МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії програмного забезпечення, канд. техн. наук, доцент,

_____Є.О. Давиденко

«____»_____2022 р.

КОМПЛЕКСНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ЗНАНЬ. FRONT-END ЧАСТИНА

Том 3

Спеціальність «Інженерія програмного забезпечення»

121 – ККРБ.3 – 409.21810910

Студент _____О. А. Гроза

підпис

«___» _____ 2022 р.

Керівник старший викладач _____С. Ю. Боровльова

підпис

«___» _____ 2022 р.

Консультант канд. техн. наук, доцент кафедри екології

_____А. О. Алексєєва

підпис

«___» _____ 2022 р.

Миколаїв – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри інженерії програмного

забезпечення, канд.техн.наук, доцент,

_____Є.О. Давиденко

«____»_____2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи бакалавра

Видано студенту групи 409 факультету комп'ютерних наук

Грозі Олені Андріївні.
(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи

«Програмне забезпечення автоматизації тестування знань. Front-end частина»

Затверджена наказом по ЧНУ від «25» травня 2022 р. № 95

2. Строк представлення кваліфікаційної роботи «27» червня 2022 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Вхідні дані до роботи − функціональні та нефункціональні вимоги до програмного забезпечення автоматизації тестування знань. Результат − функціонуюча система автоматизації тестування знань.

4. Перелік питань, що підлягають розробці

- аналіз принципів та підходів до тестування знань, тобто дослідження предметної області;
- огляд існуючих систем тестування знань;
- розробка вимог до системи на основі зібраної інформації про предметну область та аналоги;

– проєктування автоматизованої системи тестування знань;
– програмна реалізація клієнтської частини вебзастосунку для тестування знань.

5. Перелік графічних матеріалів:

Презентація.

6. Завдання до спеціальної частини

Питання охорони праці при розробці програмого забезпечення.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алєксєєва А. О.	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи старший викладач, Боровльова С. Ю.
(посада, прізвище, ім'я, по батькові)

(підпис)

Завдання прийнято до виконання

Гроза Олена Андріївна
(прізвище, ім'я, по батькові студента)

(підпис)

Дата видачі завдання «____» _____ 2022 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: «Програмне забезпечення автоматизації тестування знань. Front-end частина» .

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання ККРБ	03.10.2021р.	04.10.2021р.	виконано
2.	Складання календарного плану ККРБ	05.10.2021р.	09.10.2021р.	виконано
3.	Огляд літератури за темою ККРБ	10.10.2021р.	10.11.2021р.	виконано
4.	Аналіз предметної області	11.11.2021р.	20.12.2021р.	виконано
5.	Розробка проєктних рішень	21.12.2021р.	14.01.2022р.	виконано
6.	Моделювання та конструювання ПЗ	15.01.2022р.	14.03.2022р.	виконано
7.	Кодування, тестування та апробація розробленого ПЗ	15.03.2022р.	25.05.2022р.	виконано
8.	Написання спеціальної частини з охорони праці	26.05.2022р.	01.06.2022р.	виконано
9.	Відгук керівника ККРБ	02.06.2022р.	05.06.2022р.	виконано
10.	Оформлення ККРБ та презентації	06.06.2022р.	12.06.2022р.	виконано
11.	Попередній захист	13.06.2022р.	13.06.2022р.	виконано
12.	Завершення оформлення ККРБ та презентації	14.06.2022р.	20.06.2022р.	виконано
13.	Рецензування	21.06.2022р.	26.06.2022р.	виконано
14.	Захист кваліфікаційної роботи	27.06.2022р.	27.06.2022р.	

Розробив студент Гроза Олена Андріївна _____
(прізвище, ім'я, по батькові студента) (*підпис*)

«____» _____ 2022 р.

Керівник роботи старший викладач, Боровльова С. Ю _____
(посада, прізвище, ім'я, по батькові) (*підпис*)

«____» _____ 2022 р.

АНОТАЦІЯ

до комплексної кваліфікаційної роботи бакалавра

«Програмне забезпечення автоматизації тестування знань. Front-end частина»

Студент 409 гр.: Гроза Олена Андріївна

Керівник: ст. викладач кафедри ІПЗ, Боровльова С. Ю.

Даний том комплексної роботи спрямований на розробку користувацького інтерфейсу застосунку. Актуальність створення графічного інтерфейсу для системи полягає у сприянню ефективній взаємодії користувача із застосунком, легкості користування ним.

Об'єкт: процес проходження тестування для перевірки знань в закладах освіти.

