

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
«__» _____ 2021 р.

Дипломна робота
на здобуття ступеня бакалавра
спеціальності 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення розподілених систем»

на тему: «Інтерактивні навчальні карти понять для мобільних пристроїв»

Виконав (-ла):
студент (-ка) IV курсу, групи ТВ-71
Феденко Владислав Андрійович _____

Керівник:
доцент, канд. техн. наук
Титенко Сергій Володимирович _____

Рецензент: _____

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без відповідних
посилань.

Студент (-ка) _____

Київ – 2020 року

**Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

спеціальності 121 «Інженерія програмного забезпечення»

освітня програма «Інженерія програмного забезпечення розподілених систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр Коваль
(підпис)

” ____ ” _____ 2021р.

ЗАВДАННЯ

на дипломну роботу студенту

Феденку Владиславу Андрійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи Інтерактивні навчальні карти понять для мобільних пристроїв

керівник роботи Титенко Сергій володимирович, к.т.н., доцент

(прізвище, ім'я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ” ____ ” травня 2021р. № _____

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи Мова програмування Dart, фреймворк Flutter, API навчального порталу

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Дослідити сучасні навчальні портали на базі карт понять та існуючі алгоритми їх побудови. Провести аналіз існуючих програмних рішень, порівняти їх. Визначити головні функціональні можливості навчальних мобільних додатків на базі карт понять в аналогічних системах. Розробити дизайн інтерфейсу користувача. Побувати архітектуру системи. Спроекувати діаграми класів прецедентів. Обрати технології для реалізації програмного забезпечення. Розробити необхідні компоненти мобільного застосунку й об'єднати їх. Розробити функціональні можливості програмного продукту на базі карт понять. Імплементувати розроблений дизайн та функціональні можливості. Об'єднати складові додатку. Протестувати готову програмну систему, виправити можливі

помилки, вдосконалити функціональні можливості навчального застосунку.

5. Перелік ілюстративного матеріалу Титульна сторінка, Актуальність, Огляд системи boos.io, Огляд системи Semantic-Portal, Засоби розробки, Діаграма прецедентів, Файлова структура та діаграма класів, Робота з API, Алгоритм Фрухтермана-Рейнгольда, Модифікований алгоритм Фрухтермана-Рейнгольда, Початкове позиціонування вершин, Додаткова Сила, Результати тестувань оригінального і модифікованого алгоритмів, Основні функції системи, Інтерактивність та навігація, Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ” ____ ” _____ 202__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі		
3.	Розробка архітектури та загальної структури системи		
4.	Розробка структур окремих підсистем		
5.	Програмна реалізація системи		
6.	Оформлення пояснювальної записки		
7.	Захист програмного продукту	11.05.2021	
8.	Передзахист	25.05.2021	
9.	Захист	14.06.2021	

Студент

_____ (підпис)

_____ (прізвище та ініціали,)

Керівник роботи

_____ (підпис)

_____ (прізвище та ініціали,)

АНОТАЦІЯ

Дипломна робота виконана на 61 аркушах, вона містить 4 додатки, перелік посилань на використані джерела з 15 найменувань. У роботі було наведено 29 рисунки. Мета даної дипломної роботи полягає в розробці системи на базі інтерактивних карт понять для мобільних пристроїв. У роботі проведено приклади існуючих рішень поставленої задачі та їх аналіз — освітні веб-портали та мобільні системи на базу карт понять. Їх порівняно з точки зору наочності та зручності побудови карт понять та ефективності в дистанційному навчанні. Було проаналізовано переваги та недоліки представлених систем та розроблено алгоритм для вирішення їхніх недоліків.

Статтю на тему проблематики побудови карт понять на мобільних застосунках було опубліковано на Міжнародній науково-практичній конференції «Недінські читання». Також було опубліковано тези щодо вибору алгоритму візуалізації карт понять.

Ключові слова: карта понять, навчальна система, алгоритм побудови карт понять, навчальний веб-портал, формалізація знань.

ABSTRACT

The thesis is presented in 61 pages. It contains 4 appendixes and bibliography of 15 references. 29 figures are given in the work. The purpose of this thesis is to develop a system based on interactive concept maps for mobile devices. The paper presents examples of existing solutions to the problem and their analysis - educational web portals and mobile systems based on concept maps. They are compared in terms of clarity and ease of construction of concept maps and effectiveness in distance learning. The advantages and disadvantages of the presented systems were analyzed and an algorithm was developed to solve their shortcomings.

An article on the topic of constructing concept maps on mobile applications was published at the International Scientific and Practical Conference "Nedin Readings". Theses on the choice of algorithm for visualization of concept maps were also published.

Keywords: concept map, educational system, algorithm for constructing concept maps, educational web portal, formalization of knowledge.

ЗМІСТ

1	ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ ІНТЕРАКТИВНИХ КАРТ ПОНЯТЬ	11
2	КАРТИ ПОНЯТЬ В НАЧАЛЬНИХ СИСТЕМАХ	13
2.1	Концепція карт понять	13
2.2	Аналіз існуючих методів візуалізації графів	13
2.2.1	Алгоритм Ідеса	14
2.2.2	Алгоритм Фрухтермана-Рейнгольда	15
2.2.3	Алгоритм Камади-Кавай	16
2.3	Огляд існуючих навчальних систем на основі карт понять	17
2.3.1	Система booc.io.....	18
2.3.2	Система Metacademy	19
2.3.3	Система Semantic-Portal.....	20
3	ЗАСОБИ РЕАЛІЗАЦІЇ	23
1.1	Середовище розробки Android Studio.....	23
1.2	Мова програмування Dart та програмний каркас Flutter	23
4	АЛГОРИТМІЧНІ ОСНОВИ СИСТЕМИ ВІЗУАЛІЗАЦІЇ КАРТ ПОНЯТЬ	25
4.1	Алгоритм Фрухтермана-Рейнгольда для побудови карт понять	25
4.2	Модифікація алгоритму Фрухтермана-Рейнгольда для побудови карт понять..	26
4.3	Опис та обґрунтування здійснених модифікацій алгоритма Фрухтермана-Рейнгольда.....	28
4.4	Результати тестувань оригінального і модифікованого алгоритмів	32
5	ПРОГРАМНА РЕАЛІЗАЦІЯ.....	35
5.1	Архітектура застосунку.....	35
5.1.1	Бібліотека класів системи.....	36
5.1.2	Функціональні можливості системи	37
5.2	Структура API навчального порталу	38
5.3	Програмні компоненти навчальної системи	40
5.3.1	Сутності системи.....	41
5.3.2	Компоненти відображення карт понять.....	42

5.3.3	Компоненти інформаційної панелі.....	43
5.3.4	Компоненти панелі навігації.....	45
5.3.5	Компоненти вкладеного лінійного списку	46
5.3.6	Компоненти побудови карт понять	47
5.4	Створення макетів інтерфейсів програмного застосунку	48
6	РОБОТА КОРИСТУВАЧА З ЗАСТОСУНКОМ	52
	ВИСНОВКИ.....	59
	СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	60
	ДОДАТОК А	62
	ДОДАТОК Б.....	64
	ДОДАТОК В	75
	ДОДАТОК Г.....	85

Умовні позначення

API – Application Programming Interface.

HTTP – Hyper Text Transfer Protocol.

JSON – JavaScript Object Notation.

ПТМ – понятійно-тезисна модель.

ООП – об’єктно-орієнтоване програмування.

MVC – Model View Controller.

ПК – персональний комп’ютер.

ВСТУП

Стрімкий розвиток сучасних технологій призводить до широкого використання інтернету в галузях професійної та наукової діяльності. Інформаційно-навчальні веб-ресурси – це доступне та зручне джерело знань в області загальноосвітнього та професійного спрямування [1]. Навчання онлайн має низку переваг та є невід’ємною частиною сьогодення. Таким чином, зараз науковці дуже зосереджені на інтелектуалізації систем дистанційного навчання. Популярності набирають такі напрямки, як використання штучного інтелекту або семантичного веб-простору в освіті.

В цей же час, тенденцією сучасного світу є широке використання смартфонів. Новітні мобільні пристрої володіють значними обчислювальними можливостями та здатні навіть замінити комп’ютер чи планшет. Великою перевагою смартфона перед комп’ютером є його мобільність, що розширює коло можливостей користувача. Саме тому цей пристрій може залучити студентів до навчання більш динамічно.

Велику роль в онлайн навчанні грає візуалізація знань, адже вона позитивно впливає на успіхи студентів. Формалізація інформаційного наповнення навчального порталу, онтологічне моделювання предметної області та дидактична функція забезпечують найкращий рівень візуалізації навчальної інформації. Саме тому навчальні веб-сервіси мають бути пристосовані до онтологічно-орієнтованої моделі навчального контенту. Способом досягнення даної мети є метод візуалізації навчально-професійного контенту у вигляді карт понять, які мають великий вплив на когнітивні процеси студентів, що призводить до підвищення ефективності навчання.

Кarti понять є важливим інструментом для організації знань. Їхньою великою перевагою є стислість та чіткість. Кarti понять можуть зробити меншим когнітивне навантаження, що призводить до створення міцніших когнітивних структур в пам’яті користувачів. Побудова карти понять передбачає створення ациклічного орієнтованого графа, де вершинами слугують поняття, а ребра відображають зв’язки між ними. Поняття – це деякий об’єкт в області знань, представлений для вивчення студенту.

Інформацію про певне поняття містить у собі теза. Інтерактивність карти понять, фарбування вершин у різні кольори та надання їм різноманітних геометричних форм також позитивно впливає на доцільність карт та засвоєння знань студентами.

Метою дипломної роботи є розробка навчального мобільного додатку на базі карт понять, що призначений для дистанційного навчання. Основною аудиторією такого застосунку є студенти та учні. Застосунок розроблено на базі даних, отриманих з API навчального онлайн-порталу. Однією з найважливіших задач побудови системи є створення алгоритму побудови карт понять.

1 ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ ІНТЕРАКТИВНИХ КАРТ ПОНЯТЬ

Мета дипломної роботи полягає у створенні мобільної навчальної системи на базі інтерактивних карт понять. Основною задачею є розробка алгоритму для автоматизованої побудови, що передбачає побудову наочних карт понять, у яких вершини та ребра не перетинаються й не заважають одне одному. Також створення навчального застосунку передбачає розробку унікального дизайну, зрозумілого інтерфейсу, а також модулів навігації по картам понять та предметним областям. Перед початком написання програмного коду продукту було виконано побудову архітектури мобільного застосунку та побудовано діаграми, що дозволяють дослідити основні функції системи та її статичну структуру.

Розроблена мобільна система призначена для дистанційного навчання і є інформаційним порталом. Карти понять будуються за допомогою онтологічно-орієнтованої моделі знань та представляють формалізацію контенту. Карти розроблені на базі ПТМ [2]. Для підвищення засвоєння навчальної інформації, карти понять будуються у вигляді графів, адже графова візуалізація позитивно впливає на успіхи в навчанні. Вершинами у графі є власне поняття – певні об'єкти в предметній області, а ребрами виступають зв'язки між ними. Під час навчання та дослідження нової інформації щодо певного поняття, користувачу пропонуються для вивчення тези – ознаки, характеристики та відомості про поняття.

Однією з основних задач під час розробки програмного продукту була розробка інтерактивності карт понять. Доведено, що інтерактивність підвищує інтерес до предметної області, що вивчається, а це у свою чергу призводить до кращого засвоєння матеріалу студентом.

Таким чином, основними задачами під час розробки мобільної системи є:

- Забезпечення користувачу доступу до карт понять та розробка алгоритму для їх побудови.

- Забезпечення інтерактивності карт понять.
- Розробка модулю навігації по курсам та картам понять.
- Розробка унікального та зрозумілого користувачу дизайну, легкого у користуванні

Розробка начального мобільного застосунку виконується у такі етапи:

1. Дослідження вже існуючих систем, аналіз їхніх недоліків та переваг
2. Дослідження відповідної літератури та наукових статей, що дозволяє знайти спосіб вирішення проблем, які постають перед початком розробки
3. Побудова архітектури системи та відповідних діаграм
4. Проектування майбутнього дизайну застосунку та побудова макетів
5. Розробка алгоритму побудови інтерактивних карт понять
6. Розробка програмного застосунку
7. Тестування розробленої системи

2 КАРТИ ПОНЯТЬ В НАЧАЛЬНИХ СИСТЕМАХ

2.1 Концепція карт понять

Карти понять виникли в результаті досліджень когнітивної психології. Це важливий інструмент для організації, представлення та обміну знаннями. Побудова карти понять передбачає створення текстових вузлів, що з'єднані між собою дугами. Вузол означає поняття, а дуга означає зв'язок між певними поняттями [1]. Використання карт понять є корисним у великій кількості галузей. Карти понять мають більш стислу форму представлення інформації, в порівнянні з текстовими блоками того самого змісту.

Перегляд або складання карт понять разом з презентаціями або еквівалентним усним текстом може допомогти у когнітивній репрезентації даної інформації у вербальну та зорово просторову пам'ять людини [3].

Дослідження [4] показує, що імплементація карт понять у навчання дає низку переваг, а саме: вони сприяють кращій організації знань у конкретній понятійній області, забезпечуючи візуалізацію взаємозв'язків між ідеями просто та візуально привабливо. У картах понять використовують різні геометричні фігури та кольори, що сприяє запам'ятовуванню інформації. Карти понять забезпечують швидке та легке розуміння нових концепцій, дають можливість пояснити власними словами вивчений матеріал, та сприяють аналізу та синтезу ідей.

2.2 Аналіз існуючих методів візуалізації графів

На даний момент існує дуже багато алгоритмів візуалізації навантажених графів. Серед них розрізняють сіткові, полігональні, довільні, ортогональні, прямолінійні тощо. Далі буде розглянуто найпопулярніші алгоритми побудови графів.

- Методи спектральної компонування використовують власні вектори матриці, виведені з матриці суміжності графіка, як координати [5].

- Ортогональні методи розмітки, що дозволяють краям графіка проходити горизонтально або вертикально, паралельно осям координат макета.
- Методи багат шарового малювання графів розташовують вузли графа в горизонтальних шарах за допомогою таких методів, як алгоритм Коффмана-Грехема, таким чином, що більшість ребр рухаються вниз від одного шару до наступного. Після цього кроку вузли в кожному шарі розташовуються таким чином, щоб мінімізувати перетинання [6].
- Методи дугових діаграм розміщують вершини на лінії, краї можуть бути намальовані як півкола вище або нижче лінії, або як плавні криві, пов'язані між собою з декількох півкіл.
- Методи кругового макетування розміщують вершини графіка на колі, ретельно обираючи впорядкування вершин навколо кола, щоб зменшити перетинання і розміщувати сусідні вершини близько одна до одної.
- Силкові алгоритми візуалізації графів модифікують початкове розміщення вершин шляхом безперервного переміщення їх відповідно до системи сил, заснованої на фізичних концепціях, що відносяться до систем пружин або молекулярної механіки.

Надалі буде розглянуто силкові алгоритми візуалізації графів через їх переваги для вирішення задач, поставлених у даній роботі. Мета алгоритму полягає у тому, щоб розташувати вузли графа в двовимірному або тривимірному просторі так, що всі ребра мали б більш-менш однакову довжину, і звести до мінімуму кількість перетинів ребр шляхом призначення сил для безлічі ребр і вузлів ґрунтуючись на їх відносних положеннях [7]. Розглянемо основні силкові алгоритми візуалізації графів.

2.2.1 Алгоритм Ідеса

Ідея пружинного алгоритму Ідеса [8] полягає в моделюванні мережі з намагніченими вершинами, і ребрами, представленими пружинами. Алгоритм Ідеса був першим алгоритмом, у якому використовувались сили тяжіння та відштовхування.

Притягаюча сила f_a застосовується до вузлів, що мають пряме з'єднання ребром, а сила відштовхування f_r застосовується до вузлів, що мають непряме з'єднання. Сили тяжіння та відштовхування алгоритму Ідеса визначаються таким чином:

$$f_a(i, j) = C_a \log \frac{d(i, j)}{d_0}, \quad (1)$$

$$f_r(i, j) = C_r \frac{1}{d(i, j)^2}, \quad (2)$$

де $d(i, j)$ — ідеальна відстань між вузлами i та j ;

d_0 — ідеальна довжина ребра;

C_a та C_r — константи.

Мета алгоритму — знайти місця розташування нульових сил для всіх вузлів, щоб досягти стану рівноваги для пружинної системи.

2.2.2 Алгоритм Фрухтермана-Рейнгольда

Даний алгоритм базується на алгоритмі Ідеса [8]. Як і в алгоритмі Ідеса, він використовує сили тяжіння f_a та відштовхування f_r , які визначаються таким чином:

$$f_a(i, j) = \frac{d^2}{k}, \quad (3)$$

$$f_r(i, j) = \frac{k^2}{d}, \quad (4)$$

де d — відстань між двома вершинами;

k — постійна ідеальної попарної відстані.

Для сили притягання f_a , k можна записати як $a \times \sqrt{(W \times H)/n}$, та як $r \times \sqrt{(W \times H)/n}$ для сили відштовхування f_r . Де W – ширина фрейму, H – висота фрейму, n – загальна кількість вершин у графі, a – константа для помножувача притягання, r – константа для помножувача відштовхування.

Алгоритм Фрухтермана-Рейнгольда [9] виконується ітеративно. У кожній ітерації усі вузли переміщуються одночасно після того, як біли розраховані усі сили. Під час кожної ітерації алгоритм додає кожному вузлу параметр “переміщення” для зберігання зміщення позиції тодішніх вузлів. На початку кожної ітерації обчислюється значення сили відштовхування f_r для кожної вершини, далі для кожного ребра обчислюється парне значення сили притягання f_a для вершин, які належать даному ребру. Далі відбувається оновлення положення вузлів, використовуючи значення переміщення. На початку роботи алгоритму значення переміщення може бути занадто великим, тому у алгоритм введено характеристику температури t . Температура регулює максимальне значення переміщення за допомогою деякого алгоритму симуляції відпалу. Так після кожної ітерації значення температури t зменшується за допомогою спеціальної функції. Це відбувається для оптимізації та запобіганню застрягання вершин у локальних мінімумах.

2.2.3 Алгоритм Камади-Кавай

У алгоритмі Камади-Кавай [10] вершини розміщені таким чином, щоб їх візуальна відстань у межах креслення була пропорційною їх теоретичній графічній відстані. Ключовою ідеєю алгоритму є використання пружинної моделі таким чином, щоб мінімізувати енергію системи. Енергетична функція E :

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{1}{2} k_{i,j} (|p_i - p_j| - l_{i,j})^2, \quad (5)$$

де $k_{i,j}$ – жорсткість пружини між вузлами i та j ;

$l_{i,j}$ – ідеальна довжина пружини між вузлами i та j ;

p_i та p_j – візуальні положення вузлів i та j відповідно.

