МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ВИКОРИСТАННЯ ТЕХНОЛОГІЙ GO, STATELESS І REACT.JS ДЛЯ РОЗРОБКИ ІГРОВОГО ДОДАТКУ»**

Виконав: студент 2 курсу, групи 8.1218
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
В.І. Тихоновський
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н, Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент
_____ Лісняк А.О.
(підпис)

« 29 » травня 2019 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Тихоновському Владиславу Ігоровичу
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Використання технологій Go, Stateless і React.js для розробки ігрового додатку

керівник роботи (проекту) Гоменюк Сергій Іванович, д.т.н, професор.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 29 » травня 2019 року № 811-с

2. Строк подання студентом роботи 27.12.2019

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Реалізація продукту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 29.05.2019

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	27.05.2019-09.07.2019	
2.	Збір вихідних даних.	09.07.2019-15.09.2019	
3.	Обробка методичних та теоретичних	15.09.2019-03.10.2019	
	джерел.		
4.	Розробка першого та другого розділу.	03.10.2019-05.11.2019	
5.	Розробка третього розділу.	05.11.2019-01.12.2019	
6.	Оформлення та нормоконтроль	01.12.2019-27.12.2019	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	15.01.2020	

Студент _____ В.І. Тихоновський
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Використання технологій Go, Stateless і React.js для розробки ігрового додатку»: 78 с., 48 рис., 15 джерел, 3 додатка.

GO, ІГРОВИЙ ДОДАТОК, WEB-APPLICATION, REACT, SINGLE PAGE APPLICATION.

Об’єкт дослідження – мова GO для створення серверної частини, SQL для розробки реляційної БД, інструментарій для розробки User interface мова JavaScript бібліотека React та stateless approach.

Мета роботи: створення web-application ігровий додаток для вивчання іноземних мов засобами інтерфейсу і звукових доріжок, з класифікованою складністю і типом гри.

Метод дослідження – вивчення специфікації та документації вибраних засобів, використання їх в процесі розробки.

У кваліфікаційній роботі розглядається високорівнева мова програмування з Go. В даній роботі розглядається JavaScript - це повноцінна динамічна мова програмування, яка застосовується до HTML документу, і може забезпечити динамічну інтерактивність на веб-сайтах, та Бібліотека React, яка використовується для створення призначеного для користувача інтерфейсу завдяки компонентам та контролю стану додаток завдяки Stateless підходу та бібліотеці контролю глобального стану Unstated.

У ході розробки було створено web-application ігровий додаток, в якому є можливість вивчання іноземних мов та можливість накопичування ігрових балів. Можливість грати в single player або в multiplayer до чотирьох осіб.

Перевагами web-application є простота у використанні можливість збільшувати ігровий контент та легкий перенос на будь-які мобільні платформи.

SUMMARY

Master's Degree in "Using of GO, Stateless and React.js technologies for Gaming Software Development": 78 pages, 48 screens, 15 sources, 3 applications.

GO, GAME APP, WEB-APPLICATION, REACT, SINGLE PAGE APPLICATION.

Objects of study - GO language for server-side creation, SQL for relational database development, user interface development toolkit with JavaScript language, React library and stateless approach.

The aim of the development is to create a web-application game for learning foreign languages by means of interface and audio tracks, with classified complexity and type of game.

The research method is studying the specifications and documentation of the selected tools, using them in the development process.

The thesis deals with high-level GO programming language. In this work, JavaScript is a full-featured dynamic programming language applied to HTML documents that can provide dynamic interactivity on websites. React Library is used to create a user-friendly interface due to the components and status control application and because of the Stateless approach and the Unstated global status control library.

During the development, a web-application game was created, which has the ability to learn foreign languages and the ability to collect game points.

The ability to play single-player or multiplayer for up to four people.

The benefits of web-application are: the ease of use of, the ability to increase game content and easy portability to any mobile platform.

ЗМІСТ

ВСТУП

Існує неймовірна кількість сайтів для вивчення іноземної мови. Зараз набагато простіше почати вивчати іноземні мови через інтернет ніж ходити на курси. В наш час є велика кількість сайтів для вивчення іноземної мови, такі як Lingaleo, Inspeak ті інші, які націлені на вивчення одної або декількох іноземних мов.

Хоча сервіси для вивчення іноземних мов є дуже популярним майже всі вони платні або не мають повного функціоналу у безкоштовної версії та розраховані на декілька мов. Тому я вирішив створити безкоштовний ігровий додаток для вивчення того як звучить іноземна мову у ігровій формі. Також важливо для додатка є використання сучасних і затребуваних технологію таких як серверна мова Go та бібліотека React.

Go – являється компільованою статично типізованою мовою програмування від компанії Google. Мова Go призначена для створення різного роду додатків, але перш за все це веб-сервіси і клієнт-серверні додатки. Хоча також мову має можливості для роботі з графікою [1].

React – JavaScript бібліотека, яка ставить собі за мету спрощення розробки візуальних інтерфейсів. Розроблений в Facebook у 2013 році, він став одним з найпоширеніших прикладів використання Single Page Application в світі, приводячи в дію Facebook і Instagram, а також багато інших компаній [2].

Темою для дипломної роботи є «Використання технологій Go, stateless і React.js для розробки ігрового додатку».

Актуальність створюваного продукту полягає в наступному:

а) можливість додавання нових іноземних мов;

б) можливість додавання нових аудіо файлів;

в) гнучкий режим складності

г) responsive design

д) можливість грати в single-player multiplayer до 4 осіб;

е) можливість зміни режиму гри;

ж) безкоштовність;

з) розроблено на сучасній мові програмування.

1 ОГЛЯД ТЕХНОЛОГІЙ

1.1 Вирішення проблеми

Створення розширення ігрового додатку є проблемою, яку необхідно вирішити в даній роботі. Найважливішою проблемою є те, що у даного додатку не має аналогів, точніш ми маємо такий аналог як сайт Lingaleo який також націленний на вивчення іноземної мови. Але основний функціонал та ідея у додатків відрізняються. Тому усю логіку, взаємодію технологій, дизайн і user interface, user experience доводиться розробляти самому. Також великою частину часу розробки додатку буде займати не тільки написання коду та його тестування, та ще й додавання даних про мову та звукові треки у базу даних.

1.1.1 Опис розробки серверу на мові Go

Go компільований статично типізована мова програмування від компанії Google. Мова Go призначена для створення різного роду додатків, але перш за все це веб-сервіси і клієнт-серверні додатки.

Одним з авторів є Кен Томпсон, який, до слова, є і одним з авторів мови Сі (поряд з Денисом Рітчі). 10 листопада 2009 року Go був анонсований, а в березні 2012 року вийшла версія 1.0. При цьому мова продовжує розвиватися. Поточною версією на момент написання магістерською роботи є версія 1.12 [1].

Мова Go розвивається як open source, тобто представляє проект з відкритим вихідним кодом, і все його коди і компілятор можна знайти і використовувати безкоштовно.

Go є кросплатформним, він дозволяє створювати програми під різні операційні системи - Windows, Mac OS, Linux, FreeBSD. Код можна адоптувати під платформи: програми, написані для однієї з цих операційних систем, можуть бути легко перекомпільовані і перенесені на іншу ОС [1].

Що потрібно для роботи з Go? Перш за все необхідний текстовий редактор для набору коду і компілятор для перетворення коду в виконуваний файл. Також можна використовувати спеціальні інтегровані середовища розробки (IDE), які підтримують Go, наприклад, GoLand від компанії JetBrains. Існують плагіни для Go для інших IDE, зокрема, IntelliJ IDEA і Netbeans [3].

В даний час Go знаходить широке застосування в різних сферах. Зокрема, серед відомих проектів, які застосовують Go, можна знайти такі: Google,

Розглянемо плюси та мінуси Go.

Плюси:

а) компільований – компілятор транслює програму на Go в машинний код, зрозумілий для певної платформи;

б) статично типізований;

в) модульний;

г) присутній збирач сміття, який автоматично очищає пам'ять;

д) підтримка роботи з мережевими протоколами;

е) Підтримка багато поточності і паралельного програмування;

ж) зручність – писати код за допомогою Golang зручно, швидко та безпечно, а також підтримується між платформна підтримка, яка просто не може бути поганою. Google піклується про користувача. Вони створили багато безкоштовних офіційних посібників та зробили Golang відкритим кодом, так що тепер існує величезна колекція доповнень та пакетів. Go є частиною Google Cloud, Dropbox, MS Azure та AWS [1];

з) актуальність – У Google є гроші. Це не означає, що ви заробляєте гроші від Google, використовуючи Golang, але це означає, що ця технологія має регулярне фінансування та вдосконалення. Навряд чи Google піде куди-небудь найближчим часом. Ви можете бути впевнені, що Go довго буде грати велику роль в архітектурі Google;

и) швидкість – мова Go розроблявся в якості заміни C: його висока продуктивність майже порівнянна з мовою Сі, але більш простий синтаксис дає можливість розробляти програми набагато швидше (наприклад, як на Python).

При цьому багато розробників вивчають цю мову після Python або PHP або використовують дві мови в зв'язці (Python / Go і PHP / Go). Спрощений синтаксис полегшує не тільки написання свого власного коду, але і читання коду, написаного іншими програмістами, що особливо важливо в командній роботі. Ця особливість Go, в свою чергу, веде до іншого важливого факту: швидке освоєння Go дозволяє перевести увагу з вивчення самої мови на вивчення програмування в цілому [3].

Мінуси:

а) не можна не відзначити і чітку спрямованість мови Go: на відміну від PHP, на якому пишуть великі проекти, Go більше призначений для невеликих сервісів, які необхідно швидко написати і впровадити, але які повинні відрізнятися надзвичайної надійністю;

б) не популярна мова програмування серверів.

1.1.2 Особливості мови програмування JavaScript

Для розробки візуальної частини додатку мною була обрана мова JavaScript.

Як і всі комп'ютерні мови, JavaScript має певні переваги та недоліки. Багато плюсів і мінусів пов'язані з JavaScript, який часто виконуються в браузері клієнта, але є й інші способи використання, які дозволяють йому в той самий час буди мовою на якій написаний сервера також є можливість написання кросплатформенних додатків завдяки Electron, Expo, React native, та іншим інструментам.

Переваги використання JavaScript:

а) швидкість – для клієнта є великою перевагою, оскільки його можна запустити відразу в браузері на стороні клієнта. Якщо зовнішні ресурси не потрібні, JavaScript не перешкоджає мережевим дзвінкам на сервер [4];

б) популярність – JavaScript використовується скрізь в Інтернеті. Ресурсів для вивчення JavaScript багато. StackOverflow та GitHub мають багато

проектів, які використовують Javascript, в останні роки особливо привертає увагу в галузі. JavaScript грає чудово з іншими мовами і може використовуватися в безлічі різних додатків. На відміну від скриптів PHP або SSI, JavaScript можна вставити на будь-яку веб-сторінку незалежно від розширення файлу. JavaScript також може використовуватися усередині скриптів, написаних іншими мовами, такими як Perl і PHP. Будучи клієнтською стороною знижує попит на сервері веб-сайту. Сьогодні JavaScript може виконуватися не тільки в браузері, а й на сервері або на будь-якому іншому пристрої, який має спеціальну програму. В даний час існує безліч способів використання JavaScript за допомогою серверів Node.js. Якщо ви хотіли завантажити версію node.js з Express, використовуйте базу даних наприклад mongodb, і використовуйте JavaScript на передній панелі для клієнтів, можна розробити цілий додаток використовуючи тільки JavaScript, також можливість створювати завдяки лише JavaScript не тільки веб-додатки а ще й комп'ютерні програми завдяки таким інструментам як Electron. Корпорація Electron розробила обробник Javascript таким чином, що додатки які розробили завдяки йому сумісні з великою кількість ОС [5];

в) регулярні оновлення – з моменту появи EcmaScript 5 (специфікації сценаріїв, на які покладається Javascript), Ecma International. На сьогоднішній день ми отримали підтримку браузера для ES6 і сподіваємось на підтримку ES7 в майбутніх місяцях [5];

г) динамічна типізація – повноцінна динамічна моя, яка приєднується до HTML-документа, і може забезпечити динамічну інтерактивність на веб-сайті. Его розробляв Брендан Ейх, який співпрацює з проектами Mozilla, Mozilla Foundation та Mozilla Corporation [7];

д) компактний – JavaScript сам по собі досить компактний, але дуже гнучкий. Написано велику кількість інструментів язика JavaScript, які розблокують величезне число додаткових функцій. Усередині середовища виконання JavaScript може бути пов'язаний з об'єктами цього середовища і надавати програмний контроль над ними. Можливість підключення бібліотек відразу в браузері є також перевагою.

1.1.3 Огляд бібліотеки React

Для пришвидшення и полегшення розробки було вирішено використовувати для візуальної частини сторонні бібліотеку. А саме бібліотеку React.

React – відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка були створена для вирішувати проблеми часткового оновлення вмісту веб-сторінки. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників [2].

React дозволяє розробникам створювати великі web-application, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс.

Щоб досягти цього, Facebook довелося оптимізувати сам процес розробки, і Джордан Уолк вирішив зробити це за допомогою JavaScript. Він запропонував включити XHP, синтаксис розмітки Facebook, в систему координат JS. Ідея видалася неможливою, але в 2011 році його команда випустила бібліотеку ReactJS на основі симбіозу JavaScript та XHP. Тоді Facebook зрозумів, що ReactJS працює швидше, ніж будь-яке інше подібне впровадження [2].

ReactJS - це бібліотека JavaScript, яка поєднує в собі швидкість JavaScript і використовує новий спосіб візуалізації веб-сторінок, роблячи їх дуже динамічними та чутливими до вводу користувачів.

Через два роки та ж група інженерів випустила React Native, гібридну структуру розробки мобільних додатків для iOS та Android. Інструмент базувався на тих самих принципах, що й ReactJS, і незабаром його прийняло інженерне співтовариство та компанії, які дотримуються стратегії мобільних телефонів.

Декларативний

React дозволяє створювати інтерактивні інтерфейси користувача безболісно. Створюйте прості представлення даних для кожного стану у вашій

програмі, і React буде ефективно оновлювати та відтворювати лише потрібні компоненти, коли ваші дані змінюються.

Компонентний

Можливість побудування інкапсульовані компоненти, які керують своїм станом або станом інших компонентів, а потім складайте їх, щоб створювати складні інтерфейси користувача.

Оскільки компонентна логіка написана на JavaScript замість шаблонів, ви можете легко переносити данні через свій додаток і залишати стан поза DOM. Ми можемо записати елементи інтерфейсу, як кнопка або поле введення, як компонент React. Компоненти є композиційними. Компонент може включати в свій вихід один або більше інших компонентів.

Загалом, для написання програм React ми пишемо компоненти, які відповідають різним елементам інтерфейсу. Потім ми організовуємо ці компоненти всередині компонентів вищого рівня, які визначають структуру нашої програми.

Наприклад, візьміть форму. Форма може складатися з багатьох елементів інтерфейсу, таких як поля введення, мітки або кнопки. Кожен елемент всередині форми може бути записаний як компонент React. Потім ми запишемо компонент вищого рівня, сам компонент форми. Компонент форми вказував би структуру форми і включав би кожен із цих елементів інтерфейсу всередині неї.

Можливість додавання бібліотеки та використання її не переписуючи існуючий код.

Одностороння передача даних

Властивості передаються в рендерер компоненту, як властивості html тега. Компонент не може напряму змінювати властивості, що йому передані, але може їх змінювати через callback функції. Такий механізм називають «властивості донизу, події нагору» [13].

Віртуальний DOM

DOM (об'єктна модель документа) - це логічна структура документів у форматах HTML, XHTML або XML. React підтримує віртуальний DOM, а не

покладається виключно на DOM браузера. Це дозволяє бібліотеці визначити, які частини DOM змінилися, порівняно зі збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера. Таким чином програміст працює зі сторінкою, вважаючи що вона оновлюється вся, але бібліотека самостійно вирішує які компоненти сторінки треба оновити. Характеризуючи це в простому розумінні, це угода про перегляд даних та входів та даних, яка має форму дерева. Веб-браузери використовують механізми компонування для перетворення або розбору HTML-синтаксису представлення в об'єктну модель документа, яку ми можемо побачити в браузерах. Основна проблема, що стосується традиційної конструкції DOM, полягає в тому, як вона обробляє зміни, тобто введення користувачів, запити тощо. Сервер постійно перевіряє різницю, викликану цими змінами, щоб дати необхідну відповідь. Щоб відповісти належним чином, йому також потрібно оновити дерева DOM усього документа, що не є ергономічним, оскільки дерева DOM сьогодні досить великі, містять тисячі елементів [15].