Предмет: засоби автоматизації процесу тестування, підрахунку та аналізу результатів.

Мета роботи полягає у підвищенні ефективності перевірки знань здобувачів освіти за рахунок зменшення кількості помилок при перевірці тестів, підвищення швидкості та точності підрахунку результатів шляхом розробки автоматизованої системи тестування.

Даний том складається зі вступу, двох розділів, висновків та додатків, а також спеціальної частини «Охорона праці».

У вступі викладається актуальність теми, мета та поставлені завдання щодо реалізації front-end частини застосунку. В першому розділі обґрунтовується вибір технологій для розробки front-end, наводиться їх опис. Другий розділ охоплює процес створення клієнтської частини застосунку, використовуючи фреймворк Angular та опис користувацьких інтерфейсів.

У висновках проводиться аналіз роботи та отриманих результатів.

В спеціальній частині «Охорона праці» розглянуто питання безпеки праці під час розробки програмного забезпечення.

Том ККРБ викладений на 46 сторінок, містить 2 розділи, 38 ілюстрацій, 5 таблиць, 9 джерел в переліку посилань.

Ключові слова: *тестування знань, автоматизована система, графічний інтерфейс користувача, вебдизайн, Angular framework, Tailwind CSS.*

ABSTRACT

of the Bachelor's Thesis

"Software for automation of knowledge testing. Frontend"

Student of group 409: Hroza Olena Andriivna

Supervisor: Senior Lecturer of the Department of Software Engineering

Borovlova S. Yu.

This volume of the complex work is aimed at the development of the user interface of the application. The topicality of creating a graphical user interface for the system lies in facilitating effective interaction between the user and the application, as well as ensuring ease of use.

The object of the work is the process of test taking for the assessment of knowledge in education institutions.

The subject of the work are the means for automating the process of testing as well as calculating and analyzing test results.

The purpose of this work is to increase the efficiency of knowledge assessment of students by reducing the number of errors in checking tests, increasing the speed and accuracy of calculating results through the development of an automated testing system.

This volume consists of an introduction, three sections, conclusions and appendixes, as well as a special “Workplace safety” section.

In the introduction we define the topicality of the topic, the purpose and the tasks pertaining to implementing the frontend part of the application. In the first section we justify the choice of technologies and provide their description. The second section encompasses the process of developing the client side of the application using Angular framework and a description of user interfaces.

In the special section titled “Workplace safety” the questions of the safety of labor during the process of software development are discussed.

In the conclusion we analyze the work and the obtained results.

This volume is 46 pages long and includes 2 sections, 38 illustrations, 5 tables, and 9 sources.

Keywords: *knowledge testing, automated system, graphical user interface, web design, Angular framework, Tailwind CSS.*

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – Application Programming Interface (прикладний програмний інтерфейс)

DOM – Document Object Model (об’єктна модель документа)

GUI − Graphical User Interface (графічний інтерфейс користувача)

БД – база даних

ООП – об’єктно-орієнтоване програмування

ПЗ – програмне забезпечення

ВСТУП

Застосунок, що розроблюється, має клієнт-серверну архітектуру. Back-end частина відповідає за зберігання даних та бізнес-логіку. Завданням Front-end є презентаційна логіка та забезпечення взаємодії системи з користувачем.

Як відомо, враження користувача від роботи з програмним продуктом значною мірою залежить від його зовнішнього вигляду [6]. Важливими факторами є інтуїтивна зрозумілість та естетична привабливість інтерфейсу.

Сфера веброзробки постійно розвивається, з'являються нові технології створення користувацьких інтерфейсів та тенденції в дизайні. Сучасні інструменти роблять створення вебінтерфейсів більш ефективним, а отже їх вивчення є актуальним. Попит на front-end розробників на ринку праці є стабільно високим [7], адже багато власників сайтів намагаються модернізувати та покращити їх для залучення нових клієнтів. Не бракує також попиту на створення нових вебсайтів, адже представленість в інтернеті в наш час є критично важливою для бізнесів.

Метою даного розділу є створення користувацького інтерфейсу для ефективної взаємодії із back-end частиною застосунку для автоматизації тестування знань. Для досягнення цілі необхідно виконати наступні завдання:

1. Проаналізувати наявні технології розробки front-end частини застосунку та стилізації GUI, визначити їх основні характеристики, виявити переваги та недоліки. На основі аналізу здійснити вибір технологій розробки.
2. Розглянути основні принципи створення користувацьких інтерфейсів. Розробити макети інтерфейсів застосунку.
3. Реалізувати та протестувати front-end частину програмної системи. Забезпечити взаємодію із back-end.
4. Навести опис створених користувацьких інтерфейсів.