Тобто, алгоритм Камади-Кавай знаходить візуальне положення для кожної пари вузлів i та j , а їх евклідова відстань пропорційна $l_{i,j}$. Далі алгоритм визначає матрицю діаметрів, яка зберігає теоретичні графічні відстані ($d_{i,j}$) вузлів. Ідеальна довжина пружини ($l_{i,j}$) між вузлами i та j визначається наступним чином:

$$l_{i,j} = \frac{L_0}{\max_{i < j} d_{i,j}} \times d_{i,j}, \quad (6)$$

де L_0 – довжина сторони креслення;

$\max_{i < j} d_{i,j}$ – діаметр топології мережі.

Більш того, жорсткість пружини між вузлами i та j обчислюється таким чином:

$$k_{i,j} = \frac{K}{d_{i,j}^2} \quad (7)$$

де K – масштаб і $d_{i,j}$ представляє теоретичну графічну відстань між вузлами i та j .

Потім алгоритм шукає візуальне положення для кожного вузла v в топології мережі і намагається зменшити енергетичну функцію в цілій мережі.

2.3 Огляд існуючих навчальних систем на основі карт понять

У даному розділі буде оглянуто приклади існуючих навчальних систем на основі карт понять, проаналізовано їх плюси і мінуси. На основі цих даних буде зроблено

висновки стосовно того, який функціонал може бути потрібен чи не потрібен у застосунку, який розробляється.

2.3.1 Система boos.io

Воос.іо [11] – це інтерактивна навчальна веб-система, яка впорядковує поняття та дерево їх залежностей у ієрархічні кругові макети. Відображення карти понять саме у круговій формі робить дизайн компактним та допомагає забезпечити контекст.

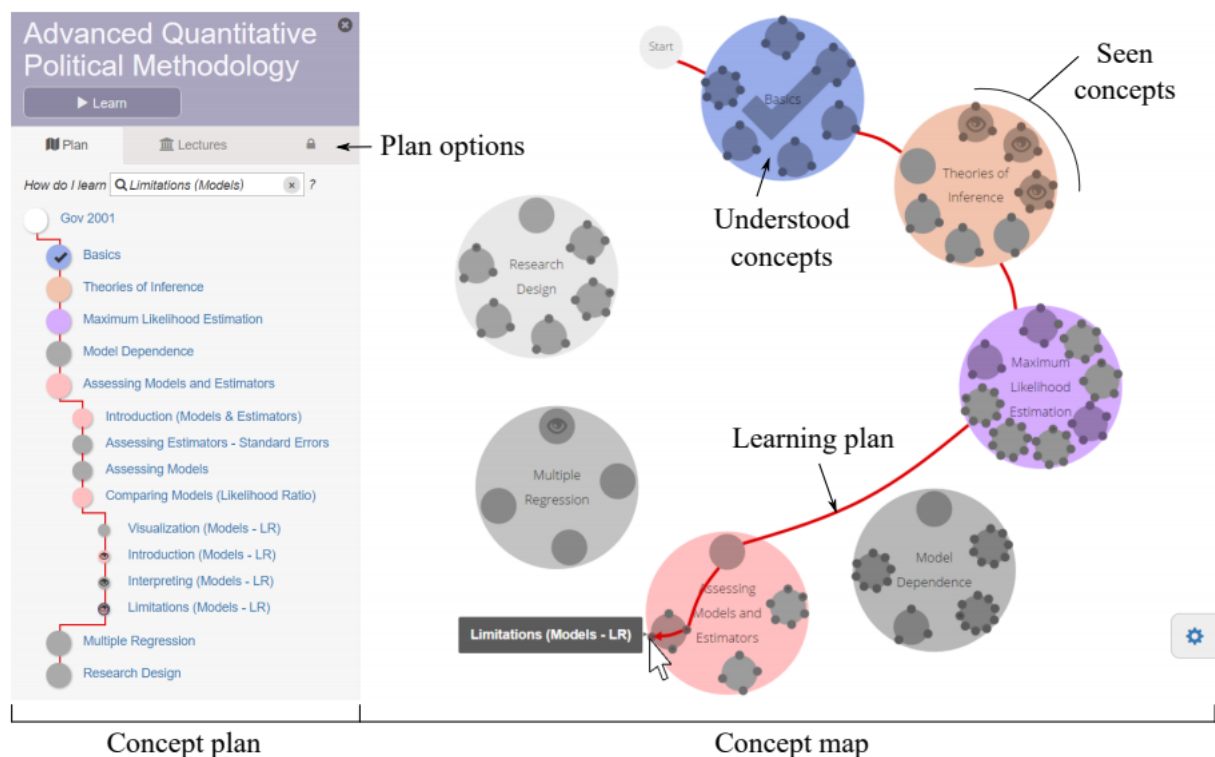


Рисунок 2.1 – Приклад інтерфейсу системи boos.io

Можливість масштабування дозволяє розширювати рівні ієрархії та підтримувати велику кількість понять. Лінійна подача матеріалу представляється як навчальний план, який проходить навколо циферблата кола ієрархічних понять. У системі підтримується

нелінійна подача матеріалу, студент має можливість прокласти свій власний навчальний план на основі вже здобутих знань. Boos.io є першою навчальною системою, яка поєднує у собі карти понять з динамічними лінійними та нелінійними персоналізованими навчальними планами. На рисунку 2.1 представлено приклад інтерфейсу системи boos.io.

Серед плюсів системи: новий погляд на дизайн карт понять, можливість побудови власного нелінійного плану навчання, автоматична побудова зрозумілого навчального плану на основі знань студента, розділення концептів по тим, які вже було опрацьовано, які було переглянуто, а з якими ще слід ознайомитись. Серед мінусів: представлення головних вузлів карти понять та їх нащадків виглядає не дуже зручним для користувача. Заплутати може постійна необхідність у масштабуванні концептів задля отримання інформації про їх нащадків.

2.3.2 Система Metacademy

Metacademy – це платформа з відкритим програмним кодом, метою якої є побудова експертами мережі знань. Система виступає хостинг-сервісом, підтримуючи інфраструктуру та організаційну структуру, що дозволяє користувачам додавати данні, редагувати їх або вчитися. Система надає користувачу доступ до мережі концептів, які представлені колами, пов'язаних між собою зв'язками, які позначаються кривими відрізками. Структури понять та зв'язків згруповано у курси та статті. Для більшої наочності велика кількість заплутаних зв'язків не видима користувачу одразу. Потрібно спочатку навести курсор на невидимий зв'язок або поняття для увімкнення видимості зв'язків. На рисунку 2.2 представлено приклад карти понять у системі Metacademy.

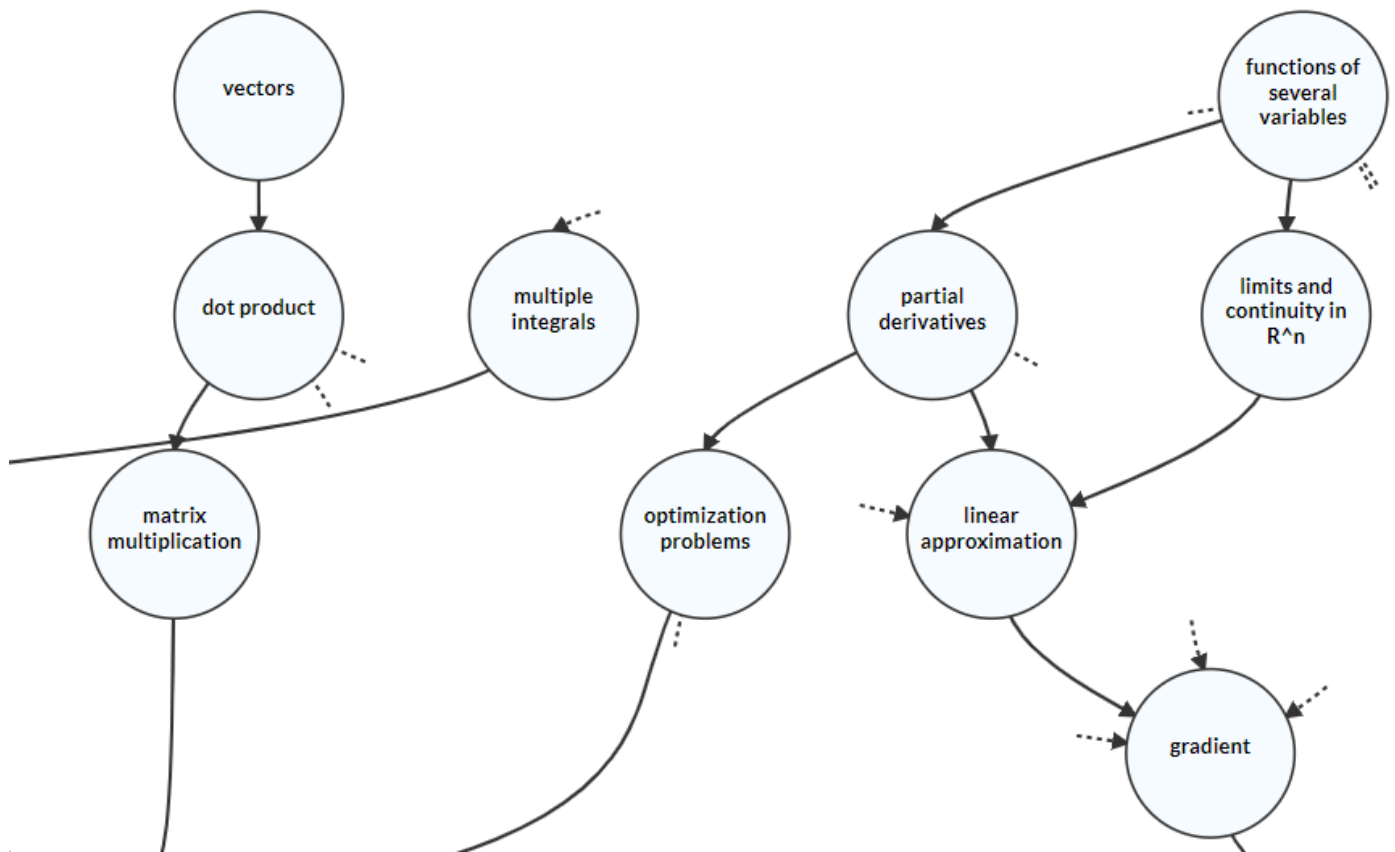


Рисунок 2.2 – Приклад карти понять у системі Metacademy до статті “Deep Belief Network” [12]

Плюсами даної системи є можливість додавати поняттям статусу прочитано або фаворит, можливість кому завгодно створювати структури понять та зв’язів, відкритий код. Серед мінусів: велика навантаженість карт понять та їх неупорядкована візуалізація. Невидимі зв’язки, які призначені зменшити навантаженість графів, створюють умови, у яких вивчення теми по даним картам понять стає нелогічним для користувача.

2.3.3 Система Semantic-Portal

Semantic-Portal – це навчальний веб-портал, який використовує понятійно-тезисну модель [2] для побудови карт понять. Сайт надає користувачу доступ до переліку курсів

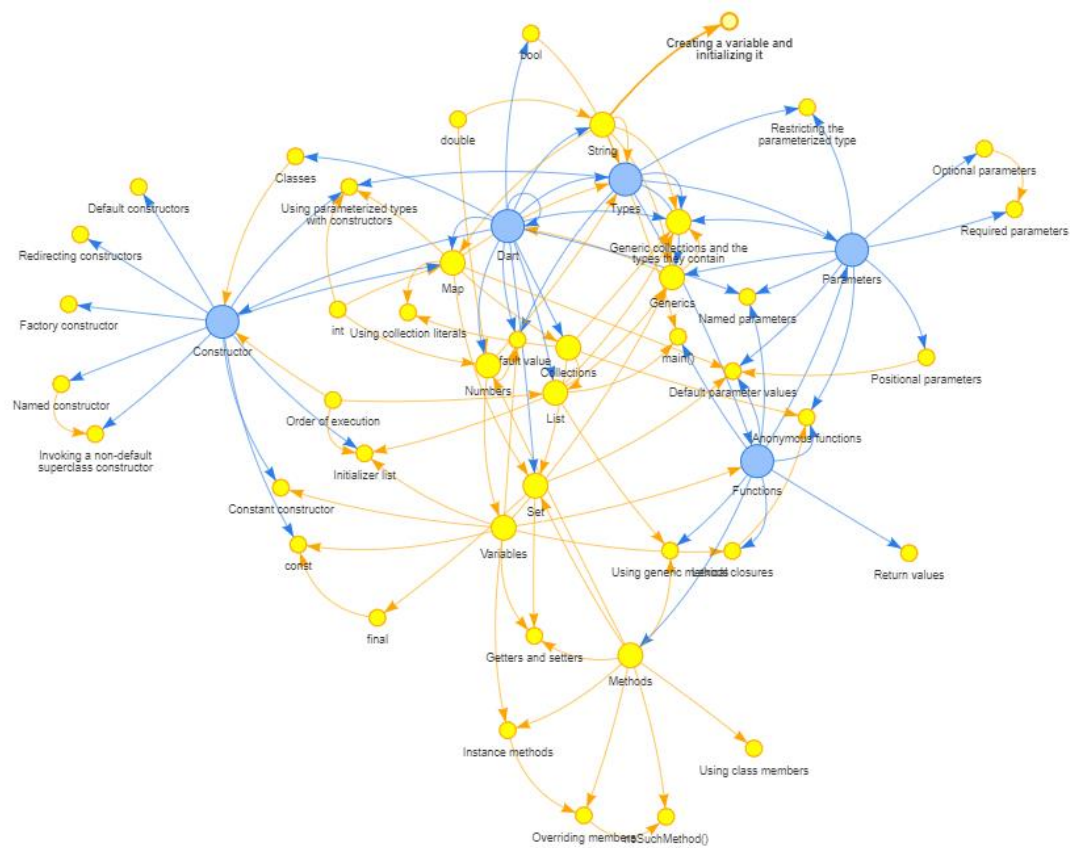


Рисунок 2.4 – Приклад перевантаженої карти понять на базі дидактичних зв'язків.

Серед плюсів даної системи: онтологічна формалізація контенту, наявність пошуку по поняттям, зручний інтерфейс, асоціація різних понять з певними кольорами, що допомагає зменшити когнітивне навантаження при запам'ятовуванні інформації. Серед мінусів велика навантаженість графів на основі дидактичних зв'язків. Карти понять, що охоплюють велику предметну область та будуються на основі дидактичних зв'язків дуже часто мають велику кількість вершин та ребр, які мають дуже велику кількість перетинів.

В результаті аналізу існуючих рішень, завданням бакалаврської роботи є розробка навчального мобільного застосунку, що впроваджує у себе позитивні сторони розглянутих систем та уникає негативні. У даному застосунку повинні бути реалізовані карти понять, побудовані власним методом, який має відповідати методам візуалізації графів, описаних у підпункті 2.2.

3 ЗАСОБИ РЕАЛІЗАЦІЇ

1.1 Середовище розробки Android Studio

Для створення програмного забезпечення на базі інтерактивних карт понять було обрано середовище розробки Android Studio. Дане середовище є універсальним та оптимізує роботу не лише мобільних застосунків, а й додатків для планшетів та ПК. До того ж, це середовище безкоштовне для використання та активно розвивається та є перспективним. У першу чергу Android Studio підходить для розробки застосунків на операційну систему Android, однак у ньому можна розроблявати також додатки для IOS. Беззаперечною перевагою даного середовища розробки перед своїми конкурентами є можливість за допомогою gradle контролювати збірку, створювати різноманітні варіанти застосунків, конфігурації підпису. Android Studio також дозволяє краще виконувати рефакторинг.

Середовище Android Studio має декілька незначних недоліків – певну нестабільність, а також відсутність деяких бібліотек, які не підтримуються через відносну новизну середовища.

Також під час розробки мобільної системи у середовищі розробки Android Studio було використано систему контролю версій git. Git дозволяє керувати версіями програмного забезпечення під час його розробки та може використовуватися з багатьма інтегрованими середовищами розробки.

1.2 Мова програмування Dart та програмний каркас Flutter

Для створення мобільного навчального застосунку на базі інтерактивних карт понять було обрано мову програмування Dart, яка є молодим та потужним інструментом розробки й має низку переваг перед іншими мовами програмування. Dart – це гнучка і потужна мова програмування, що у першу чергу вирішує наявлі недоліки мови

JavaScript. Дану мову можна використовувати для розробки як і серверної, так і клієнтської частини програми. Також завдяки мові програмування Dart можна розробляти не лише мобільні застосунки, а й веб-додатки, додатки для ПК і програми для embedded-систем. Мова Dart містить багато вбудованих інструментів та бібліотек, вони не конфліктують між собою.

Для створення мобільної системи також було використано фреймворк Flutter, що був випущений компанією Google у 2017 році. Найголовнішим плюсом фреймворку у створенні мобільних застосунків є кросплатформність – можливість розробляти додаток одночасно і для Android, і на IOS. Також завдяки функції «гарячого перезапуску», фреймворк дозволяє зберегти прописані в коді зміни та перезавантажити додаток за кілька секунд й уникнути додаткових обчислювальних витрат. Додаток можна оновлювати за секунди прямо на мобільному телефоні, або користуватися емулятором.

Таким чином, мобільний застосунок на базі інтерактивних карт понять було розроблено для операційної системи Android. Мінімальними вимогами для коректної роботи додатку є версія системи 6.0 або вища. Спершу застосунок потрібно завантажити, інсталювати, потім він готовий до використання.

4 АЛГОРИТМІЧНІ ОСНОВИ СИСТЕМИ ВІЗУАЛІЗАЦІЇ КАРТ ПОНЯТЬ

Для візуалізації карт понять в ході розробки було обрано алгоритм Фрухтермана-Рейнгольда [9]. Візуалізація графу, отримана у ході роботи даного алгоритму найбільше відповідає тим, які було спроектовано перед початком роботи над алгоритмом. Також алгоритм Фрухтермана-Рейнгольда є одним з класичних алгоритмів візуалізації графів, тому дає велике наукове поле для експериментів та власних удосконалень.

4.1 Алгоритм Фрухтермана-Рейнгольда для побудови карт понять

Даний алгоритм є фундаментальним силовим алгоритмом візуалізації графів, за допомогою нього можна достатньо ефективно будувати візуально-приємні графічні структури. Даний алгоритм є дуже інтуїтивні, бо працює на основі фізичних концептів на кшталт сили пружності пружин та сили відштовхування заряджених об'єктів, це дозволяє розуміти як себе можуть поводити сили та легше вносити власні зміни та модернізувати алгоритм. Також плюсом даного алгоритму є його ітеративність. Система може отримувати проміжні положення вершин, що можна використати для створення анімації.

Серед мінусів даного алгоритму можна виділити основні: можливість вершин застрягати у локальних мінімумах та час виконання. Даний алгоритм є ітераційним, час роботи однієї ітерації можна представити як $O(n^3)$, де n – кількість вершин графу. Це відбувається через те, що для кожної вершини потрібно розрахувати сили відштовхування та притягання, а для цього потрібно перебрати усі вершини двічі. Даний алгоритм в результаті роботи видає граф з мінімальною енергією, але може статися так, що це лише локальний мінімум. Локальний мінімум у багатьох випадках гірший за глобальний, це призводить до поганої візуалізації.