Команді React вдалося збільшити швидкість оновлень за допомогою віртуальної DOM. На відміну від інших бібліотек, що працюють з DOM, React використовує свою абстрактну копію - Virtual DOM. Він оновлює навіть мінімалістичні зміни, застосовані користувачем, але не впливає на інші частини інтерфейсу значно прискорює швидкість додатку. Це також можливо завдяки ізоляції компонентів React.

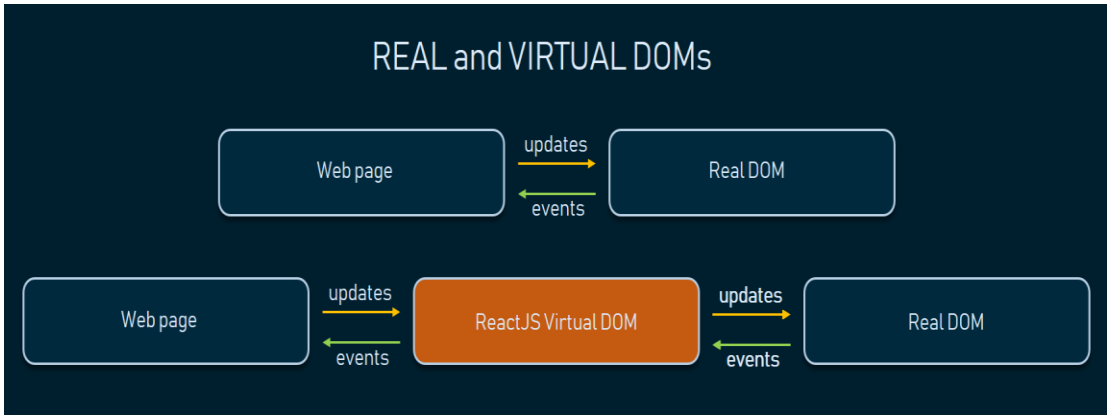

Рисунок. 1.1 – Взаємодія Virtual DOM з DOM веб-браузера

JSX

Компоненти React зазвичай написані на JSX. Розробники можуть так само писати на чистому JavaScript. JSX нагадує іншу мову, яку створили у компанії Фейсбук для розширення PHP, XHP [2].

React виходить із принципу, що логіка рендеринга нерозривно пов'язана з іншою логікою UI: з тим, як обробляються події, як стан змінюється в часі і як дані готуються до відображення.

Замість того, щоб штучно розділити технології, поміщаючи розмітку і логіку в різні файли, React розділяє відповідальність за допомогою слабо пов'язаних одиниць, компоненти які містять і розмітку, і логіку.

Продуктивність

ReactJS, як відомо, чудовий виконавець. Ця функція робить його набагато кращим, ніж інші бібліотеки сьогодні. Причиною цього є те, що він управляє віртуальним DOM.

1.1.4 Особливості Statless

У React компоненти дозволяють розділити користувальницький інтерфейс на незалежні багаторазові шматки та подумати про кожну частину окремо. Створюючи свою програму React, у вас буде багато різних компонентів. Деякі, швидше за все, змінюватимуть стан і, можливо, доведеться навіть отримувати доступ до вашого сервера. Також існують компоненти які вкілючають в себе інші компненти. Є компоненти які просто відповідають за візуальні частину вашого інтерфейсу.

У React є лише 2 типу компонентів stateless та stateful.

Stateless (Functional) – компонент без стану зазвичай асоціюється з тим, як представлена концепція користувачеві, це лише функція Javascript, яка повертає елемент React. Оголошуючи функціональний компонент, ви також можете використовувати функції стрілок. Ви можете використовувати рендери, та

контекстні API з функціональним компонентом. Ви також можете передати props до функціонального компонента [13].

Statefull – компонент завжди є компонентом класу. Він створюється шляхом розширення класу React.Component. Компонент, що зазначає стан, залежить від об'єкта стану і може змінити власний стан. Компонент відображається на основі змін у його стані та може передавати властивості його стану дочірнім компонентам як властивості об’єкта реквізиту [13].

Але ж Stateless це не тільки компоненти а ще підхід до розробки додатку. Тому в даному додатку ми будемо використовувати global state management. В міру ускладнення вашого додатку управління станом може стати виснажливим. У React стан компонента повинні бути самодостатніми, що утруднює передачу стану між декількома компонентами. Redux є передову бібліотеку для управління станом в React, однак, в залежності від того, наскільки складна ваша програма, вам може Redux і не знадобиться [2].

Unstated – альтернатива, яка надає можливість керувати станом декількох компонентів за допомогою компонентів класу Container, а також компонентів Provider і Subscription. Це доволі нова бібліотека, яка набирає певної популярності у громаді. Він використовує потужність API React Context, щоб зробити управління надзвичайно простим [6] .

За останні кілька років у спільноті React з'явилася низка різних моделей управління станом. Вони варіюються від прямого setState API, до сторонніх бібліотек, таких як Redux і MobX (їх більше, але це найбільш широко використовувані), і до нещодавно оновленого API Context [6].

1.1.5 Огляд допоміжних бібліотек

Для більш швидкої розробки графічного інтерфейсу ігрового додатку, було обрано фреймворк Bootstrap. Будь-який веб-розробник рано чи пізно замислюється про те, як йому спростити і прискорити процес верстки сайту також великим плюсом є можливість розробляти додаток для всіх пристойїв. У

зв'язку з цим, він вдається до допомоги css-фреймворків. Най популярніший з них - Bootstrap. Bootstrap це не тільки css, а ще й js-фреймворк. У Bootstrap написано готові стилі та скрипти, для прикладу яких вам достатньо всього лише прописати необхідні стильні класи та атрибути html-елемента [8].

Також були застосовані наступні бібліотеки:

а) Lodash – для спрощення обробки даних у JavaScript;

б) React Router – для керування переходами між вікнами додатку, та збереження стану цих переходів;

в) React Wavesurfer – для відображення стану аудіо файлу.

1.2 Порівняння обраних засобів з альтернативними

1.2.1 Порівняння GO з PHP

Lingaleo використовує PHP для серверу я ж для свого додатку буду використовувати мову Go бо вважаю що мова Go має такі переваги над PHP як.

Варіативність розробки.

Go в основному мова програмування, яка може бути використана для швидкого складання машинного коду, мова Go має призначеннння не тількі як серверна мова тоді як PHP в основному сценарій на стороні сервера, а також мова програмування загального призначення, розроблена для веб-розробки [3].

Шаблони

PHP використовує для цілей шаблону, і таким чином браузер через відправлений HTML-код обробляє код PHP, а вихід відправляється в браузер, тоді як у випадку GO зазвичай використовується проста система шаблонів [3].

Типізація

Go статично типізована мова, тоді як PHP має динамічну типізацію.

Менеджер пакетів

Go має свій менеджер пакетів, плюс є можливість використання бібліотек просто завдяки додаванню link у той час для PHP необхідно використовувати Composer [1].

1.2.2 Порівняння React з Angular

Lingaleo використовую jQuery а для даного додатку я вирішив використовувати React але зрівнювати jQuery и React на мою думку на є дуже коректним бо технологія jQuery вже давно відійшла на другий план. Тому я вважаю основним конкурентом React на даний час э Angular.

а) JSX це HTML подібний синтаксис, компільований в JavaScript. Розмітка і код знаходяться в одному файлі. Це рішення дозволяє вставляти посилання на функції, компоненти і змінні. І навпаки, малі шаблони Angular тягнуть за собою явні мінуси: немає підсвічування синтаксису в багатьох редакторах, немає повної підтримки автокомпіляції і підсвічування помилок в коді. Здавалося б це повинно привести до жахливого висновку повідомлень про помилки, але команда Angular написали свій власний парсер HTML коду;

б) React обробка помилок – коли ви робите помилку в JSX у React, він не буде компілюватися. Це чудова річ. Ви відразу точно знаєте в якому ряду помилка. Зрозуміло що це незакритий тег або посилання на оголошену змінну. Насправді, JSX компілятор позначе номер рядка в якій ви допустили помилку. Це істотно збільшує швидкість розробки. І навпаки, коли ви помиляєтеся з посиланням на змінну в Angular, нічого не відбудеться. Angular тихенько собі впаде під час виконання, а не компіляції;

в) React JavaScript – у цьому полягає ключова відмінність React і Angular. На жаль, Angular залишається HTML орієнтованим. Angular не вдалося вирішити найбільш принципову проблему архітектури. Angular продовжує поміщати JS в HTML. React ж поміщає HTML в JS. Це принципово впливає на досвід розробки. У Angular HTML орієнтований дизайн залишається його слабким місцем. Як я і підкреслив в JSX JavaScript більш потужний ніж HTML.

Набагато логічніше посилювати можливості JavaScript для підтримки розмітки, ніж HTML розширювати логікою. HTML і JavaScript все одно повинні бути якось склеєні і JavaScript орієнтований підхід React є безсумнівною перевагою над Angular, Ember і Knockout с їх HTML орієнтованим підходом.

1.3 Технічне завдання

Темою роботи є «Використання технологій Go, stateless і React.js для розробки ігрового додатку».

Було вирішено написати ігровий додаток який буде націлений на вивчення іноземної мови. Перевагами додатку є те що в ньому буде підтримка багатої кількості мов та діалектів, можливість додавання мов та звукових доріжок.

Також великим плюсом додатку буде не тільки те що його можна використовувати на комп’ютері а ще й на планшетах і мобільних пристроях.

Додаток повинен мати такий функціонал як:

а) вибір типу гри;

б) вибір складності гри;

в) перехід по сторінкам;

г) підрахунок балів;

д) авторизація для адміністратора;

е) можливість додавання нових мов та діалектів;

ж) завантаження аудіо файлів;

з) використовувати multiplayer до 4 осіб;

и) задання користувачам імена;

к) відображення таблицю результатів.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1 Проектування додатку з погляду користувача

Для користувача найважнішим аспектом додатку є web-interface. Починаючи з того як користуватися основними функціями додатку та закінчуючи адмін панеллю, де можлива додавати нові мови та аудіо файли до додатку. Основні критерії якості додатку:

а) дизайн – додаток повинен буди зрозумілим для користувача;

б) швидкість завантаження – додаток повинен швидко відкриватися та показувати інформацію користувачу;

в) швидкість додатку – швидкість анімацій, переходів по сторінкам, швидкість підсумку результатів та швидкість запитів. Це все є не мало важливим для користувача;

г) коректне відображення на всіх розширення екрану.

На діаграмі (див. рис. 2.1) зображено взаємодію користувача з додатком, на головній сторінці.

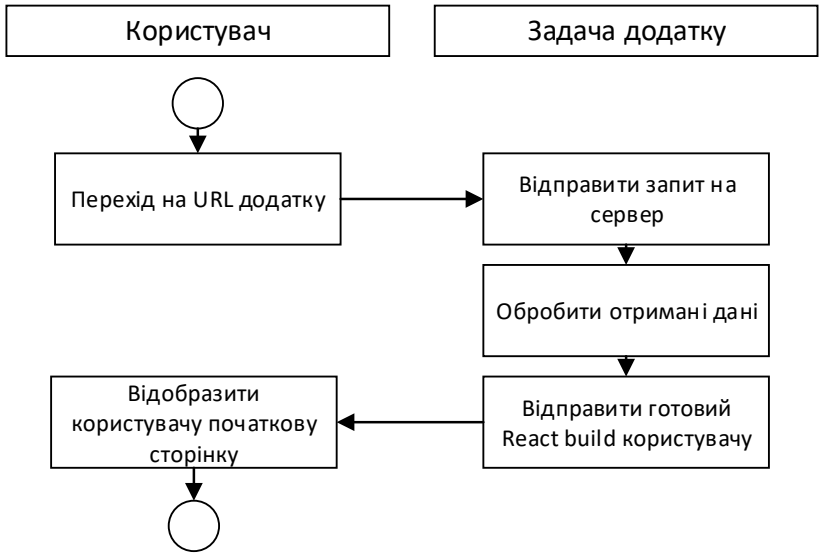

Рисунок 2.1 – Діаграма входу на головну сторінку

Також важливим в розробці є візуальний прототип. Перш ніж почати розробку додатку було вирішено спочатку розробити мокап. Мокав – це шаблон за яким потрібно робити користувальницький інтерфейс. Всі мокапи зберігаються в сервісі Figma.

Візуальне представлення головної сторінки на мокапі (див. рис. 2.2).

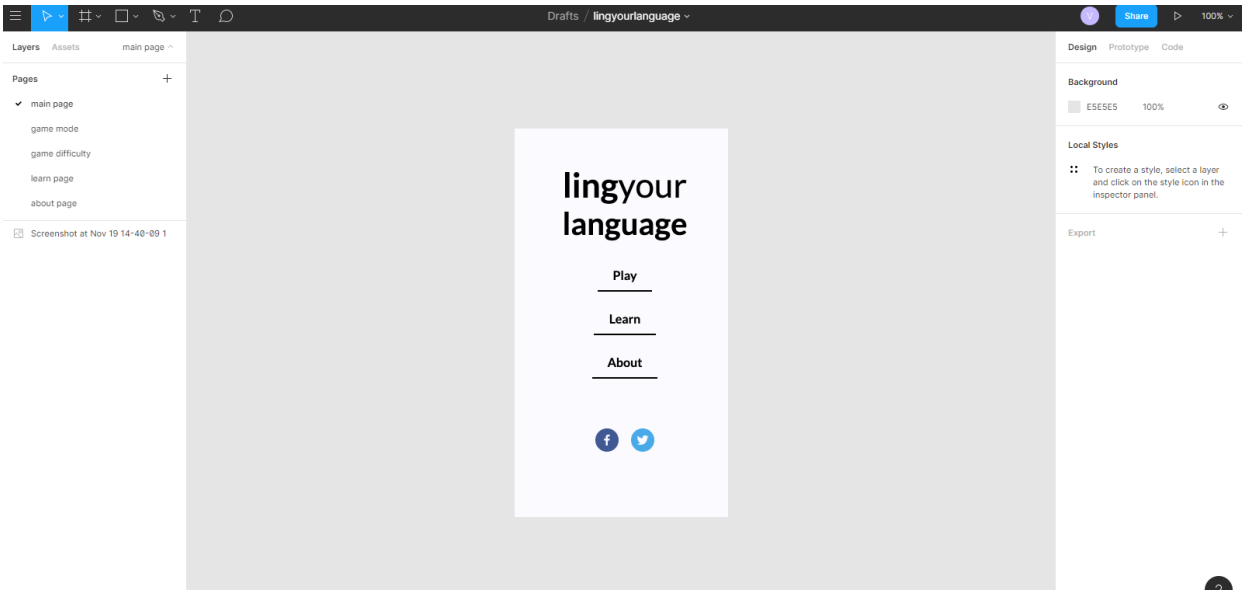

Рисунок 2.2 – Мокав головної сторінки

Далі розглянемо діаграму взаємодії при переході на сторінку Learn у додатку рисунок (див. рис. 2.3).

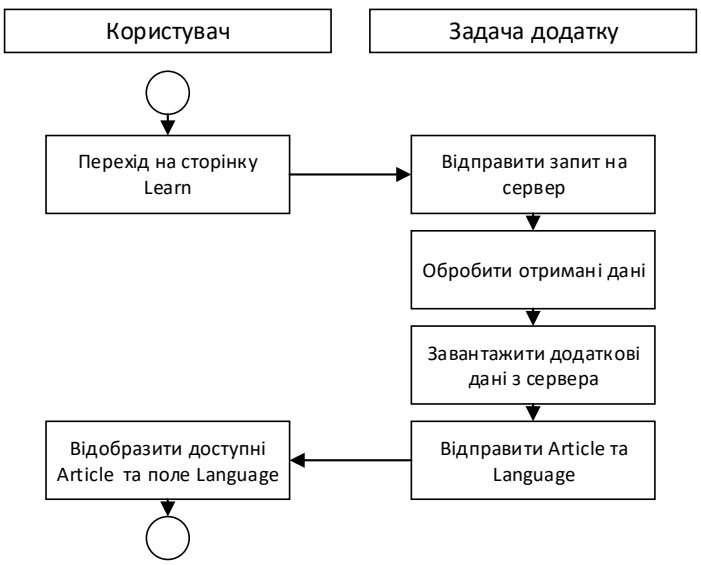

Рисунок 2.3 – Діаграма переходу на сторінку Learn

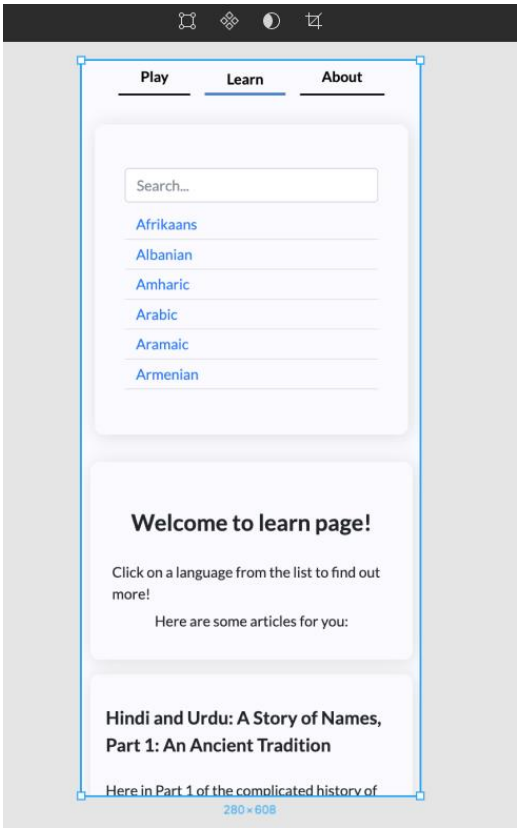

Рисунок 2.4 – Мокав сторінки Learn

Також у користувача є можливість пошуку мов зі списку, та відображення їх характеристик. На діаграмі (див. рис. 2.5) зображено взаємодію користувача з фільтром.

