення основних характеристик компоненту на основі синтаксичного аналізу природомовних текстових даних користувача.

5.5. Клієнти. Сегменти ринку споживання

Як вже зазначалося вище, клієнтами даного продукту є розробники програмного забезпечення.

На сьогоднішній день кількість випускників технічних університетів та студентів ІТ-академій зростає, тому кожен розробник має покращувати якість та швидкість розроблення програмного коду для того, щоб залишитися конкурентноспроможним.

Найбільш вагомим критерієм під час розроблення програмного забезпечення є відношення витраченого часу до об’єму зробленої роботи. Тому розробники намагаються пришвидшити процес написання коду при цьому не погіршити якість продукту. З вище сказаного, можна зробити висновок, що є необхідність у сегментації ринку за показником досвіду у сфері програмного розроблення.

Однак, оскільки у наш час достатньо активно росте технологічний прогрес, тому виникає попит на гарних спеціалістів. Тому є сенс у сегментації ринку за сферою використання. Кількість користувачів даного програмного продукту залежить від сфери використання та відповідно від

вікових ознак. Для студентів та школярів даний продукт є гарним сервісом з точки зору навчання, а для розробників гарним продуктом для покращення процесу написання того чи іншого модулю чи компоненту.

Окрім розміру ринку, сегментація ринку за мовою також зумовлена інженерно-технічними факторами. Складність реалізації цього методу в інших мовах пов'язана з тим, що нині домінує кілька мов програмування, тобто є деякі відмінності в основній структурі модулів або компонентів. Хоча метод, описаний у цій роботі, можна застосувати до будь-якої мови програмування, але для цього потрібно впровадити програмне забезпечення для кожної мови окремо. Також складність впровадження цього методу в інших мовах пов’язана з тим, що інші методи, передобробка тексту (стемінг та лематизація), спрямовані на використання саме англійської мови. Це пов’язано з тим, що англійська мова є мовою міжнародного спілкування, а наукове співтовариство набуло значного розвитку саме у сфері використання методів для англійської мови.

5.6. Унікальна ціннісна пропозиція

Ціннісна пропозиція – це пояснення того, як продукт вирішує проблему. Його можна скласти за формулою:

Ціннісна пропозиція = Проблема + Рішення / Продукт.

У дереві проблем зазначено основні проблеми, а також визначено очікування щодо зацікавлених сторін у галузі генерування програмного коду. Розробники бажають отримати економний та надійний спосіб розроблення базової структури програмного продукту, навчальні заклади, такі як школи та ліцеї – можливість надання практичних завдань для розвитку мислення щодо особливостей реалізації логіки, а не базової структури компоненту, ІТ академії хочуть мати можливість надання знань поступово через розуміння принципів роботи та взаємодії компонентів, а не особливостей реалізації, а тестувальники – економний спосіб розроблення програмного коду.

Насправді, запропоноване рішення дозволяє частково задовольнити всі вищезазначені вимоги зацікавлених сторін та вирішити проблеми, зазначені нижче.

Отже, основною унікальною цінністю пропозицією є розроблений метод генерування коду шаблонів компонентів веб-застосунків, що дозволить описати звичайною мовою основні характеристики розроблюваного компоненту, код якого згодом буде згенеровано.

5.7. Доходи та витрати

Сумарний дохід обчислюється як сума доходу на продаж ліцензії на використання програмного забезпечення та надання послуг на його підтримку.

Загальний дохід обчислюється як дохід від продажу ліцензій на використання програмного забезпечення та послуг, що підтримують програмне забезпечення. План полягає у введенні ліцензій на програмне забезпечення для планування продажу програмного забезпечення, тобто підписання угоди, яка надає право на використання програмного забезпечення. Тип ліцензії, яку планується продати, – комерційна. Ця ліцензія передбачає використання закритого, власницького або пропрієтарного ПЗ.

Пропрієтарне програмне забезпечення, (від англ. proprietary software) – це програмне забезпечення, яке зберігає як невласні, так і власні авторські права. Отримавши або придбавши таке програмне забезпечення, користувач отримає обмежене право на використання: заборонити або закрити доступ до коду (дослідження), модифікації, копіювання, розповсюдження та перепродажу. Якщо існує хоча б одне з цих обмежень, програмне забезпечення вважається власним програмним забезпеченням.

Зазвичай основним методом захисту майнових прав власного програмного забезпечення є те, що поза ліцензійною угодою власник вирішує закрити оригінальний код, щоб захистити свій продукт від

модифікацій та вбудовування в систему, обмежуючи тим самим його використання за допомогою авторизації. Цей вид програмного забезпечення називається закритим програмним забезпеченням. Проте власний код товару може бути відкритим, але власник може обмежити права користувача на умови ліцензії. Запатентоване програмне забезпечення та комерційне програмне забезпечення не є синонімами; запатентоване програмне забезпечення може бути безкоштовним (тобто некомерційним) програмним забезпеченням. Тому ліцензія поширюватиметься на комерційне програмне забезпечення.

Платна технічна підтримка включає контракт на реагування на вимоги користувача та усунення недоліків програми протягом часу, зазначеного в угоді про використання програмного забезпечення, та на інтеграцію методу розробки в існуюче середовище розробки.

У вартість розробки входить:

- утримання персоналу, який надає технічну підтримку (заробітна плата, соціальне забезапечення);
- утримання робочих місць для персоналу (оплата за оренду офісу та комунальні послуги);
- податкові витрати;
- оплата послуг юриста, бухгалтера, прибиральниці.

Витрати та очікуваний прибуток з урахуванням того, що процес розробки мінімального готового продукту займатиме приблизно пів-року представлено у табл. 5.2.

Після завершення фази розробки мінімально ціннісного продукту планується розпочати активну рекламну кампанію та розпочати продаж ліцензій. Таким чином очікується, що перші прибутки почнуться з 7-го місяця старту розробки додатку, а по результатам другого півріччя планується продати ліцензій на 310 тис. дол. Отже, після року розробки та впровадження додатку очікуваний прибуток становить 103 тис. дол.

Таблиця 5.2

Фінансовий план

Місяць	Заробітна плата, тисяч $	Інші витрати, тисяч $	Сума витрат, тисяч $	Заплан. прибутки, тисяч $	Результат, тисяч $
1-й	2	1	3	–	–3
2-й	8	4	12	–	–12
3-й	8	4	12	–	–12
4-й	8	4	12	–	–12
5-й	8	4	12	–	–12
6-й	8	4	12	–	–12
7-й	20	4	24	30	6
8-й	20	4	24	40	16
9-й	20	4	24	60	36
10-й	20	4	24	60	36
11-й	20	4	24	60	36
12-й	20	4	24	60	36
Заг. рез.	162	45	207	310	103

5.8. Бізнес модель

Загалом, все написане вище можна об’єднати просту бізнес-модель у вигляді lean canvas, де:

1. Споживачі – розробники програмного забезпечення.
2. Проблема – полягає у відсутності автоматизованих рішень; відсутності зручної документації щодо нових технологій; людському факторі, який впливає на кількість помилок друку при розробці програмного забезпечення.
3. Рішення – програмне забезпечення, засноване на аналізі природних текстових даних для генерації програмного коду.
4. Унікальна ціннісна пропозиція – програмне забезпечення, що реалізує новий метод генерації програмного коду на основі опису основних особливостей розроблених компонентів або модулів; скорочення часу на написання базової структури програмного модуля.
5. Потоки доходів – доходи від продажу ліцензій; доходи від підтримки програмного забезпечення.
6. Структура витрат – утримання персоналу технічної підтримки (заробітна плата, соціальні виплати); підтримка роботи персоналу (оплата оренди офісу та оплата комунальних послуг); податкові витрати; оплата послуг юристів, бухгалтерів та прибиральників; виплати за контрактом із середовищем розробки для інтеграції цього методу.

Також канва бізнес-моделі включає наступні структурні блоки: прихована перевага (переваги, які неможливо скопіювати або придбати), ключові метрики (основні показники, що вимірюються) та канали (власне шляхи до користувачів), а саме:

1. Канали: через соціальні мережі, Міністерство освіти та науки.
2. Ключові метрики: кількість проданих ліцензій.

3. Прихована перевага: синтаксичний аналіз тексту для виділення основних характеристик програмного компоненту.

Бізнес-модуль представлений у зведеному вигляді у табл. 5.3.

Таблиця 5.3

Канва бізнес-моделі

Проблема	Рішення	Унікальна ціннісна пропозиція	Прихована перевага	Споживачі
відсутність автоматизованих рішень; відсутність зручної документації для нових технологій; людський фактор, який впливає на кількість друкарських помилок при розробленні програмного продукту.	програмне забезпечення, що генерує програмний код на основі синтаксичного аналізу природомовних текстових даних.	програмне забезпечення, що реалізує новий метод генерування програмного коду на основі опису основних характеристик розроблюваного компоненту чи модуля; зменшення часу для написання базової структури програмного модуля.	синтаксичний аналіз тексту для виділення основних характеристик програмного компоненту.	розробники програмного забезпечення.

Продовження табл. 5.3

<table>
<tr><td></td><td>Ключові метрики

кількість проданих ліцензій.</td><td></td><td>Канали

соціальні мережі;
Міністерство освіти та науки.</td><td></td></tr>
<tr><td colspan="3">Структура витрат

утримання персоналу для надання технічної підтримки (виплати заробітних плат, соціальних виплат);
утримання робочих місць для персоналу (оплата за оренду офісу та комунальні послуги);
податкові витрати;
оплата послуг юриста, бухгалтера, прибиральниці;
оплата контрактів із середовищами розробки для інтегрування даного методу.</td><td colspan="2">Потоки доходів

доходи від продажу ліцензій;
доходи від підтримки програмного забезпечення.</td></tr>
</table>

Отже, виходячи з даних у таблиці, можна зробити висновок, що запропонований проєкт реалізує метод генерації коду для шаблону компонента веб-додатку, описаний у роботі, і має перспективи для подальшого впровадження. Звичайно, цей аналіз не враховує всі ризики та фактори, такі як конкретні обставини оподаткування в країні / регіоні, де ведеться бізнес, але навіть наявних досліджень достатньо для прогнозування комерційного успіху товару та його прибутковості.