4.2 Модифікація алгоритму Фрухтермана-Рейнгольда для побудови карт понять

На рисунку 4.1 представлено псевдокод модифікованого алгоритму Фрухтермана-Рейнгольда. Як вхідні дані алгоритм приймає масив вершин (V) та ребер (E) між ними $G=(V,E)$. Алгоритм починається з ініціалізації сторони фрейму, радіусу сили відштовхування, постійна ідеальної попарної відстані тощо. Коли усі ініціалізації відбулись, починає свою роботу основний цикл, який продовжується задану кількість ітерацій. Для кожної вершини відбувається розрахунок сили відштовхування. Для цього кожній вершині у відповідність ставиться інша вершина та відбувається перевірка відстані однієї до одної. Якщо відстань менша за максимальний радіус дії сили відштовхування, то взаємодія вершин розраховується. Також якщо відстань менша за `hotDistance`, то параметру вершини `.isHot` надається значення `true`. Для того, щоб знайти вектор переміщення, спочатку знаходиться вектор різниці двох вершин. Далі цей вектор форматується у одиничний та помножається на значення сили відштовхування. Отриманий вектор додається до параметру вершини `.displacement`. Аналогічний процес відбувається і для розрахунку сил відштовхування вершини від середини ребра. Кожній вершині у відповідність ставиться ребро, перевіряється, щоб вершина не входила у дане ребро. Після цього розраховується позиція середини ребра і відбуваються дії, описані вище. Далі відбувається обчислення сил притягання. Для кожної пари вершин кожного ребра знаходиться вектор різниці. Після цього цей одиничний вектор множиться на значення сили притягання. Цей вектор додається до параметру `.displacement` кожної з двох вершин ребра.

Фінальною дією кожної ітерації є корегування положення кожної вершини залежно від вектору `.displacement`. Через те, що модуль вектора `.displacement` іноді може мати дуже великі значення, спочатку треба формувати це значення за допомогою функції `displacement()`.

```

fAtr(l, x) => x * x / l //Функція сили притягання
fRep(l, x) => l * l / x //Функція сили відштовхування
displacement(i, forceRadius, l, hotDistance, v) //Функція контролю зміщення
G = (V, E) //Множина вершин і ребр. Початкові значення позиції вершин
ініціалізовано за допомогою рекурсивного перебору дерева
side = 30 * V.length
area = side * side //Розміри фрейму
l = sqrt(area / V.length)
forceRadius = 300 //Радіус дії сили відштовхування
hotDistance = 100 //Радіус "гарячої" відстані
while і досягло максимального числа ітерацій iter
    //Обчислення сил відштовхування
    for v in V
        v.displacement = 0
        for u in V
            if u != v
                if відстань між v та u < forceRadius
                    Δ = v.position - u.position
                    v.displacement += Δ / |Δ| * fRep(l, |Δ|)
                if відстань між v та u < hotDistance
                    v.isHot = true
            end
        end
        for e in E
            if e.v != v && e.u != v
                if відстань між v та серединою e < forceRadius
                    Δ = v.position - u.position
                    v.displacement += Δ / |Δ| * fRep(l, |Δ|)
                if відстань між v та серединою e < hotDistance
                    v.isHot = true
            end
        end
    end
    //Обчислення сил притягання
    for e in E
        Δ = e.v.position - e.u.position
        e.v.displacement += Δ / |Δ| * fAtr(l, |Δ|)
        e.u.displacement += Δ / |Δ| * fAtr(l, |Δ|)
    end
    for v in V
        d = displacement(i, forceRadius, l, v)
        v.position += ((v.displacement) / |v.displacement|) * d
        v.position.x = min(side / 2, max(-side / 2, v.pos.x))
        v.position.y = min(side / 2, max(-side / 2, v.pos.y))
    end
end

```

Рисунок 4.1 – Модифікований алгоритм Фрухтермана – Рейнгольда

Дана функція нормалізує значення переміщення на базі поточної ітерації, параметру `.isHot`, радіусу дії сили притягання, постійної ідеальної попарної відстані тощо. Значення параметру `.position` змінюється з урахуванням максимальних доступних розмірів інтерфейсу для уникнення подальших помилок. Після цього відбувається нова ітерація, алгоритм не закінчує свою дію, поки значення змінної `i` не досягне значення максимальної кількості ітерацій.

4.3 Опис та обґрунтування здійснених модифікацій алгоритма Фрухтермана-Рейнгольда

Основними проблемами візуалізації графі є великий час виконання, перетин ребр та застрягання вершин у їх локальних мінімумах. Усі доробки до базового алгоритму спрямовані на вирішення даних проблем. Перша річ, яка відрізняється від оригінального алгоритму Фрухтермана-Рейнгольда це початкове положення вершин. У оригінальному алгоритмі на початку виконання вершинам надається випадкові позиції у межах деякого фрейму. Через це кількість ітерацій на побудову одного й того самого графу може відрізнитись від разу до разу, у вершин більша вірогідність застрягти у локальних мінімумах, а у ребр більше шансів на перетин. Алгоритм потребує більшу кількість ітерацій, а через це збільшується час виконання. У модифікованій версії алгоритму початкові положення вершинам надається не випадково, а за допомогою деякого алгоритму. Для того, щоб на побудову дерев витрачався менший час, було вирішено перед виконання силового алгоритму будувати схематичне дерево за допомогою рекурсивного перебору дерева. Для цього множина вершин та ребр $G=(V,E)$ перетворюється у множину нод $G=(N)$. Кожна нода зберігає у собі інформацію про дочірні ноди, що дозволяє будувати дерева методом перебору. Результат модифікованого модулю задання начальних положень вершин можна побачити на рисунку 4.2.

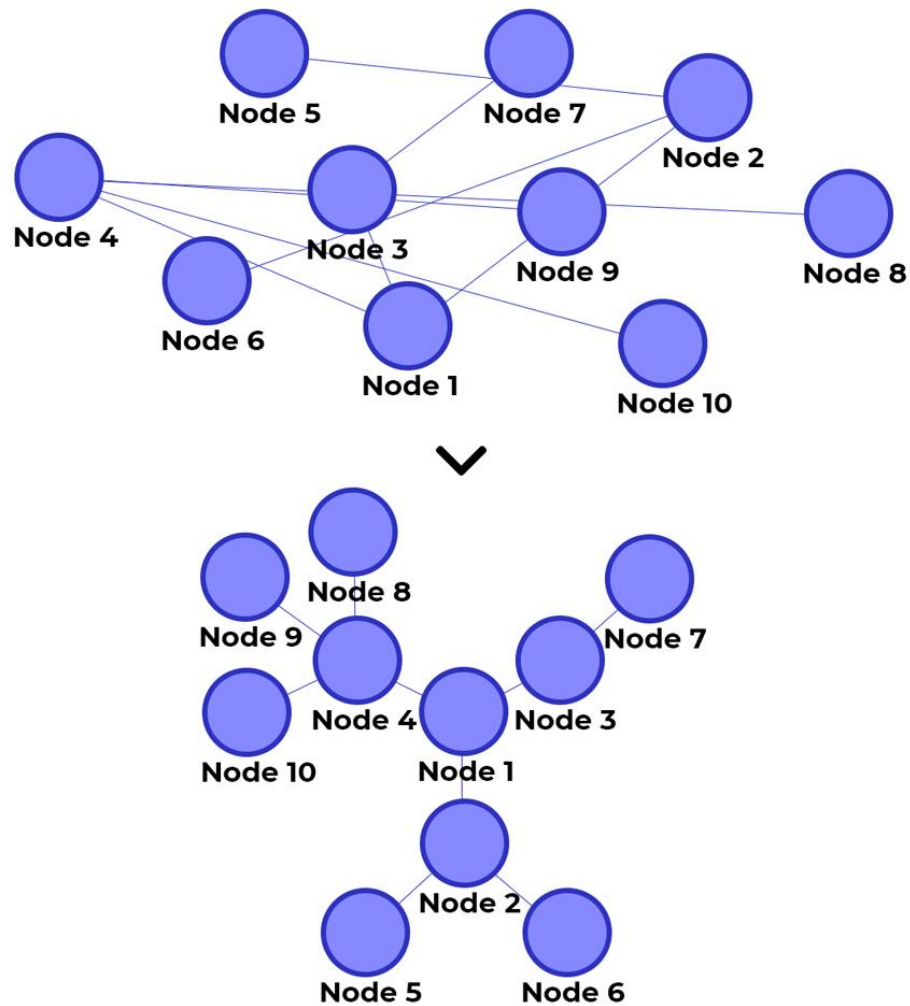


Рисунок 4.2 – Приклад початкового позиціювання вершин у оригінальному та модифікованому алгоритмах Фрухтермана – Рейнгольда

Даний підхід дозволяє знизити час роботи алгоритму та отримувати майже однаковий граф кожного разу, коли виконується алгоритм, коли у оригінальному алгоритмі кожного разу будується абсолютно новий візуально граф. Це виконує свою роль, бо у навчальному застосунку на основі карт понять дуже важливим є асоціативність форм і кольорів графу через їх зв'язок з когнітивними властивостями мозку людини [13].

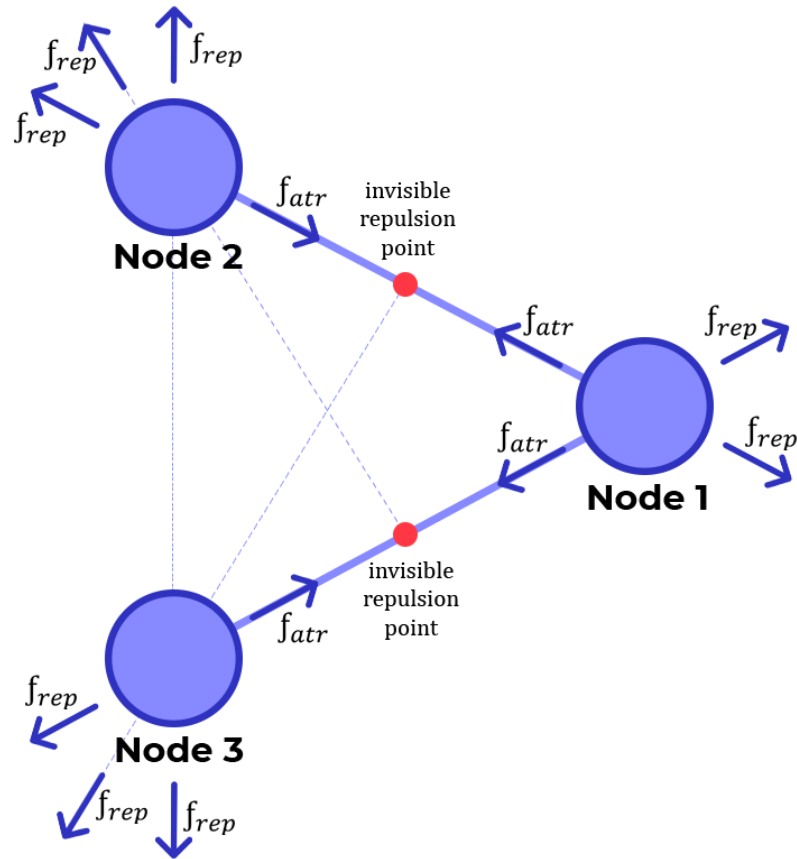


Рисунок 4.3 – Схематичне представлення сил відштовхування f_{rep} та притягання f_{atr} у модифікованому алгоритмі Фрухтермана – Рейнгольда

У ході розробки було помічено, що місце на ребрі між двома вершинами дуже часто є локальним мінімум для деяких вершин, бо у цьому місці на вершину з схожою силою діють дві інші вершини з протилежних сторін. Для вирішення цієї проблеми було вирішено до сил відштовхування додати ще одну, яка б діяла з середини кожного ребра на вершини. Хоча ця дія збільшує час виконання одної ітерації алгоритму, вона також зменшує загальну кількість ітерацій. Схематичне представлення сил, які діють на вершини під час роботи модернізованого алгоритму Фрухтермана-Рейнгольда можна побачити на рисунку 4.3. Червоними точками позначено невидимі точки відштовхування, які розміщено на середині ребр.

Якщо розраховувати вектор переміщення вершини на основі даних про відштовхування від усіх інших вершин, можлива ситуація, коли вектор переміщення не є коректним через вершини, які знаходяться занадто далеко від даної. Для вирішення даної проблеми було додано поняття радіусу дії сили. Перед розрахунку сили відштовхування кожна вершина спочатку перевіряє відстань до другої вершини. Якщо ця відстань більша за задану, то розрахунок сили відштовхування від вершини не відбувається. Даний метод дозволяє будувати візуально кращі графи та скоротити час роботи алгоритму, бо не відбувається розрахунок непотрібних даних. На рисунку 4.4 показано як Node 4 відштовхується від Node 3, яка у радіусі дії сил відштовхування, та не відштовхується від Node 2, яка не попадає у даний радіус.

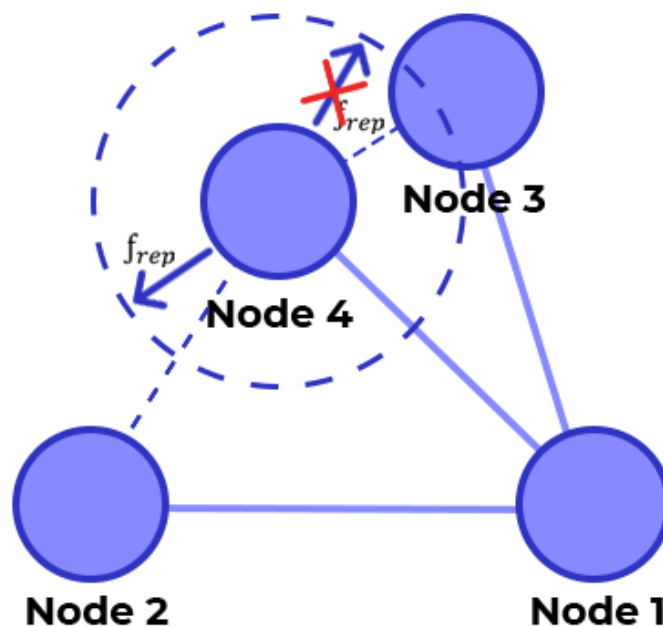


Рисунок 4.4 – Схематичне представлення радіусу дії сил відштовхування

З ходом ітерацій максимальне дозволене переміщення вершин зменшується. Для запобігання застрягання вершин поряд з іншими на фінальних етапах роботи алгоритму було додано поняття гарячої відстані. Суть полягає у тому, що коли відстань між двома вершинами стає меншою за мінімальну “гарячу”, вершинам надається велике

переміщення і вони відскакують одна від одної. Даний підхід дозволяє будувати візуально гарний граф за меншу кількість ітерацій. На рисунку 4.5 схематично представлено роботу даної модифікації.

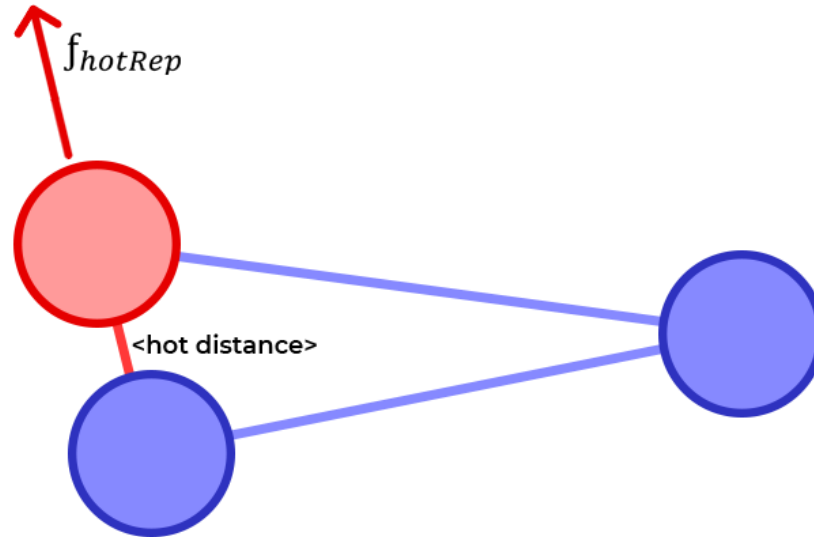


Рисунок 4.5 – Схематичне представлення поведінки вершини, яка знаходиться на відстані до іншої вершини, меншій за “гарячу”

Дані доповнення до оригінального алгоритму Фрухтермана-Рейнгольда було розроблено у ході роботи над візуалізацію карт понять експериментальним шляхом. Результати тестувань часу виконання та вигляду графів надано у наступному підрозділі.

4.4 Результати тестувань оригінального і модифікованого алгоритмів

Опис оригінального алгоритму Фрухтермана-Рейнгольда [9] оминає опис алгоритму симуляції опалу. Даний алгоритм відповідає за довжину вектору переміщення вершини та запобігає застряганню вершин у локальних мінімумах. Через те, що даний модуль дуже важливим і він не описаний у оригінальному алгоритмі, було вирішено розробити алгоритм контролю переміщення вершин. Даний алгоритм використовує концепцію

гарячих відстаней, яка описана у підрозділі 4.1.2. У результатах тестувань обидва алгоритми використовують даний алгоритм замість оригінального алгоритму симуляції опалу.