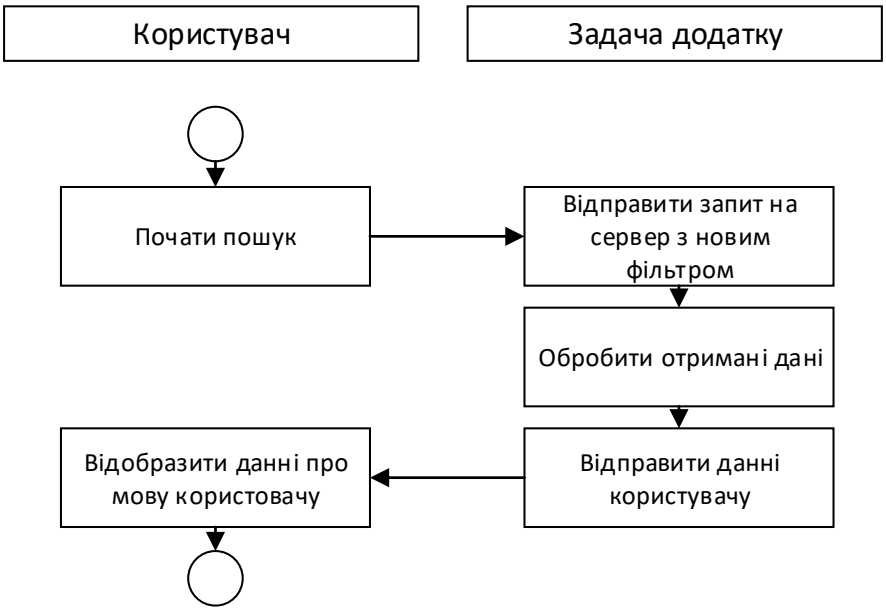

Рисунок 2.5 – Діаграма фільтрації мов

Найважливішим у ігровому додатку є сам процес гри. Користувач може обирати складність гри. Залежно від його поточних знань (див. рис. 2.6).

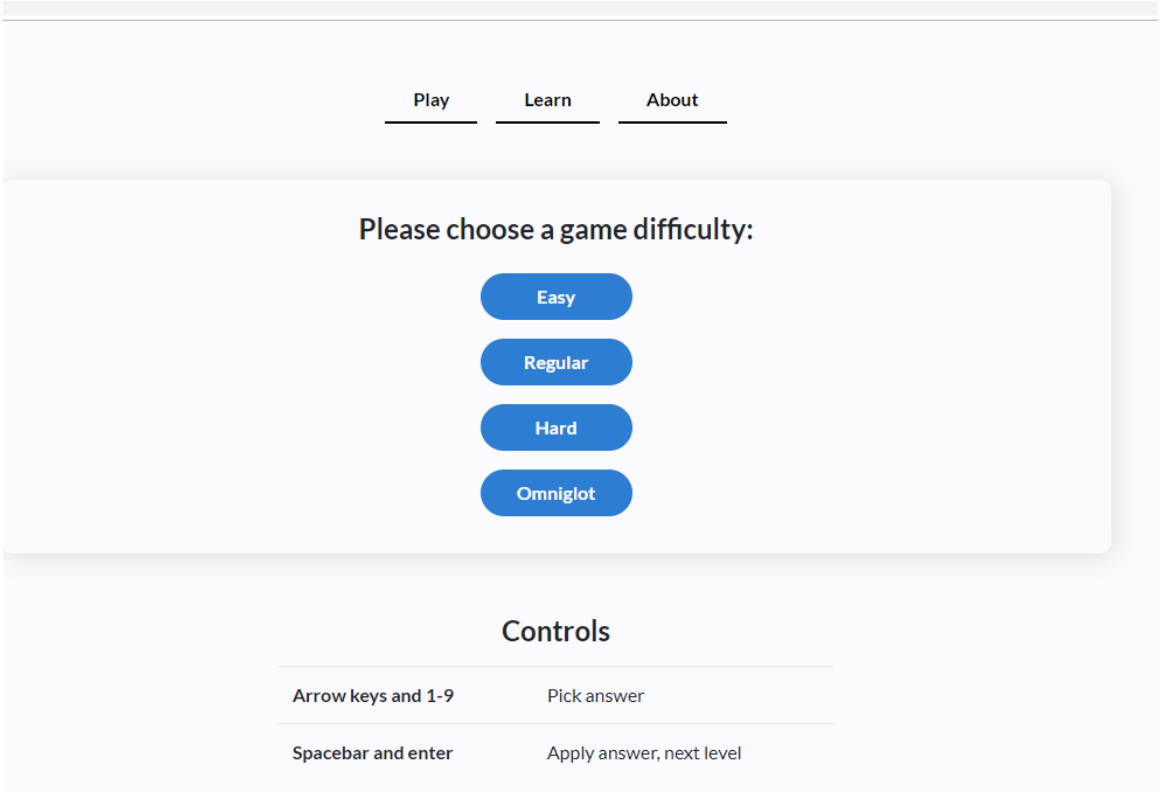

Рисунок 2.6 – Мокап складності гри

Для кожної складності підбираються окремі завдання. Адміністратор при додаванні даних про мову повинен вказати данні про її складність сприйняття на слух від 1 до 9 (див. рис. 2.7).

name	family	region	branch	subbranch	grp	subgroup	subsubgroup	variety	rank
Bengali	Indo-European	ME/Asia	Indo-Iranian	Indo-Aryan	Eastern Indo-Aryan	Bengali-Assamese			3
Maithili	Indo-European	ME/Asia	Indo-Iranian	Indo-Aryan	Eastern Indo-Aryan	Bihari			7
Russian	Indo-European	E Europe	Balto-Slavic	Slavic	East Slavic				1
French	Indo-European	S/W Europe	Italic (Romance)	Western Romance	Gallo-Romance	Oïl		European French	1
French	Indo-European	S/W Europe	Italic (Romance)	Western Romance	Gallo-Romance	Oïl		Quebec French	1
Catalan	Indo-European	S/W Europe	Italic (Romance)	Western Romance	Gallo-Romance	Occitano-Romance			5
Ukrainian	Indo-European	E Europe	Balto-Slavic	Slavic	East Slavic				5
Belarusian	Indo-European	E Europe	Balto-Slavic	Slavic	East Slavic				8
Portuguese	Indo-European	S/W Europe	Italic (Romance)	Western Romance	Ibero-Romance	Portuguese-Galician		Brazilian Portuguese	2
Portuguese	Indo-European	S/W Europe	Italic (Romance)	Western Romance	Ibero-Romance	Portuguese-Galician		European Portugues	2

Рисунок 2.7 – Таблиця усіх мов в додатку.

Переходячи безпосередньо до геймплею можна побачити що UX дуже простий і зним може розібратися навіть дитина.

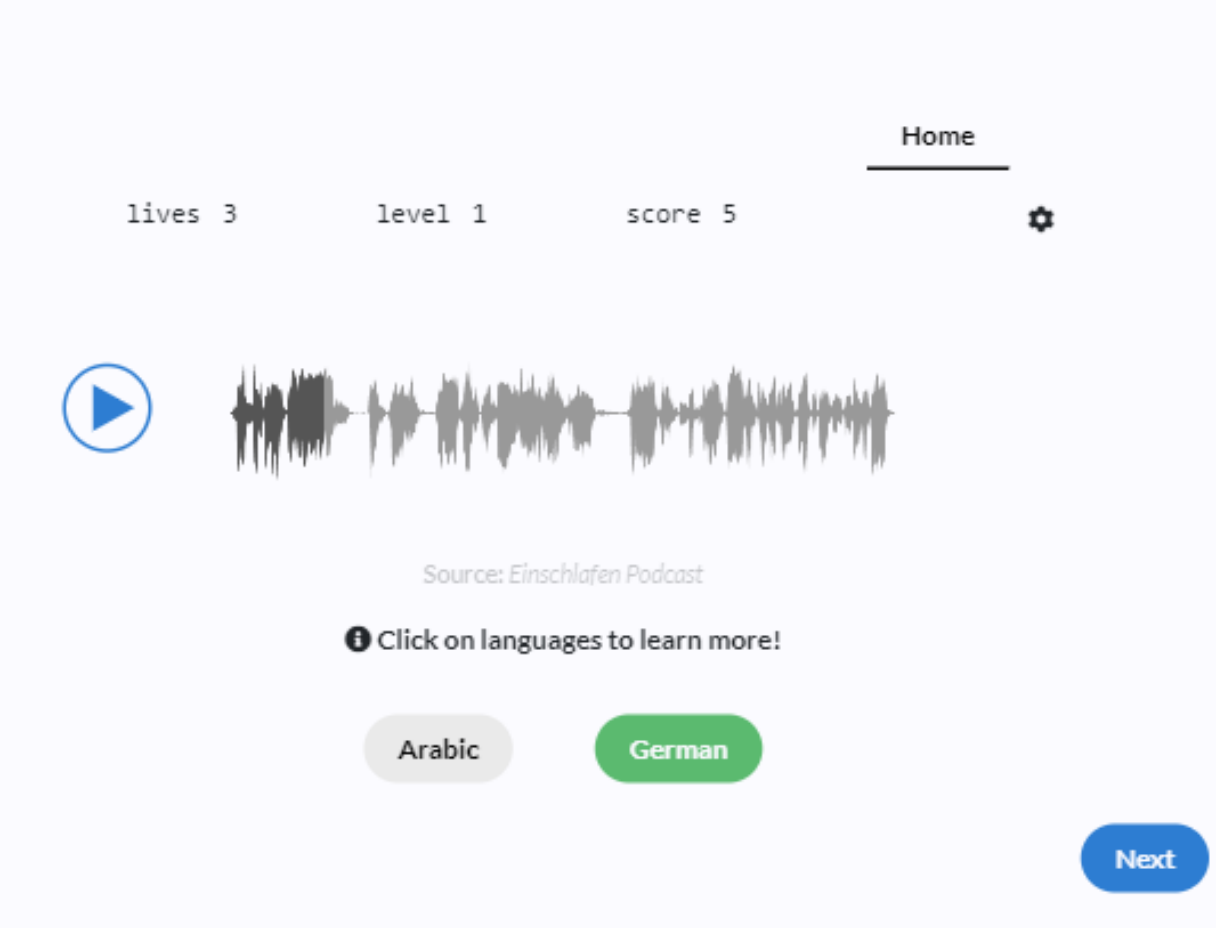

Рисунок 2.8 – Основний функціонал гри Single Player.

Описання основних функцій гри для режиму single-player(див. рис. 2.8):

а) Верхня панель основних характеристик гри;

б) Кнопка home вихід в основне меню;

в) Кнопка плей та паузи для відтворення аудіо файлу;

г) Аудіо доріжка з можливістю зміни часу аудіо файлу;

д) Можливі відповіді;

е) Перехід до іншого питання.

Також є можливість побачити інформацію про мову не виходячи з гри. Для цього необхідно два рази натиснути на відповідь (див. рис. 2.9).

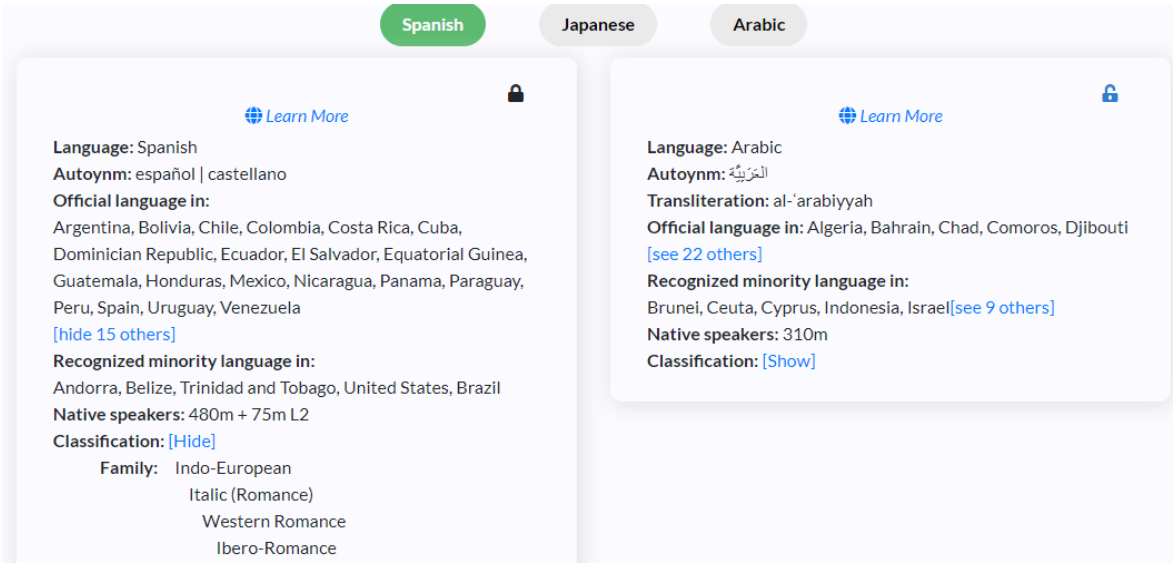

Рисунок 2.9 – Данні про мову у грі.

Основною відмінністю multiplayer від single-player є (див. рис. 2.10):

а) можливість вибору кількості гравців від 2 до 4;

б) можливість надання гравцям ім’я;

в) режим Score в якому гравці намагаються першими здобути необхідну кількість очок;

г) режим Lives де гравці мають обмежену кількість помилок, якщо гравець помилиться більша разів ніж дозволено гра для нього припиняється.

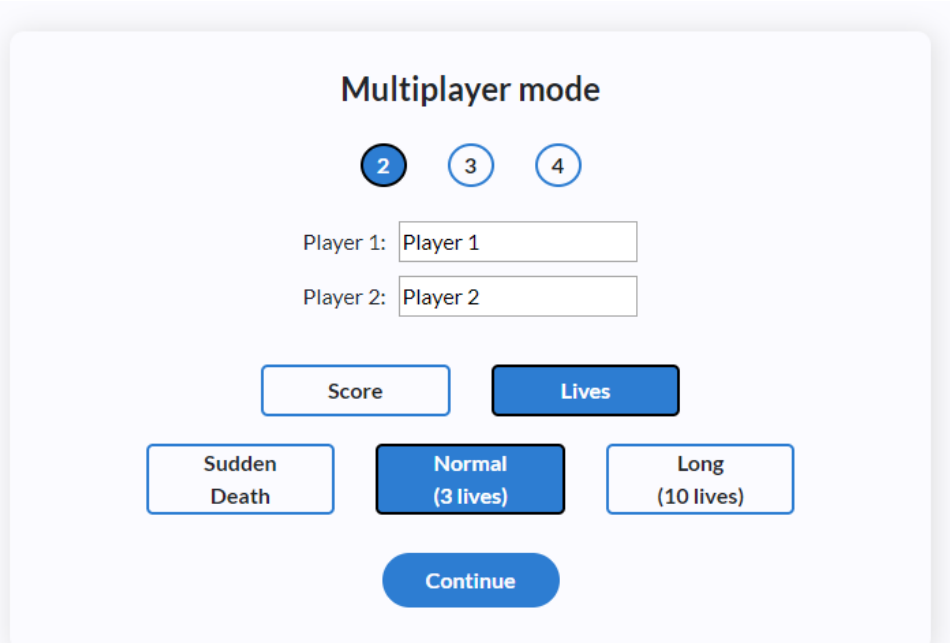

Рисунок 2.10 – Мокав вибору режиму Multiplayer.

2.2 Архітектура програмного продукту

2.2.1 Планування архітектури бізнес-логіки додатку

Архітектуру програми можна і потрібно розробляти виходячи з цілей і поставлених завдань для цього додатку.