5.9. Висновки до розділу 5

У даному розділі було проведено аналіз поточної ситуації у сфері генерування програмного коду, виявлено наявні проблеми та підсумовано їх у відповідному дереві проблем. Окрім проблем, визначаються також основні зацікавлені сторони, які вирішують існуючі недоліки, і ступінь впливу цих сторін на проблему. В результаті було запропоновано конкурентне бізнес-рішення, яке задовольняє інтереси зацікавлених сторін, та визначено унікальну цінність пропонованого продукту.

Проаналізувавши майбутніх споживачів та дослідивши ринкові сегменти споживчого ринку спрогнозовано потенційний дохід і вартість реалізації продукції. В результаті була описана бізнес-модель, яка підтвердила доцільність продажу товару та передбачила його потенційну віддачу від інвестицій та прибутковість у майбутньому.

ВИСНОВКИ

У даній магістерській дисертації були виконані такі завдання:

1. Проаналізовані існуючі методи та системи автоматизованого генерування програмного коду, визначено їх переваги та недоліки.
2. Розглянута та проаналізована специфіка генерування коду шаблонів веб-застосунків, а саме React-компонентів.
3. Було розглянуто види React-компонентів, а саме: функціональні та класові компоненти, а також Router-компонент. Для кожного з них виділено ключові слова, які складають власне компонент, а саме: *function*, *class*, *export*, *default*, *return*, *render*, *router*.
4. Проаналізовано процеси аналізу даних та генерування програмного коду. Для отримання більш точних результатів аналізу вхідних даних запропоновано модифікувати даний етап.
5. Розроблено метод генерування коду шаблонів компонентів веб-застосунків.
6. Створено програмне забезпечення для генерування коду шаблонів компонентів веб-застосунків, що реалізує запропонований у роботі метод.
7. Проведено аналіз результатів роботи розробленого програмного забезпечення.

Наразі критерії оцінювання швидкості та якості роботи розробника або тестувальника ПЗ змінюються, тому кожен претендент повинен швидко реагувати на такі зміни. Однак у даній роботі було виділено лише генерування React-компонентів, тому одним із перспективних напрямків для наступних досліджень є вдосконалення та розширення створеного у даній роботі методу для можливості генерування компонентів будь-якими мовами.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Ricardo A. Automatic Inductive Programming [Електронний ресурс] / Aler Mur Ricardo // ICML – Режим доступу до ресурсу: http://www.sigplan.org/Conferences/GPCE/ – Дата доступу: жовтень 2020. – Назва з екрану.
2. Parnas D. L. Software Aspects of Strategic Defense Systems [Електронний ресурс] / Parnas – 1985 November. – Режим доступу до ресурсу: http://web.stanford.edu/class/cs99r/readings/parnas1.pdf – Дата доступу: жовтень 2020. – Назва з екрану.
3. Chun. On Software, or the Persistence of Visual Knowledge / Chun, Wendy. – Boston: Grey Room 18, 2004. – 30 с.
4. About Generative Programming [Електронний ресурс] – Режим доступу: https://scala-lms.github.io/tutorials/01_overview.html – Дата доступу: жовтень 2020. – Назва з екрану.
5. Cointe P. Towards Generative Programming / Cointe., 2005. – 315-325 с. – Дата доступу: жовтень 2020.
6. Generative Programming: Concepts and Experiences (GPCE) [Електронний ресурс] – Режим доступу: http://www.sigplan.org/Conferences/GPCE/ – Дата доступу: жовтень 2020. – Назва з екрану.
7. Wilcox J., Paying Too Much for Custom Application Development/ James Wilcox – March 2011.
8. Blokdyk G. Extensible Stylesheet Language XSL A Complete Guide Paperback / Gerardus Blokdyk., 2018. – 290 с.
9. Stevens P. Using UML: Software Engineering with Objects and Components / P. Stevens, R. Pooley., 1999. – 256 с.
10. Chadwick J. Programming Razor: Tools for Templates in ASP.NET MVC or WebMatrix / J. Chadwick, R. Pooley., 2011. – 120 с.

11. Metadrone [Електронний ресурс] – Режим доступу: http://www.metadrone.com/documentation/ – Дата доступу: жовтень 2020. – Назва з екрану.
12. Child A. The Regenerator / A. Child, R. Pooley., 2015. – 198 с.
13. Hazzard K. Metaprogramming in .NET / K. Hazzard, J. Bock., 2012. – 360 с.
14. Radzen [Електронний ресурс] – Режим доступу: https://www.apriorit.com/dev-blog/629-web-radzen-app-development-platform-review – Дата доступу: жовтень 2020. – Назва з екрану.
15. CodeSmith [Електронний ресурс] – Режим доступу: https://codesmith.io/ – Дата доступу: жовтень 2019. – Назва з екрану.
16. ASP.NET [Електронний ресурс] – Режим доступу: https://aspnetzero.com/ – Дата доступу: жовтень 2019. – Назва з екрану.
17. Mandic D. Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability / D. Mandic, J. Chambers., 2017. – 102 с.
18. K.S.Tai. Improved semantic representations from tree-structured long short-term memory networks. In Proceedings of the Annual Meeting of the Association for Computational Linguistics / K.S.Tai, R.Socher, C.D.Manning., 2015.
19. S. Karaivanov, V. Raychev, and M. Vechev. Phrase-based statistical translation of programming languages. In Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, pages 173–184. ACM, 2014.
20. T. Nguyen, T. T. Nguyen, and T. N. Nguyen. Lexical statistical machine translation for language migration. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pages 651–654. ACM, 2013.
21. T. Nguyen, T. T. Nguyen, and T. N. Nguyen. Divide-and-conquer approach for multi-phase statistical migration for source code (t). In Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, pages 585–596. IEEE, 2015.

22. D. He, Y. Xia, T. Qin, L. Wang, N. Yu, T. Liu, and W.-Y. Ma. Dual learning for machine translation. In Advances in Neural Information Processing Systems, pages 820–828, 2016.
23. X. Chen, C. Liu, E. C. Shin, D. Song, and M. Chen. Latent attention for if-then program synthesis. In Advances in Neural Information Processing Systems, 4574–4582 c., 2016.
24. McKinney, Python for Data Analysis [Text]/Wes McKinney. – O'Reilly, 2017. – 10-26p.
25. NUMPY [Електронний ресурс] – Режим доступу: https://numpy.org/ – Дата доступу: березень 2020. – Назва з екрану.
26. Pandas [Електронний ресурс] – Режим доступу: pandas.pydata.org – Дата доступу: березень 2020. – Назва з екрану.
27. Scikit-learn. Machine learning for Python [Електронний ресурс] – Режим доступу: https://scikit-learn.org/stable/ – Дата доступу: березень 2020. – Назва з екрану.
28. Dr. Axel Rauschmayer. Speaking JavaScript /Dr. Axel Rauschmayer, 2014. – 460 c.
29. Сарницкий Я. ES6, ES8, ES2017: что такое ECMAScript и чем это отличается от JavaScript [Електронний ресурс] / Ярослав Сарницкий. – 2017. – Режим доступу до ресурсу: https://tproger.ru/translations/wtf-is-ecmascript/.
30. Sidelnikov G. React.js Book: Learning React JavaScript Library From Scratch / Greg Sidelnikov., 2017. – 102 c.