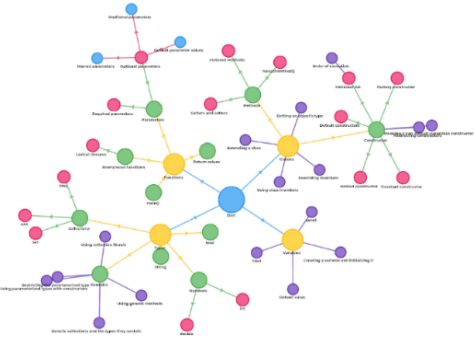
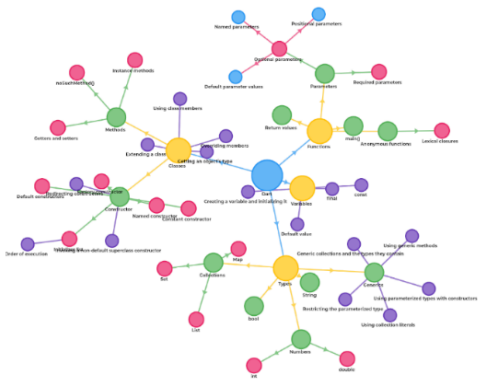
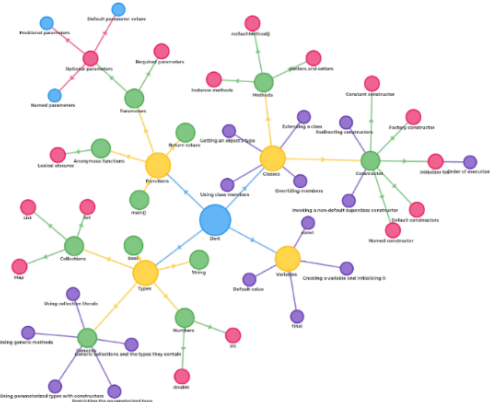
	Оригінальний алгоритм Фрухтермана-Рейнгольда	Модифікований алгоритм Фрухтермана-Рейнгольда
Середній час виконання для 50 ітерацій (мс)	82.8	72.4
Середній час виконання для 200 ітерацій (мс)	394.1	352.6
Середній час виконання для 500 ітерацій (мс)	1349.3	1266.5
Приклад карти понять при 50 ітераціях		
Приклад карти понять при 200 ітераціях		

Рисунок 4.6 – Порівняння результатів роботи оригінального і модифікованого алгоритмів Фрухтермана-Рейнгольда

На рисунку 4.6 можна побачити, що у середньому модифікований алгоритм працює швидше за оригінальний. Хоча кількість сил, які треба прорахувати у модифікованому алгоритмі більше, у середньому на одну ітерацію витрачається менше часу. Це можливо перевірки вершиною зони дії сил. Через початкове позиціювання вершин, вже після перших ітерацій деякі вершини знаходяться на оптимальних позиціях, що дозволяє пропускати розрахунок деяких їх сил. Також можна побачити, що якість побудови карт понять відрізняється. При п'ятдесяти ітераціях, модифікований алгоритм будує візуально прийнятний граф.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ

Під час розробки мобільного навчального застосунку на базі карт понять було побудовано архітектуру мобільного застосунку, розроблено зручний користувацький інтерфейс та макети мобільної системи, описано структуру API навчального веб-порталу, з якого беруться усі дані для побудови курсів і карт понять, описано роботу наступних модулів мобільного навчального застосунку: API-сервіс, компонента побудови карт понять, компонента навігації, компонента побудови вкладеного лінійного списку, пошуку тощо.

5.1 Архітектура застосунку

Архітектура програмного застосунку відображає структуру системи та дає пояснення щодо того, як вона поводить себе. Система представляє з себе набір компонентів, що виконують певні функції та взаємодіють один з одним. Архітектура програмного застосунку допомагає створити міцну основу для розробки якісного програмного продукту. Ряд архітектурних рішень впливають на продуктивність, якість і загальний успіх застосунку. Можливість врахувати довгострокові проблеми може врятувати систему від ризику.

У книзі *An Introduction to Software Architecture* автори пишуть: “Крім створення алгоритмів і структур даних, необхідно вирішити ще одну принципове завдання - розробити загальну структуру системи. Процес розробки структури включає в себе створення загальної інфраструктури організації системи і управління нею, вибір протоколів і методів синхронізації і доступу до даних, розподіл функцій системи між компонентами, фізичний розподіл, об'єднання елементів проекту, масштабування, оптимізацію продуктивності і вибір оптимальних варіантів серед доступних альтернатив” [14].

Діаграма класів займає головне місце у проектуванні систем, основаних на парадигмах ООП. Діаграма класів відображає структуру класів системи, полів, методів, інтерфейсів та зв'язків між ними. Діаграма класів не несе за мету представлення динамічної структури системи, уся динаміка повинна буди відображена на діаграмах інших нотацій. Дана діаграма має у собі представлення статичної структури декларативних елементів системи. Вона надає найбільш повне представлення про програмний код та функціоналу окремих класів.

Наприклад, дана діаграма показує як клас контролер ForceDirectedController модифікує екземпляри класів Edge і Vertice, та на їх основі, за допомогою класу BalloonTreeController, створює сутності класу Node.

5.1.2 Функціональні можливості системи

На рисунку 5.1 представлено діаграму прецедентів для розроблюваної системи. На даній діаграмі представлено як користувач (Студент) може взаємодіяти з системою.

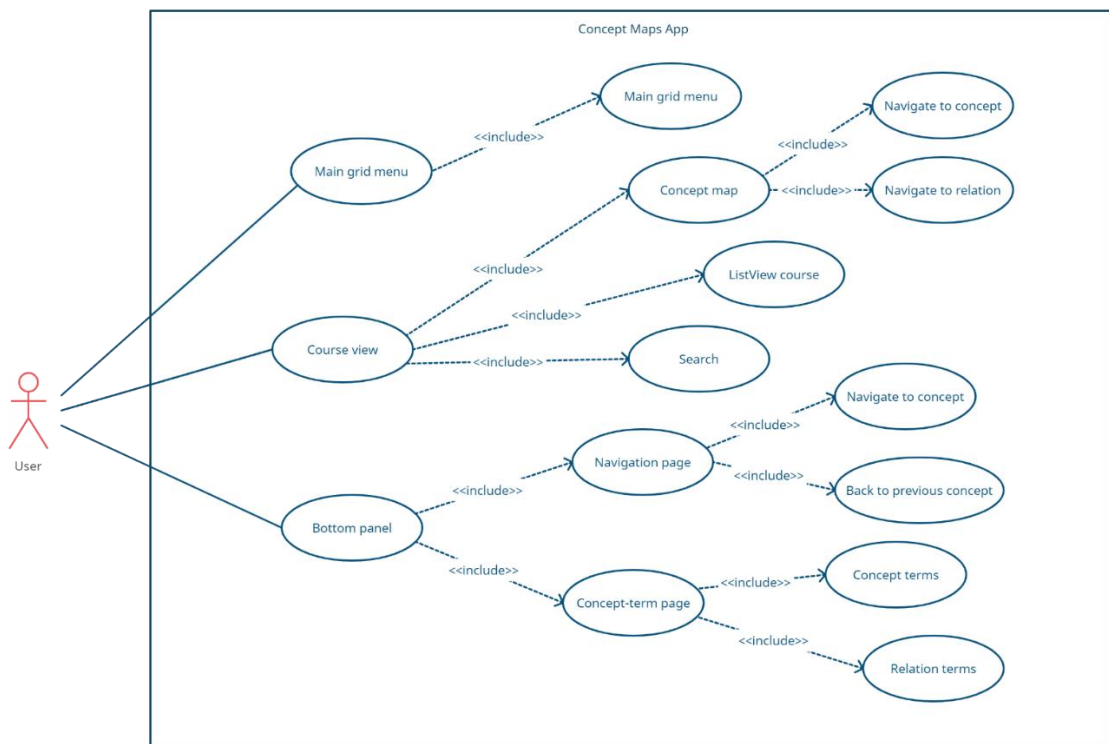


Рисунок 5.1 – Діаграма прецедентів мобільного застосунку

Діаграма прецедентів представляє з себе граф, який складається з множин акторів та прецедентів, що пов'язані відношеннями між ними. Дані діаграми використовують для візуалізації функціоналу системи та взаємодії різних типів користувачів з нею, не поглиблюючись у внутрішню роботу системи. Специфікація далі стане основою для документації та тестування програмного продукту. Дана діаграма є достатньо простою, тому її можна використовувати у технічному завданні.

У головному меню Main Grid Menu користувач має можливість обрати курс, після чого потрапляє у Course View, звідки має можливість перейти до карти понять та навігації по ній, перейти у розділ лінійного представлення курсу або до пошуку.

При переході до Bottom Panel користувач має можливість скористатися одним з варіантів використання даної панелі: навігації та вивчення інформації. Панель навігації дозволяє користувачу переходити до наступного або попереднього поняття, а інформаційний розділ переглядати тези, пов'язані з поняттям або зв'язком двох понять.

5.2 Структура API навчального порталу

Усі данні для побудови карт понять та курсів було взято з відкритого API навчального веб-порталу. Умовну структуру даних у даному API представлено на рисунку 5.3. API представляє з себе набір HTTP-запитів та деякі структури HTTP-відповідей у форматі JSON. Даний веб-портал надає користувачам доступ до онтологічно-формалізованих даних у ПТМ [2].

У розділі Branch користувач має доступ до даних повних курсів, їх понять, відношень та тез. Даний розділ потрібен для побудови карт понять у мобільному застосунку. У розділі Page надається доступ до інформації про сторінку, а саме підписи, картинки, заголовки тощо. Також у даному розділі надається інформація про входження понять до заголовків сторінок. Ця інформація потрібна для ініціалізації початкових понять у картах.

Розділ Concept надає детальну інформацію про будь яке поняття. Користувач має змогу отримати один або декілька понять по їх id, отримати інформацію про тези даного поняття, дидактичні та інші зв'язки з іншими поняттями. Через даний розділ API також можна отримати доступ до такого важливого поля, як CinT. Дане поле надає інформацію про знаходження конкретного поняття у тезах інших понять та про знаходження інших понять у тезах конкретного. Дана інформація дуже корисна при аналізі зв'язків між поняттями. Розділ Thesis надає доступ до тез. Користувач має доступ до усіх тез системи, або до декількох тез по їх id.

Ontology API

CinH - Concepts in Headers of all pages.

Branch

branch/{branch} - all branch concepts, relations, didactic relations,
branch concepts in theses, branch concepts in concepts

where {branch} - semantic root of this branch. Page code of root page.

branch/{branch}/theses - theses of branch

branch/{branch}/concepts - concepts of branch

branch/{branch}/concepts/relations - relations of branch

branch/{branch}/concepts/didactic - didactic relations of branch

Page

branch/{branch}/children - children of branch page

branch/{branch}/view - page information (caption, intro, image, text)

Concepts in Headers of Page

Concept

concept/{id} - concept by id

concept/id1,id2,...,idN - concepts by ids

concept/{id}/thesis - theses of concept with id

* concept/{id}/CinT - this concept in other theses & other concepts
in this concept theses

concept/{id}/relations - concept relations

concept/{id}/didactic/before - concepts before this concept didactically

concept/{id}/didactic/after - concepts after this concept didactically

Theses

thesis - all theses

* thesis/id1,id2,... - theses by ids

Auth

user/check/{login}/{psw} (POST) - check user login

Рисунок 5.3 – Структура API навчального веб-порталу

Дані API навчального веб-порталу стали основою і каркасом навчального мобільного застосунку на карті понять. На основі проаналізованої інформації про поняття та їх взаємозв'язків будується основна інтерактивна карта понять, користувач має доступ до навчальних матеріалів, відбувається пошук та навігація по поняттях. Отримання даних за допомогою API відбувається дуже швидко, тому користувач не має проблем з очікуванням даних з серверу.

5.3 Програмні компоненти навчальної системи

Програмний продукт виконано з дотриманням парадигм ООП, використано архітектурний шаблон MVC, що передбачає розділення файлів на три окремих розділи: модель даних програми, представлення інтерфейсів та логіку взаємодії користувача з застосунком. Даний підхід дозволяє мінімізувати вплив на інші компоненти при модифікації одного з них. Основною метою MVC є розділення даних і логіки від візуалізації. За рахунок такого підходу з'являється можливість повторного використання коду. На рисунку 5.4 показано схематичну структуру зв'язків між модулями програмного застосунку. У даному застосунку є два типи моделей: сутності, які репрезентують об'єкти з API та додаткові сутності, які було розроблено для малювання графу. Контролери відповідають за побудову і роботу графу та панелі навігації. Представлення – це віджети, які відповідають за те, що бачить користувач. Також у додатку присутні провайдер, сервіси та утиліти.

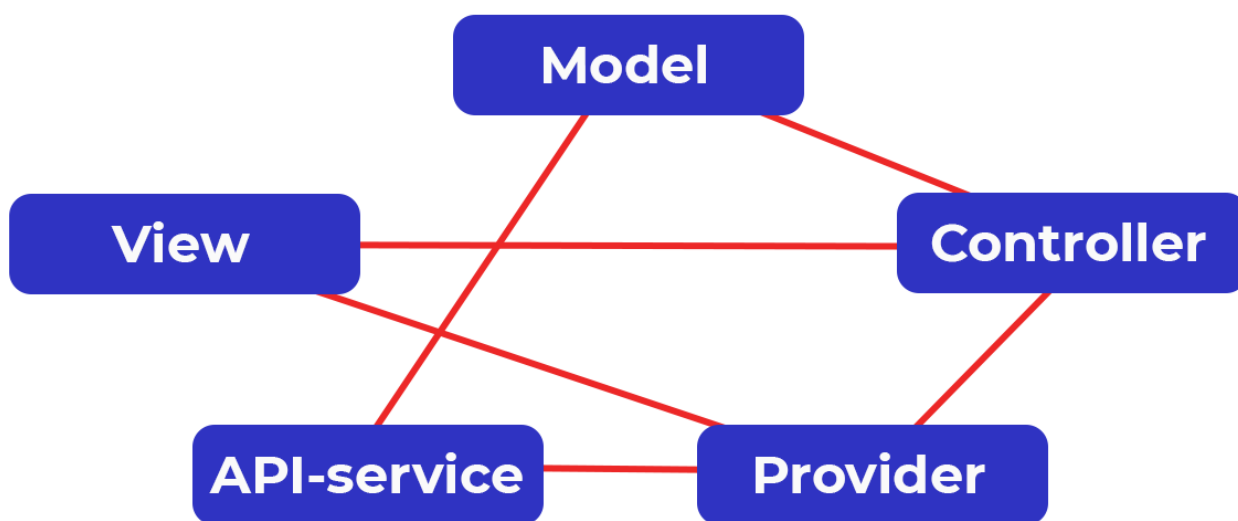


Рисунок 5.4 – Схематична структура зв'язків між модулями мобільної системи

Провайдер це клас з фіксованим станом, доступ до якого можна отримати з будь якої точки проекту у будь який момент виконання. Головна перевага провайдера полягає

у тому, що дані у ньому не перезаписуються з кожним переходом між інтерфейсами. Провайдер містить у собі поля, які використовуються у декількох місцях системи та методи для їх модифікації. Сервіси мають у собі логіку взаємодії застосунку з API навчального веб-порталу. Для усіх методів сервісів, які використовуються у ході роботи застосунку існує метод у провайдері, для зв'язку з API у будь якій точці програмного застосунку.

5.3.1 Сутності системи

Модель складається з сутностей, які відповідають записам у API навчального веб-порталу. Модель реагує на запити з контролера, змінює свій стан або повертає дані до нього. Модель не взаємодіє з користувачами та не містить у собі інформацію про контролер та представлення. На рисунку 5.5 представлено файлову структуру моделі мобільного застосунку.

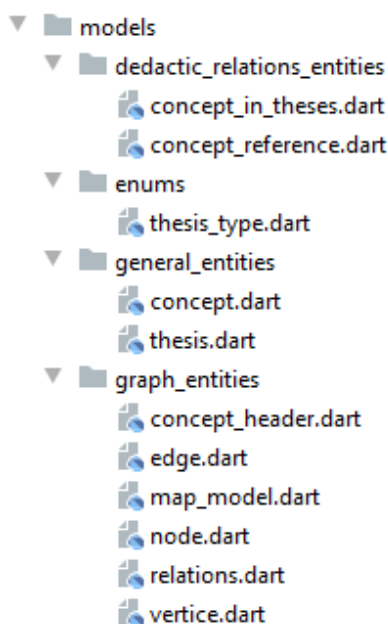


Рисунок 5.5 – Файлова структура моделі мобільного застосунку

Основними є класи Concept та Thesis, які є основними елементами понятійно-тезисної моделі [15], яка лежить у основі усієї роботи. Також дуже важливим є клас

Node, який є елементом списку tree, на основі якого будується покращене початкове розташування вершин карти понять, навігація та вкладений список. Основна відмінність Node від Concept полягає у тому, що кожен Node зберігає інформацію про дочірні елементи всередині себе, на відміну від Concept, який використовує додатковий клас Relation. Класи Vertice та Edge є аналогами Concept та Relation, які мають параметри положення, переміщення та інші, які необхідні для побудови карт понять за допомогою силового алгоритму побудови графів. Класи ThesisType, ConceptReference, ConceptInTheses, Thesis пов'язані з API та слугують для коректного отримання та парсингу тез. Клас MapModel потрібен для зручного доступу до поточних листів об'єктів класів Concept, Relation та ConceptHeader.

5.3.2 Компоненти відображення карт понять

Представлення відповідає за те, що користувач бачить на екрані. Це різні шаблони віджетів, візуалізація карт понять, пошуку, різних елементів навігації тощо. Класи представлення не мають у собі бізнес-логіки та звернень до сервера, дані функції прописано у контролерах та API-сервісах. На рисунку 5.6 продемонстровано файлову структуру представлення.

Розглянемо дану структуру більш детально. Клас CoursesMenu відповідає за початкове меню вибору курсу. Будується сітка з різнокольорових прямокутників з назвою курсу, при натисканні на які відбувається перехід до відповідного курсу. Коли користувач переходить до певного курсу, він бачить карту понять даного курсу. За побудову карти понять відповідає клас ForceDirected. Клас названо за аналогією з алгоритмом, за допомогою якого карта понять будується. Даний клас отримує данні про вершини та їх зв'язки з провайдера та передає їх до контролера. Там відбувається надання вершинам, початкових положень, кольорів та відбувається розрахунок кінцевих положень на основі виконання силового алгоритму візуалізації графів.

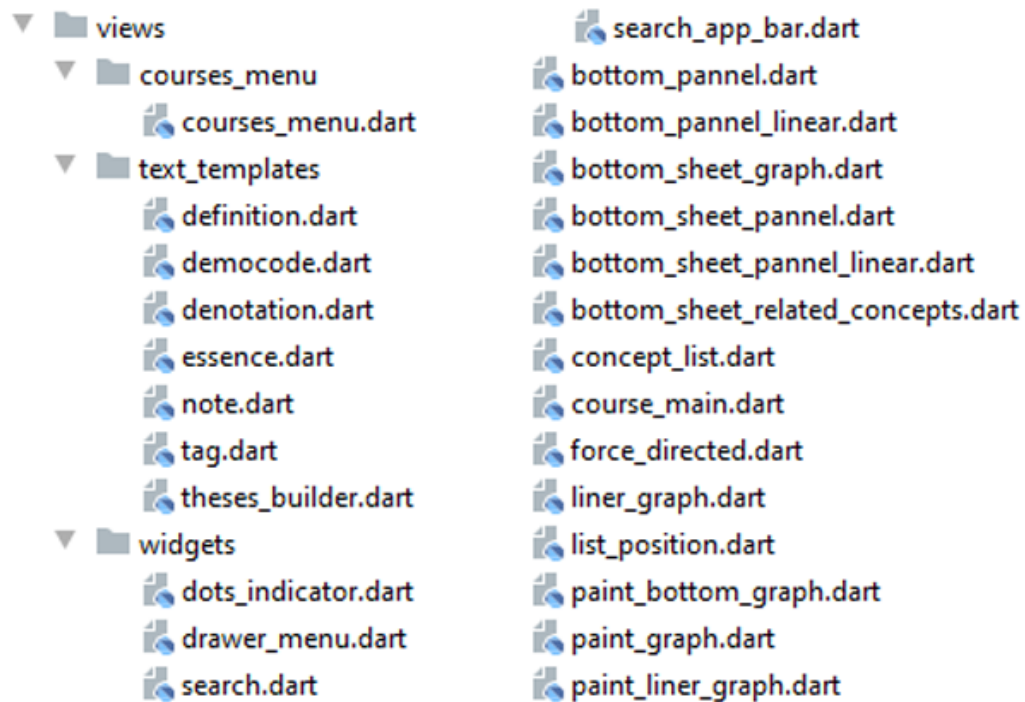


Рисунок 5.6 – Файлова структура представлення мобільного застосунку

Данні, отримані з контролерів передаються до спеціального методу, який відповідає за створення списку віджетів, кожен з яких відповідає вершинам, їх назвам або ребрам. Дані віджети будуються з можливістю користувача натискати на них. При натисканні на вершину запускається анімація плавного переходу до даного елементу системи, відкриття нижньої панелі та збереження до провайдера інформації про вершину, яка наразі знаходиться у фокусі та інформацію про нижню панель. Анімація допомагає користувачу орієнтуватись у іноді складних структурах карт понять.