Цей додаток можна розділити на дві частини сама гра і адмінських частина. Відштовхуючись від функціоналу цих частин нам потрібно розробляти нашу архітектуру.

С першу нам потрібно написати сервер який має виконувати такі функції:

а) обробляти запити;

б) обробка помилок;

в) зчитування та запис даних у базу даних;

г) фільтрування даних з бази даних.

Обробляти запити ми будемо через сервер роутеру, завдяки ньому ми зможемо розширювати кількість різноманітних запитів на сервер без страху за збільшення коду.

Обробляти помилки ми будемо завдяки хендлерам, вони допоможуть нам відпрацьовувати усі помилки як зі сторони клієнта так зі сторони бази даних без страху зупинки додатку.

Для безпечного і простого використання бази даних потрібно використовувати моделі даних. Це набір об'єктів, поділених за типом. Кожен тип об'єктів повинен мати однакові поля, це значно полегшить роботу з даними та допоможе уникнути помилок.

Також потрібно розробити глобальні функції для зчитування та запису даних до бази даних. Це полегшить розробку та зменшить код.

Для фільтрації даних ми будемо використовувати окремі методи які будуть використовуватися не тільки для простих маніпуляцій з базою даних, а ще ї для режимів складності у грі.

Потрібно зауважити що описаний вище функціонал для серверу, підходить як для адмінської частини так і для гри.

З даних функцій які потрібні для нашого сервера, ми можемо зробити висновок що сервер потрібен лише як зв'язок бази даних і клієнта, тому потрібність в шаблонізаторах чи в роутінгу завдяки серверу в нас нема а завдання роутінгу додатку ми перекладемо на React (див. рис. 2.11).

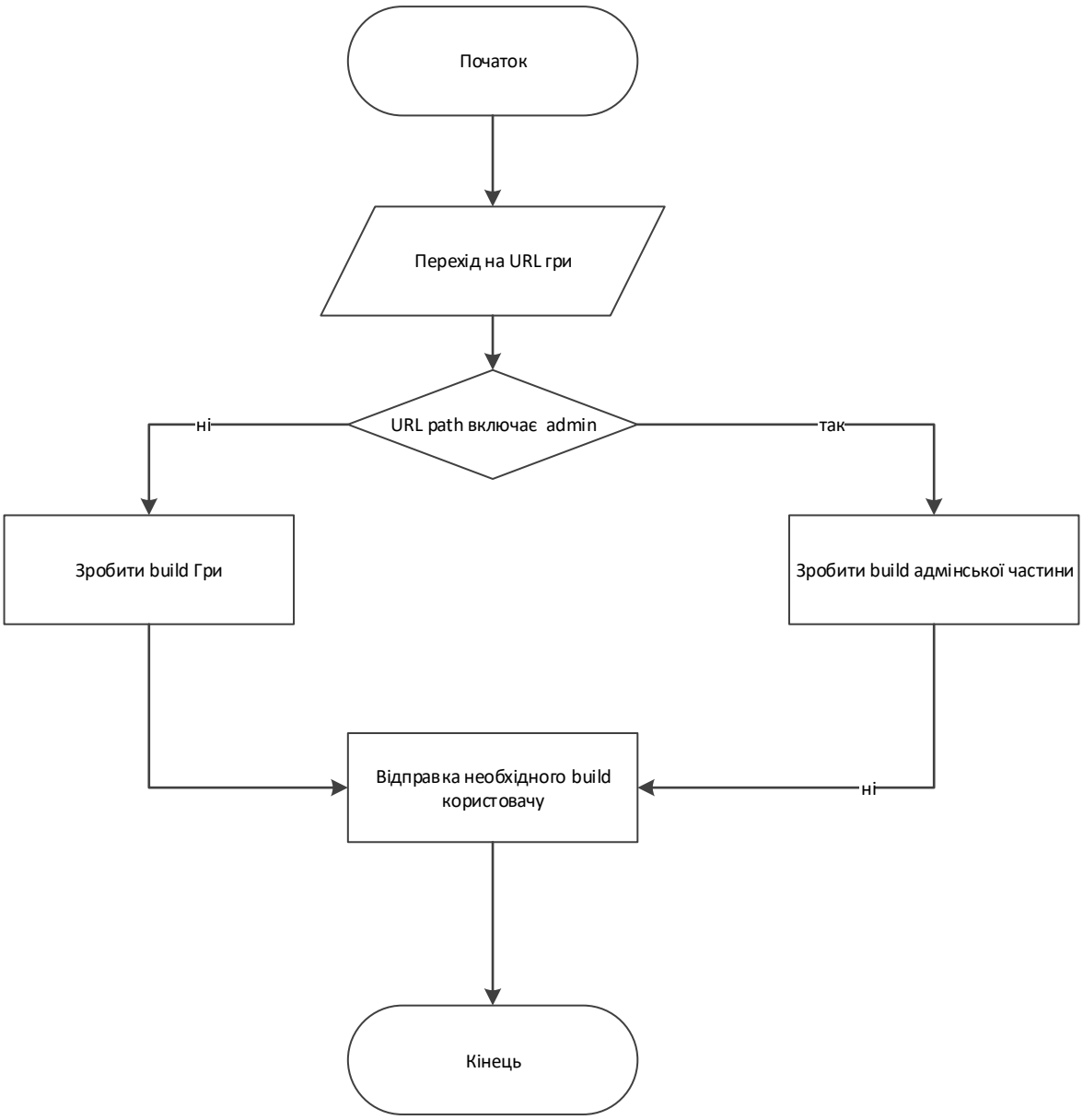

Рисунок 2.11 – Блок-схема алгоритму входу в гру

На наступній блок-схемі (див. рис. 2.12) зображено алгоритм дій у разі зміни фільтрації мов.

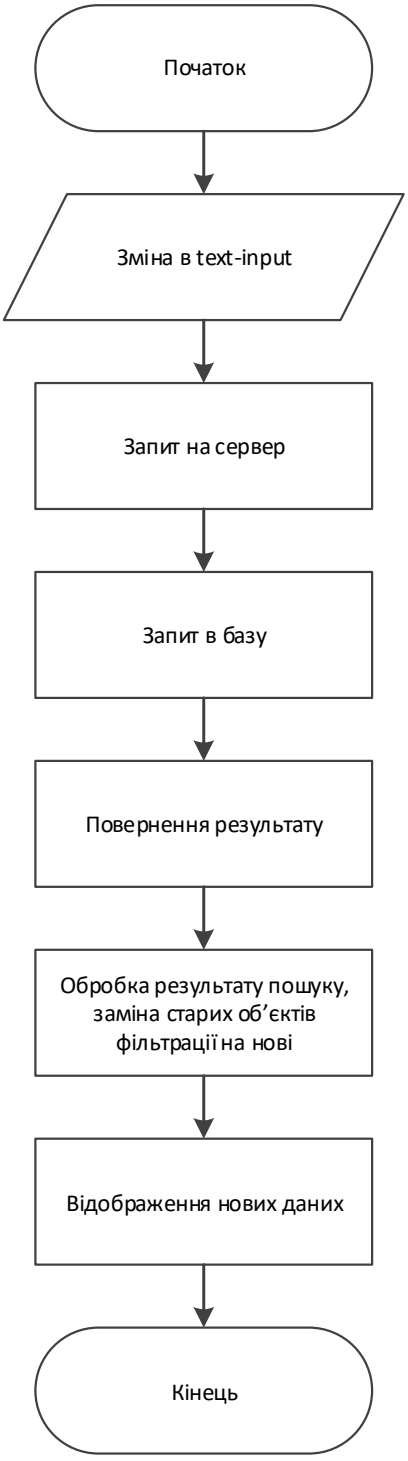

Рисунок 2.12 – Блок-схема алгоритму обробки подій при зміні фільтра мов

На третій блок-схемі (див. рис. 2.13) зображено алгоритм гри в одиночному режимі.

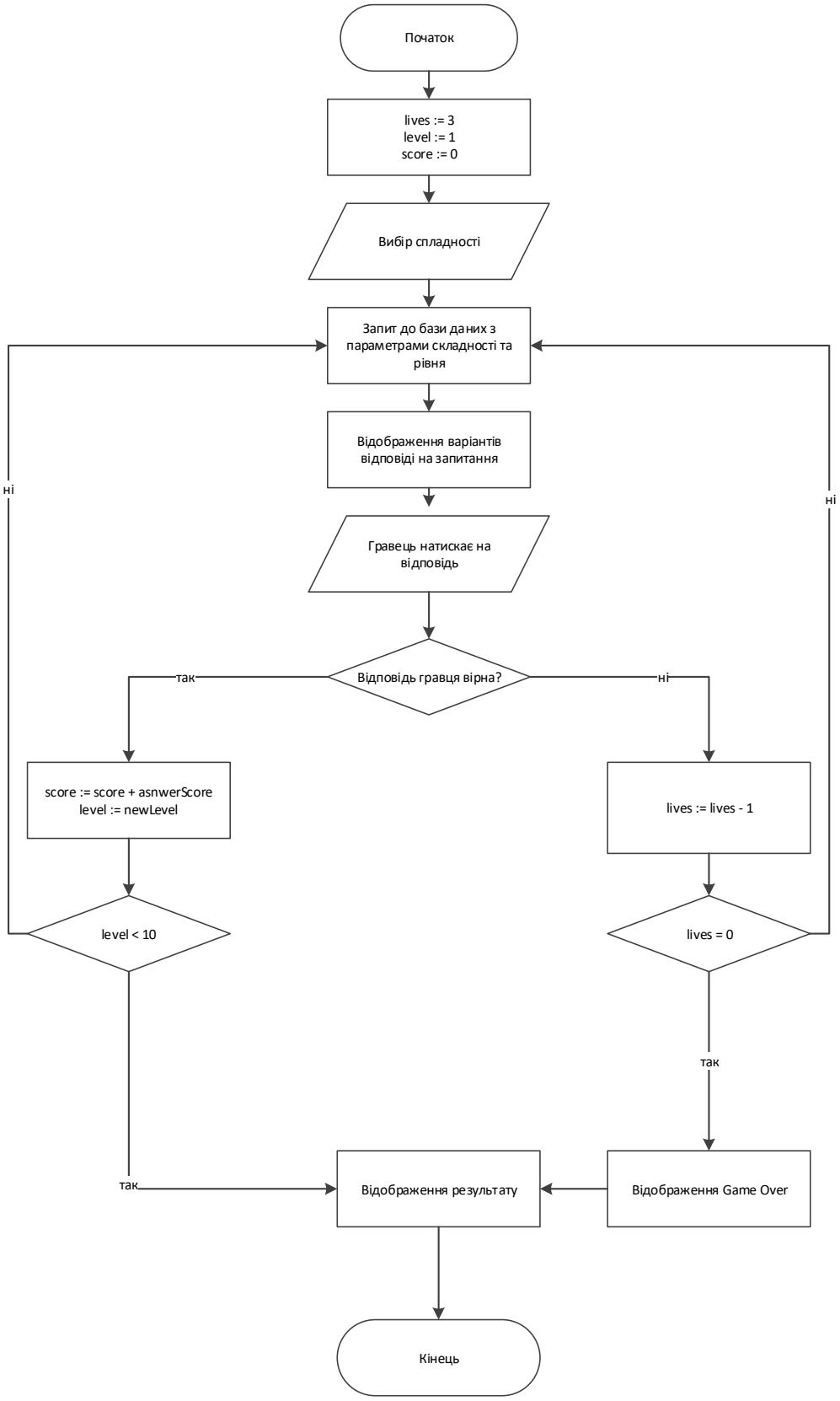

Рисунок 2.13 – Блок-схема алгоритму гри в single-player режимі

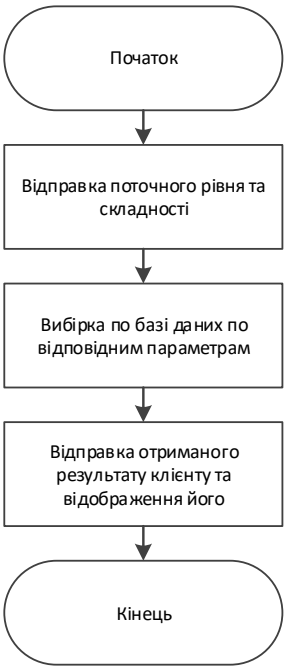

Рисунок 2.14 – Блок-схема алгоритму синхронізації даних з сервером

На схемі (див. рис. 2.14) зображено алгоритм вибірки запитання.

2.2.2 Планування архітектури візуальної частини додатку

Важливим в написанні будь якого додатку є не тільки бізнес логіка але і візуальна частина, особливо для гри. Так як ми вже маємо готові макети додатку гарним тоном буде роздроблення всі візуальні частини на під компоненти а також написання одного компонента для однакових елементів який ми зможемо використовувати у всьому додатку.

Так як ми вирішили що використовувати React і Go сервер нам потрібен лише як зв'язок з БД. Тому для написання логіки переходу між сторінками та адмін панеллю нам потрібен маршрутизатор. У React нема встроенного маршрутизатора це зробленно для зменшення розміру бібліотеки але є популярне рішення бібліотека React Router але для задач гри нам не потрібен увесь її функціонал. Тому було прийнято рішення написати свій кастомний маршрутизатор який має бути «обгорнутим» Unstated сховищем для контролю стану маршрутизатора.

Щоб декомпозувати код для спрощення розробки додатку потрібно реалізовувати компоненти, які будуть виконувати певний ряд дій. Компоненти можуть буди як обгортки для взаємодії бізнес-логіки додатку так і виконувати роль звичайних візуальних компонентів наприклад кнопка або модальні вікна.

Також ми будемо розмежування компоненти на під компоненти. Кожен компонент потрібен мати свій файл стилю. Це значно прискорить розробку полегшить дебаг а також попередить можливість помилок які можуть бути при конфлікті стилів. Замість звичайних CSS файлів ми використовуємо SCSS (див. рис. 2.15).

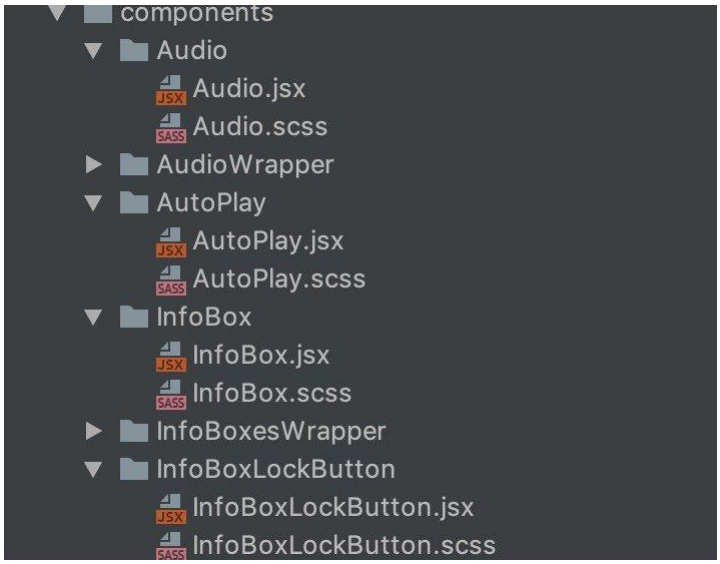

Рисунок 2.15 – Візуалізація структури компонентів додатку

Після дослідження було виявлено що немає візуальних тем зі схожим дизайном. Для швидкої розробки візуальної частини було вирішено використовути сітку Bootstrap [8], яка э дуже простим та ефективним рішенням для адаптивного сайту.

Додаток потрібен бути повністю адаптований до мобільних пристроїв, використання сітку Bootstrap та компонентів дуже полегшить завдання.

Візуальне представлення режиму single-player для мобільного пристрою (див. рис. 2.16).

Рисунок 2.16 – Режим signle-player на мобільному пристрої.

Режим multiplayer має інакший вигляд. Кожен гравець має свою особисту кількість життів та кількість очок. Перемагає той гравець який набрав найбільшу кількість очок.

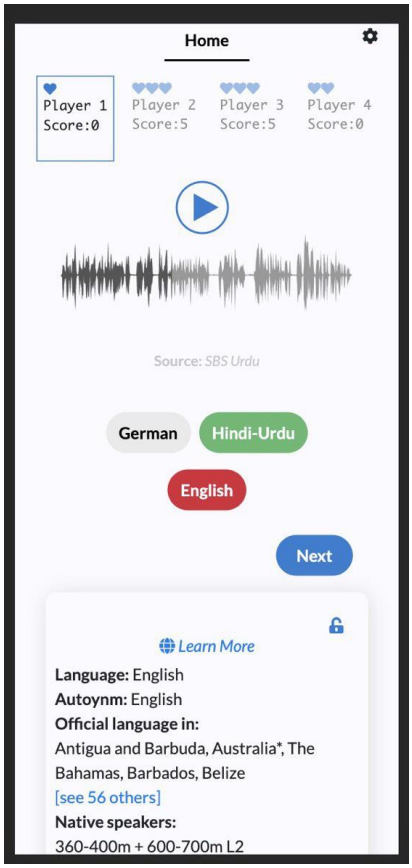

Рисунок 2.17 – Режим multiplayer на мобільному пристрої.