Результатом роботи має бути готова front-end частина програмного забезпечення автоматизації тестування знань.

1 ТЕХНОЛОГІЇ СТВОРЕННЯ FRONT-END ЧАСТИНИ КЛІЄНТ-СЕРВЕРНОГО ЗАСТОСУНКУ

1.1 Огляд front-end фреймворку Angular

1.1.1 Загальні відомості про технологію

Angular – популярний фреймворк для створення клієнтських застосунків за допомогою HTML, CSS та TypeScript. Фреймворк безкоштовний для використання та має відкритий код. Технологія розроблена компанією Google.

Важливо розрізняти версії фреймворку, адже його попередник AngularJS кардинально відрізняється від сучасного Angular. У 2010 році був створений JavaScript фреймворк ***AngularJS***, відомий також як Angular1. Він швидко набрав популярність, проте його внутрішня архітектура була занадто складною, до того ж він не відповідав потребам сучасної веброзробки. Тому в середині 2016 року команда Angular випустила повністю перероблену версію, ***Angular2***. Цього разу фреймворк був написаний на мові TypeScript. Версії настільки сильно відрізнялися, що їх можна вважати різними фреймворками. Для зручності усі версії після Angular2 називають просто ***Angular*** (для розрізнення з AngularJS). Актуальною версією на момент написання даної роботи є Angular 13. В кожному новому релізі з'являються нові можливості та відбувається оптимізація процесів, проте різниця між версіями після другої є відносно малою.

Фреймворк Angular стабільно зберігає високу популярність впродовж останніх років. Фреймворк Angular використовує велика кількість всесвітньо відомих компаній, таких як Google, Nike, PayPal, HBO, The Guardian та багатьох інших. Деякі з них наведені на рисунку 1.1.

Рисунок 1.1 – Компанії, що використовують фреймворк Angular

Одним з найкращих прикладів, що демонструють гнучкість та потужність фреймворку Angular, є сервіс Gmail. Це односторінковий застосунок, яким щоденно користуються більш ніж 1,5 мільярди людей. Застосунок відображає дані в реальному часі при надходженні нових електронних листів, окрім того він підтримує миттєвий обмін повідомленнями в Google Hangouts. Іншими вартими уваги функціями є створення чернеток листів, вбудована перевірка тексту на граматичні та орфографічні помилки, прогнозування введення. Отже, фреймворк підходить для створення складного та нестандартного функціоналу і може використовуватися у великих проєктах.

1.1.2 Обґрунтування вибору фреймворку розробки

Використання фреймворку, порівняно із класичним JavaScript або jQuery, має такі основні переваги:

- надає застосунку чітку та зрозумілу структуру, полегшує його підтримку;
- полегшує процес тестування ПЗ;
- автоматичне створення компонентів системи та генерація коду за допомогою Angular CLI;
- легкість повторного використання коду в різних місцях проєкту.

Окрім Angular, існують інші front-end фреймворки. Для обґрунтування доцільності використання саме Angular, порівняємо фреймворк з його конкурентом React.js. Зіставлення основних характеристик та особливостей технологій наведене у таблиці 1.1.

Таблиця 1.1 – Порівняння front-end технологій Angular та React

	Angular	**React**
Мова	TypeScript, HTML	JavaScript, JSX (JavaScript Syntax Extension)
Розробник	Google	Facebook

Кінець таблиці 1.1

	Angular	**React**
Короткий опис	Фреймворк, заснований на компонентах для створення масштабованих вебзастосунків. Включає набір добре інтегрованих бібліотек та широкий функціонал (маршрутизація, обмін інформації із сервером та ін.)	JavaScript бібліотека для front-end розробки. Дозволяє легко створювати складні інтерфейси користувача, розробляти швидкі та масштабовані застосунки.
Особливості мови	TypeScript є надбудовою над JavaScript, в якій присутня типізація даних та інструментарій для ООП	JSX – розширення мови JavaScript, впроваджене в React. Використовується для розмітки та має подібний синтаксис до HTML.
Тенденції використання	При створенні складних програм корпоративного рівня, таких як односторінокові застосунки	Використовується для створення компонентів UI для будь-яких застосунків із змінними даними
Функціонал	Багатий вбудований функціонал	Значно менший функціонал, для його розширення необхідне додаткове встановлення додаткових бібліотек
Легкість вивчення	Порівняно важкий в опануванні	Середньої складності

На відміну від React, у фреймворку Angular підтримується механізм двосторонньої прив’язки даних. Це сприяє легкій синхронізації даних між представленням та моделлю, пришвидшує створення інтерактивного інтерфейсу. Прив’язка даних дозволяє позбутися необхідності написання функцій для оновлення даних і таким чином робить код більш лаконічним і простим.