ДОДАТКИ

Додаток 1

Копії графічних матеріалів

Дерево проблем

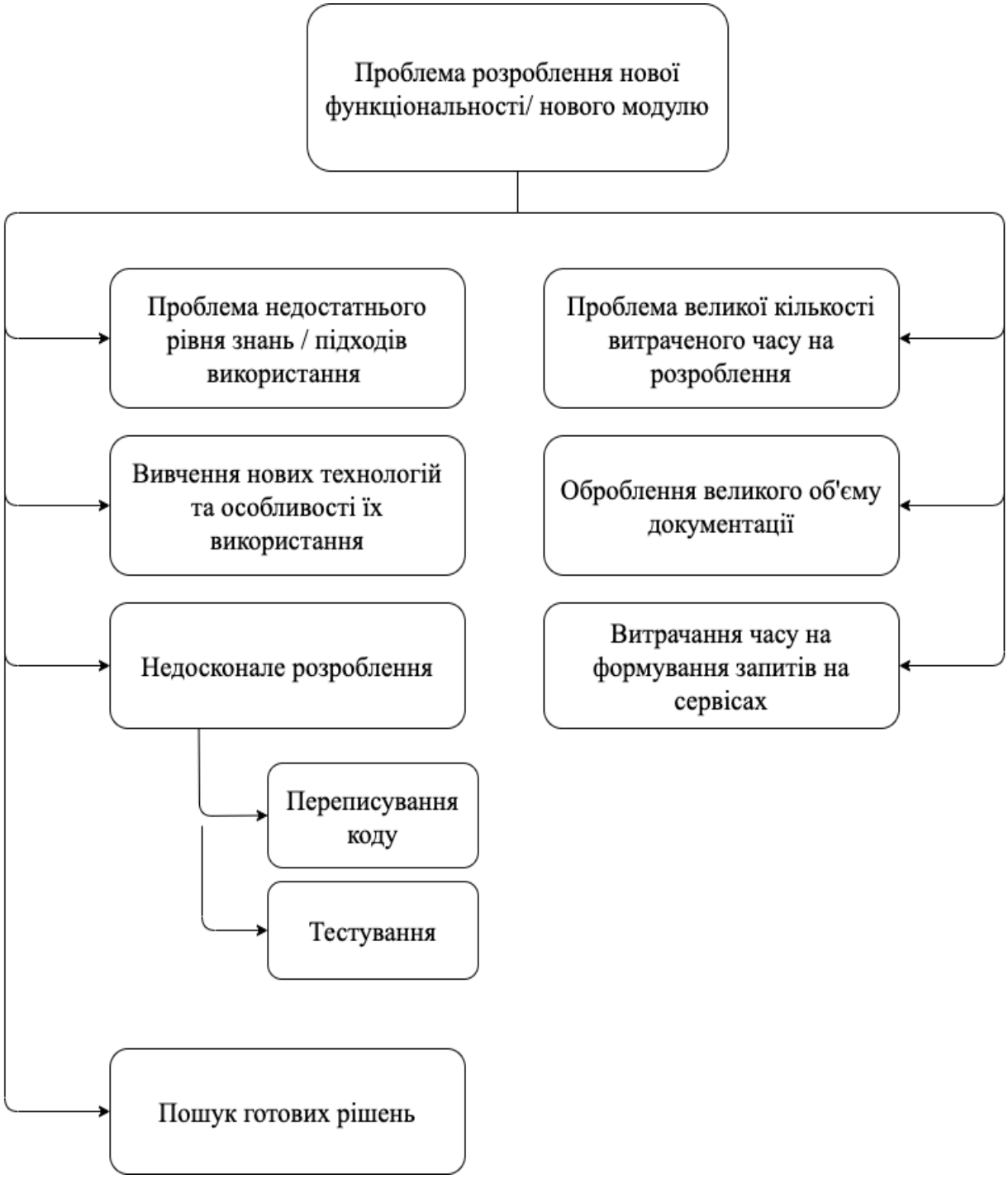
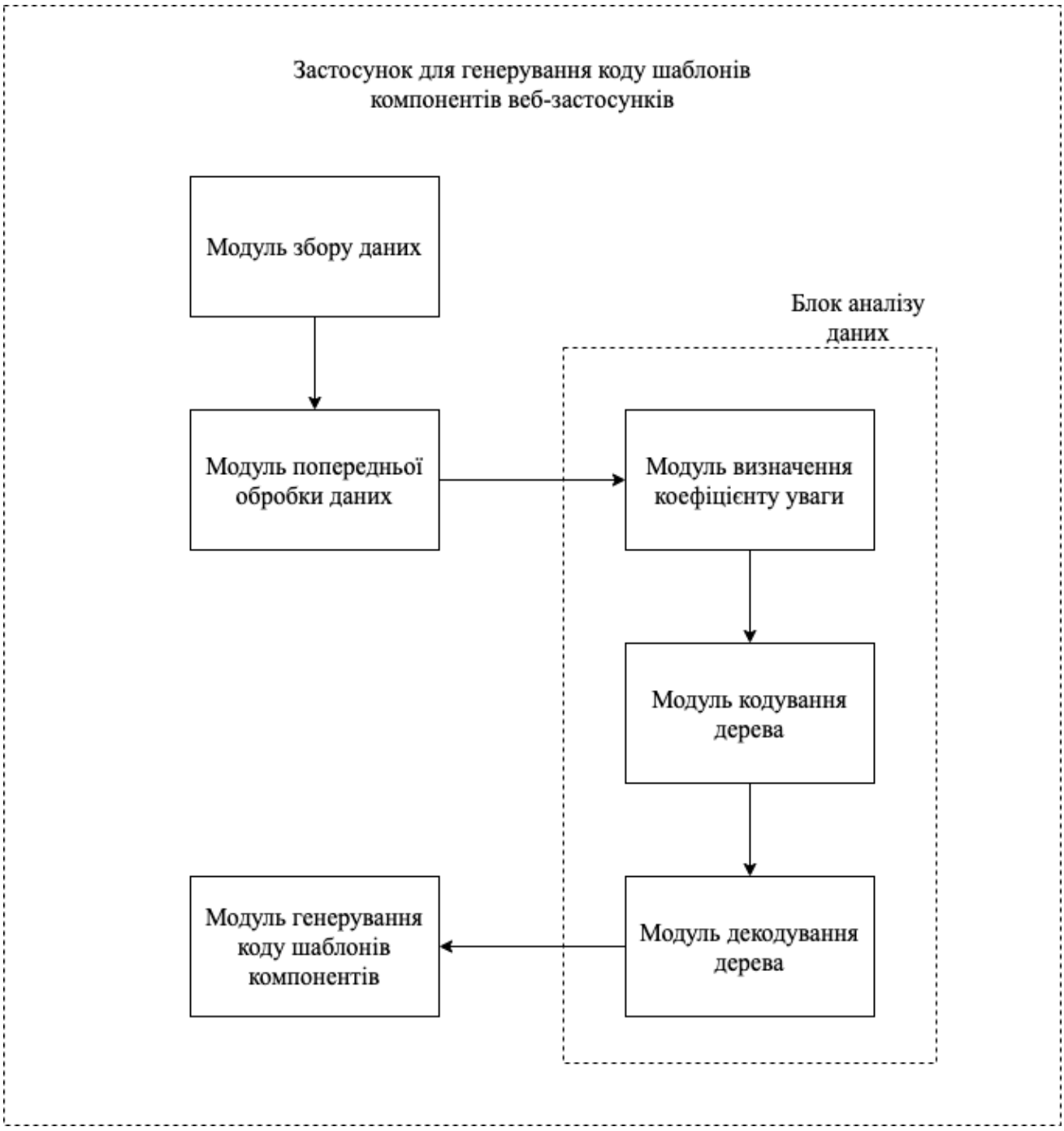
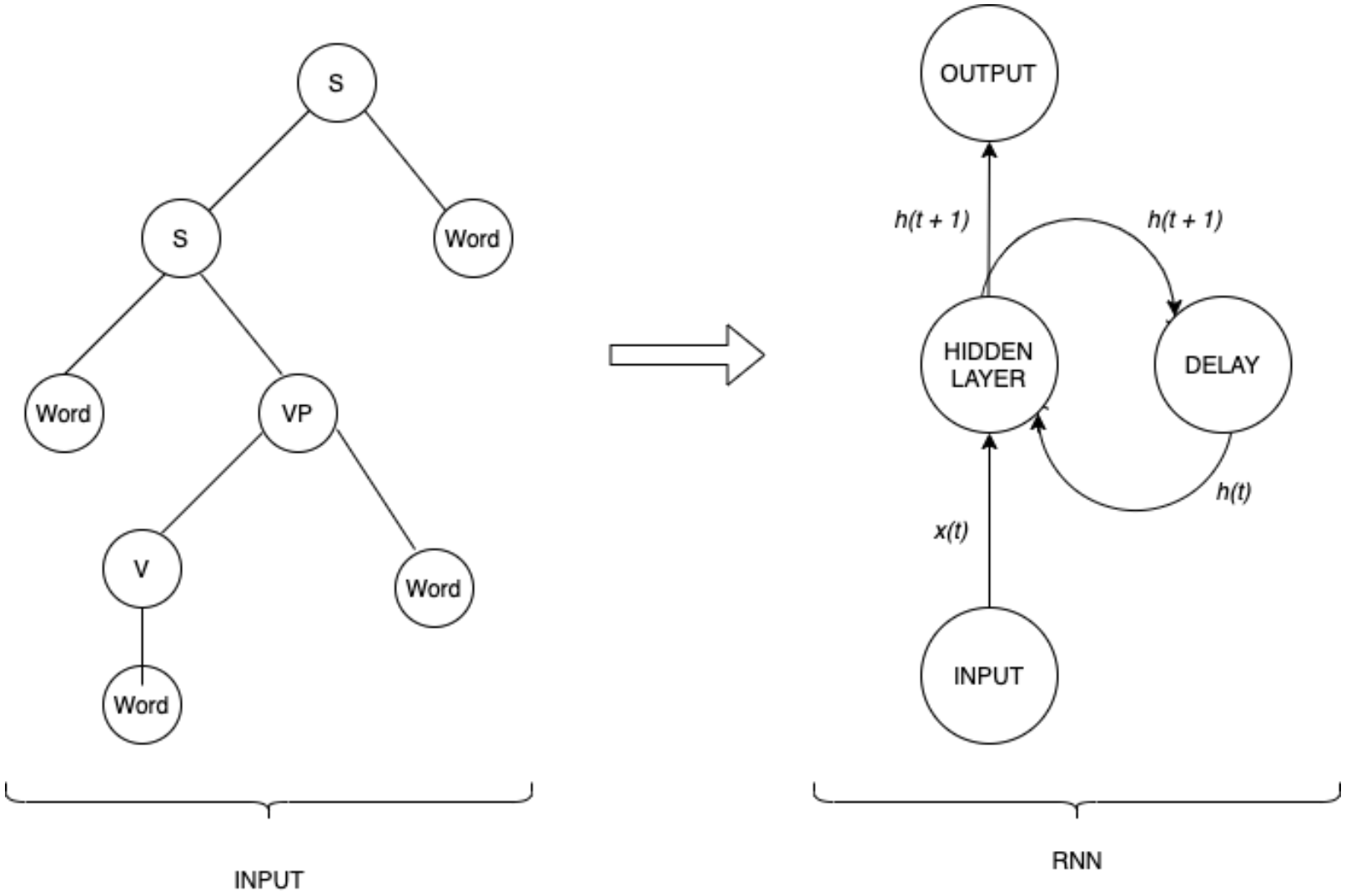