Загальний цикл роботи візуалізації компонентів (див. рис. 2.18). Це спрощена схема роботи React та її можна застосувати до будь-якого додатку.

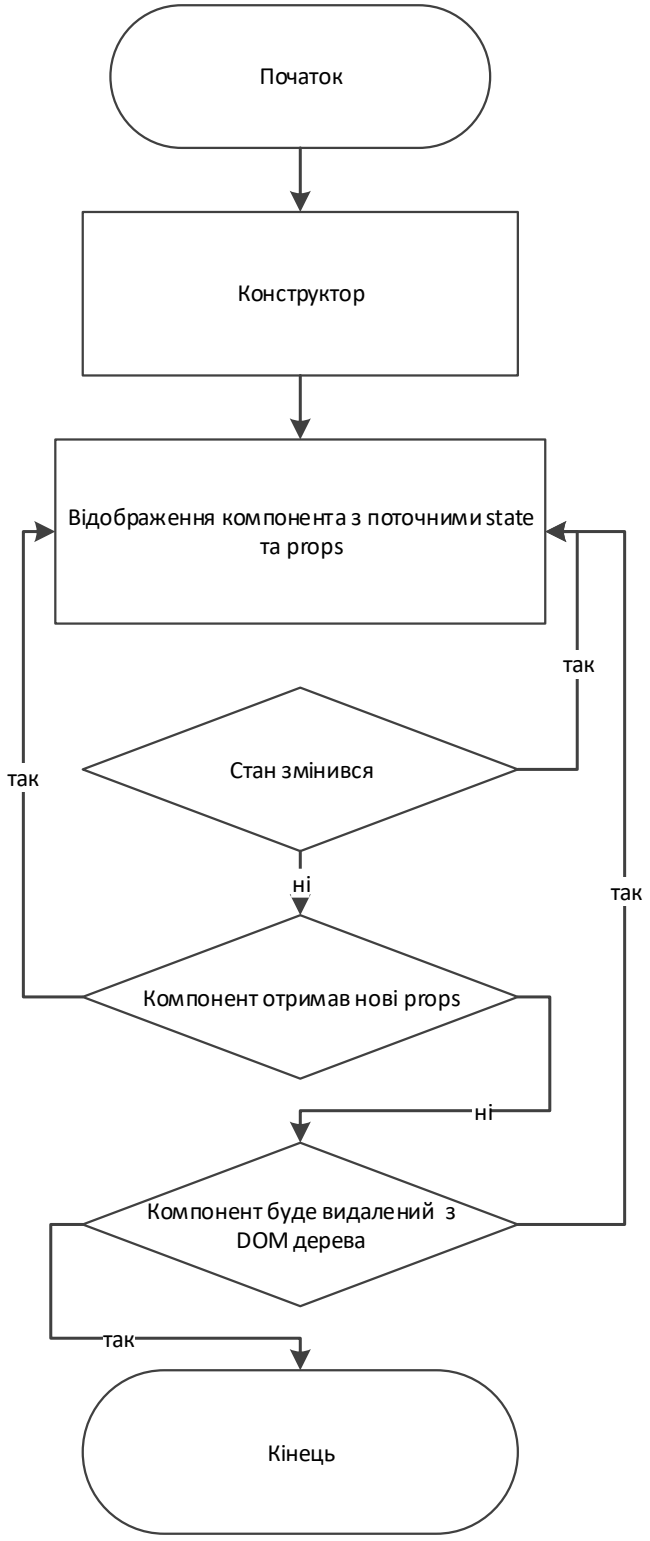

Рисунок 2.18 – Блок-схема алгоритму циклу візуалізації компонентів

3 РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Реалізація компонентів додатку

Додаток можна поділити на дві частини реалізацію серверу Go та React build.

```
package server
 // Set CORS config.
 corsConfig := cors.DefaultConfig()
 corsConfig.AddAllowHeaders("Authorization")
 corsConfig.AllowMethods = []string{"GET", "POST", "OPTIONS"}
 corsConfig.AddExposeHeaders("Authorization")
   corsConfig.AllowOrigins = []string{
    "https://www.lingyourlanguage.com",
    "https://lingyourlanguage.com",
   }
 // Validate CORS corsConfig and use.
 err := corsConfig.Validate()
 if err != nil {
  panic(err)
 }
```

Рисунок 3.1 – Задання Cors

На рисунку 3.1 зображені конфігурація cors. Cross-Origin Resource Sharing (CORS) - механізм, який використовує додаткові HTTP-заголовки, щоб дати можливість агенту користувача отримувати дозволи на доступ до обраних ресурсів з сервера на джерелі (домені), відмінному від того, що сайт використовує в даний момент. Кажуть, що агент користувача робить запит з

іншого джерела (cross-origin HTTP request), якщо джерело поточного документа відрізняється від запитуваного ресурсу доменом, протоколом або портом [9]. Цей фрагмент коду вказує на те що робити запити можна буде лише з домену lingyourlanguage [10].

```
package server
// setupHandler sets up the HTTP handler and returns it. All routes are defined here.
func setupHandler(h interfaces.Handler, cfg *config.Config, useTLS bool) http.Handler {
 root := router.Group("")
 root.GET("/q", h.GetQuestion)
 root.GET("/a/:checksum", h.GetSound)
 root.GET("/languages", h.GetLearnPageLanguages)
 root.GET("/languages/:prop_id", h.GetLanguageData)
 root.GET("/articles", h.GetArticles)
 root.GET("/articles/:id", h.GetArticleURL)

 admin := root.Group("/admin")
 admin.GET("/login", h.Login)

 session := admin.Group("/session").Use(h.AuthMiddleware)
 session.GET("/view/:table", h.SessionView)
 session.POST("/update/:table/:id/:column", h.SessionUpdate)
 session.POST("/create/:table", h.SessionCreate)
 session.POST("/delete/:table/:id", h.SessionDelete)
 session.POST("/upload", h.SessionUpload)
 return router
}
```

Рисунок 3.2 – Роутинг додатку

Як вже було сказанно в другому розділі сервер Go буде мати лише роутер для запитів (див. рис. 3.2).

Розглянемо основні роути:

а) «root.GET("/q", h.GetQuestion)» – Get запит який повертає запитання для користувача;

б) «root.GET("/a/:checksum", h.GetSound)» – Get запит який повертає аудіо файл для користувача;

в) «root.GET("/languages", h.GetLearnPageLanguages)»– Get запит який повертає список мов;

г) «session.GET("/view/:table", h.SessionView)» – Get запит який повертає таблицю для адмінського додатку, треба зауважити що цей запит може відправляти тільки адміністратор;

д) «session.POST("/create/:table", h.SessionCreate)» – Post запит для створення нової таблиці для адмінського додатку.

На наступному рисунку 3.3 зображено основні функції для взаємодії з грою.

```
…
 func (g *Game) newQuestion(points, difficulty int) (*models.Question, error) {…}
 func (g *Game) makeQuestion(level *models.Level, difficulty int) {…}
 func randomAnswerCount(level *models.Level) int {…}
 func calculatePointsForQuestion(level *models.Level, rankOfCorrectAnswer int,
  answerCount int, relationalWeight float64) int {…}
…
```

Рисунок 3.3 – Код реалізація методів гри

Розглянемо кожну функцію гри окремо:

а) «newQuestion» – функція, яка створює правильний варіант відповіді;

б) «makeQuestion» – функція, яка бере з бази даних за відповідними параметрами аудіо файл;

в) «randomAnswerCount» – функція яка створює лист з відповідями розмір цього листа задаєтеся в адміньскій таблиці де є мінімальна кількість та максимальна кількість відповідей ;

г) «calculatePointsForQuestion» – функція яка в залежності від рівня, складності, та кількості запитань кількує кількість очок за правельну відповідь.

Тепер перейдем до візуаьності частини і почати потрібно з точки входу в додаток рисунку (див. рис. 3.4)

```
…
function App(props) {
  return (
    <AppMainContainer>
      <NavButtonsComponent />
      <PageComponent />
    </AppMainContainer>
  );
}
function AppMainContainer(props) {
  return (
    <div className="container-fluid mt-5 main-header justify-content-center">
      {props.children}
    </div>
  );
}



function NavButtonsComponent(props) {
  // Do not render top nav bar when home page is shown.
```

```
  return (
    <Subscribe to={[AppContainer, GameContainer]}>
      {(app, game) => ( <NavButtons app={app.state} game={game.state} /> )}
    </Subscribe>
  );
}
function PageComponent(props) {
  return (
    <Subscribe to={[AppContainer]}>
      {app => {
        let mainView;
        switch (app.state.page) {
        case Pages.HOME:
          mainView = <Home switchPage={app.switchPage} />;
          break;
        case Pages.ABOUT:
          mainView = <About switchPage={app.switchPage} />;
          break;
        case Pages.PLAY: // Valid state but handled differently below.
          break;
        case Pages.LEARN:
          mainView = <Learn switchPage={app.switchPage} />;
          break;
        default:
          return;
        }
...
}
export default App;
```

Рисунок 3.4 – Точка входу в додаток

Функція App є головною точкою входу в додаток, як можна побачити вона повертає JSX [2]. Функція повертає дерево функціональних компонентів.

AppMainContainer – так званий компонент обгортка який приймає до себе інші об’єкти JSX [2] в якості children. Всі props обгорнуті в div який включає в себе Bootstrap класи за для того щоб усі елементи які були передані в якості children були в центрі екрану [8].

NavButtonsComponent – функціональний компонент який відображає навігацію в додатку. Він підписаний на зміну глобального стану додатку завдяки Subscribe [6]. AppContainer, GameContainer глобальні стани додатку, передають свої стани NavButtons, якщо AppContainer, GameContainer будуть змінено NavButtons отримає нові данні.

PageComponent – функціональний компонент який замінює React Router, він також обгорнутий Subscribe [6].

На наступному рисунку 3.5 зображено основні функції AppContainer.

```
…
type AppState = {
  page: Page,
  played: boolean,
  learnLanguage?: string,
  readArticle?: string,
  searchParams?: SearchParams
}

// AppContainer contains main states of application.
export class AppContainer extends Container<AppState> {
  constructor(searchParams: SearchParams) {
    super();
    this.state = {
      page: Pages.HOME,
```

```
    played: false,
    learnLanguage: undefined,
    readArticle: undefined,
    loaded: false
  }
  this._loadAllArticles(searchParams)
 }

 switchPage = (page: Page, learnLanguage?: string) => {…}
 goToHomePage = () => {…}
 loadAllArticles = async (searchParams: any) => {…}
…
```

Рисунок 3.5 – Код класу AppContainer.

Для типізації використаєтеся typescript, AppState задає строгу типізацію стану для класу AppContainer [12].

Розглянемо кожну функцію синхронізації окремо:

а) «switchPage» – функція для зміни теперішньої сторінки, якщо викликати цю функцію додаток автоматично відкриє нову сторінку;

б) «goToHomePage» – відкриває початкову сторінку додатку;

в) «loadAllArticles» – завантажити усі статті для відображення.

На наступному рисунку 3.6 зображено основні функції класу GameContainer.

```
…
export class GameContainer extends Container<GameState> {
  state = {
    maxLevel: 10
    difficulty: Difficulty.NotChosen,
    gameMode: GameMode.NotChosen,
```

```
    lives: 3,
    level: 1,
    score: 0,
    gameOver: false,
    question: {},
    settingsVisibility: false,
    multipleMode: initMultipleType,
    masOfPlayerIndex: [],
    volume: 100,
    autoPlay: true,
    theHighestScore: 0,
    arrayOfPlayer: initPlayers(2)};
  initPlayers(count, arrayOfPlayer) {…}
  setupKeyboardControls = () => {…}
  setCountOfPlayer = (number) => {…}
  setPlayerName = (index, name) => {…}
  setMultipleType = (value) => {…}
  setNewRandomOrderToUser = () => {…}
  changeVolume = (value) => {…}
  changeSettingsVisibility = () => {…}
  setDifficulty = (difficulty: DifficultyLevel) => {…}
  setGameMode = (gameMode) => () => {…}
  gameOver = (): boolean => {…}
  setUserAnswer = (index, value) => {…}
  nextQuestion = async () => {…}
  answerQuestion = (chosenAnswer: string) => {…}
  restart = () => {…}
}
```

Рисунок 3.6 – Код класу GameContainer

Початковий state має основні налаштування гри. Так як ми передаємо state цього класу як props для інших компонентів завдяки ustated, при виклику функцій цього класу ми буде змінювати властивості гри [6]. Розглянемо більш детально основні функції цього сценарію:

а) «initPlayers» – ініціація початкових імен та властивостей гравців в режимі мультіплей;

б) «setupKeyboardControls» – регіструє усі натиснення кнопок на клавіатурі;

в) «setCountOfPlayer» – змінює кількість гравців;

г) «setNewRandomOrderToUser» – якщо вибраний режим multiplayer то після запитання можуть змінюватися порядок відповідачів;

д) «restart» – перезапуск гри встановлення state на початкове значення;

е) «gameOver» – викликає сторінку game over а також підраховую кількість очок у гравців та виводить таблицю результатів для гравців де на першим місці буде гравець який набрав найбільше очок на останнім буде з найменшою, але це тільки для режиму multiplayer, також треба зазначити що кожне запитання може давати різну кількість очків, це залежить від складності запитання та кількості варіантів відповіді.

3.2 Тестування роботи додатка

Всі перераховані раніше вимоги до додатка були реалізовані. Додаток буду протестований не тільки мануально а ще й за допомогою сучасного підходу до тестування unit тести.

Для тестування роботи додатку було вирішено використовувати бібліотеки Jest та Enzyme.

Давайте почнемо з тестування простого компонента без стану (також відомого як дурний компонент), який відображає простий елемент посилання, що містить заголовок і URL.

```
import React from 'react';
import { string } from 'prop-types';
const Link = ({ title, url }) => <a href={url}>{title}</a>;
Link.propTypes = {
  title: string.isRequired,
  url: string.isRequired
};
export default Link;
```

Рисунок 3.7 – Простий компонент Link

На рисунку 3.7 зображено компонент Link, ми хочемо протестувати його, перевіривши, чи приходять властивості в компонент і чи правильно вони відображаються. За допомогою Jest у нас є дуже простий спосіб перевірки цього створення знімка (snapshots) [14]. У перший раз, коли запускається тест, створюється файл знімка. Після цього ви можете подивитися створений файл, щоб перевірити, чи відповідає компонент очікуваному результату.

```
import React from 'react';
import { shallow, configure } from 'enzyme';
import { shallowToJson } from 'enzyme-to-json';
import Link from './Link';
import Adapter from 'enzyme-adapter-react-16';
describe('Link', () => {
  it('should render correctly', () => {
    const output = shallow(
      <Link title="mockTitle" url="mockUrl" />);
    expect(shallowToJson(output)).toMatchSnapshot();
  });
});
```

Рисунок 3.8 – Код тестування компоненту Link

Як видно на рисунку 3.8, ми імпортуємо Link компонент до нашого файлу с тестом та перевіряємо чи зміг компонент відобразитись, результати тесту можна перевірити, усі створені знімки можна знати у папці snapshots (див. рис. 3.9) та результат цього тесту (див. рис. 3.10).