5.3.3 Компоненти інформаційної панелі

Клас `BottomSheetPanel` відповідає за представлення нижньої панелі. Даний клас є межовим між класами `BottomPanel` та `BottomSheetGraph`, які у верстці представлено різнокольоровими стікерами. За аналогією з зошитом-конспектом. Ці класи відповідають за представлення панелі з інформацією про поняття та панелі навігації. У класі `BottomSheetPanel` прописано методи для анімації панелі та умови переходів до

одного з класів панелей. Зелений стікер відповідає класу `BottomPanel`, у якому будується представлення тез, які стосуються понять, яке знаходиться у фокусі. Клас посилає до провайдера, а той у свою чергу до API-сервіса ід поняття, по якому отримується список тез, які стосуються даного поняття. Ці сирі дані відправляються до класу `ThesesBuilder`, у якому відбувається пошук відповідного класу представлення тези до її типу.

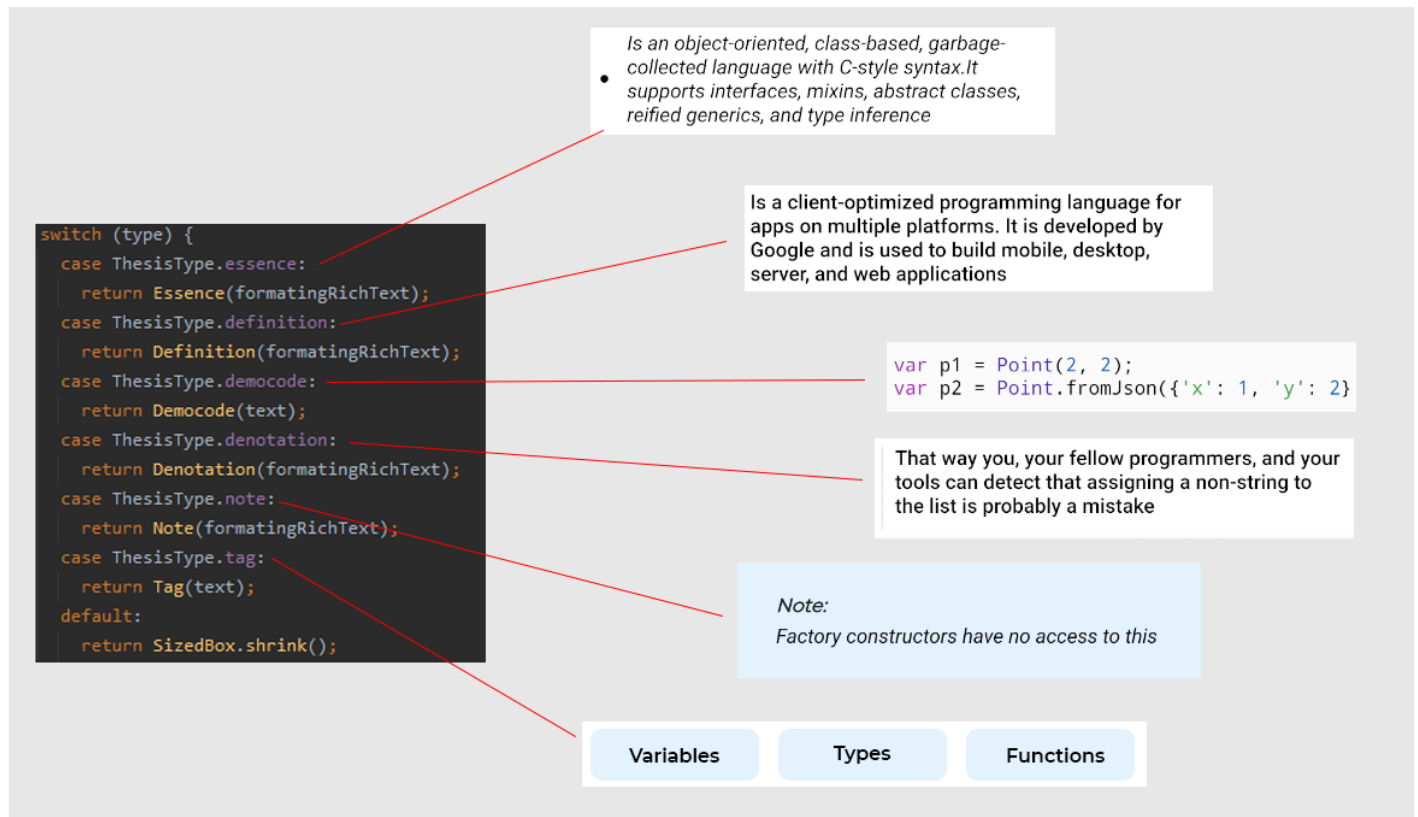


Рисунок 5.7 – Схематичне представлення роботи класу `ThesisBuilder`

Тека `text_templates` містить у собі класи, які відповідають за представлення тез у інформаційній панелі. Клас `ThesisBuilder` перевіряє тип тези, отриманої з сервера та запускає відповідний контролер. Існує шість типів тез: суть, означення, код, позначення, нотатка та тег. Таке ромаїття типів дозволяє більш детально засвоїти відповідну тему під час проходження курсу. На рисунку 5.7 схематично представлено роботу класу `ThesisBuilder`.

5.3.4 Компоненти панелі навігації

Клас BottomSheetGraph відповідає за представлення нижньої панелі навігації застосунку. Панель представляє з себе круговий список дочірніх елементів з батьківським елементом по центру. На рисунку 5.8 схематично представлено роботу класу BottomSheetGraph.

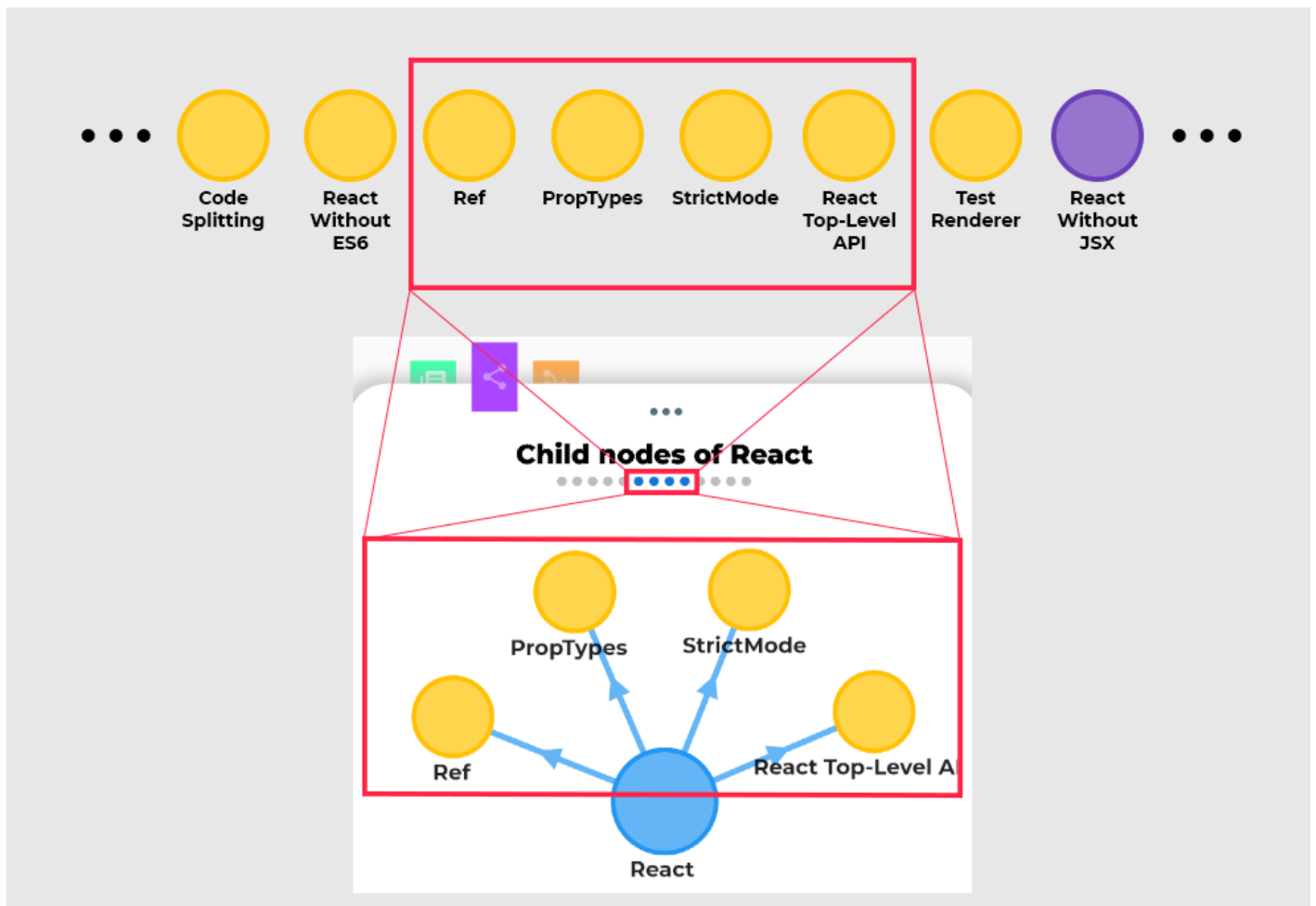


Рисунок 5.8 – Схематичне представлення відображення активних елементів кругового списку

У зоні видимості користувача завжди знаходиться лише чотири дочірніх поняття, які схематично відображаються у меню відображення активних понять, яке має вигляд списку точок, серед який синім відображаються активні, а сірим – неактивні. Навігаційний список є інтерактивним, користувач має змогу крутити його як рулетку.

Для більш приємного користувацького досвіду список має аналог фізичного прискорення, тому від сили руху пальцем залежить швидкість переміщення елементів у списку. Список є циклічним, це означає, що користувач може крутити елементи у одну з сторін без кінця. Даний ефект досягається за допомогою спеціального алгоритму аналізу списків активних і неактивних елементів.

При натисканні на елементи списку дочірніх понять користувач здійснює перехід до даного поняття, що активізує анімацію головної карти понять та задає у провайдері активне поняття. Даний перехід є аналогічним тому, який відбувається при навігації по самій карті понять. Також користувач має змогу переміщуватися у зворотному напрямку за допомогою стрілки, яка знаходиться під активним поняттям.

Даний дизайн списку дочірніх елементів вирішує проблему навігації у маленькому дисплеї мобільного пристрою, тому що може у просторі, обмеженому півколом може відтворювати нескінченну кількість елементів. Також за допомогою умовних позначень активних та неактивних елементів у вигляді точок, даний дизайн є зрозумілим користувачу.

5.3.5 Компоненти вкладеного лінійного списку

Клас `ConceptList` відповідає за візуалізацію лінійного вкладеного списку понять. Даний список будується рекурсивним методом на основі структури екземплярів класу `Node`. Схематичну структуру класу `ListPosition` представлено на рисунку 5.9. Даний клас складається з поля, яке зберігає поняття, прапору згорнутості та списку нащадків. Нащадки у даному класі представляють з себе об'єкти класу `ListPosition`. Після того, як головний `ListPosition` було рекурсивно сформовано, маємо деревовидну структуру з екземплярів `ListPosition`. Даний клас є віджетом, який складається з головного тексту та декількох рядків тексту нащадків, які знаходяться знизу з невеликим відступом вправо. Текст нащадок може бути у згорнутому положенні. Дана візуалізація і рекурсивність

класу `ListPosition` дозволяють будувати повноцінний вкладений список з функцією згортання понять.

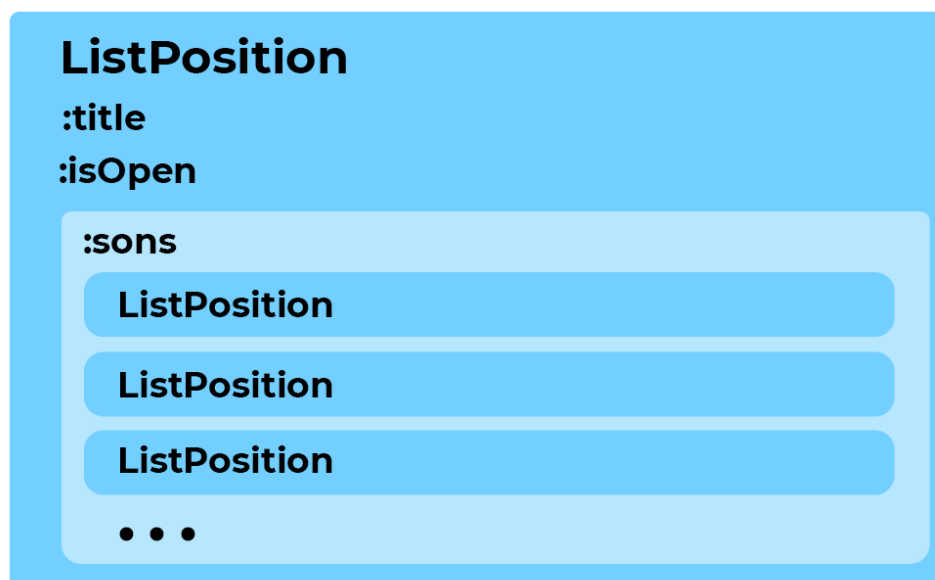


Рисунок 5.9 – Схематична структура класу `ListPosition`

Класи, назва який починається з `Paint` відповідають за візуалізацію ребр з стрілками. Для побудови використовується системний клас `Canvas`, який дозволяє будувати на екрані будь які геометричні фігури. Лінія ребра будується по двом точкам: центрам вершин, які його утворюють. Стрілка посередині ребра будується по трьом точкам, які вираховуються автоматично за допомогою геометричних розрахунків.

5.3.6 Компоненти побудови карт понять

Контролер забезпечує зв'язок між користувачем та системою, передає дані користувача до систему та навпаки. У контролері представлена уся логіка застосунку, яка може змінювати об'єкти моделі. У системі навчального мобільного застосунку контролер відповідає за логіку побудови карт понять, логіку побудови навігаційних панелі, пошук тощо. Контролер працює з сутностями моделі, які було описано у підпункті 5.3.1.

Клас `ForceDirectedController` відповідає за логіку побудови, задання кольорів та поведінки карт понять. У даному класі прописано модифікований алгоритм Фрухтермана-Рейнгольда [9], який описано у розділі 4. Клас `BalloonTreeController` відповідає за надання початкового положення вершин карти понять. Свою назву даний контролер бере від техніки візуалізації графів, у якій нащадки кожної вершини розміщуються на колі деякого одного діаметру. Клас `BottomSheetController` відповідає за логіку роботи нижньої комбінованої інформаційної панелі.

5.4 Створення макетів інтерфейсів програмного застосунку

Макет програмного застосунку – це графічне представлення застосунку, його дизайн. У ньому відображається кольорова схема, приблизний порядок елементів, шрифти тощо. Від того наскільки якісно виконано макет залежить досвід використання програмного застосунку користувачем та його загальне сприйняття. Створення графічного макету передуює етапу верстки та написання логіки застосунку. Схематичне представлення елементів макету дуже спрощує роботу з майбутнім ресурсом.

Для навчального мобільного застосунку на базі карт понять на етапі проектування було створено декілька макетів основних інтерфейсів. На рисунку 5.10 представлено приклад макету сектору карти понять і інформаційної панелі. На даному макеті продемонстровано один з концептів представлення карти понять, який було розроблено на початкових етапах роботи над проектом. Також на макеті можна побачити елементи навігації у вигляді різних кнопок для зуму, від яких було вирішено відмовитись, через можливість замінити їх на звичні для сенсорних мобільних пристроїв жести. На макеті є початкова концепція навігації, яка далі при розробці буде доповнено та винесено у окремий інтерфейс.

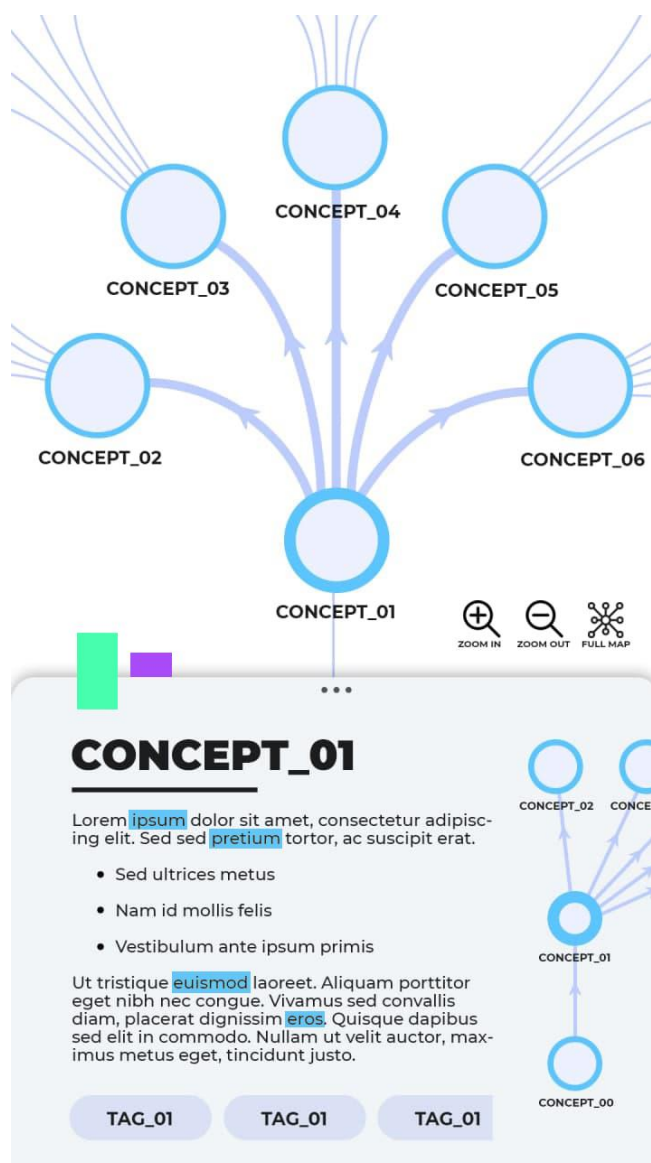


Рисунок 5.10 – Макет сектору карти понять і інформаційної панелі

На початкових етапах розробки дизайну було запропоновано концепцію зв'язку навчального мобільного застосунку та паперового конспекту. Різнокольорові елементи угорі інформаційної панелі є аналогами стікерам, які використовуються у конспектуванні для швидкого доступу до потрібної сторінки. У мобільному застосунку дані стікери є перемикачами між інформаційним і навігаційними станами нижньої панелі. Також на макеті можна побачити деякі слова у інформаційній панелі, які виділено синім кольором. Даний елемент використовується для звернення уваги на деякі частини навчального тексту та є аналогом до маркерів, які використовують для

підкреслення важливої інформації при конспектуванні. Дані аналогії з реальним світом у дизайні допомагають створити унікальний стиль застосунку, який забезпечить користувачу позитивний досвід користування застосунком.