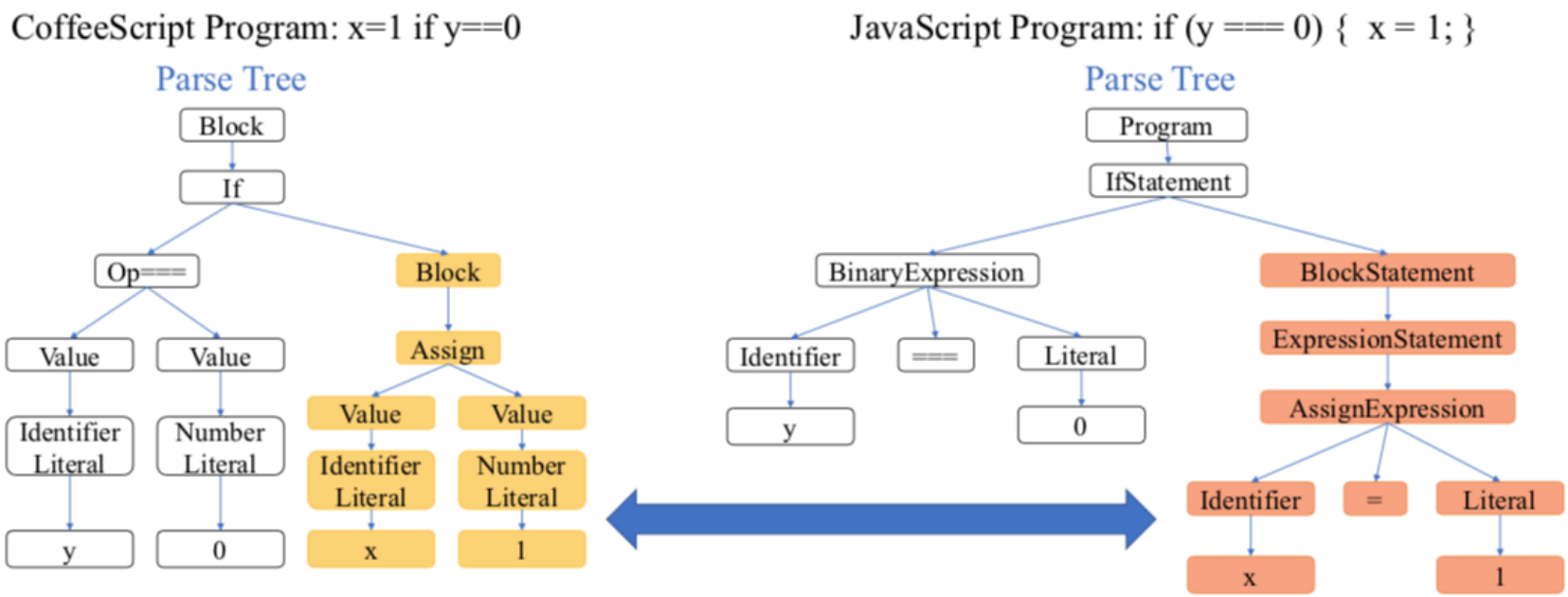
Схема архітектури системи



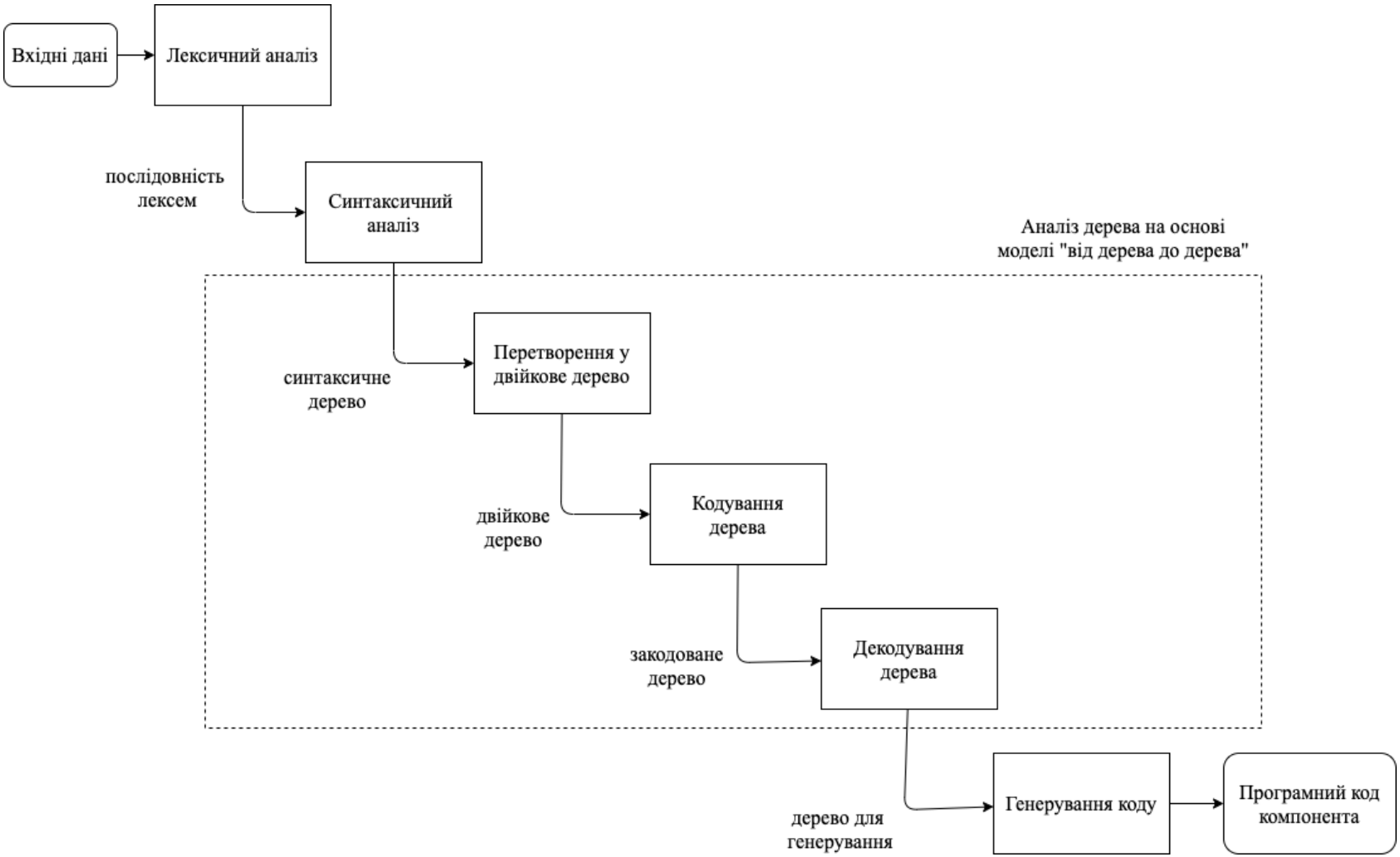
Узагальнена схема методу генерування коду шаблонів програмних компонентів



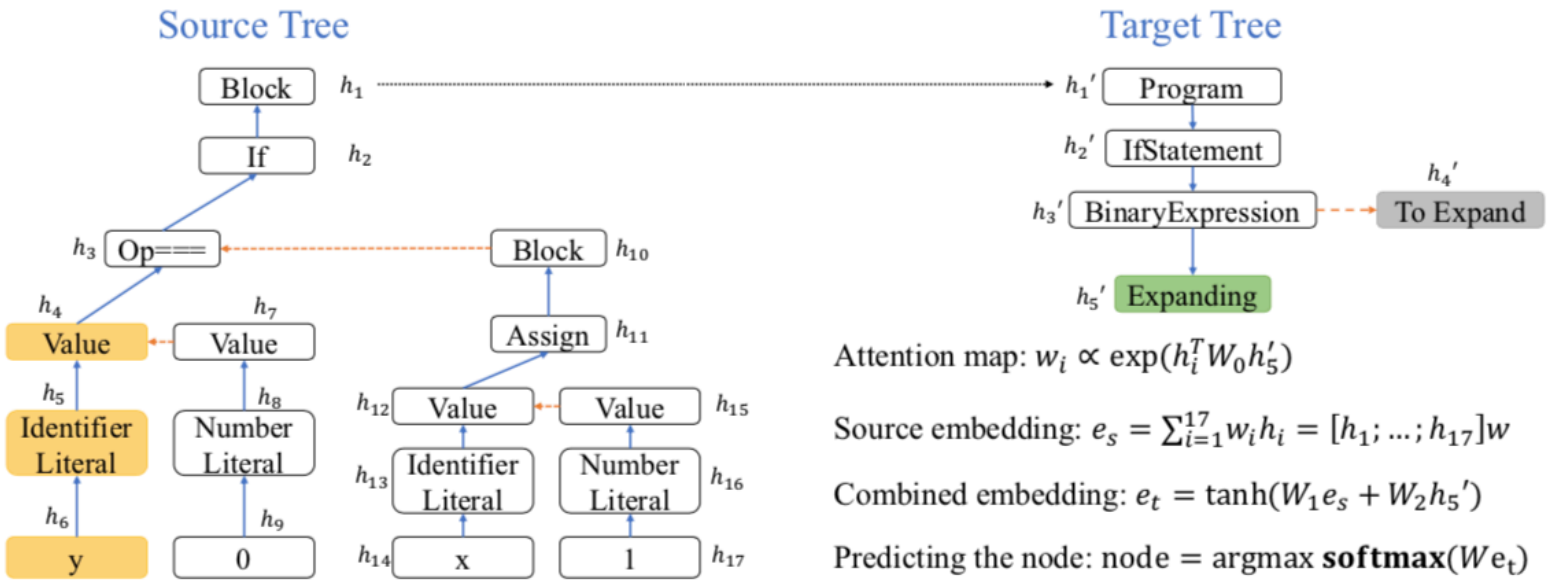
Графічне представлення перекладу програми у JavaScript