Обидві технології – і Angular, і React – представляють собою сучасний та гнучкий підхід до створення користувацького інтерфейсу. Проте перевагою Angular є багатий функціонал, що надається одразу при встановленні фреймворку. Інтеграція всіх модулів налагоджена, тому не постає проблема із додаванням безлічі зовнішніх бібліотек та забезпеченням їх сумісності. Синтаксис TypeScript зручний у використанні в об’єктно-орієнтованому програмуванні та наближений до таких мов, як Java або C#.

Фреймворк Angular складніше вивчити, проте використання Angular CLI прискорює цей процес, адже за допомогою простих команд дозволяє автоматично створювати компоненти, налаштовувати конфігурації тощо.

Застосунок, що розробляється, має саме односторінкову структуру, для створення якої чудово пристосований Angular. Зважаючи також на велику кількість різноманітного вбудованого функціоналу в Angular для розробки віддано перевагу саме цьому фреймворку.

1.2 Особливості мови TypeScript

Мовою, якою написано фреймворк Angular, та яка використовується при розробці на базі нього, є TypeScript. Мова була розроблена компанією Microsoft.

TypeScript – мова програмування, що є надбудовою JavaScript. Тобто кожна програма, написана на JavaScript є правильною і в TypeScript [8]. Проте TypeScript надає розробнику низку можливостей, відсутніх у JavaScript:

- строга типізація: правильність типів даних перевіряється під час компіляції;

- об’єктно-орієнтовані риси: TypeScript підтримує інтерфейси, а JavaScript – ні;
- зазначення помилок ще до компіляції: через це менш вірогідні помилки під час виконання програми, тоді як JavaScript є інтерпретованою мовою.

Програми, написані на TypeScript, транслюються в JavaScript. Якщо використовувати мову самостійно, перед запуском програми її потрібно транслювати за допомогою TSC (TypeScript Compiler) [3]. Але у застосунках Angular немає необхідності робити це вручну: трансляція відбувається засобами самого фреймворку. Коли мова йде про TypeScript, доречно вживати саме термін «транспіляція».

Компіляція – загальний термін на позначення переведення вихідного коду з однієї мови на іншу. На практиці зазвичай термін використовують для процесу переведення коду високорівневої мови програмування в машинний код.

Транспіляція – трансформація коду з однієї мови на іншу, для якої характерний подібний рівень абстракції. TypeScript та JavaScript обидві є високорівневими мовами програмування, тому має місце саме цей процес. Трансформація в код JavaScript відбувається для того, щоб браузери могли розпізнавати та виконувати код.

На рисунку 1.2 показано в дії механізм строгої типізації у TypeScript. При спробі присвоїти змінній інший тип, ніж задекларований, з'являється повідомлення про помилку.

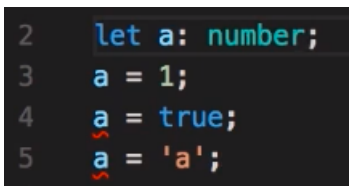

Рисунок 1.2 – Строга типізація в TypeScript

Мова пропонує широкий набір простих та складних типів даних: boolean, string, number, array, enum, object, interface та ін. Слід звернути увагу на кілька

нестандартних типів: *tuple* є масивом заздалегідь визначеної довжини та з визначеними типами кожного значення; тип *union* дозволяє задати кілька альтернативних варіантів типів для змінної. Також існє тип *any*, який може приймати значення будь-якого типу та фактично є способом обійти механізм строгої типізації. Використання цього типу не рекомендується, окрім випадків, коли це конче необхідно. Якщо у змінної не вказаний тип, вона є *any* за замовченням. Отже, TypeScript надає багатий та гнучкий набір типів даних, а також підтримку ООП.