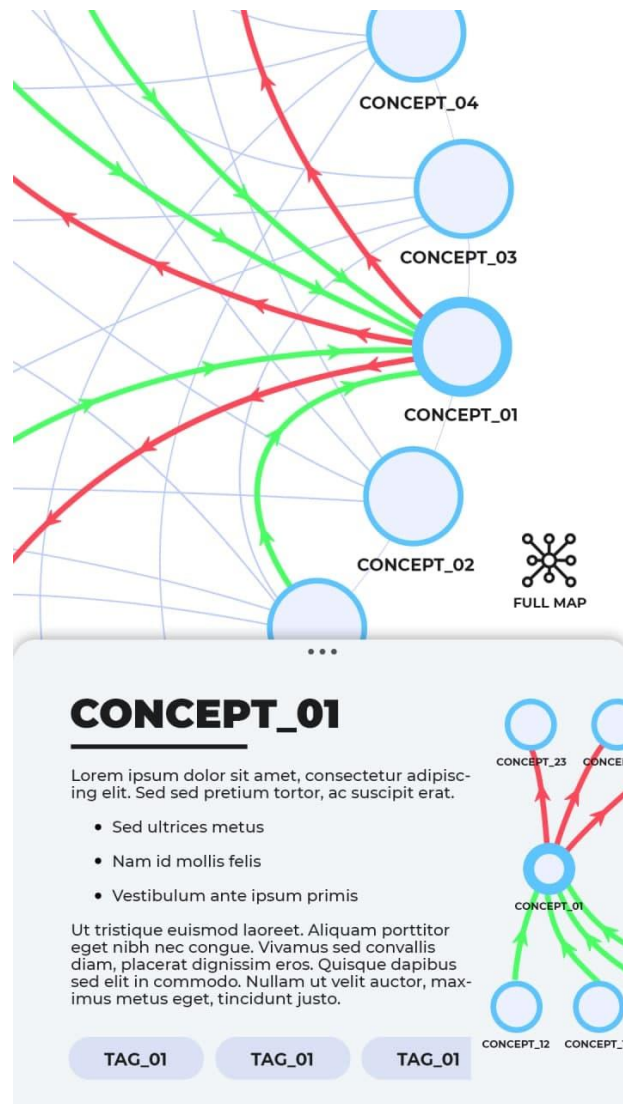


Рисунок 5.11 – Макет кругової карти понять

У ході розробки було запропоновано унікальний дизайн кругових інтерактивних карт понять. Концепція полягала у тому, щоб розмістити вершини на колі деякого радіуса з центром за видимими межами екрану. Зв'язки між вершинами повинні були відображатися лініями з деяким викривленням. Для навігації по даній карті понять,

даний модуль мав бути інтерактивним, для цього даний граф повинен був обертатись навколо центра головного кола після конкретних жестів пальцями користувача. При натисканні на вершину, вона повинна була ставати активною, а її зв'язки підсвічуватись. Але через велику завантаженість даної карти для сприйняття користувачем, було прийнято відмовитись від даної концепції. Даний дизайн став основою для кругової навігаційної панелі, яка має схожий функціонал.

6 РОБОТА КОРИСТУВАЧА З ЗАСТОСУНКОМ

Мобільний застосунок працює на платформі Android 6 та вищої версії. Користувач має встановити застосунок на мобільний пристрій через Google Play Market або виконуваний файл. Також можливий варіант використання емулятору мобільного пристрою для ПК. Після інсталяції застосунок не потребує ніяких додаткових дій та готовий до роботи.

Одразу після запуску користувач може побачити список доступних до вивчення курсів. На рисунку 6.1 продемонстровано дане меню. Користувач може обрати область знань, яку він вирішив вивчати.

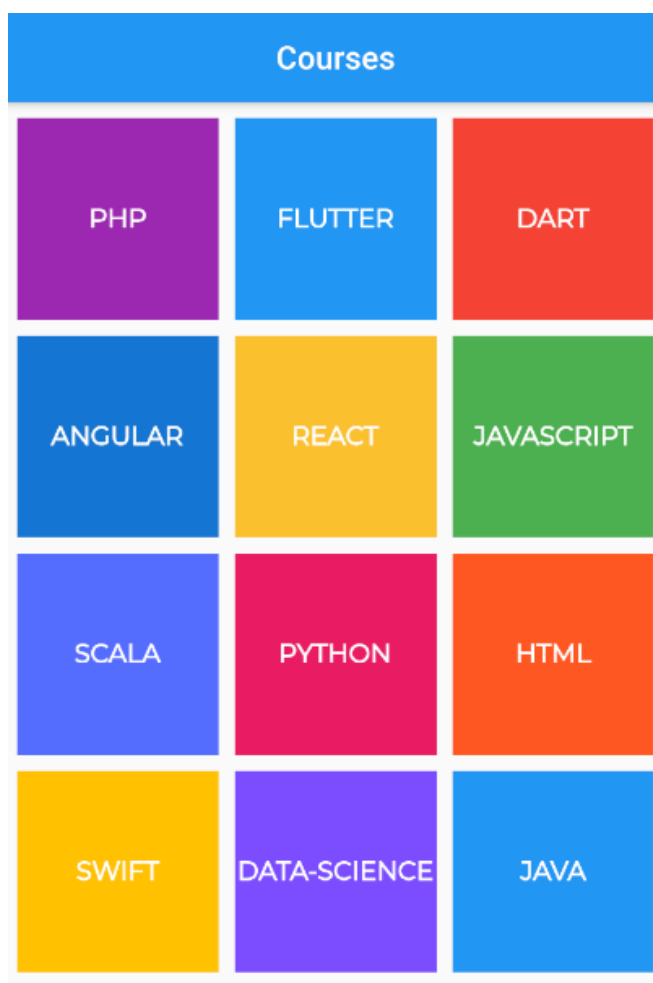


Рисунок 6.1 – Меню вибору курсу

Після вибору курсу користувач бачить на екрані побудовану різнокольорову карту понять по даній області знань, згорнуту нижню панель, іконки пошуку та меню. Користувач може вільно переміщатись по карті понять та зумити до вершин за допомогою стандартних для сенсорних мобільних пристроїв жестів. Також користувач може натиснути на вершину або ребро карти понять, дані дії активують нижню панель та нададуть користувачу інформацію про поняття або зв'язок між ними відповідно. На рисунку 6.2 представлено інтерфейс активної вершини та розгорнутого інформаційного меню.

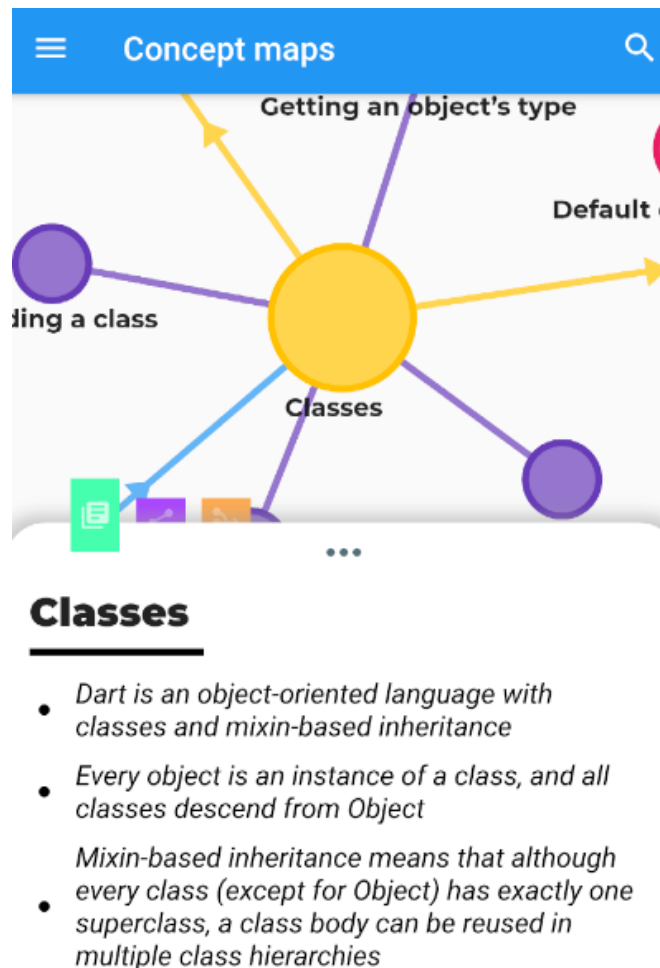
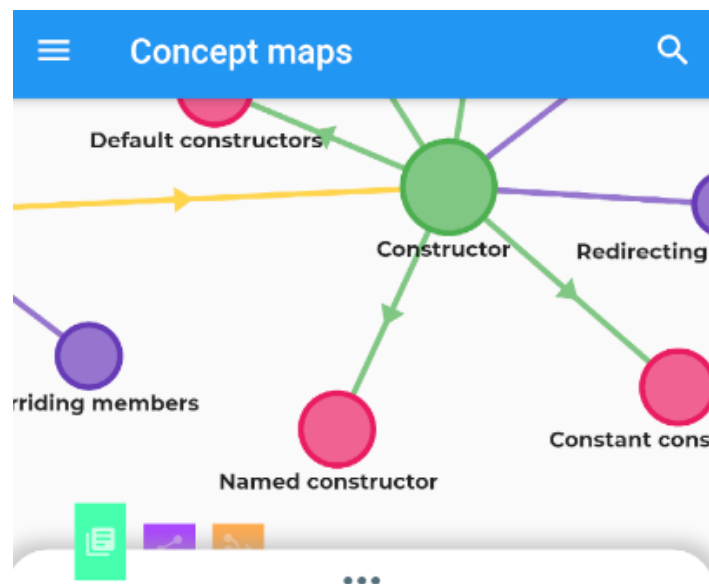


Рисунок 6.2 – Інтерфейс з активною вершиною і розгорнутим меню з інформацією про поняття

При натисканні на ребро карти понять користувач може побачити інформацію про тези вершин, які утворюють дане ребро. Для зручності, знаходження понять у даних тезах підсвічується різними кольорами, які відповідають кольорам вершин на головній карті понять. На рисунку 6.3 представлено приклад даного інтерфейсу, де зеленим підсвічено входження до тез поняття конструктора, а рожевим поняття іменованого конструктора. Даний підхід дозволяє вивчати курс більш детально та різносторонньо, бо розкриває тему зв'язків понять.



Constructor — Named constructor

Constructor in Named constructor theses

- Use a **named constructor** to implement multiple **constructors** for a class or to provide extra clarity
 - Constructors** are not inherited, which means that a superclass's **named constructor** is not inherited by a subclass
- If you want a subclass to be created with a

Рисунок 6.3 – Інтерфейс з активним зв'язком і розгорнутим меню з інформацією про зв'язок

Користувач може побачити різнокольорові стікери угорі інформаційної панелі. Дані стікери відповідають за стан нижнього меню. Доступно два стани: інформаційний та навігаційних. В інформаційній користувач може бачити тези, які стосуються даного поняття або зв'язку, кожен тип тези має свій особистий дизайн, що дозволяє користувачу отримувати різне інформаційне навантаження від різних тез.

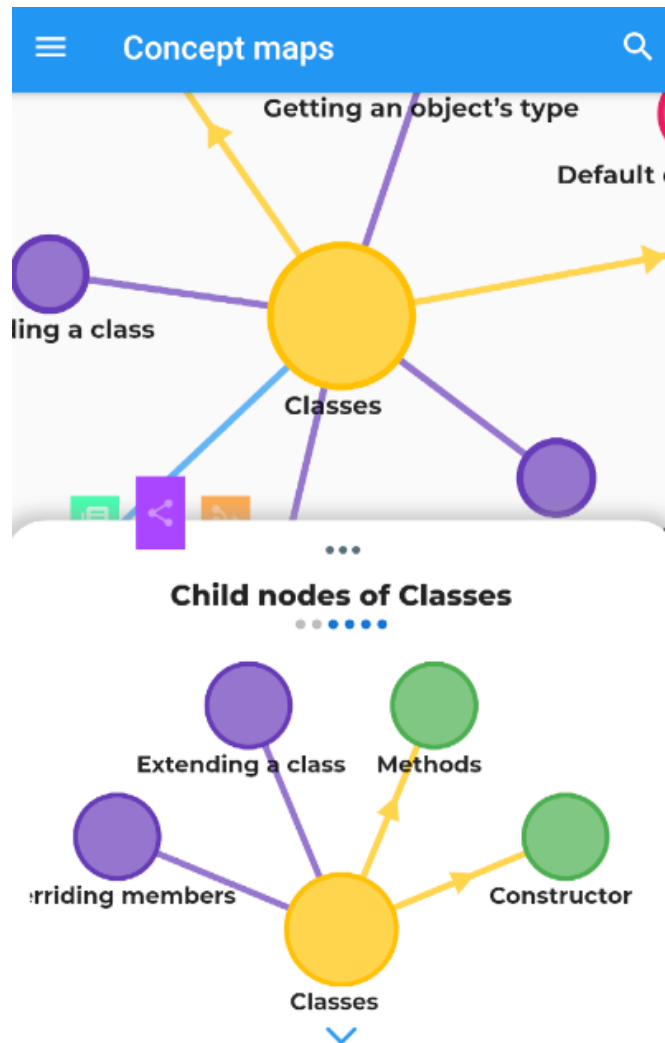


Рисунок 6.4 – Інтерфейс з активною вершиною і розгорнутим навігаційним меню

В навігаційному стані користувач може побачити зручне кругове навігаційне меню. Дане меню дозволяє побачити усі доступні дочірні поняття та здійснити навігацію до них або до попереднього поняття. Користувач може прокручувати кругове меню у різні

сторони. При натисканні на дочірнє поняття здійснюється плавний перехід до відповідної вершини на головній карті понять та навігаційне меню автоматично перезавантажується під нове активне поняття. При натисканні на іконки стрілки знизу активного поняття користувач здійснює перехід до попередньої активної вершини. Таким чином, користувач має змогу без перешкод досліджувати усю карту понять.

У ситуаціях, коли кількість дочірніх вершин є дуже великою, користувач може заплутатись. Для вирішення даної проблеми було додано індикатор вершин, які знаходяться у полі зору користувача. Даний індикатор знаходиться у верхній частині меню. Кола у даному індикаторі відповідають усім дочірнім вершинам, сіре відповідають тим, які не у полі зору користувача, сині – ті, які у полі зору. На рисунку 5.4 користувач може побачити навігаційне меню.

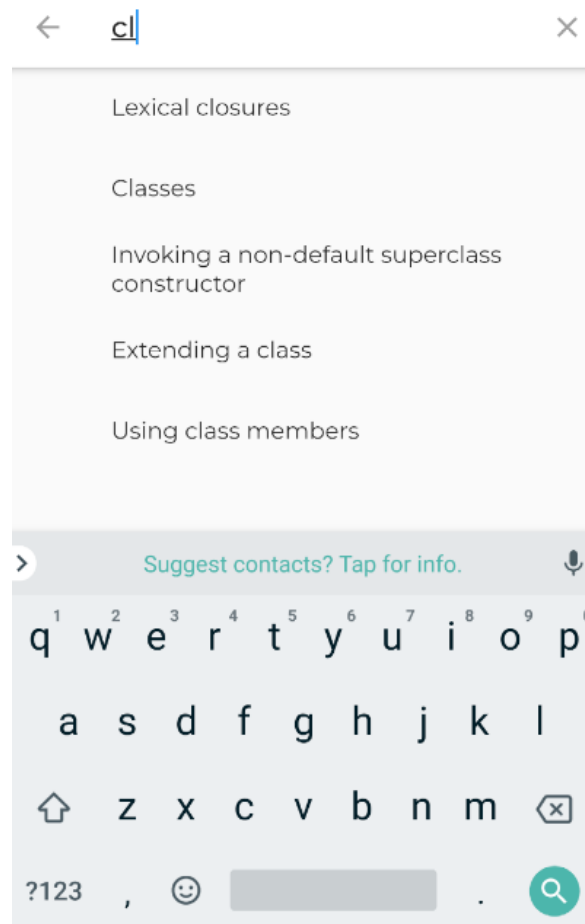


Рисунок 6.5 – Інтерфейс пошуку

Для більшої зручності користувача, було додано пошук по вершинах. На рисунку 6.5 представлено інтерфейс пошуку у мобільному додатку. Користувачу у реальному часу надаються варіанти понять, які відповідають вже введеному рядку. Після вибору користувачем потрібного поняття, відбувається плавний перехід до відповідної вершини на головній карті понять. Пошук допомагає користувачу знаходити інформацію про конкретні поняття та робить процес навчання більш оптимальним.