Графічне представлення послідовності виконання кроків методу



Графічне представлення роботи системи на основі моделі від дерева до дерева



Додаток 2

Лістинг програми

Лістинг 1. Модуль генерування програмного коду

```
const path = require("path");
const fs = require("fs");
const chalk = require("chalk");
const async = require("async");
const metalsmith = require("metalsmith");
const render = require("consolidate").handlebars.render;
const toSlugCase = require("to-slug-case");
const toCamelCase = require("to-camel-case");
const toPascalCase = require("to-pascal-case");
const toSnakeCase = require("to-snake-case");
const toSpaceCase = require("to-space-case");
const isTextOrBinary = require("istextorbinary");

module.exports = function generate(type, options, settings) {
  if (settings.templatePath == "") {
    settings.templatePath = path.join(__dirname, "../template");
  }

  if (!pathExists(settings.templatePath)) {
    console.log();
    console.error(
      chalk.red(
        `Template folder (${path.resolve(settings.templatePath)}) doesn't
exist`
      )
    );
    return;
  }

  const fullTemplatePath = path.join(settings.templatePath, "/" + type);

  if (!pathExists(fullTemplatePath)) {
    console.log();
    console.log(
      chalk.red(
        `'${options.type}' template folder doesn't exist in ${path.resolve(
          settings.templatePath
        )}`
      )
    );
    return;
  }

  console.log();
  console.log(
    chalk.green(
      chalk.bold(
        `Generating files from '${type}' template with name:
${options.name}`
      )
    )
  );

  metalsmith(fullTemplatePath)
    .metadata(Object.assign({}, getNames(options.name)))
    .source(".")
    .destination(path.resolve(options.destination))
    .clean(false)
    .use(renderPaths)
    .use(renderTemplates)
    .build(function(err) {
```

```
      if (err) {
        console.error(chalk.red(err));
      } else {
        console.log();
        console.log(chalk.green("Done!"));
      }
    });
};

function getNames(name) {
  return {
    name,
    name_cc: toCamelCase(name),
    name_pc: toPascalCase(name),
    name_sc: toSlugCase(name),
    name_uc: toUpperCase(name),
    name_lc: toLowerCase(name),
    name_sn: toSnakeCase(name),
    name_sp: toSpaceCase(name)
  };
}

function toUpperCase(name) {
  return name.toUpperCase();
}

function toLowerCase(name) {
  return name.toLowerCase();
}

function pathExists(value) {
  return fs.existsSync(path.resolve(value));
}

function renderPaths(files, metalsmith, done) {
  const keys = Object.keys(files);
  const metadata = metalsmith.metadata();

  keys.forEach(key => {
    let newKey = replaceVars(key, metadata);

    if (newKey != key) {
      files[newKey] = files[key];
      delete files[key];
    }
  });

  done();
}

function renderTemplates(files, metalsmith, done) {
  const keys = Object.keys(files);
  const metadata = metalsmith.metadata();

  async.each(keys, run, done);

  function run(file, done) {
    if (
      isTextOrBinary.isBinarySync(path.basename(file),
files[file].contents)
    ) {
      done();
      return;
```

```
    }

    let str = files[file].contents.toString();
    render(str, metadata, function(err, res) {
      if (err) {
        return done(err);
      }
      files[file].contents = new Buffer(res);
      done();
    });
  }
}

function replaceVars(value, object) {
  return value.replace(/\$?\{([@#$%&\w\.]*)(\((.*?)\))?\}/gi, (match, name)
=> {
    const props = name.split(".");
    const prop = props.shift();
    let o = object;

    if (o != null && prop in o) {
      return o[prop];
    }
    return "";
  });
}
```

Лістинг 2. Модуль кодування

```
class Encoder(nn.Module):
    def __init__(self, in_dim, h_dim, embedding_size):
        super().__init__()
        self.initial_h = 0
        self.initial_c = 0
        self.in_dim = in_dim
        self.h_dim = h_dim

        # Dropout Layer (may be useful)
        self.drop = nn.Dropout(p=0.5, inplace=False)

        # Binary LSTM cell and embedding layer
        self.tree_cell = BinaryTreeLSTMCell(in_dim, h_dim)
        self.embed = nn.Embedding(in_dim, embedding_size)

    # compute embeddings for source tree and subtrees
    def forward(self, batch):
        #
        binary_cell = BinaryTreeLSTMCell(in_dim, h_dim)

        # iterate through each tree in batch
        for tree in batch:
            # hidden state
            hr = 0
            hl = 0

            # cell state
            cr = 0
            cl = 0

            # iterate postorder over the tree, passing each layer to the
lstm cell
            current = 0
            nodes_stack = []

            while(True):
                # while root is not empty
                while (tree.successors(current).size() != 0):
                    nodes_stack.append(tree.successors(current[1]))
                    nodes_stack.append(current)

                current = tree.successors(current)[0]
                current = nodes_stack.pop()

                if (tree.successors.size() != 1 and
tree.successors(current)[1] in nodes_stack):
                    nodes_stack.pop()
                    nodes_stack.append(current)
                    current = tree.successors(current)[1]
                else:
                    # run binary lstm for node
                    x = tree[current].ndata['info']
                    h, c = binary_cell.forward(x, hl, hr, cl, cr)
                    tree[current].ndata['e'] = self.embed(x)
                    tree[current].ndata['h'] = h
                    tree[current].ndata['c'] = c
                # stack is empty
                if (len(nodes_stack) == 0):
                       break
```

Лістинг 3. Модуль декодування

```
# Decoder generates the target tree starting from a single root node
class Decoder(nn.Module):
    def __init__(self, e_t, h_dim, vocab_size):
        super().__init__()
        # trainable matrix of vocab size of outputs and embedding dimension
        self.W_tt = nn.Linear(h_dim, vocab_size)
        self.B_t = nn.Linear(h_dim, vocab_size)

        # attention mechanism
        self.attention = Attention(h_dim)

    # generate target tree from source tree
    def forward(self, batch):
        for tree in batch:
            # make tree with one node
            target_tree = dgl.DGLGraph(1)

            # copy LSTM state from encoder of root of source tree and
attach to root of target tree until empty list
            target_tree[0].ndata['h'] = tree[0].ndata['h']

            # initialize expanding node queue
            nodes_queue = [0]
            current = 0

            # stop if there are no nodes left to expand
            while (nodes_queue):
                # current node is the first one in queue
                current = nodes_queue.pop(0)

                # compute e_t
                e_t = attention.forward(tree,
target_tree[current].ndata['h'])

                # feed it into softmax regression network to get our token
                t_t = th.max(F.softmax(W_tt(e_t)))

                # if t_t isn't EOS, make two children nodes
                if (t_t != "EOS"):
                    # make two children
                    target_tree.add_nodes(2)
                    target_tree([current, current], [len(target_tree) - 1,
len(target_tree) - 2])

                    # add children to queue

nodes_queue.append(target_tree[current].successors()[0])

nodes_queue.append(target_tree[current].successors()[1])
```

Лістинг 4. Модуль визначення коефіцієнту уваги

```
# Attention class to locate the source sub-tree
class Attention(nn.Module):
    def __init__(self, h_dim):
        # Weights matrices of size d * d (d is the embedding dimension)
        self.h_dim = h_dim
        W_0 = nn.Linear(h_dim, h_dim)
        W_1 = nn.Linear(h_dim, h_dim)
        W_2 = nn.Linear(h_dim, h_dim)

    # get the source tree
    def forward(self, tree, h_t):
        # calculate probability while doing post-order traversal through
tree
        current = 0
        nodes_stack = []

        while(True):

            # expectation
            e_s = th.zeros(h_dim)

            # while root is not empty
            while (tree.successors(current).size() != 0):
                nodes_stack.append(tree.successors(current[1]))
                nodes_stack.append(current)

                current = tree.successors(current)[0]
                current = nodes_stack.pop()

                if (tree.successors.size() != 1 and
tree.successors(current)[1] in nodes_stack):
                    nodes_stack.pop()
                    nodes_stack.append(current)
                    current = tree.successors(current)[1]

                else:
                    # calculate probability
                    p = th.exp(tree[current].ndata['h'].transpose() *
self.W_0(h_t))

                    # compute expectation of h_t to be throughout all the
nodes in the tree
                    e_s += tree[current].ndata['h'] * p

                if (len(nodes_stack) == 0):
                    break

            # compute e_t by combining W_1, W_2, e_s, and h_t and pass
through activation function tanh
            e_t = F.tanh(self.W_1(e_s) + self.W_2(h_t))

            return e_t
```

Лістинг 5. Модуль попередньої обробки даних

```
class BinaryTreeLSTMCell(nn.Module):
    def __init__(self, in_dim, h_dim):
        super().__init__()
        self.iou_x = nn.Linear(in_dim, h_dim * 3)         # i, o, u
matrices for x (cell state)
        self.iou_hl = nn.Linear(h_dim, h_dim * 3)         # i, o, u
matrices for left h (hidden state)
        self.iou_hr = nn.Linear(h_dim, h_dim * 3)         # i, o, u
matrices for right h (hidden state)
        self.f_x = nn.Linear(in_dim, h_dim)               # forget for x

        # forget for hidden state
        self.f_h = nn.ModuleList([[nn.Linear(h_dim, h_dim),
nn.Linear(h_dim, h_dim)],
                                  [nn.Linear(h_dim, h_dim), nn.Linear(h_dim,
h_dim)]])

    # takes in input, cell states, and hidden states
    def forward(self, x, hl, hr, cl, cr):
        # i, o, u, gates
        self.iou = self.iou_x(x) + self.iou_hl(hl) + self.iou_hr(hr)

        # split
        i, o, u = th.split(iou, iou.size(1) // 3, dim=1)

        # apply activation functions
        i = F.sigmoid(i)
        o = F.sigmoid(o)
        u = F.tanh(u)

        # forget for left and right
        fl = F.sigmoid(self.f_x(x) + self.f_h[0][0](hr) +
self.f_h[0][1](hl))
        fr = F.sigmoid(self.f_x(x) + self.f_h[1][0](hr) +
self.f_h[1][1](hl))

        # calculate hidden state and cell state
        c = i * u + fl * cl + fr * cr
        h = o * F.tanh(c)

        # return hidden state and cell state
        return h, c
```

Додаток 3

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

АЛГОРИТМІЧНО-ПРОГРАМНИЙ КОМБІНОВАНИЙ МЕТОД ГЕНЕРАЦІЇ КОДУ ШАБЛОНІВ КОМПОНЕНТІВ ВЕБ-ЗАСТОСУНКІВ

Виконала: Довганюк Ліна Олегівна

Науковий керівник: доц., к.т.н. Заболотня Т.М.