Іншою особливістю мови є спрощений синтаксис написання функцій – так звані *arrow functions*. Приклад синтаксису наведено на рисунку 1.3.

```
let log = function(message) {
    console.log(message);
}

let doLog = (message) => {
    console.log(message);
}
```

Рисунок 1.3 – Звичайний синтаксис написання функції та arrow function

Якщо функція складається лише з одного рядка, запис можна скоротити ще більше, усуваючи фігурні дужки:

```
let doLog = (message) => console.log(message);
```

В TypeScript кожен файл вважається окремим модулем. Для отримання доступу до класу з інших модулів, перед його оголошенням додається ключове слово *export* (рисунок 1.4).

```
export class Point {
    constructor(private x?: number, private y?: number) {
    }

    draw() {
        console.log('X: ' + this.x + ', Y: ' + this.y);
    }
}
```

Рисунок 1.4 – Оголошення класу в TypeScript

Також для скорочення обсягу коду створити поля класу можна шляхом додавання модифікатора доступу перед назвою параметра в конструкторі (рисунок 1.4). В такому випадку поле не треба окремо оголошувати, воно буде створене автоматично.

Отже, TypeScript має суттєві переваги порівняно із JavaScript. Мова пропонує багато способів скоротити обсяг коду, завдяки чому робить розробку швидшою та легшою, а код – більш лаконічним. Строга типізація дозволяє швидко виявити та виправити помилки.

1.3 Архітектура та основні поняття фреймворку Angular

1.3.1 Архітектура Angular

Стара версія Angular – AngularJS була заснована на паттерні MVC [9], який передбачає модель (Model) даних, представлення (View), тобто шаблон, через який відображаються дані користувачу та контролер (Controller), який відповідає за маніпуляцію даними та взаємодію між моделлю та представленням. Проте паттерн було дещо видозмінено для імплементації важливої характерної риси фреймворку – двосторонньої прив’язки даних. Недоліком стандартного MVC є те, що при внесенні змін у представлення, вони не оновлюються автоматично в моделі. Для оновлення повинен спрацювати метод контролера. В AngularJS було запроваджено механізм особливий механізм зв’язку між моделлю та представленням. Якщо дані змінюються в представленні, вони автоматично оновлюються в моделі і навпаки – таким чином функціонує механізм двосторонньої прив’язки (Two-Way Data Binding). Тому іноді паттерн AngularJS класифікують як MVVM (Model View ViewModel) або більш широко, як MV*.

Сучасний фреймворк Angular використовує ***компонентну архітектуру***. В ній все ще проглядаються риси MVC, проте не її вже не можна вважати дійсно імплементацією цього паттерну. Фреймворк надає чіткі вказівки щодо того, як має бути структурована програма. В ній вже немає контролерів, а основними

структурними одиницями є компоненти. Нижче будуть детально розглянуті найважливіші поняття фреймворку Angular.

1.3.2 Angular CLI

Angular CLI (Command-Line Interface) – інструмент для взаємодії із застосунком через командний рядок. Використовується для ініціалізації проєктів, створення компонентів та сервісів тощо. Простий командний інтерфейс позбавляє розробника необхідності вручну працювати зі складними конфігураціями та інструментами збірки, такими як TypeScript Compiler, WebPack та ін.

Нижче наведені декілька прикладів найбільш розповсюджених команд:

ng new <project-name> – створення проєкту;

ng serve – збірка та запуск проєкту;

ng g c <component-name> – створення нового компоненту;

ng g s <service-name> – створення сервісу.

Angular CLI є зручним та універсальним інструментом керування проєктом, дозволяє автоматично генерувати код та прискорює розробку.

1.3.3 Component

Component є основною структурною одиницею в фрейворку Angular. Компонент слугує для інкапсуляції:

1. даних;
2. HTML шаблону;
3. логіки, пов’язаної із представленням.

При створенні компоненту AngularCLI генерує 4 файли (рисунок 1.5). Файл із розширенням .ts визначає клас із властивостями, в яких зберігаються дані представлення, та методами, що відповідають за презентаційну логіку. Файл із розширенням .html є шаблоном з розміткою компонента. Цей шаблон здійснює обмін даними із класом компонента.

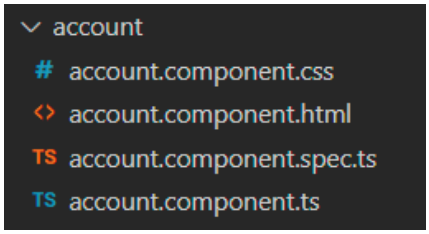

Рисунок 1.5 – Структура компоненту Angular

Файл з розширенням .css слугує для визначення стилів конкретно для даного компоненту (спільні стилі, якими можуть користуватись декілька компонентів одразу, визначаються окремо, в загальному файлі стилів для модулю або всього проєкту). Файл з розширенням .spec.ts призначений для написання unit тестів. Кожен проєкт Angular повинен мати принаймні один компонент (кореневий).