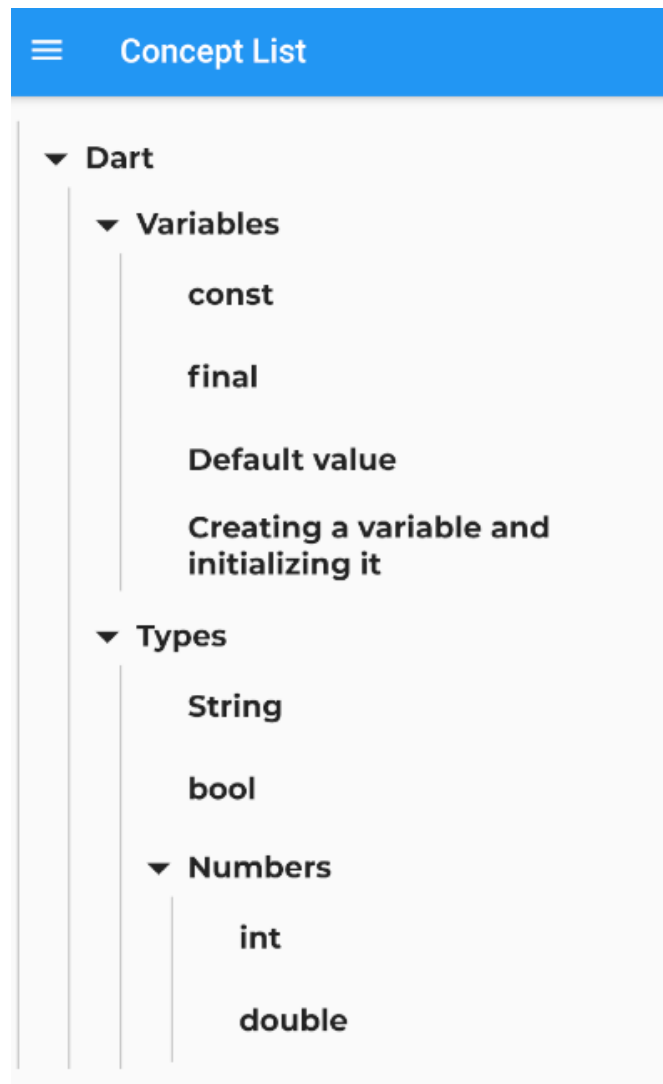


Рисунок 6.6 – Інтерфейс лінійного вкладеного списку понять

Для різностороннього вивчення однієї теми користувачу надається доступ до лінійного списку понять. На рисунку 6.6 продемонстровано інтерфейс лінійного

вкладеного списку понять. Даний список представляє конструкцію з вкладених понять. Користувач має змогу згорнути та розгорнути підписки для більш наглядного спостереження ієрархії понять. Надання користувачу можливості подивитись на матеріал для вивчення під іншим кутом є дуже важливим, бо допомагає краще засвоїти матеріал. Функціонал, описаний вище, спрямований на якісне та різностороннє вивчення користувачем деякої області знань. Користувачу надано інструменти для нестандартного підходу до навчання.

ВИСНОВКИ

Під час роботи над бакалаврською роботою було розроблено навчальний мобільний застосунок на базі карт понять, який має зручний інтерфейс, функцію аналізу відношень, пошук та навігацію. Додаток дозволяє користувачам вивчати потрібну їм інформацію одразу зі своїх смартфонів, дає можливість подивитись на візуальну інтерпретацію курсу у вигляді графу та проаналізувати відношення між його вершинами, що дозволяє краще зрозуміти та запам'ятати матеріал. Мобільний додаток написано за допомогою фреймворку Flutter мовою програмування Dart у середі розробки Android Studio. Загалом під час виконання дипломної роботи було:

- 1) досліджено наукові статті та літературу за темою бакалаврської роботи. Проведено аналіз існуючих навчальних систем на базі карт понять і методів побудови графів;
- 2) розроблено власний алгоритм візуалізації карт понять на основі алгоритму Фрухтермана-Рейнгольда. Проведено тестування продуктивності оригінального і модифікованого алгоритмів;
- 3) розроблено унікальний дизайн мобільного застосунку, який включає у себе інтерфейси опису курсів, карти понять, інформаційної панелі, панелі навігації, пошуку, лінійного вкладеного списку тощо;
- 4) побудовано архітектуру програмного застосунку, на основі якої було розроблено програмний продукт, дотримуючись парадигм ООП та використовуючи шаблон проектування MVC;
- 5) протестовано готову мобільну систему та розроблено технічну специфікацію та інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Полєнова В. А., Феденко В. А., Коваленко Д. Р., Титенко С. В. Методи підвищення наочності карт понять у навчальних мобільних застосунках // Сталий розвиток — XXI століття. Дискусії 2020: колективна монографія / Національний університет “Києво-Могилянська академія” / за ред. проф. Хлобистова Є.В. — Київ, 2020. — С. 435-441 с. — Електронне видання. ISBN: 978-617-7668-22-9
2. S.V. Tytenko, “Construction of didactic ontology based on the analysis of concept-thesis model elements”, *Naukovi Visti NTUU KPI*, no. 1, pp. 82—87, 2010.
3. Paivio, A. (1986). *Mental representations: A dual coding approach*. Oxford, UK: Oxford University Press.
4. Romero, C., Cazorla, M., & Buzón, O. (2017). Meaningful learning using concept maps as a learning strategy. *Journal of Technology and Science Education*, 7, 313-332.
5. Beckman, Brian (1994), *Theory of Spectral Graph Layout*, Tech. Report MSR-TR-94-04, Microsoft Research
6. Bastert, Oliver; Matuszewski, Christian (2001), "Layered drawings of digraphs", in Kaufmann, Michael; Wagner, Dorothea (eds.), *Drawing Graphs: Methods and Models*, Lecture Notes in Computer Science, 2025, Springer-Verlag
7. Di Battista, Giuseppe; Eades, Peter; Tamassia, Roberto; Tollis, Ioannis G. (1994), "Algorithms for Drawing Graphs: an Annotated Bibliography", *Computational Geometry: Theory and Applications*
8. Peter Eades. A heuristic for graph drawing. *Congressus numerantium*, 42:149–160, 1984.
9. Thomas M. J. Fruchterman; Edward M. Reingold (1991). *Graph drawing by force-directed placement.* , 21(11), 1129–1164. doi:10.1002/spe.4380211102
10. Kamada T and Kawai S. An algorithm for drawing general undirected graphs. *Inf Proces Lett* 1989; 31(1): 7–15.
11. Schwab, Michail; Strobelt, Hendrik; Tompkin, James; Fredericks, Colin; Huff, Connor; Higgins, Dana; Strezhnev, Anton; Komisarchik, Mayya; King, Gary; Pfister, Hanspeter

- (2017). booc.io: An Education System with Hierarchical Concept Maps and Dynamic Non-linear Learning Plans. *IEEE Transactions on Visualization and Computer Graphics*, 23(1), 571–580. doi:10.1109/tvcg.2016.2598518
12. R. Grosse and C. Reed. Metacademy Deep Belief Networks Concept Map. https://metacademy.org/graphs/concepts/deep_belief_networks#mode=explore
13. Nesbit, J. C., & Adesope, O. O. (2006). Learning With Concept and Knowledge Maps: A Meta-Analysis. *Review of Educational Research*, 76(3), 413–448. doi:10.3102/00346543076003413
14. Garlan, D. and M. Shaw. “An Introduction to Software Architecture.” *Advances in Software Engineering and Knowledge Engineering* (1993).
15. Tytenko, S. V. INTERACTIVE CONCEPT MAPS IN ONTOLOGY-ORIENTED INFORMATION AND LEARNING WEB-SYSTEMS. *KPI Science News*, no. 2, pp. 24–36, 2019. doi:10.20535/kpi-sn.2019.2.167515

ДОДАТОК А

Інтерактивні навчальні карти понять для мобільних пристроїв

Специфікація

УКР.НТУУ «КПІ»_ТЕФ_АПЕПС_ ТВ7180_21А 1

Аркушів 2

Київ – 2021

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ"КП" _ТЕФ_АПЕПС_ ТВ7180 _20Б 2	Записка Феденко В.А. ТВ-71.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ"КП" _ТЕФ_АПЕПС_ ТВ7180 _20Б 2-1	force_directed_con troller.dart	Компонент побудови карт понять
УКР.НТУУ"КП" _ТЕФ_АПЕПС_ ТВ7180 _20Б 2-2	bottom_sheet_grap h.dart	Компонент побудови панелі навігації
УКР.НТУУ"КП" _ТЕФ_АПЕПС_ ТВ7180 _20Б 2-3	api_service.dart	Компонент роботи з API

ДОДАТОК Б

Інтерактивні навчальні карти понять для мобільних пристроїв

Лістинг програми

УКР.НТУУ «КПІ»_ТЕФ_АПЕПС_ТВ7180_21Б 2

Аркушів 11

Київ – 2021

force_directed_controller.dart

```

import 'package:concept_maps/controllers/balloon_tree_controller.dart';
import 'package:concept_maps/models/graph_entities/concept_header.dart';
import 'package:concept_maps/models/graph_entities/edge.dart';
import 'package:concept_maps/models/graph_entities/map_model.dart';
import 'package:concept_maps/models/graph_entities/vertice.dart';
import 'package:concept_maps/utlis/node_value_list.dart';
import 'package:flutter/material.dart';
import 'package:vector_math/vector_math_64.dart';
import 'dart:math';
import 'package:concept_maps/models/graph_entities/node.dart';

class ForceDirectedController {
  ForceDirectedController(this.map);

  MapModel map;

  BalloonTreeController balloon;

  List<Vertice> vertices = [];
  List<Edge> edges = [];
  List<Widget> widgets = [];
  List<Widget> titles = [];
  int rootId;

  //Модуль перетворення об'єктів понять і зв'язків у вершини і ребра
  crToVE() {
    balloon = BalloonTreeController()..relationToNodes(map);
    rootId = int.parse(balloon.firstId);
    map.concepts.forEach((a) {
      vertices.add(Vertice(a.id, a.concept, a.isAspect, a.forms));
    });
    map.relations.forEach((a) {
      edges.add(Edge(vertices[vertices.indexWhere((b) => b.id == a.conceptId)],
        vertices[vertices.indexWhere((b) => b.id == a.toConceptId)]));
    });
    map.concepts.forEach((a) {
      if (a.isAspect == "1") {
        vertices.firstWhere((element) => element.id == a.id).fullTitle =
          vertices.firstWhere((element) => element.id == a.aspectOf).title
            +": "
            +a.concept;
        edges.add(Edge(vertices[vertices.indexWhere((b) => b.id == a.id)],
          vertices[vertices.indexWhere((b) => b.id == a.aspectOf)]));
      } else {
        vertices.firstWhere((element) => element.id == a.id).fullTitle =
          a.concept;
      }
    });
  }

  //Модуль надання вершинам початкового положення рекурсивно
  void nodeRecursion(Node node, double originDeg, List<Node> three) {
    if (node.child.length > 0) {
      for (var j = 0; j < node.child.length; j++) {
        for (var i = 0; i < node.child.length - 1; i++) {
          if (i < node.child.length / 2.floor()) {
            if (three[three.indexWhere((a) => a.id == node.child[i])]
              .child
              .length >
              three[three.indexWhere((a) => a.id == node.child[i + 1])]
              .child
              .length) {
              String temp = node.child[i];
              node.child[i] = node.child[i + 1];
              node.child[i + 1] = temp;
            }
          }
        }
      }
    }
  }

```

```

    } else {
        if (three[three.indexOf((a) => a.id == node.child[i])]
            .child
            .length <
            three[three.indexOf((a) => a.id == node.child[i + 1])]
            .child
            .length) {
            var temp = node.child[i];
            node.child[i] = node.child[i + 1];
            node.child[i + 1] = temp;
        }
    }
}
}

double deltaDeg = 15.0;
double deltaDegNum = (node.child.length - 1) / 2.0;
double deg = originDeg - deltaDegNum * deltaDeg;
double r;

if (node.child.length > 0) {
    node.child.forEach((element) {
        r = 200;
        double x = r * cos(deg * (pi / 180.0)) + node.x;
        double y = r * sin(deg * (pi / 180.0)) + node.y;
        three[three.indexOf((a) => a.id == element)].x = x;
        three[three.indexOf((a) => a.id == element)].y = y;
        //print([x, y]);
        nodeRecursion(
            three[three.indexOf((a) => a.id == element)], deg, three);
        deg = deg + deltaDeg;
    });
} else {}
}

void setVerticesEdgesColors(List<Node> tree) {
    vertices.forEach((a) {
        int treeIndex = tree.indexOf((element) => element.id == a.id);
        a.mainColor = tree[treeIndex].mainColor;
        a.sideColor = tree[treeIndex].sideColor;
        a.size = tree[treeIndex].d;
    });

    edges.forEach((a) {
        if (a.v.mainColor == NodeValueList.color[0][0] ||
            a.u.mainColor == NodeValueList.color[0][0]) {
            a.edgeColor = NodeValueList.color[0][1];
        } else {
            a.edgeColor =
                tree.firstWhere((element) => element.id == a.u.id).sideColor;
        }
    });
}

//Модуль надання вершинам початкового положення
void setVerticesPos(Offset size) {
    int x;
    int y;
    Random rand = new Random();
    balloon.three[balloon.three.indexOf((element) => element.parent == "-1")]
        .x = size.dx / 2;
    balloon.three[balloon.three.indexOf((element) => element.parent == "-1")]
        .y = size.dy / 2;
    List<Node> branch = balloon.three.where((element) =>
        element.parent ==
        balloon
        .three[
            balloon.three.indexOf((element) => element.parent == "-1")])

```



```

        (v.position.y - u.position.y));
    if (dis <= forceRadius) {
        if(dis < hotDisp){
            v.isHot = true;
            v.hotDistance = forceRadius;
        }
        delta.x = v.position.x - u.position.x;
        delta.y = v.position.y - u.position.y;

        delta.x = delta.x * fRep(l, delta.length) / delta.length;
        delta.y = delta.y * fRep(l, delta.length) / delta.length;

        v.displacement.x = v.displacement.x + delta.x;
        v.displacement.y = v.displacement.y + delta.y;
    }
}
});

//edges repulsion force
edges.forEach((element) {
    if (element.u.id != v.id && element.v.id != v.id) {
        double halfX = (element.u.position.x + element.v.position.x) / 2;
        double halfY = (element.u.position.y + element.v.position.y) / 2;
        dis = sqrt((v.position.x - halfX) * (v.position.x - halfX) +
            (v.position.y - halfY) * (v.position.y - halfY));
        if (dis <= forceRadius) {
            if(dis < hotDisp){
                v.isHot = true;
                v.hotDistance = forceRadius;
            }
            delta.x = v.position.x - halfX;
            delta.y = v.position.y - halfY;

            if (delta.length == 0) {
                delta.x++;
            }

            delta.x = delta.x * fRep(l, delta.length) / delta.length;
            delta.y = delta.y * fRep(l, delta.length) / delta.length;

            v.displacement.x = v.displacement.x + delta.x;
            v.displacement.y = v.displacement.y + delta.y;
            //}
        }
    }
});

//vertices attraction force
edges.forEach((e) {
    delta.x = e.v.position.x - e.u.position.x;
    delta.y = e.v.position.y - e.u.position.y;

    delta.x = delta.x * fAttr(l, delta.length) / delta.length;
    delta.y = delta.y * fAttr(l, delta.length) / delta.length;

    e.v.displacement.x = e.v.displacement.x - delta.x;
    e.v.displacement.y = e.v.displacement.y - delta.y;

    e.u.displacement.x = e.u.displacement.x + delta.x;
    e.u.displacement.y = e.u.displacement.y + delta.y;

    e.u.atr = Vector2(delta.x, delta.y);
    e.v.atr = Vector2(delta.x, delta.y);
});

vertices.forEach((v) {

```

```

if (v.isOn == false) {
    double d = test3(iter, i, forceRadius, l, v.isHot, v.hotDistance, force);
    v.isHot = false;
    v.hotDistance -= forceRadius/((iter-i)/100);
    v.position.x =
        v.position.x + v.displacement.x * test3(iter, i, forceRadius, l, v.isHot, v.hotDistance, v.displacement.x) / v.displacement.length;
    v.position.y =
        v.position.y + v.displacement.y * test3(iter, i, forceRadius, l, v.isHot, v.hotDistance, v.displacement.y) / v.displacement.length;

    v.displacementPrev = v.displacement.length;
    v.position.x = min(size.dx, max(0, v.position.x));
    v.position.y = min(size.dy, max(0, v.position.y));

    v.prevPosition = Vector2(v.position.x, v.position.y);

    avX = v.displacement.x;
    avY = v.displacement.y;
    if((avX <= maxDisp && avX >= -maxDisp) && (avY <= maxDisp && avY >= -maxDisp)){
        satCount++;
    }
}
t = test(size, iter, i);
});
i++;
if(satCount >= vertices.length){
    isSat = false;
}

if(i >= 1000){
    isSat = false;
}
satCount = 0;
durationsList.add(st.elapsed);
}
durationsList.add(st2.elapsed);
return durationsList;
}

bool inRange(double x1, double x2, double x3) {
    if (x1 > x2) {
        if (x3 >= x2 && x3 <= x1) {
            return true;
        } else {
            return false;
        }
    } else if (x1 <= x2) {
        if (x3 >= x1 && x3 <= x2) {
            return true;
        } else {
            return false;
        }
    }
}

//Модуль контролю переміщення вершин
double test3(int iter, int i, double rad, double l, bool isHot, double hotDistance, double force) {
    Random rand = Random();
    int r = rand.nextInt(i+1);
    if(iter == 1){
        if(force < 500 && force > -500){
            return 5;
        } else if(force < 1000 && force > -1000){
            return 45;
        } else if(force < 2000 && force > -2000){
            return 65;
        } else if(force < 5000 && force > -5000){
            return 110;
        }
    }
    else if(force < 10000 && force > -10000) {

```

```

    return 190;
  }
  else{
    return 250;
  }
}
else if(isHot){
  return hotDistance/(i+1);
}
else{
  return rad - (rad/iter)*i;
  //return rad - (rad/force) + 5;
}
}
}
}

```

api_service.dart

```

import 'package:concept_maps/models/dedactic_relations_entities/concept_in_theses.dart';
import 'package:concept_maps/models/general_entities/concept.dart';
import 'package:concept_maps/models/general_entities/thesis.dart';
import 'package:concept_maps/models/graph_entities/map_model.dart';
import 'package:concept_maps/utills/course_key_list.dart';
import 'package:http/http.dart' as http;

```

```

abstract class ApiService {
  static const String _hostUrl = "https://semantic-portal.net";

  //Отримуємо дані про концепт
  static Future<MapModel> _fetchConceptRelations(String field) async {
    final String relationsUrl =
      "$_hostUrl/api/branch/$field/concepts/relations";
    final String conceptsUrl = "$_hostUrl/api/branch/$field/concepts";
    final String headerUrl = "$_hostUrl/api/CinH/$field";

    final response = await Future.wait<http.Response>(
      [http.get(relationsUrl), http.get(conceptsUrl), http.get(headerUrl)],
    );

    final relationsResponse = response.first;
    final conceptsResponse = response[1];
    final headerResponse = response[2];

    if (relationsResponse.statusCode == 200 &&
        conceptsResponse.statusCode == 200) {
      final List relationsResult = jsonDecode(relationsResponse.body);
      final List conceptsResult = jsonDecode(conceptsResponse.body);
      final List headerResult = jsonDecode(headerResponse.body);

      return MapModel.fromMap(
        field, relationsResult, conceptsResult, headerResult);
    }

    throw Exception("Error occurred during fetch of map model");
  }

  //Отримуємо дані про гілку
  static Future<List<MapModel>> fetchBranches() async {
    final List<MapModel> maps = await Future.wait<MapModel>(
      CourseListInfo.courseKeyList
        .map<Future<MapModel>>((key) => _fetchConceptRelations(key))
        .toList(),
    );

    return maps;
  }

  //Отримуємо дані про концепти у тезах
  static Future<ConceptInTheses> fetchConceptsInTheses(int conceptId) async {
    assert(conceptId != null);
  }