Київ – 2020

Постановка задачі

Метою дослідження є підвищення ефективності автоматизованого створення програмного коду складових веб-застосунків за рахунок розроблення та реалізації методу генерування коду шаблонів веб-компонентів та відповідного програмного забезпечення.

Об’єктом дослідження є процес генерування програмного коду шаблонів компонентів веб-застосунків.

Предметом дослідження є методи і алгоритми генерації програмного коду.

Актуальність

1. Зростання обсягів розроблюваного програмного забезпечення

2. Підвищення рівня вимог до швидкості написання коду

3. Пошук та подальша модифікація готових рішень з точки зору наявної задачі

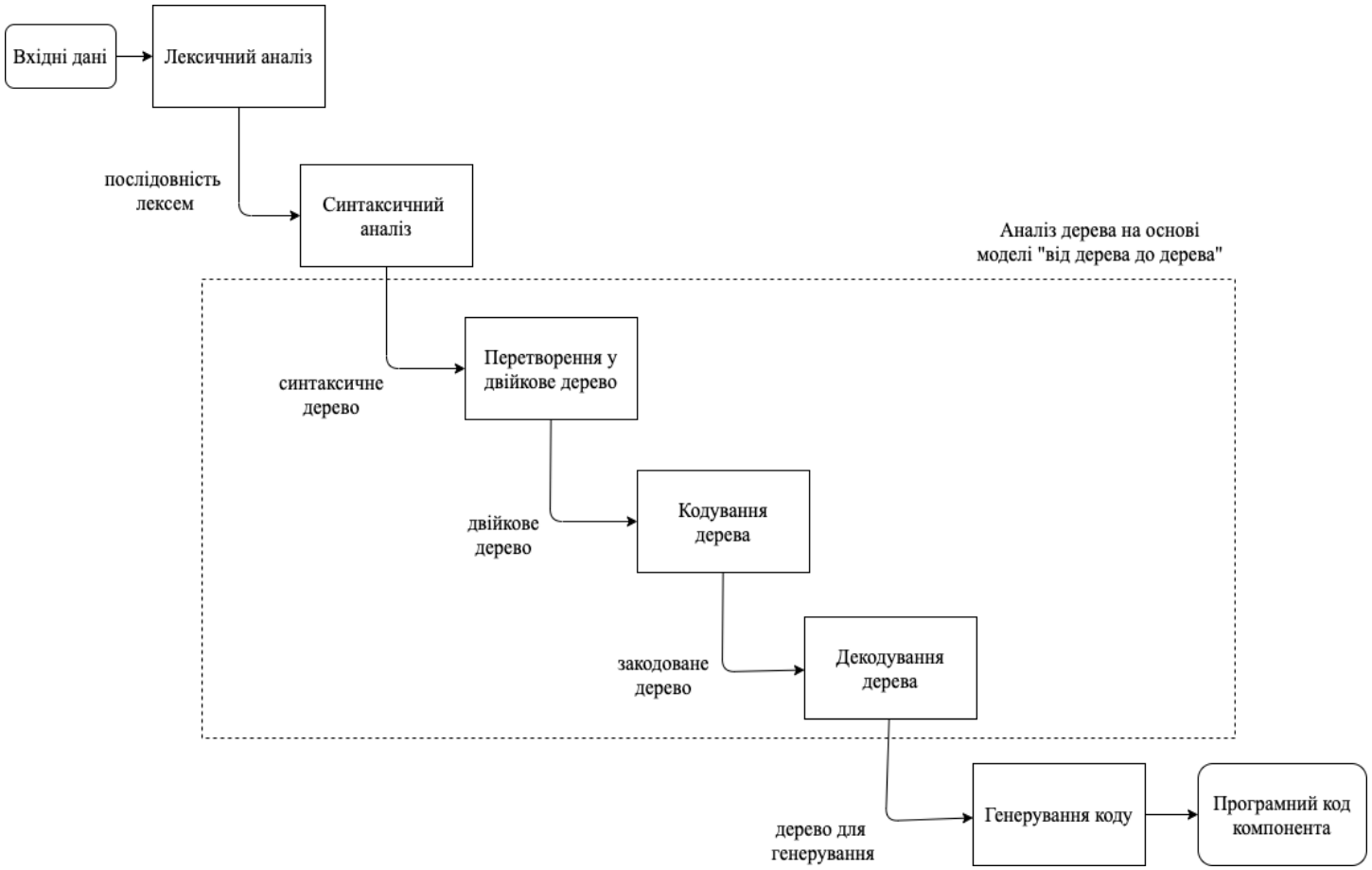
4. Підвищення рівня вимог до якості отримуваних застосунків

Порівняння існуючих рішень

Назва рішення	Ресурсоємність	Можливість налаштування	Орієнтованість на веб-застосунки	Універсальність
XSLT	+	+	+	+/–
UML	–	+	+/–	+/–
Razor Generator	–	–	–	–
Metadrone	+	+	–	–
Reegenerator	+	+	–	+
T4	+	+	–	–
Radzen	–	+	+	+/–
Генератор CodeSmith	–	+	–	+
ASP.Net Zero	+	–	+/–	+

Запропонований метод генерації коду шаблонів компонентів

Загальна схема методу



Аналіз вхідних даних. Подання у вигляді послідовності символів або токенів

1. При використанні нейронних мереж генерування синтаксично правильних програм стає більш складнимта довготривалим процесом при збільшенні довжини послідовностей.

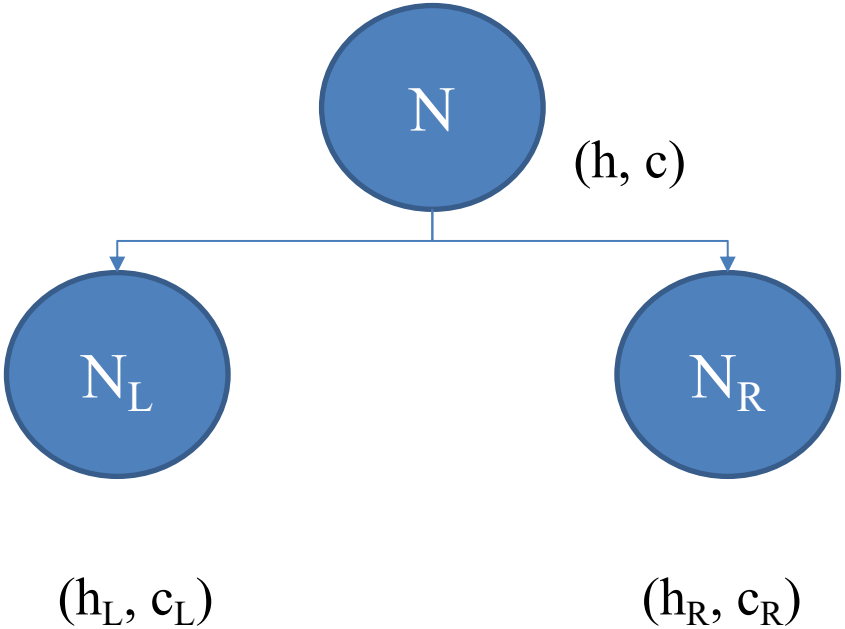
Аналіз вхідних даних. Подання у вигляді дерева

1. Дозволяє аналізувати вхідні дані за допомогою рекурсивних нейронних мереж.

2. Для детального аналізу даних використовується механізм уваги.

Кодування дерева

$$(h,\ c)\ =\ LSTM(([h_L;h_R],[c_L;c_R]),\ x)$$



x – вектор батьківського вузла

(h_L, c_L) – стан лівого дочірнього вузла

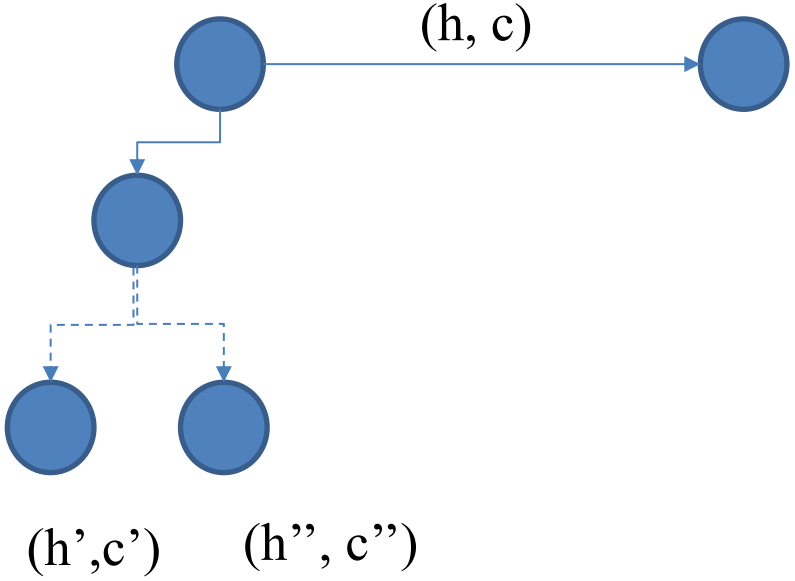
(h_R, c_R) – стан правого дочірнього вузла