Рисунок 3.9 – Старенна папка snapshots

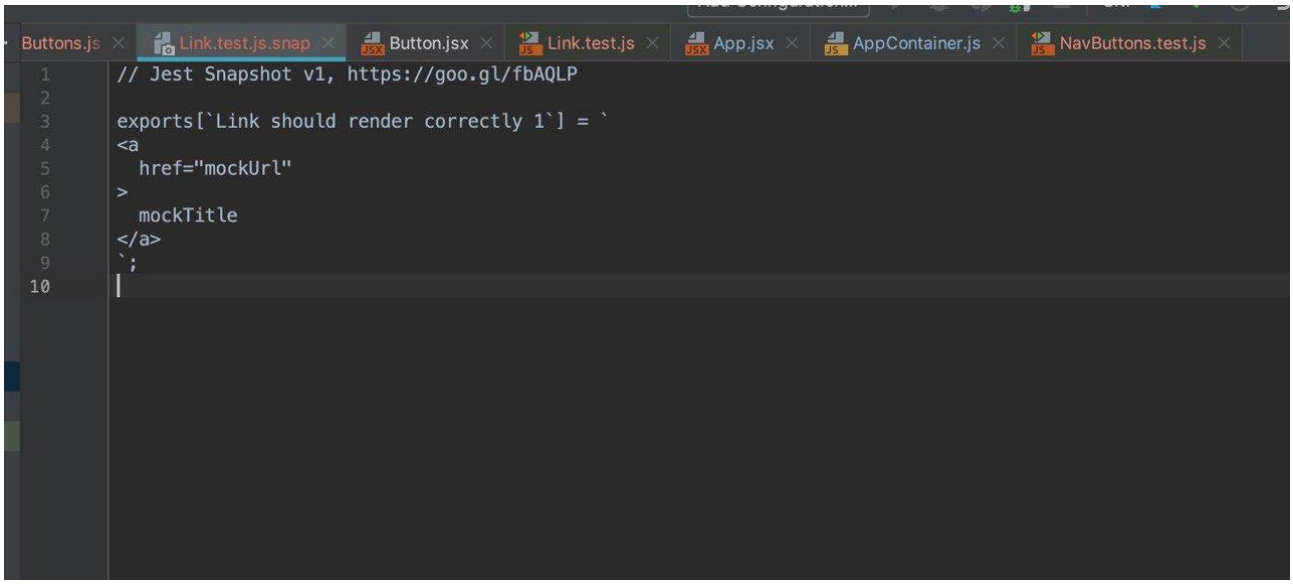

Рисунок 3.10 – Знімок Link тесту.

Далі розглянемо навігаційну панель додатку. Компонент навігаційної моделі має декілька завдань, по перше перехід на сторінки а також відображення тільки потрібних сторінок переходу. Для анімації навігаційний компонент має декілька класів для відображення та переходу у невидимий стан. Кожен з класів обгартує компонент залежачи від переданих параметрі props (див. рис. 3.11).

```
import React from 'react';
import { NavButton } from "../Buttons/Button";
```

```
…
export default class NavButtons extends React.Component {
  render() {
    let  classes = "disp-none";
    if (this.props.game.ready === true) {
      classes = "hide-nav-bar";
    }
    if (this.props.game.gameOver === true) {
      classes = "show-nav-bar";
    }
    if (this.props.game.difficulty === Difficulty.NotChosen) {
      classes = "";
    }
    if (this.props.app.page === Pages.HOME) {
      classes = "disp-none"
    }
    return (
      <NavContainer className={classes}>
        <NavButtonContainer page={Pages.PLAY} />
        <NavButtonContainer page={Pages.LEARN} />
        <NavButtonContainer page={Pages.ABOUT} />
      </NavContainer>
    );
  }}
…
```

Рисунок 3.11 – NavButtons класс

Для тестування потрібно використовувати Enzyme [11]. Перевіремо як змінюється результати тесту залежачи від переданних параметрів (див. рис. 3.12).

```
import React from 'react';
import { shallow, configure } from 'enzyme';
import { shallowToJson } from 'enzyme-to-json';
import NavButtons from './NavButtons';
import Adapter from 'enzyme-adapter-react-16';
configure({adapter: new Adapter()});

describe('NavButtons', () => {
  it('should render correctly', () => {
    const output = shallow(
      <NavButtons game={{ready: true}} app={{page: 'Home'}}/>
    );
    expect(shallowToJson(output)).toMatchSnapshot();
  });

  it('should find disp-none class if home page open', () => {
    const output = shallow(
      <NavButtons game={{ready: true}} app={{page: 'Home'}}/>
    );
    expect(output.hasClass('disp-none')).toBe(true);
  });
});
```

Рисунок 3.12 – Простий компонент Link

Як бачимо у першому тесті ми перевіряємо чи може компонент бути відображенний та створюймо знімок компонента. У другому тесті при передачі до props app об’єкт з ключем page та значенням Home цей тест буде відпрацьовувати коректна якщо після відображення він матиме клас 'disp-none'. Результат тесту (див. рис. 3.13), треба зауважити що при зміні значення ключа page на інше (див. рис. 3.14) Jest відправить нам помилку (див. рис. 3.15) бо клас

вже не буде дорівнювати 'disp-none', це і є основною ідею тестування, при написанні нового коду можуть виникати помилки та завдяки тестування ми не тільки зменшимо кількість помилок та ще й зможемо моментально їх знайти.

```
Test Suites: 9 passed, 9 total
Tests:       75 passed, 75 total
Snapshots:   44 written, 30 passed, 74 total
Time:        3.981s
Ran all test suites.
Vlads-MacBook-Pro:lyl-fe-new vlad$
```

Рисунок 3.13 – Результати тестування

```
it('should find disp-none class if home page open', () => {
  const output = shallow(
    <NavButtons game={{ready: true}} app={{page: 'About'}}/>
  );
  expect(output.hasClass('disp-none')).toBe(true);
});
```

Рисунок 3.14 – Зміна значення на не валідне

```
● NavButtons › should find disp-none class if home page open

  expect(received).toBe(expected) // Object.is equality

  Expected: true
  Received: false

    18 |       <NavButtons game={{ready: true}} app={{page: 'About'}}/>
    19 |     );
  > 20 |     expect(output.hasClass('disp-none')).toBe(true);
```

Рисунок 3.15 – Помилка яку відпрацював Jest

Далі перевіримо основний функціонал додатку. На сторінці learn ми можемо побачити фільтр по мовам (див. рис. 3.16) а також результат фільтрації (див. рис. 3.17).

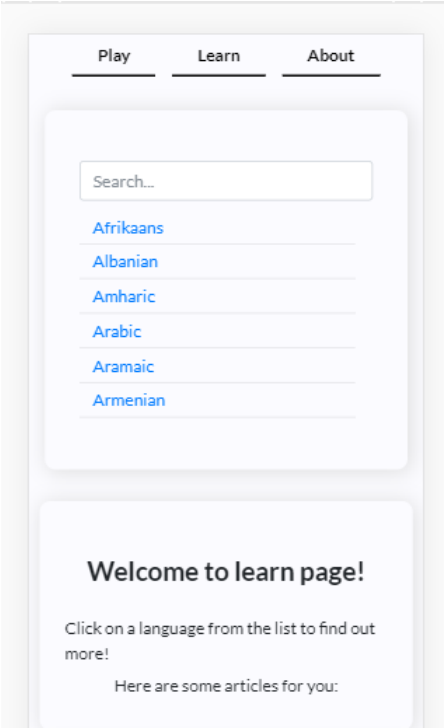

Рисунок 3.16 – Фільтр на сторінці learn

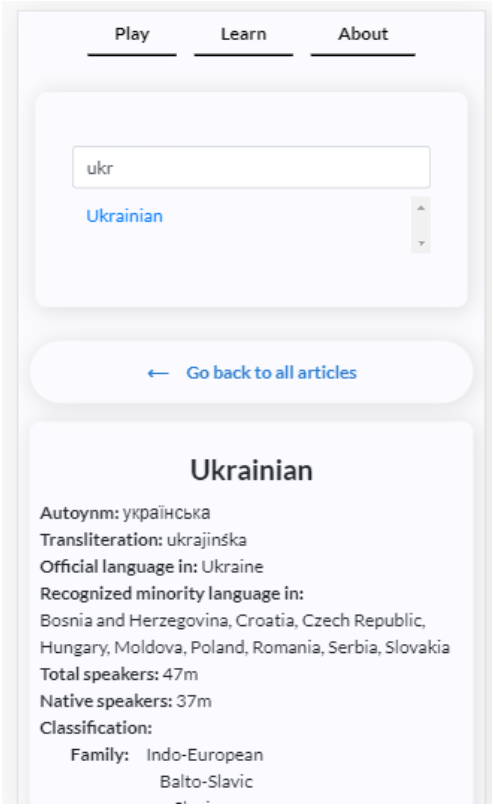

Рисунок 3.17 – Результат роботи фільтру

Вибір режиму гри (див. рис. 3.18) обравши режим single-payer потрібно вибрати режим складності (див. рис. 3.19), після вибору складності відкривається

гра у гравця є можливість зупиняти аудіо файл (див. рис. 3.20) та відчиняти вікно налаштування (див. рис. 3.21), якщо гравець відповів вірно ми бачимо що гравець отримає очки та варіант відповіді підкреслюєтеся зеленим (див. рис. 3.22), якщо гравець відповідає не вірно то в нього віднімається одне життя та варіант відповіді підкреслюєтеся червоним (див. рис. 3.23). Після того як гравець відповіси тричі не вірно він побачить свій кінцевий результат (див. рис. 3.24).

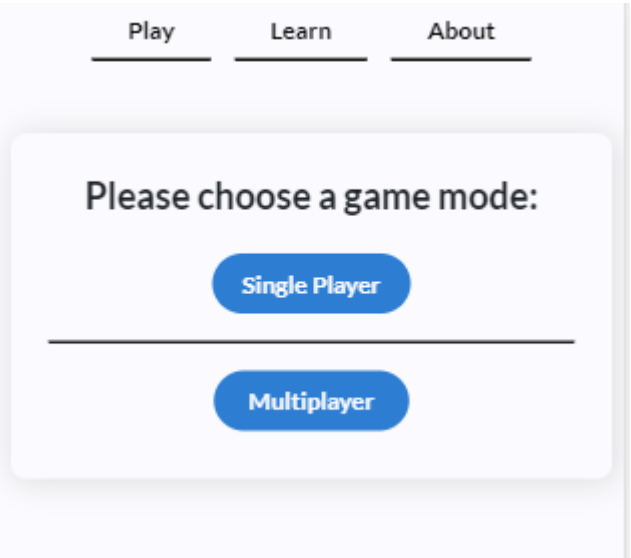

Рисунок 3.18 – Варіант гри

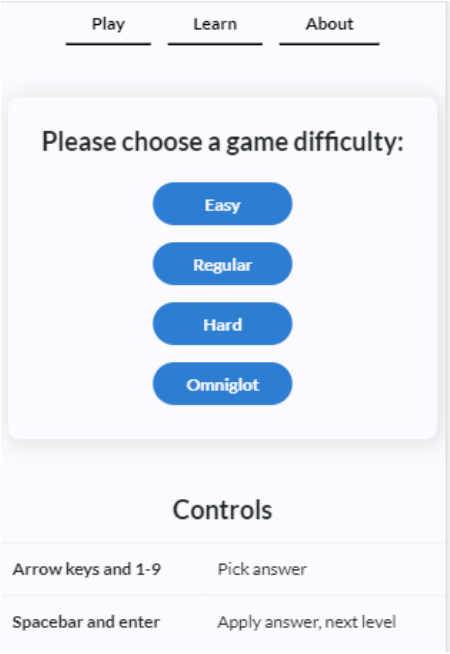

Рисунок 3.19 – Вибір режиму складності.

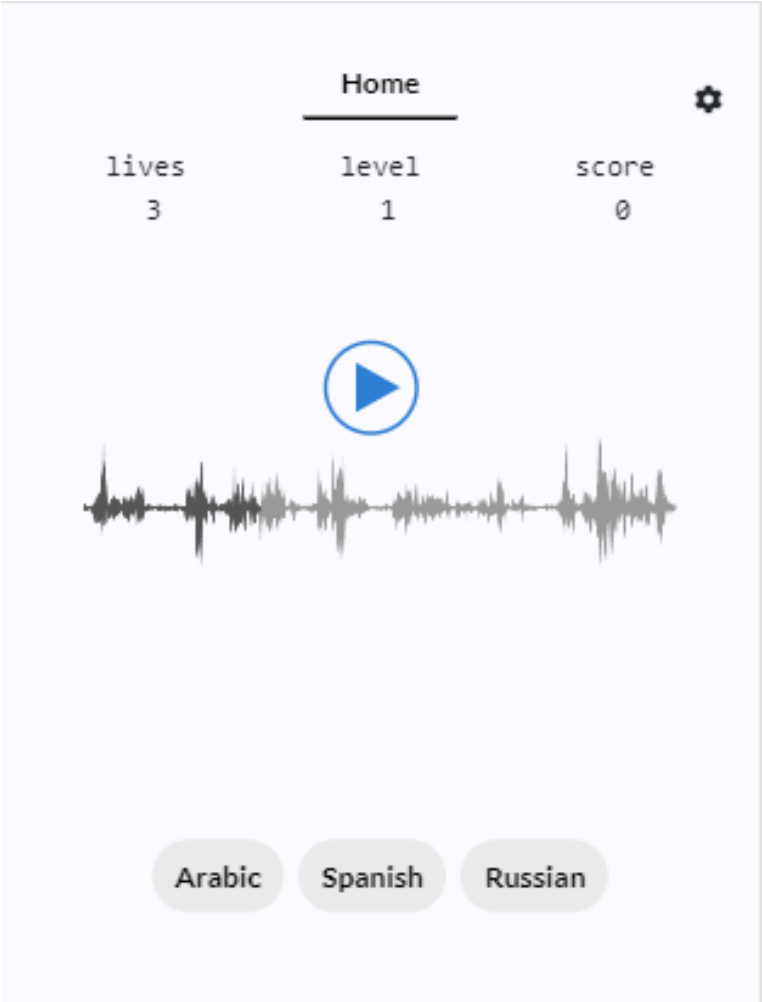

Рисунок 3.20 – Зупинка аудіо файлу.

Рисунок 3.21 – Модульне вікно налаштувань.

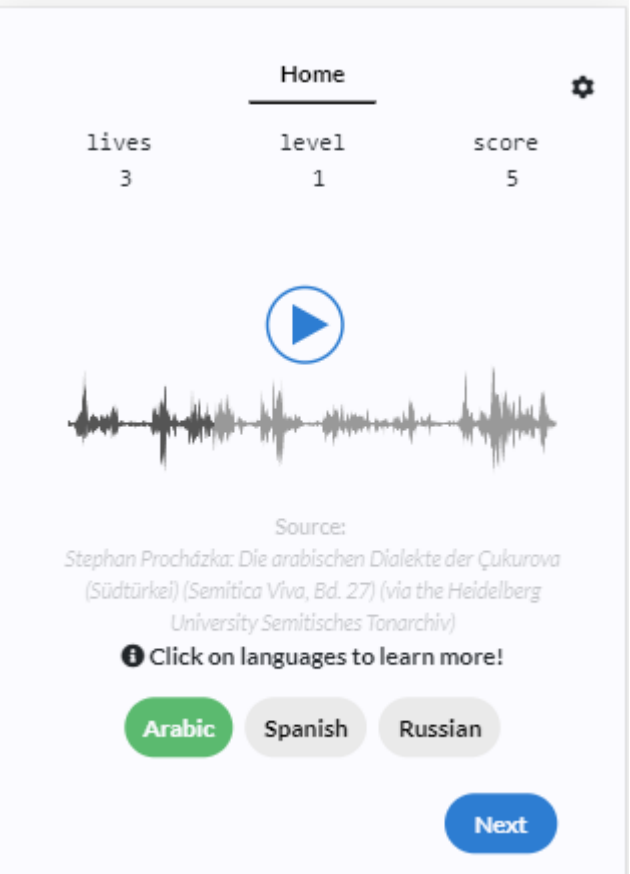

Рисунок 3.22 – Гравець обрав вірну відповідь.

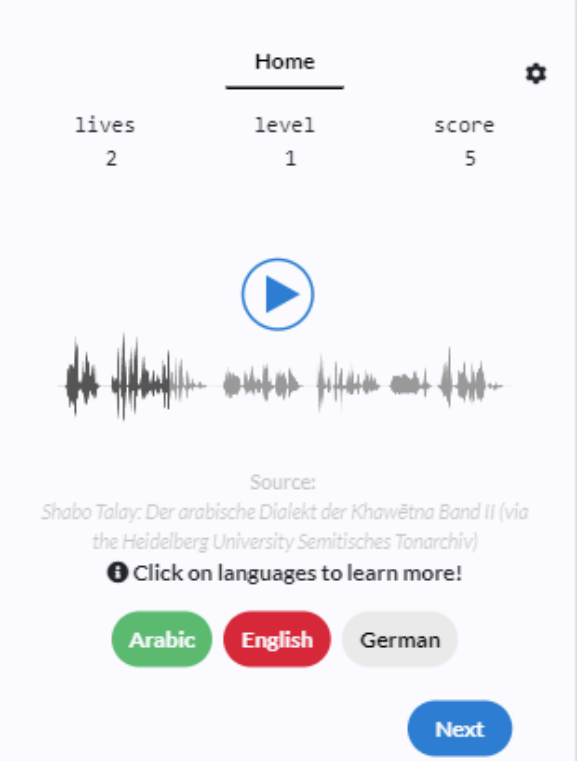

Рисунок 3.23 – Гравець обрав не вірну відповідь.