```

```

final url = '$_hostUrl/api/concept/$conceptId/CinT';

final response = await http.get(url);
if (response.statusCode == 200) {
  final result = jsonDecode(response.body);

  return ConceptInTheses.fromMap(result, conceptId);
}

throw Exception("Error occurred during fetch of concepts in theses");
}
//Отримуємо дані дидактичні зв'язки
static Future<List<Concept>> fetchConceptsDidacticBefore(
  int conceptId) async {
  assert(conceptId != null);
  final url = '$_hostUrl/api/concept/$conceptId/didactic/before';

  final response = await http.get(url);
  if (response.statusCode == 200) {
    final result = jsonDecode(response.body);

    return List<Concept>.from(
      result?.map((x) => Concept.fromJson(x)),
    );
  }

  throw Exception(
    "Error occurred during fetch of concepts in didactic before");
}
//Отримуємо дані дидактичні зв'язки
static Future<List<Concept>> fetchConceptsDidacticAfter(int conceptId) async {
  assert(conceptId != null);
  final url = '$_hostUrl/api/concept/$conceptId/didactic/after';

  final response = await http.get(url);
  if (response.statusCode == 200) {
    final result = jsonDecode(response.body);

    return List<Concept>.from(
      result?.map((x) => Concept.fromJson(x)),
    );
  }

  throw Exception(
    "Error occurred during fetch of concepts in didactic after");
}
//Отримуємо тезу по айди поняття
static Future<List<Thesis>> fetchThesesByConceptId(int conceptId) async {
  assert(conceptId != null);
  final url = '$_hostUrl/api/concept/$conceptId/thesis';

  final response = await http.get(url);
  if (response.statusCode == 200) {
    final result = jsonDecode(response.body);

    return List<Thesis>.from(
      result?.map((x) => Thesis.fromMap(x)),
    );
  }

  throw Exception("Error occurred during fetch of theses by concept id");
}
//Отримуємо дані про тезу
static Future<List<Thesis>> fetchTheses(List<int> thesisIds) async {
  assert(thesisIds != null);
  assert(thesisIds.isNotEmpty);

  final thesisIdsStr = thesisIds.join(",");

```

```

final url = '$_hostUrl/api/thesis/$thesisIdsStr';

final response = await http.get(url);
if (response.statusCode == 200) {
  final result = jsonDecode(response.body);

  return List<Thesis>.from(
    result?.map((x) => Thesis.fromMap(x)),
  );
}

throw Exception("Error occurred during fetch of selected theses");
}
}

```

bottom_sheet_graph.dart

```

import 'package:concept_maps/models/graph_entities/map_model.dart';
import 'package:concept_maps/providers/app_provider.dart';
import 'package:concept_maps/views/widgets/dots_indicator.dart';
import 'dart:math';

import 'package:flutter/animation.dart';
import 'package:flutter/material.dart';
import 'package:concept_maps/models/graph_entities/node.dart';
import 'package:flutter/services.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';
import 'package:concept_maps/views/paint_bottom_graph.dart';

class BottomSheetGraph extends StatefulWidget {
  @override
  _BottomSheetGraphState createState() => _BottomSheetGraphState();
}

class _BottomSheetGraphState extends State<BottomSheetGraph>
  with SingleTickerProviderStateMixin {
  Node node;
  List<Node> tree;
  List<Node> newNodes = [];
  MapModel map;
  List<Widget> widgets = [];

  Animation<double> animation, curve;
  AnimationController animationController;
  double bottomSheetCof = 0.5;
  //модуль анімації панелі навігації
  void runAnimation(double velocity) {
    curve =
      CurvedAnimation(parent: animationController, curve: Curves.easeInOut);

    animation = animationController
      ..drive(CurveTween(curve: Curves.easeInOut))
      ..drive(Tween<double>(begin: velocity, end: 0));
    animationController.forward(from: 0.0);
  }
  //модуль додавання віджета до представлення
  void addWidget(Node node) {
    TextPainter textPainter = TextPainter(
      text: TextSpan(text: node.title),
      maxLines: 1,
      textDirection: TextDirection.ltr)
      ..layout(minWidth: 0, maxWidth: double.infinity);
    double textWidth = textPainter.width;

    widgets.add(Positioned(
      top: node.y - node.r,
      left: node.x - node.r,
      child: GestureDetector(

```

```

onTap: () {

  Provider.of<AppProvider>(context, listen: false)
    .setAnimationParam(node.id);
},
child: Container(
  height: 2 * node.r,
  width: 2 * node.r,
  decoration: BoxDecoration(

    color: node.sideColor,
    border: Border.all(color: node.mainColor, width: 3),
    borderRadius: BorderRadius.circular(2*node.r)),
  ),
),
));

widgets.add(Positioned(
  top: node.y + node.r,
  left: node.x - textWidth * 0.63,
  child: Container(
    child: Text(
      node.title,
      style: GoogleFonts.montserrat(
        fontSize: 14,
        fontWeight: FontWeight.w700,
      ),
    ),
  ),
));
}

//модуль додавання до представлення граничного віджета
void addBorderWidget(Node node) {
  final size = MediaQuery.of(context).size;
  widgets.add(Positioned(
    top: node.y,
    child: GestureDetector(
      onTap: () {
        Provider.of<AppProvider>(context, listen: false)
          .setAnimationParam(node.id);
      },
      child: Container(
        height: 4 * node.r,
        width: node.x * 2,
        decoration: BoxDecoration(color: Color(0xffffffff)),
      ),
    ),
  ));

  List<bool> dots = [];
  tree.firstWhere((a) => a.id == node.id).child.forEach((element) {
    if(node.child.contains(element)){
      dots.insert(0, true);
    }
    else{
      dots.insert(0, false);
    }
  });

  widgets.add(Positioned(
    top: 0,
    child: Container(
      width: size.width - 30,
      child: Text(
        "Child nodes of "+node.title,
        textAlign: TextAlign.center,
        style: GoogleFonts.montserrat(
          fontSize: node.r/2,
          fontWeight: FontWeight.w800,

```

```

    ),
  ),
),
));

widgets.add(Positioned(
  top: node.r/2 + 8,
  child: DotsIndicator(dots),
));
}
//модуль додавання до представлення заднього віджета
void addBackWidget(Node node) {
  widgets.add(Positioned(
    top: node.y - 1.3 * node.r,
    left: node.x - node.r,
    child: Container(
      height: 2 * node.r,
      width: 2 * node.r,
      //color: Color(0xff2f0df4),
      child: Container(
        child: Transform.rotate(
          angle: 90 * pi / 180,
          child: IconButton(
            onPressed: () {
              Provider.of<AppProvider>(context, listen: false)
                .setAnimationParam(node.id);
            },
            icon: Icon(
              Icons.arrow_forward_ios_rounded,
              color: Color(0xff2a9df4),
            ),
          ),
        ),
      ),
    ),
  ));
}
}
//представлення панелі навігації
@override
Widget build(BuildContext context) {
  final size = MediaQuery.of(context).size;
  return Consumer<AppProvider>(
    builder: (context, value, _) => GestureDetector(
      onPanDown: (details) {},
      onPanUpdate: (details) {
        setState(() {
          resetPosition(details.delta);
        });
      },
      onPanEnd: (details) {
        setState(() {
          runAnimation(details.velocity.pixelsPerSecond.dx / 50);
        });
      },
      child: Container(
        width: size.width - 30,
        height: size.height * 0.4,
        //color: Color(0xffd04fff),
        child: CustomPaint(
          painter: PaintBottomGraph(newNodes, value.focusNode),
          child: Stack(
            children: widgets,
          )),
      )),
  );
}
}

```

ДОДАТОК В

Інтерактивні навчальні карти понять для мобільних пристроїв

Опис програмного коду

УКР.НТУУ«КПІ»_ТЕФ_АПЕПС_ТВ7180_21В 3

Аркушів 10

Київ – 2021

АНОТАЦІЯ

Начальний мобільний застосунок на базі карт понять має такі функціональні можливості:

- Доступ до курсів на базі інтерактивних карт понять;
- Доступ до понять;
- Доступ до тез понять;
- Доступ до навігації;
- Доступ до пошуку по поняттям;

Мобільну систему розроблено у середовищі розробки Android Studio, використовуючи фреймворк Flutter та мову програмування Dart. Для побудови карт понять було використано дані з API навчального веб-порталу.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ.....	77
2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ	78
3. ОПИС ЛОГІЧНОЇ СТРУКТУРИ	79
4. ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ.....	80
5. ВИКЛИК І ЗАВАНТАЖЕННЯ.....	81
6. ВХІДНІ ДАНІ	82
7. ВИХІДНІ ДАНІ.....	83

ЗАГАЛЬНІ ВІДОМОСТІ

Додаток А містить специфікацію мобільної системи.

Додаток Б містить програмний код головних функціональних модулів системи.

Додаток В містить опис основних класів мобільної навчальної системи на основі інтерактивних карт понять.

Для коректної роботи застосунку потрібен мобільний девайс з встановленою операційною системою Android версії 6.0 або вище. Також застосунком можна користуватися за допомогою емулятора.

Мобільна система розроблена у середовищі розробки Android Studio мовою програмування Dart, використовуючи фреймворк Flutter.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Навчальний мобільний застосунок на базі інтерактивних карт понять призначений для студентів та має такі функціональні можливості:

1. Доступ до карт понять;
2. Доступ до нижньої інформаційної панелі;
3. Доступ до інформації про поняття;
4. Доступ до навігаційної панелі;
5. Можливість пошуку поняття на карті;
6. Можливість навігації по курсу.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Згідно з архітектурою системи, навчальний мобільний застосунок складається з трьох основних модулів:

- Модуль побудови карт понять;
- Модуль побудови навігації;
- Модуль аналізу зв'язків між поняттями і тезами.

Модуль побудови карт понять складається з класів, що містять опис алгоритмів побудови графів, сутності карт понять, такі як класи понять, тез, зв'язків тощо, допоможні сутності для побудови карт понять.

Модуль побудови панелі навігації складається з контролера побудови кругового меню навігації та класу представлення даного меню у інтерфейсі користувача.

Модуль аналізу зв'язків між поняттями і тезами складається з API-сервіса, за допомогою якого здійснюється зв'язок з API навчального порталу, контролери і представлення понять і тез тощо.

ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Для навчання за допомогою мобільного застосунку користувач потребує мобільний девайс або емулятор з встановленою операційною системою Android. Для користування застосунком користувач завжди має мати підключення до мережі інтернет для завантаження даних з API навчального порталу. Після відключення від мережі вже завантажені дані зберігаються і користувач має можливість користуватися завантаженою картою понять.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Для користування мобільним застосунком потрібно завантажити та інсталювати програму. Додаток можна використовувати на смартфоні з операційною системою Android. Завантажити програмне забезпечення можна за посиланням <https://drive.google.com/drive/folders/1I4pnzIDBgRkaMoBjdeHtAvjoUDoOY0pE?usp=sharing>.

Для інсталяції програмного забезпечення потрібно запустити завантажений файл на мобільному пристрої. Після завершення інсталяції програма готова до її використання.

ВХІДНІ ДАНІ

Вхідними даними для роботи мобільного застосунку є дані з API навчального веб-порталу. Для побудови карт понять і курсів використовуються дані про поняття, їх зв'язки, тези, форми назв понять, інформація про наявність понять у тезах та наявність понять у заголовках.

ВИХІДНІ ДАНІ

Вихідними даними розробленої системи є побудовані карти понять, інформація про обрані поняття у вигляді тез, що йому належать, та результат аналізу відношень між поняттями у вигляді списку тез з виділеними ключовими словами.

ДОДАТОК Г

Інтерактивні навчальні карти понять для мобільних пристроїв

Публікації

УКР.НТУУ«КПІ»_ТЕФ_АПЕПС_ТВ7180_21Г 4

Аркушів 2

Київ – 2021

АЛГОРИТМИ ПОБУДОВИ КАРТ ПОНЯТЬ У НАВЧАЛЬНИХ МОБІЛЬНИХ ЗАСТОСУНКАХ

У зв'язку із широким використанням інтернету та стрімким розвитком інформаційних технологій навчальні веб-сервіси набирають усе більшу популярність серед користувачів. Інформаційно-навчальні веб-ресурси є зручним та доступним джерелом професійної та загальноосвітньої інформації. Так як візуалізація грає важливу роль у засвоєнні знань, карти понять мають позитивний вплив на когнітивні здібності й підвищують ефективність навчання [1]. Формалізація інформаційного наповнення та онтологічне моделювання предметної області забезпечують подання та візуалізацію навчальної інформації, саме тому навчальні ресурси на базі карт понять є ефективними. Водночас, використання мобільних застосунків набирає усе більших масштабів, саме тому існує попит у навчальних мобільних системах.

Під час розробки навчального мобільного застосунку на базі карт понять з'являється низка проблем. Найбільш обмежуючим фактором є відносно невеликий розмір екрану, що зменшує наочність навантажених карт. Іншою проблемою є розробка алгоритму побудови дерева, у якому вузли не будуть заважати одне одному, а також будуть відсутні заплутані зв'язки між поняттями.

Існує багато різних алгоритмів побудови навантажених графів, вони поділяються за способом реалізації. Серед них розрізняють довільні, прямолінійні, сіткові, полігональні, ортогональні тощо. Далі буде розглянуто найпопулярніші алгоритми побудови графів.

- Методи спектральної компонування використовують власні вектори матриці, виведені з матриці суміжності графіка, як координати [2].
- Ортогональні методи розмітки, що дозволяють краям графіка проходити горизонтально або вертикально, паралельно осям координат макета.
- Методи багат шарового малювання графів розташовують вузли графа в горизонтальних шарах за допомогою таких методів, як алгоритм Коффмана – Грехема, таким чином, що більшість ребер рухаються вниз від одного шару до наступного. Після цього кроку вузли в кожному шарі розташовуються таким чином, щоб мінімізувати перетинання [3].
- Методи дугових діаграм розміщують вершини на лінії, краї можуть бути намальовані як півкола вище або нижче лінії, або як плавні криві, пов'язані між собою з декількох півкіл.
- Методи кругового макетування розміщують вершини графіка на колі, ретельно обираючи впорядкування вершин навколо кола, щоб зменшити перетинання і розміщувати сусідні вершини близько одна до одної.
- Силкові алгоритми візуалізації графів модифікують початкове розміщення вершин шляхом безперервного переміщення їх відповідно до системи сил, заснованої на фізичних концепціях, що відносяться до систем пружин або молекулярної механіки.

Надалі буде розглянуто силкові алгоритми візуалізації графів через їх переваги для вирішення задач, поставлених у даній роботі. Мета алгоритма полягає у тому, щоб розташувати вузли графа в двовимірному або тривимірному просторі так, що всі ребра мали б більш-менш однакову довжину, і звести до мінімуму кількість перетинів ребер шляхом призначення сил для безлічі ребер і вузлів ґрунтуючись на їх відносних положеннях [4].

Як тільки сили на вузлах і ребрах визначені, поведінку всього графа під дією цих сил можна ітеративно промодельовувати, як якщо б це була фізична система. У такій ситуації сили, що діють на вузли,

намагаються стягнути їх ближче або відштовхнути їх один від одного подальше. Це триває, поки система не прийде в стан механічної рівноваги, тобто положення вузлів не змінюється від ітерації у ітерації. Положення вузлів в цьому стані рівноваги використовується для генерації малюнка графа.

Цей метод є ідеальним для мобільних застосунків, тому що завдяки йому будується естетично приємний граф, який займає мінімальний простір, що дуже важливо через сильну обмеженість екранного простору.

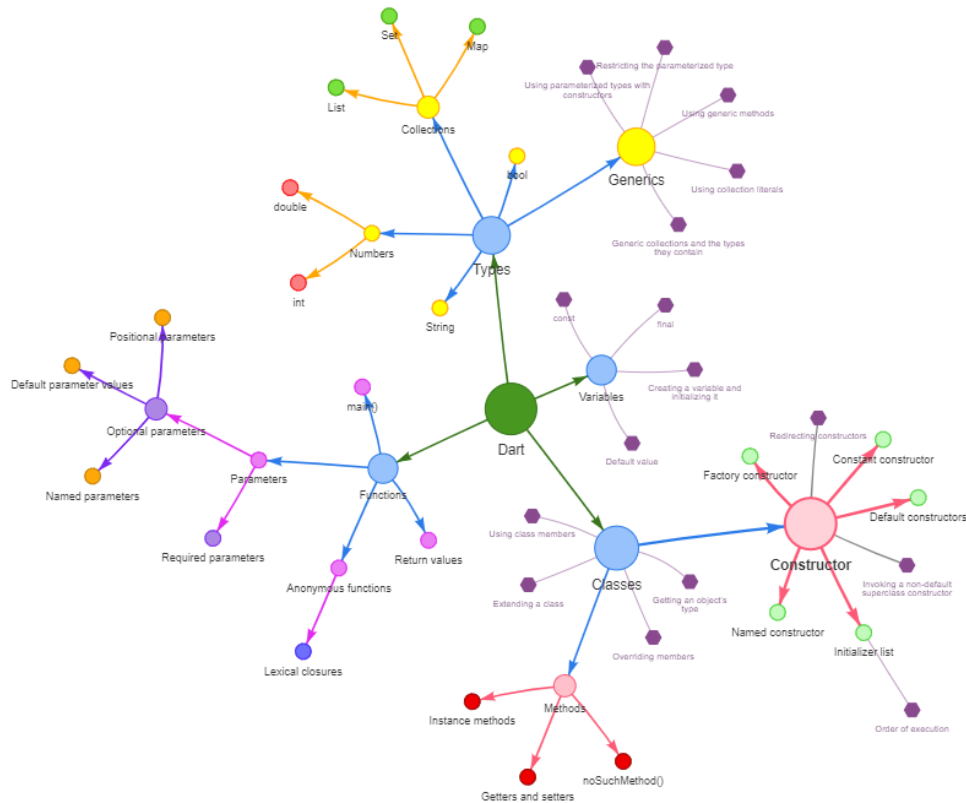


Рисунок 1. Приклад карти понять, побудованої за допомогою силового алгоритму візуалізації графів[5]

1. Coffey JW, Hoffman R, Cañas A. Concept Map-Based Knowledge Modeling: Perspectives from Information and Knowledge Visualization. *Information Visualization*. 2006;5(3):192-201. doi:10.1057/palgrave.ivs.9500129
2. Beckman, Brian (1994), *Theory of Spectral Graph Layout*, Tech. Report MSR-TR-94-04, Microsoft Research
3. Bastert, Oliver; Matuszewski, Christian (2001), "Layered drawings of digraphs", in Kaufmann, Michael; Wagner, Dorothea (eds.), *Drawing Graphs: Methods and Models*, Lecture Notes in Computer Science, 2025, Springer-Verlag
4. Di Battista, Giuseppe; Eades, Peter; Tamassia, Roberto; Tollis, Ioannis G. (1994), "Algorithms for Drawing Graphs: an Annotated Bibliography", *Computational Geometry: Theory and Applications*
5. Semantic Portal [Електронний ресурс] – Режим доступу до ресурсу: <http://semantic-portal.net>.