Декодування дерева

$$t_t = argmax\, softmax(W_{e_t})$$

$$(h',\ c') = LSTM_L((h,\ c),\ B_{t_t})$$

$$(h'',\ c'') = LSTM_R((h,\ c),\ B_{t_t})$$



W – навчальна матриця розмірності $d \times V_t$

V_t – розмір словника вхідних даних

d – розмірність вектора

B – матриця для вектора слова розмірністю $d \times V_t$

Механізм уваги для пошуку вихідного дерева

$$P\left(N_s - \textit{піддерево джерела, що відповідає } N_t \mid N_t\right)$$

$$P\left(N_s \mid N_t\right) \propto exp\left(h_s^T W_0 ht\right)$$

$$e_s = E[h_{N_s} \mid N_T] = \sum_{N_s} h_{N_s} * P(N_s \mid N_t)$$

$$e_t = tanh(W_1 e_s + W_2 h)$$

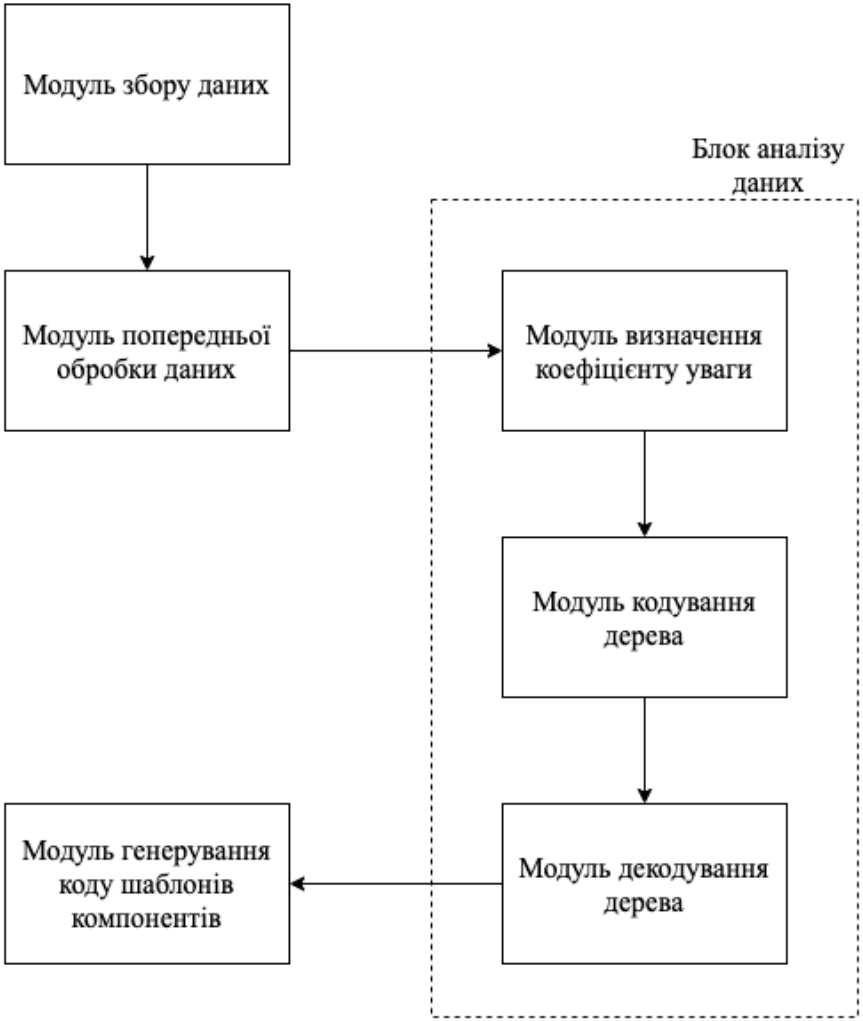
Механізм уваги. Батьківський механізм уваги

$$(h', c') = LSTM_L((h, c), [B_{t_t}; e_t])$$

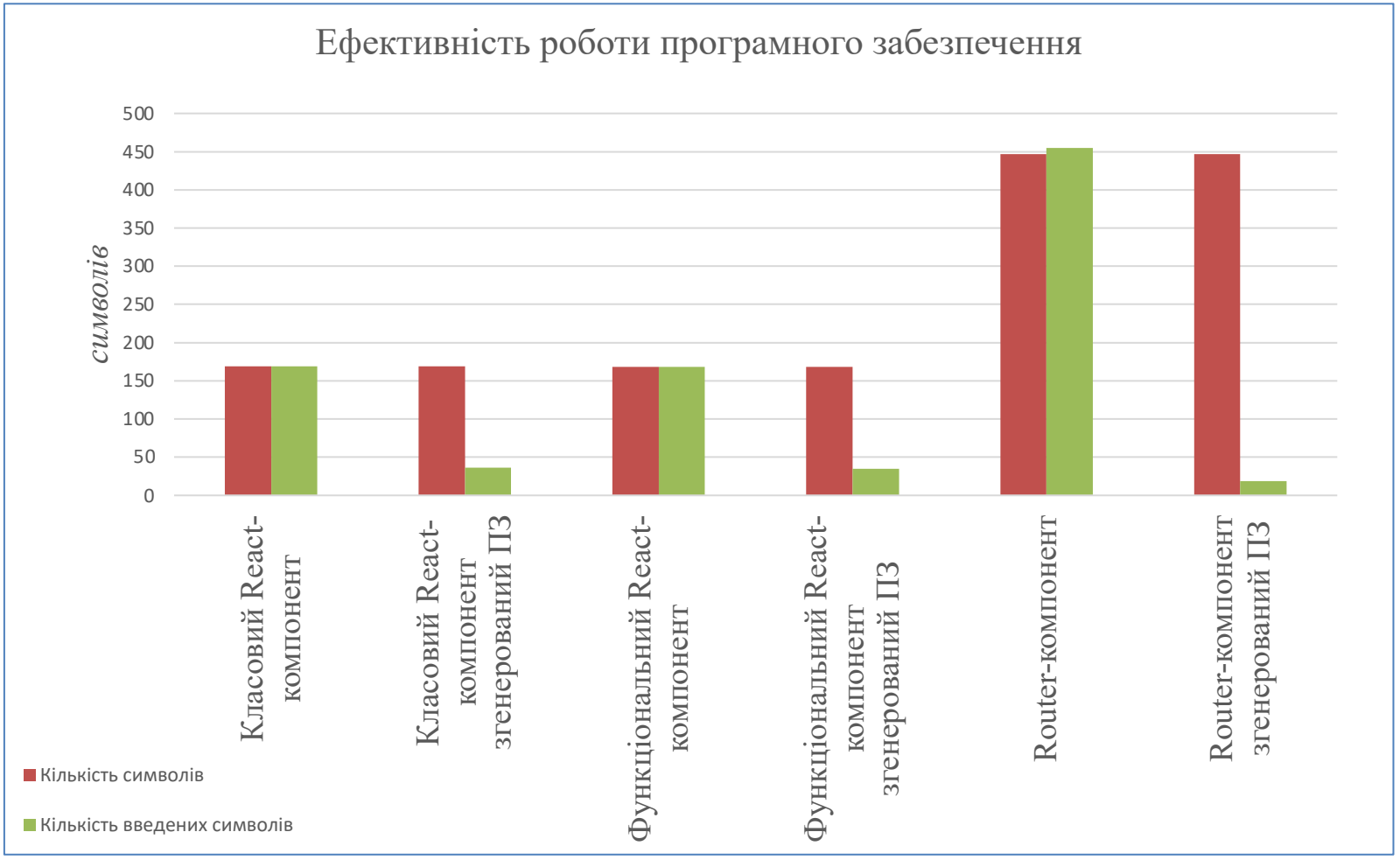
$$(h'', c'') = LSTM_R((h, c), [B_{t_t}; e_t])$$