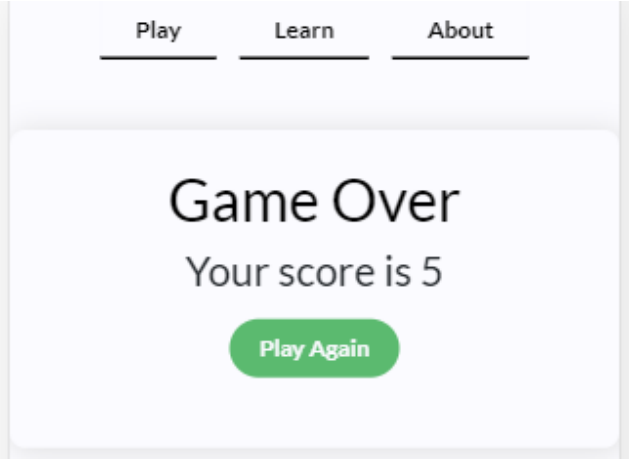

Рисунок 3.24 – Сторінка game over для режиму single-player.

Якщо гравець обирає режим multiplayer (див. рис. 3.18) він потрапить до режиму вибору типу де він зможе обрати кількість гравців, їх ім’я та тип гри (див. рис. 3.25).Обравши режим score переможе той гравець який перший здобуте певну кількість очок обравши режим live гра буде тривати доки в останнього гравця хоч одне життя. Після вибору режиму score ми можемо бачити що кожен гравець має нуль очок та певну кількість життів в залежності від обраного режиму (див. рис. 3.26). Після цього кожен гравець по черзі повинен відповідати на питання (див. рис. 3.27). Якщо в гравця закінчуються житті він більше не може відповідати (див. рис. 3.28). Коли всі гравці втратили всі життя ми побачимо кінцевий результат в таблиці (див. рис. 3.29).

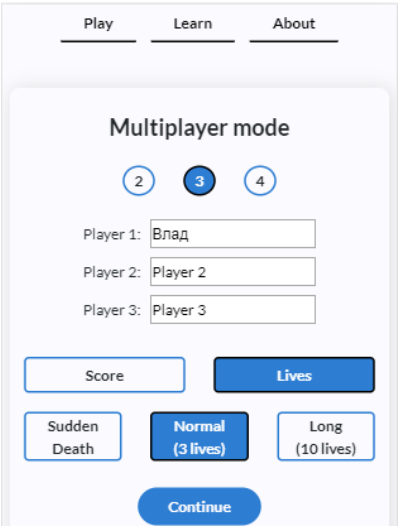

Рисунок 3.25 – Режим вибору гри multiplayer

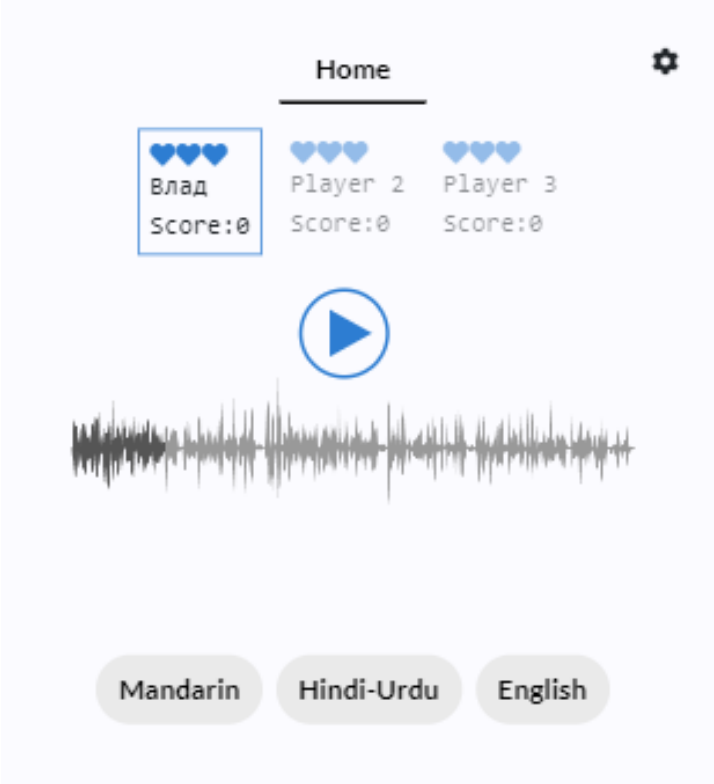

Рисунок 3.26 – Стартовий вигляд режиму score

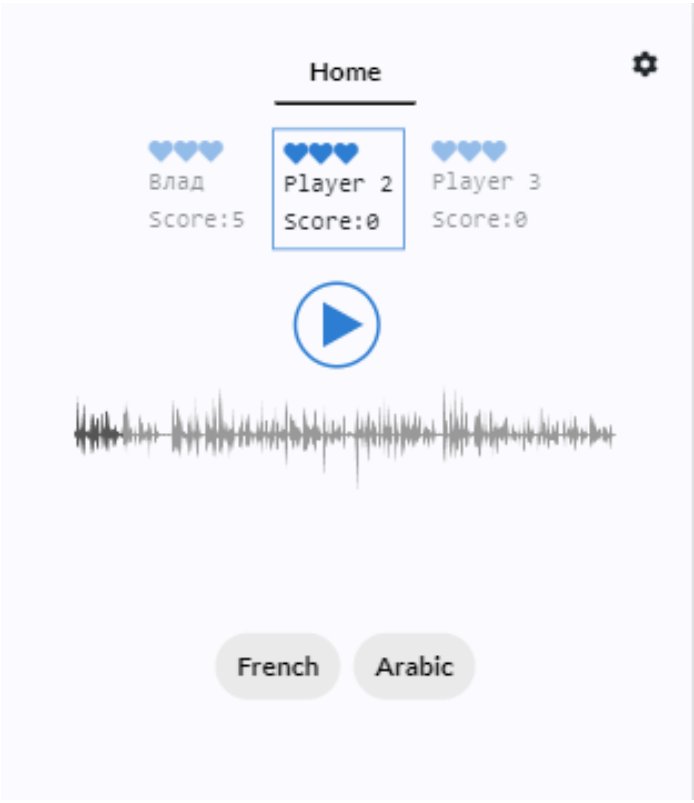

Рисунок 3.27 – Черга гравців

Рисунок 3.28 – Гравець який вибув з гри

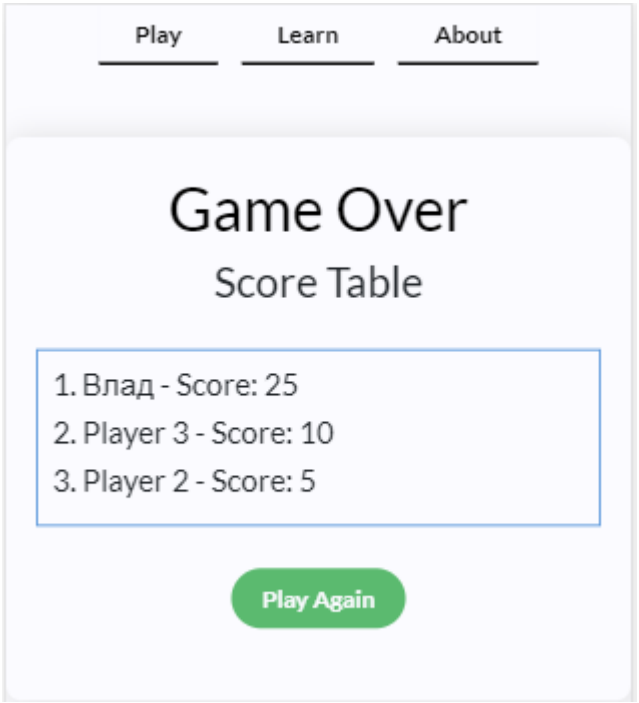

Рисунок 3.29 – Сторінка game over для режиму multiplayer.

ВИСНОВКИ

В ході виконання дипломної роботи на тему: «Використання технологій Go, stateless і React.js для розробки ігрового додатку» було отримано багато нових навичок та знань у області розробки веб додатків. Було вивчено як працює сервер Go як застосовувати React у поєднанні з stateless.

Загалом вирішено наступні задачі та проблеми:

а) розробка адаптивних додатків за допомогою React;

б) вивчення нюансів роботи React;

в) роздріб великих частин код на маленькі під компонент;

г) використання нових підходів для регулювання глобального state додатку;

д) планування роботи і функціоналу додатку, відштовхуючись від поставленого технічного завдання;

е) дослідження конкурентів та їх помилок для написання найкращого ігрового додатку.

Розроблений додаток має приємний інтерфейс, швидко та стабільно працює, може працювати з будь якого браузеру на комп’ютері чи на мобільному пристрою.

Основними його перевагами є те, що на відміну від конкурентів він є повністю безкоштовним та націлений на вивчення не однієї мови а одразу багатьох. Можливість додавати нові мови та аудіо файли які будуть створювати нові питання без зміни коду наддасть можливість збільшувати контент гри без втручання в код. Можливість вибору складності гри де мови будуть дуже відрізнятися на слух чи навпаки будуть окремими під діалектами однієї мови надає можливість вибрати будь-якому гравцю складність під себе. Також є великім плюсом можливість грати з іншими гравцями в режимі multiplayer.

ПЕРЕЛІК ПОСИЛАНЬ

1. Батчер Ф. Go на практиці. Москва: ДМК-прес, 2017. 376 с.

2. Бэнкс А. React і Redux.Функціональная веб-розробка. Петербург: Эксмо 2018. 336 с.

3. Мова Go : https://metanit.com/go/tutorial/1.1php (дата звернення: 15.10.2019).

4. Резіг Д. Секрети JavaScript ніндзя. Москва: Вільямс, 2016. 416 с.

5. Стефанов С. JavaScript. Шаблони. Москва: Вільямс, 2011. 272 с.

6. Управління станом React за допомогою Unstated. URL : https://webformyself.com/upravlenie-sostoyaniem-v-react-s-pomoshhyu-unstated/ (дата звернення: 12.11.2019).

7. Хавербеке М. Виразний JavaScript. Петербург: Эксмо, 2019. 480 с.

8. Bootstrap layout. URL : https://getbootstrap.com/docs/4.4/layout/overview/ (дата звернення: 13.10.2019).

9. Cross-Origin Resource Sharing (CORS) requests. URL : https://developer.mozilla.org/ru/docs/Web/HTTP (дата звернення: 13.10.2019).

10. HTTP request methods. URL : https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (дата звернення: 12.11.2019)

11. Introduction to Enzyme. URL : https://airbnb.io/enzyme (дата звернення: 19.11.2019).

12. Introduction to React and TypeScript. URL : https://www.typescriptlang.org/docs/handbook/react-&-webpack.html (дата звернення: 29.11.2019).

13. React Components and Props. URL : https://reactjs.org/docs/components-and-props.html (дата звернення: 12.11.2019).

14. Snapshot Testing. URL : https://jestjs.io/docs/ru/snapshot-testing (дата звернення: 17.10.2019).

15. What is the Virtual DOM? URL : https://reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom (дата звернення: 17.10.2019).

ДОДАТОК А

Код головного сценарію

```
import React from 'react';
import {Pages} from './containers/enums';
import {Subscribe} from 'unstated';
import {AppContainer} from './containers/AppContainer';

import Home from './pages/Home/Home.jsx';
import About from './pages/About/About.jsx';
import Play from './pages/Play/Play.jsx';
import Learn from './pages/Learn/Learn.jsx';
import {NavButtons} from './components/Buttons/Button.jsx';
import {GameContainer} from './containers/GameContainer';
import { LearnContainer } from './containers/LearnContainer';
import DefaultLearnPageSpinner from
'./pages/Learn/components/DefaultLearnPageSpinner/DefaultLearnPageSpinner.jsx';
import './App.scss';

function App(props) {
  return (
    <AppMainContainer>
      <NavButtonsComponent />
      <PageComponent />
    </AppMainContainer>
  );
}

function AppMainContainer(props) {
  return (
    <div className="container-fluid mt-5 main-header justify-content-center">
      {props.children}
    </div>
  );
}

function NavButtonsComponent(props) {
  // Do not render top nav bar when home page is shown.
  return (
    <Subscribe to={[AppContainer, GameContainer]}>
      {(app, game) => ( <NavButtons app={app.state} game={game.state} /> )}
    </Subscribe>
  );
}

function PageComponent(props) {
  return (
```

```
    <Subscribe to={[AppContainer]}>
      {app => {
        let mainView;
        switch (app.state.page) {
        case Pages.HOME:
          mainView = <Home switchPage={app.switchPage} />;
          break;
        case Pages.ABOUT:
          mainView = <About switchPage={app.switchPage} />;
          break;
        case Pages.PLAY: // Valid state but handled differently below.
          break;
        case Pages.LEARN:
          mainView = <Learn switchPage={app.switchPage} />;
          break;
        default:
          console.log("Unknown page in app state: "+app.state.page);
          return;
        }

        // If player clicks on "Play" once, have component instance but hide it.
        return (
          <div className="row">
            {app.state.loaded ?
              <React.Fragment>
              {mainView}
              {app.state.played && <Play shouldHide={app.state.page != Pages.PLAY}/>}
              </React.Fragment>
              :

              <div className="row">
                <div className="pl-4 col-md-4 offset-md-4 learn-content-spinner-margin justify-
content-center">
                  <DefaultLearnPageSpinner className="pl-5 ml-4" />
                </div>
              </div>
            }
          </div>
        );
      }}
    </Subscribe>
  );
}
type SearchParams = { language: string, article: string }

type AppState = {
  page: any,
  played: boolean,
  learnLanguage?: string,
  readArticle?: string,
  searchParams?: SearchParams
}
```

```
// AppContainer contains main states of application.
export class AppContainer extends Container<AppState> {
  constructor(searchParams: SearchParams) {
    super();
    this.state = {
      page: Pages.HOME,
      played: false,
      learnLanguage: undefined,
      readArticle: undefined,
      loaded: false
    }

    this._loadAllArticles(searchParams)
  }

  switchPage = (page: Page, learnLanguage?: string) => {
    this.setState({
      page: page,
      played: (this.state.played || page == Pages.PLAY), // If at least once the Play page is visited, or
not
      learnLanguage: learnLanguage,
    });
  }

  clearDefaultContent = () => {
    if (this.state.learnLanguage || this.state.readArticle) {
      this.setState({
        learnLanguage: undefined,
        readArticle: undefined
      });
    }
  }

  goToHomePage = () => {
    this.setState({
      page: Pages.HOME,
      played: false,
      learnLanguage: undefined,
      readArticle: undefined
    })
  }

  _loadAllArticles = async (searchParams: any) => {
    try {
      const allArticles: Array<any> = await API.allArticles();

      let checkIfArcicle = false;

      allArticles.map((item) => {
        if (item.thumbnail_url === searchParams.all) {
          checkIfArcicle = true;
```

```
        }
      })

      if (checkIfArcicle) {
        this.setState({
          page: Pages.LEARN,
          readArticle: searchParams.all,
          learnLanguage: undefined,
          loaded: true
        }, () => {
        })
      }
      if (searchParams.all !== "" && searchParams.all.length > 0 && searchParams.all !== "/" &&
!checkIfArcicle) {
        this.setState({
          page: Pages.LEARN,
          learnLanguage: searchParams.all,
          loaded: true
        }, () => {
        });
      }

      this.setState({
        loaded: true
      })
    }catch (e) {

      this.state = {
        page: Pages.HOME,
        played: false,
        learnLanguage: undefined,
        readArticle: undefined,
        loaded: true
      }
    }
  }
}

export default App;
```