Особливості реалізації програмного забезпечення

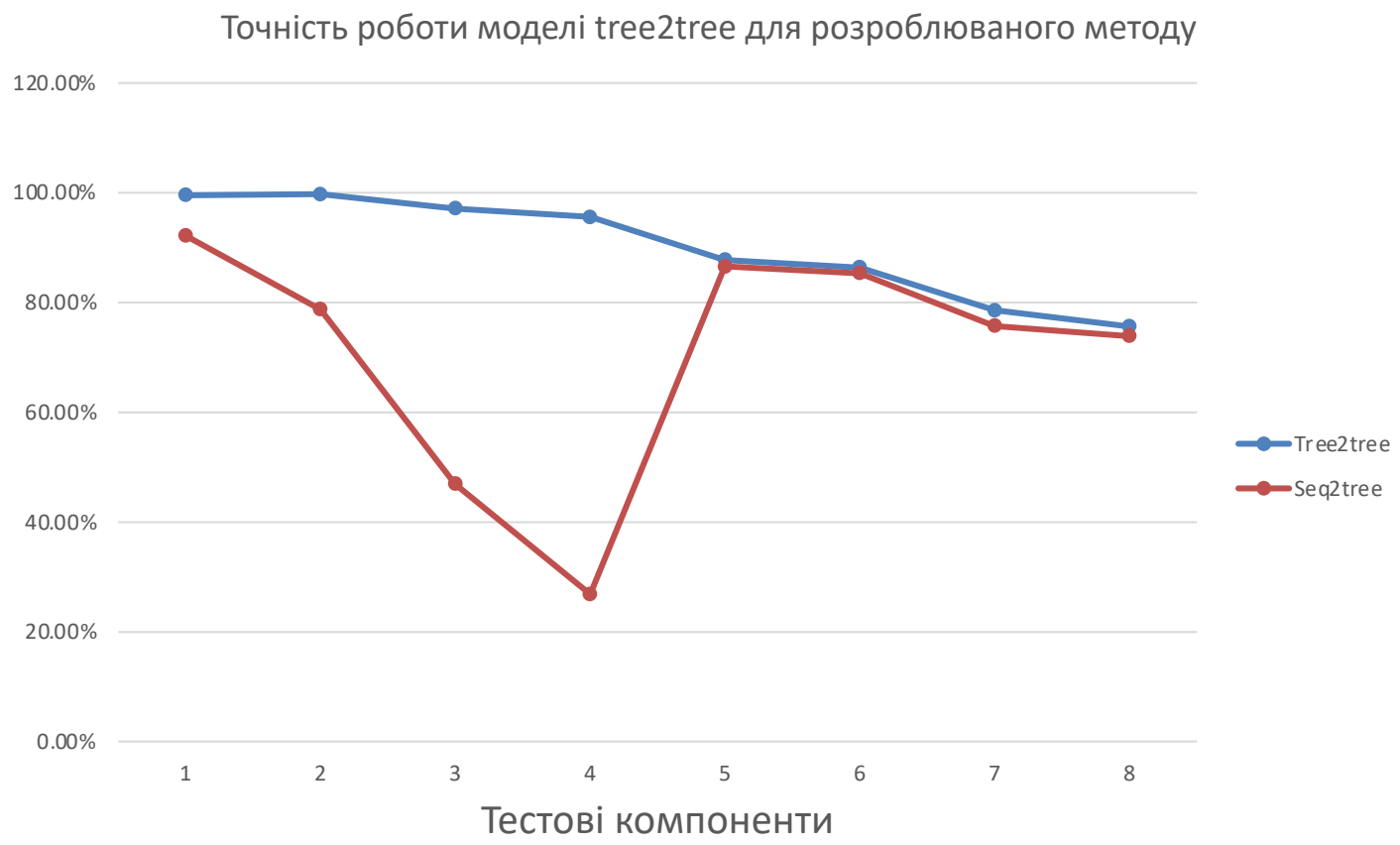
Загальна архітектура системи



Аналіз розробленого програмного забезпечення



Аналіз розробленого програмного забезпечення



Побудова бізнес-моделі

Канва бізнес-моделі

Проблема	**Рішення**	**Унікальна ціннісна пропозиція**	**Прихована перевага**	**Споживачі**
відсутність автоматизованих рішень; зручної документації для нових технологій; людський фактор, який впливає на кількість друкарських помилок.	програмне забезпечення, що генерує програмний код на основі синтаксичного аналізу природомовних текстових даних	програмне забезпечення, що реалізує метод генерації коду шаблонів компонентів, зменшення часу для написання базової структури програмного компонента	Синтаксичний аналіз тексту для виділення основних характеристик програмного компоненту	розробники програмного забезпечення
	Ключові метрики кількість проданих ліцензій		**Канали** соціальні мережі; МОН України	
Структура витрат утримання персоналу для надання технічної підтримки (виплати заробітних плат, соціальних виплат); утримання робочих місць для персоналу (оплата за оренду офісу та комунальні послуги); податкові витрати; оплата послуг юриста, бухгалтера, прибиральниці; оплата контрактів із середовищами розробки для інтегрування даного методу.			**Потоки доходів** доходи від продажу ліцензій; доходи від підтримки програмного забезпечення	

Фінансовий план

	1-й	2-й	3-й	4-й	5-й	6-й	7-й	8-й	9-й	10-й	11-й	12-й	Загальні результати
Заробітна плата, тисяч $	2	8	8	8	8	8	20	20	20	20	20	20	162
Інші витрати, тисяч $	1	4	4	4	4	4	4	4	4	4	4	4	45
Сума витрат, тисяч $	3	12	12	12	12	12	24	24	24	24	24	24	207
Заплановані прибутки, тисяч $	-	-	-	-	-	-	30	40	60	60	60	60	310
Результат исяч $	–3	–12	–12	–12	–12	–12	6	16	36	36	36	36	103

Наукова новизна

Вперше запропоновано метод організації процедури автоматизованого створення програмного коду складових веб-застосунків, який полягає у синтаксичному аналізі текстового опису майбутнього компонента та подання результату аналізу до моделі «tree2tree» з батьківським механізмом уваги.

Це дозволяє підвищити ефективність та швидкість написання програмного коду за рахунок зменшення кількості символів для введення.

Практичне значення

Практичне значення одержаних результатів визначається тим, що запропонований метод генерування програмного коду складових веб-застосунків дозволив підвищити ефективність та швидкість написання коду та розробити відповідне програмне забезпечення, придатне до використання у вигляді бібліотеки.

ВИСНОВКИ

1. Проаналізовано існуючі методи та системи автоматизованого генерування програмного коду, визначено їх переваги та недоліки; проаналізована специфіка генерування коду шаблонів веб-застосунків, а саме React-компонентів.
2. Було розглянуто види React-компонентів. Для кожного з них виділено ключові слова, які складають власне компонент, а саме: function, class, export, default, return, render, router.
3. Проаналізовано процеси аналізу даних та генерування програмного коду. Для отримання більш точних результатів аналізу вхідних даних запропоновано модифікувати даний етап.

ВИСНОВКИ

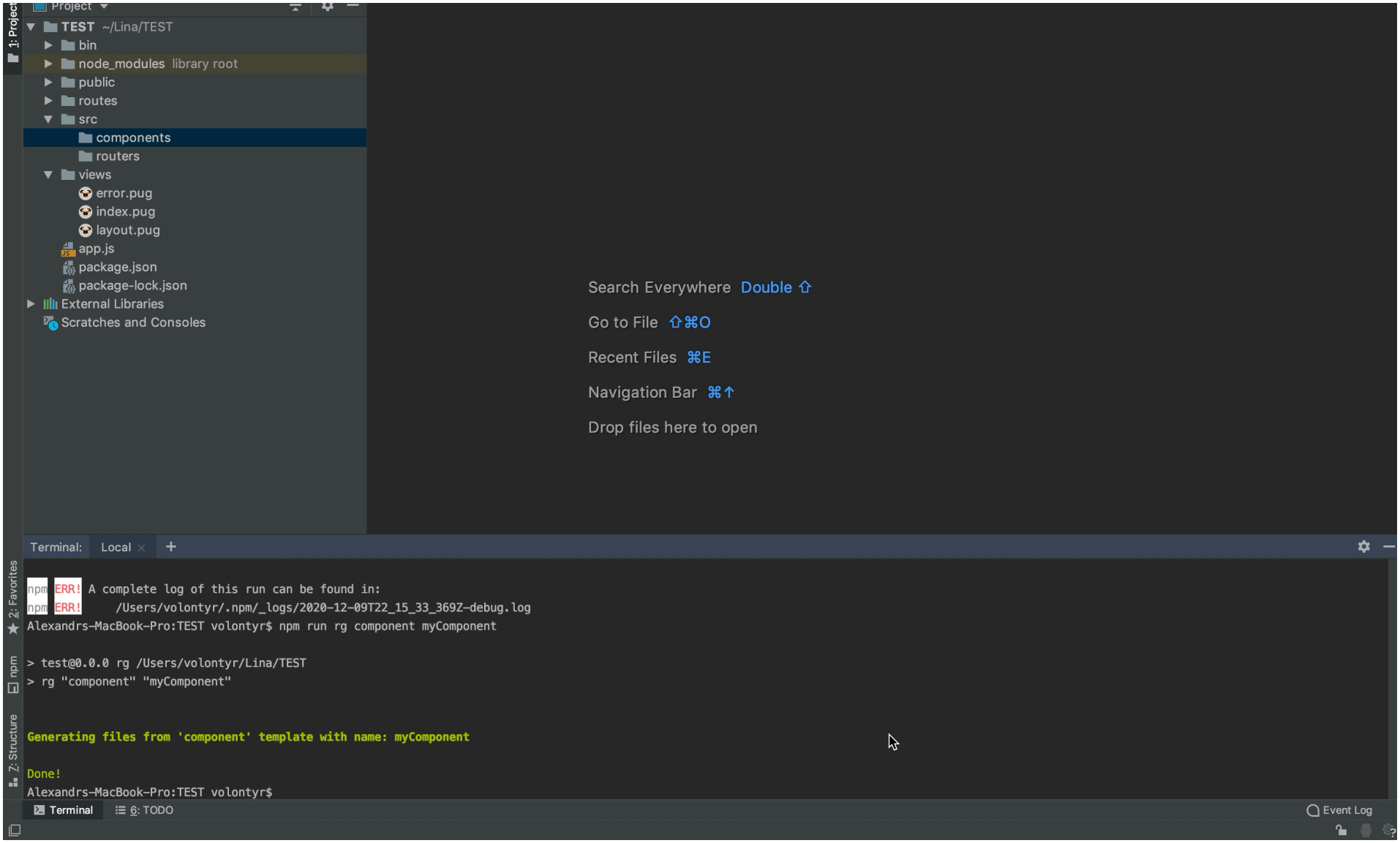
4. Розроблено метод генерування коду шаблонів компонентів веб-застосунків.
5. Створено програмне забезпечення для генерування коду шаблонів компонентів веб-застосунків, що реалізує запропонований у роботі метод.
6. Проведено аналіз результатів роботи розробленого програмного забезпечення

Апробація роботи

1. XIII наукова конференція магістрантів та аспірантів “Прикладна математика та комп’ютинг” ПМК-2020.

2. IX Міжнародна науково-практична конференція «Проблеми інформатики та комп’ютерної техніки» ПІКТ 2020.

Дякую за увагу!


Project
TEST ~/Lina/TEST
bin
node_modules library root
public
routes
src
components
routers
views
error.pug
index.pug
layout.pug
app.js
package.json
package-lock.json
External Libraries
Scratches and Consoles
Search Everywhere Double ⇧
Go to File ⇧⌘O
Recent Files ⌘E
Navigation Bar ⌘↑
Drop files here to open
Terminal: Local
npm ERR! A complete log of this run can be found in:
npm ERR! /Users/volontyr/.npm/_logs/2020-12-09T22_15_33_369Z-debug.log
Alexandrs-MacBook-Pro:TEST volontyr$ npm run rg component myComponent
> test@0.0.0 rg /Users/volontyr/Lina/TEST
> rg "component" "myComponent"
Generating files from 'component' template with name: myComponent
Done!
Alexandrs-MacBook-Pro:TEST volontyr$
Terminal
6: TODO
Event Log
1: Project
2: Favorites
npm
7: Structure