ДОДАТОК Б

Код сценарію GameContainer

```
// @flow
import {Container} from 'unstated';
import _ from 'lodash';
import {Answer, Difficulty, GameMode, User} from './enums';
import type {AnswerState, DifficultyLevel} from './enums';
import {API} from '../api/API';
import type {QuestionResponse} from '../api/models';

const initMultipleType = {
  type: "score",
  subType: 500
}

const MaxLevel = 10;

function initPlayers(count, arrayOfPlayer) {
  const array = [];
  for (let i = 0; i < count; i++) {
    const user = {...User};
    if(!arrayOfPlayer) {
      user.name = `${user.name} ${i + 1}`;
      array.push(user)
    }else {
      const newArray = [...arrayOfPlayer];
      if(newArray[i]) {
        array.push(newArray[i])
      }else {
        user.name = `${user.name} ${i + 1}`;
        array.push(user)
      }
    }
  }

  return array;
}

type Question = {
  data: QuestionResponse,
  audioURL: string,
  answerIs: AnswerState,
  chosenAnswer?: string
}

type GameState = {
```

```
  ready: boolean,
  difficulty: DifficultyLevel,
  lives: number,
  level: number,
  score: number,
  gameOver: boolean,
  question: ?Question,
  highlightButtonAt: number // index,
}

// GameContainer contains game states.
export class GameContainer extends Container<GameState> {
  state = {
    ready: false,
    difficulty: Difficulty.NotChosen,
    gameMode: GameMode.NotChosen,
    lives: 3,
    level: 1,
    score: 0,
    gameOver: false,
    question: {},
    highlightButtonAt: -1,
    settingsVisibility: false,
    multipleMode: initMultipleType,
    masOfPlayerIndex: [],
    isMultipleModeSetUp:false,
    volume: 100,
    wasAnswered: false,
    autoPlay: true,
    theHighestScore: 0,
    arrayOfPlayer: initPlayers(2)
  };

  constructor() {
    super();

    this.setupKeyboardControls();
  }

  setupKeyboardControls = () => {
    window.onkeyup = (e) => {
      if (this.state.question.data == null) {
        return;
      }

      const index = this.state.highlightButtonAt;
      const minIndex = 0;
      const maxIndex = this.state.question.data.answers.length-1;
      const keyCode = (e.which || e.keyCode);
      const minNumKeyCode = 49; // key code for 1

      switch (keyCode) { // Depending on what the browser supports
```

```
      // Left arrow key
      case 37:
        if (index > minIndex) this.setState({highlightButtonAt: index-1});
        break;

      // Right arrow key
      case 39:
        if (index < maxIndex) this.setState({highlightButtonAt: index+1});
        break;

      // Enter and spacebar
      case 13: case 32:
        if (this.isQuestionAnswered()) {
          this.nextQuestion(); // next question or game over
          break;
        }
        // i.e. the question is not answered and this does it
        if (minIndex <= index && index <= maxIndex) {
          this.answerQuestion(this.state.question.data.answers[index]);
        }
        break;

      // Numbers, 1-9
      case 49: case 50: case 51: case 52: case 53: case 54: case 55: case 56: case 57:
        let indexFromNum = keyCode - minNumKeyCode;
        if (minIndex <= indexFromNum && indexFromNum <= maxIndex) {
          this.setState({highlightButtonAt: indexFromNum});
        } else {
          this.setState({highlightButtonAt: -1}); // Correct number or nothing
        }
        break;

      default:
        return // Do not prevent default behaviour.
      }

      e.preventDefault();
    }

    // Disable HTML scroll for spacebar because it's the button to apply answer or go to next
question.
    const htmlElement: any = document.querySelector("html");
    htmlElement.onkeydown = (e) => {
      if ((e.which || e.keyCode) == 32) { // Space bar
        e.preventDefault();
      }
    };
  }

  setAutoPlayValue = (autoPlay) => {
    this.setState({
      autoPlay
```

```
  })
}

setCountOfPlayer = (number) => {
  this.setState({
    arrayOfPlayer: initPlayers(number, this.state.arrayOfPlayer)
  })
}

setPlayerName = (index, name) => {
  const setArrayOfPlayer = [...this.state.arrayOfPlayer];
  setArrayOfPlayer[index].name = name;

    this.setState({
      arrayOfPlayer: setArrayOfPlayer
    })
}

setMultipleType = (value) => {
  this.setState({
    multipleMode: {
      ...this.state.multipleMode, ...{
        type: value
      }
    }
  }, () => {
    this.setState({
      multipleMode: {
        ...this.state.multipleMode, subType: this.state.multipleMode.type === "score" ? 500 : 3
      }
    })
  })
}

setMultipleSubType = (value) => {
  this.setState({
    multipleMode: {...this.state.multipleMode, ...{subType : value}}
  })
}

setUpMultipleModeSetUp = () => {
  this.setState({
    isMultipleModeSetUp: true
  })

  const newArray = [];

  this.state.arrayOfPlayer.map((player) => {
    newArray.push({...player,lives: this.state.multipleMode.subType})
  })

  this.setState({
```

```
    arrayOfPlayer: newArray
   })

   this.setNewRandomOrderToUser();
 }

 setNewRandomOrderToUser = () => {
   const newArray = [];
   this.state.arrayOfPlayer.map((player, index) => {
     if(player.lives !== 0) {
       newArray.push(index);
     }
   })

     let check = true;
     this.state.arrayOfPlayer.map(item => {
       if(item.lives) {
         if(item.isAnswered === null) {
           check = false;
         }
       }});
   if(check) {
     newArray.sort((a, b) => {
       return Math.random() - 0.5;
     });

     this.setState({
       masOfPlayerIndex: newArray
     })
   }else {
     this.setState({
       masOfPlayerIndex: newArray
     })
   }
 }

 changeVolume = (value) => {
   this.setState({
     volume: value
   })
 }

 changeSettingsVisibility = () => {
   this.setState({
     settingsVisibility: !this.state.settingsVisibility
   })
 }

 setDifficulty = (difficulty: DifficultyLevel) => {
   this.setState({difficulty: difficulty}).then(() => { this.nextQuestion(); });
 }
```

```
setGameMode = (gameMode) => () => {
  this.setState({gameMode});
}

_gameOver = (): boolean => {
  if (this.state.gameOver) {
    return true;
  }

  if(this.state.multipleMode.type === "score") {
    let check = false;
    this.state.arrayOfPlayer.map(item => {
      if(item.score >= this.state.multipleMode.subType) {
        check = true;
      }
    })

    if(check) {
      this.setState({gameOver: true});
      return true
    }
  }

  if (this.state.lives == 0) {
    this.setState({gameOver: true});
    return true
  }
  return false
}

setUserAnswer = (index, value) => {
  let arrayOfAnswer = _.cloneDeep(this.state.arrayOfPlayer);
  const question: Question = this.state.question;
  const correctAnswer = question.data.correct;
  arrayOfAnswer[index].isAnswered =  value === correctAnswer;
  arrayOfAnswer[index].languageAnswer = value;

  this.setState({
    arrayOfPlayer: arrayOfAnswer
  })
}

isAllUserAnswered = () => {
  let check = false;
  this.state.arrayOfPlayer.map(item => {
    if(item.lives) {
      if(item.isAnswered != null) {
        check = true;
      }
    }else {
      if(item.isAnswered != null) {
        check = true;
```

```
      }
     }
    })
    if(check) {
     this.calcPlayerAfterAnswer();
    }
    return check;
  }

  calcPlayerAfterAnswer =  () => {
    const newArray = [];
    if(!this.state.wasAnswered){
    this.state.arrayOfPlayer.map(item => {
     if(item.lives > 0) {
      const check = {...item};
      if(item.isAnswered !== null && !item.isAnswered) {
       if(this.state.multipleMode.type !== "score") {
        check.lives =  check.lives -1;
       }
       newArray.push(check)
      }else {
       if(item.isAnswered) {
        check.score = check.score + this.state.question.data.points;
        newArray.push(check)
       }else {
        newArray.push(check);
       }
      }
     }else {
      newArray.push(item)
     }
    })
    this.setState({
     arrayOfPlayer: newArray,
     wasAnswered: true
    })
    }
  }

  newQustionMultiple = () => {
    let newArray = [...this.state.arrayOfPlayer];

    newArray = newArray.map(item => {
     item.isAnswered = null;
     return item;
    })

    let newMainArray = _.cloneDeep(this.state.masOfPlayerIndex);
    newMainArray.splice(0, 1);

    this.setState({
     masOfPlayerIndex: newMainArray
```

```
    }, () => {
      if(newMainArray.length === 0) {
        this.setNewRandomOrderToUser();
      }
      this.setState({
        wasAnswered: false,
        arrayOfPlayer: newArray
      })
    })
  }

  nextQuestion = async () => {
    if (this._gameOver()) {
      return
    }
    if(this.state.gameMode === GameMode.SingleMode) {
      const data: QuestionResponse = await API.newQuestion(this.state.difficulty, this.state.score);
      const question: Question = {
        data: data,
        audioURL: API.toAudioURL(data.checksum),
        answerIs: Answer.NotGivenYet
      }
      question.data.correct = data.correct;
      question.answerIs = Answer.NotGivenYet;
      this.setState({
        ready: true,
        level: question.data.level,
        question: question,
        highlightButtonAt: -1,
      });
    }else {
      let highestScoreOfPlayer = 0;
      if( this.state.level !== MaxLevel) {
        highestScoreOfPlayer = Math.max.apply(Math, this.state.arrayOfPlayer.map(function
(player) {
          return player.score;
        }))
        this.setState({
          theHighestScore: highestScoreOfPlayer
        })
      }else {
        highestScoreOfPlayer = this.state.theHighestScore
      }

      if( this.state.multipleMode.type === "score" && highestScoreOfPlayer >=
this.state.multipleMode.subType) {
        this.setState({gameOver: true});
      }

      if(this.state.multipleMode.type !== "score" ) {
        let continueGame = 0;
```

```
        this.state.arrayOfPlayer.map(player => {
          if(player.lives > 0) {
            continueGame += 1;
          }
        });

        if(continueGame <=1) {
          this.setState({gameOver: true});
        }
      }
      const data: QuestionResponse = await API.newQuestion(this.state.difficulty,
highestScoreOfPlayer);
      const question: Question = {
        data: data,
        audioURL: API.toAudioURL(data.checksum),
        answerIs: Answer.NotGivenYet
      }
      question.data.correct = data.correct;
      question.answerIs = Answer.NotGivenYet;
      this.setState({
        ready: true,
        level: question.data.level,
        question: question,
        highlightButtonAt: -1,
      });
    }
  }

  answerQuestion = (chosenAnswer: string) => {
    const question: Question = this.state.question;
    const correctAnswer = question.data.correct;
    question.chosenAnswer = chosenAnswer;
    question.data.correct = correctAnswer;
    const answerIsCorrect = (chosenAnswer == correctAnswer);

    if (answerIsCorrect) {
      question.answerIs = Answer.IsCorrect;
      this.setState({
        question: question,
        score: this.state.score+question.data.points
      });
      return;
    } else {
      question.answerIs = Answer.IsWrong;
      this.setState({
        question: question,
        lives: this.state.lives-1
      });
    }
  }

  isQuestionAnswered(): boolean {
```

```
    return this.state.question.answerIs != Answer.NotGivenYet;
  }

  _encodeString = (s: string): string => {
    return btoa(s);
  }

  _decodeString = (s: string): string => {
    return atob(s);
  }

  restart = () => {
    this.setState({
      difficulty: Difficulty.NotChosen,
      gameMode: GameMode.NotChosen,
      lives: 3,
      level: 1,
      score: 0,
      gameOver: false,
      question: {},
      settingsVisibility: false,
      multipleMode: initMultipleType,
      masOfPlayerIndex: [],
      isMultipleModeSetUp:false,
      wasAnswered: false,
      arrayOfPlayer: initPlayers(2)
    });
  }
}
```

ДОДАТОК В

Код сценарію для контролю ігрових механік на Go сервері

```
package game

import (
  "fmt"
  "gitlab.com/baburaman/checkbeck/models"
  "gitlab.com/baburaman/checkbeck/utils"
  "math"
)

// RefreshLevels sets game levels as given levels.
func (g *Game) RefreshLevels(levels []*models.Level) {
  g.mu.Lock()
  g.levels = levels
  g.mu.Unlock()
}

// NewQuestion returns a new question by taking earned points into account.
func (g *Game) NewQuestion(points, difficulty int) (*models.Question, error) {
  if points < 0 {
    return nil, fmt.Errorf("game: negative amount of points")
  }
  level := g.findLevel(points)
  //changing
  question, err := g.makeQuestion(level, difficulty)
  if err != nil {
    return nil, errorf("failed to make question: %v", err)
  }

  question.Details = make(map[string]*models.LanguageData)
```

```
  for _, language := range question.Answers {
    languageData := &models.LanguageData{}

    prop, err := g.r.GetLanguagePropByName(language)
    if err != nil {
      return nil, errorf("failed to get language prop by name: %v", err)
    }
    if prop == nil {
      continue
    }

    languageData.LanguageProp = *prop
    languageData.Language = language
    languageData.Classification, err = g.r.GetLanguageClassification(language)
    if err != nil {
      return nil, errorf("failed to get language classification: %v", err)
    }

    question.Details[language] = languageData
  }

  return question, nil
}

// findLevel finds the level which the player is at.
func (g *Game) findLevel(points int) (level *models.Level) {
  g.mu.RLock()
  defer g.mu.RUnlock()

  for _, level = range g.levels {
    if level.MaxPoints == -1 { // -1 == infinite == level never ends
      break
    }
    if level.MaxPoints > points {
      break
```

```
    }
  }
  return
}

// makeQuestion gets a random audio, prepares answers looking at level properties
// and returns them along with the point delta to use when answered correctly.
func (g *Game) makeQuestion(level *models.Level, difficulty int) (*models.Question, error) {
  groupField := level.GroupsOn
  limit := randomAnswerCount(level) - 1

  // Determine max rank for answers.
  maxRank, err := level.MaxRankAt(difficulty)
  if err != nil {
    return nil, fmt.Errorf("game: failed to determine max rank for level %d: %v", level.ID, err)
  }

  // Get random language ID from sounds table using max rank.
  languageID, err := g.r.GetRandomLanguageID(maxRank)
  if err != nil {
    return nil, fmt.Errorf("game: failed to get random language ID: %v", err)
  }
  sound, err := g.r.GetSoundForLanguage(languageID)
  if err != nil {
    return nil, fmt.Errorf("game: failed to get random sound for new question: %v", err)
  }
  lang, err := g.r.GetLanguage(languageID)
  if err != nil {
    return nil, fmt.Errorf("game: failed to get language for random sound: %v", err)
  }

  // Try to include the variety if it's a hard level.
  if level.Hard > 0 {
    if len(lang.Variety) > 0 {
      lang.Name = fmt.Sprintf("%s (%s)", lang.Name, lang.Variety)
```

```
    }
  }

  answers := make([]string, 0)
  var relationalWeight float64

  if level.Hard > 0 {
    answers, relationalWeight, err = g.aggregatedRelativesOf(lang.ID, groupField, maxRank, limit)
  } else {
    // Zero relational weight for easy levels since it's too complex to find out.
    answers, err = g.r.NonRelativesOf(lang.ID, groupField, maxRank, limit)
  }
  if err != nil {
    return nil, err
  }

  return &models.Question{
    Level:    level.ID,
    Correct:  lang.Name,
    Answers:  utils.RandomConcat(lang.Name, answers),
    Checksum: sound.Checksum,
    Source:   sound.Source,
    Points:   calculatePointsForQuestion(level, lang.Rank, len(answers)+1, relationalWeight),
  }, nil
}

func (g *Game) aggregatedRelativesOf(languageID int, groupField string,
  maxRank, limit int) ([]string, float64, error) {

  groups := models.LanguageGroups()
  invGroups := models.LanguageGroupsInverse()

  whereNot := make(map[string]string)
  answers := make([]string, 0)
  step := 1
```

```
var relationalWeight float64
var groupValue string
var err error

for {
  if len(groupField) > 0 {
    groupValue, err = g.r.GetLanguageField(languageID, groupField)
    if err != nil {
      return nil, 0, err
    }
  }
  found, err := g.r.RelativesOf(languageID, groupField, groupValue, whereNot, maxRank, limit)
  if err != nil {
    return nil, 0, err
  }

  // Step one level down and continue if null field.
  if len(groupValue) == 0 {
    index, ok := groups[groupField]
    if !ok {
      break
    }
    index--
    groupField, ok = invGroups[index]
    if !ok {
      break
    }
    step++
    continue
  }

  // Update relational weight in a decaying manner. found/1, found/2 and so on.
  relationalWeight += float64(len(found)) / float64(step)
```

```
    // Append found answers and stop if enough answers are found.
    // Update the limit as remaining amount of required answers while doing this check.
    answers = append(answers, found...)
    limit = limit - len(found)
    if limit == 0 {
      break
    }

    // Note down the group and value, step one level down.
    whereNot[groupField] = groupValue
    index, ok := groups[groupField]
    if !ok {
      break
    }
    index--
    groupField, ok = invGroups[index]
    if !ok {
      break
    }
    step++
  }

  return answers, relationalWeight, nil
}

// randomAnswerCount returns a random answer count using min and max values.
func randomAnswerCount(level *models.Level) int {
  if level.AnswerMax <= level.AnswerMin {
    return level.AnswerMin
  }
  return utils.RandN(level.AnswerMax-level.AnswerMin+1) + level.AnswerMin
}

// calculatePointsForQuestion calculates points for a question.
func calculatePointsForQuestion(level *models.Level, rankOfCorrectAnswer int,
```

```
  answerCount int, relationalWeight float64) int {
  pointsFromRank := float64(rankOfCorrectAnswer) * level.MultRank
  pointsFromAnswerCount := float64(answerCount) * level.MultAnswers
  pointsFromRelations := relationalWeight * level.MultRelation

  return int(math.Ceil(pointsFromRank + pointsFromAnswerCount + pointsFromRelations))
}
```