Компонент може представляти як сторінку застосунка, так і її частину. Кожен компонент може вміщувати інші компоненти. Таких рівнів вкладеності може бути скільки завгодно. Наочно використання вкладених компонентів демонструє рисунок 1.6.

Рисунок 1.6 – Використання вкладених компонентів в Angular

На рисунку 1.6 наводяться три рівні вкладеності компонентів на прикладі сторінки вебзастосунку з освітніми курсами. Кореневим компонентом є AppComponent (1), він складається з окремих компонентів: бокової панелі, панелі навігації та списку курсів (2), в свою чергу список курсів складається з карток з інформацією про курс (3). Тобто застосунок має деревовидну структуру компонентів з одним кореневим та декількома вкладеними.

Для рендерінгу одного компоненту всередні іншого необхідно всього лише додати тег дочірнього компоненту в шаблон батьківського. Для кожного компоненту Angular створюються власний тег, який можна використовувати так само, як теги HTML. Використання тегу компоненту side-menu у розмітці батьківського компонету наведено на рисунку 1.7.

```
<div>
  <h1>Test Your Brain</h1>
  <side-menu></side-menu>
</div>
```

Рисунок 1.7 – Використання вкладеного компоненту side-menu

Також Angular підтримує механізм для повторного використання тих самих компонентів в різних місцях проєкту (*reusability*). Батьківський компонент може легко передавати дані у дочірній та отримувати дані з нього за допомогою вхідних та вихідних властивостей (*input/output properties*). Схему обміну даних з дочірнім компонентом наведено на рисунку 1.8.

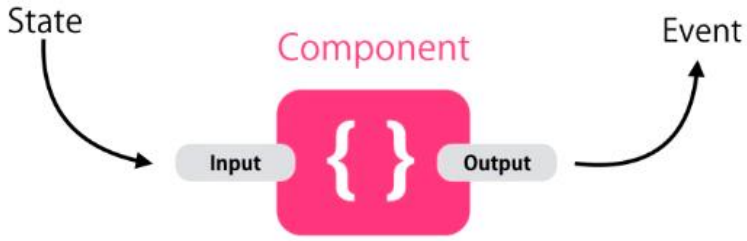

Рисунок 1.8 – Механізм обміну даних з компонентом за допомогою input та output properties

Розбиття сторінки на менші елементи робить код більш читабельним, полегшує супроводження. Компонентна архітектура Angular сприяє впровадженню принципу єдиної відповідальності (Single-Responsibility Principle) та запобігає дублюванню коду за рахунок повторного використання компонентів.

1.3.4 Module

Module – своєрідний контейнер для компонентів, сервісів та інших файлів.

В маленьких застосунках модуль може бути лише один – стандартний AppModule. Проте з розширенням програми виникає потреба структурувати компоненти. Компоненти поєднують у модуль за принципом спорідненості. Наприклад, може бути окремий модуль для функціоналу адміністрування, окремий – для взаємодії із користувачем, окремий – для обміну миттєвими повідомленнями. Компоненти в одному модулі спільно використовують контекст компіляції.Модулі, так само як компоненти, утворюють деревовидну структуру із єдиним кореневим та довільною кількістю дочірніх.

При створенні нових компонентів їх обов'язково треба зареєструвати у модулі. Інакше виникає помилка, бо Angular не зможе їх знайти. Для реєстрації компоненту треба додати його назву до масиву *declarations* в файлі визначення модулю, наприклад, app.module.ts. Для використання сервісів, їх необхідно реєструвати у масиві *providers*. Часто виникає необхідність імпортувати стандартні модулі Angular для можливості їх використання в проєкті. Наприклад, для здійснення запитів до сервера необхідний модуль HttpClientModule. Усі використовувані модулі додаються в масив *imports*. Всі метадані модуля задаються у декораторі @NgModule. Нижче наведено приклад визначення модулю в Angular:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule(
{ imports: [ BrowserModule ],
  providers: [ Logger ],
  declarations: [ AppComponent ],
  exports: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

1.3.5 Directive

Директиви є способом розширення стандартної розмітки HTML в Angular.

Директиви Angular починаються з префіксу ng. Використовуються вони в HTML шаблоні компонента.

За призначенням в Angular існує 3 види директив:

- components;
- attribute directives;
- structural directives.

Також усі директиви можна поділити на вбудовані та користувацькі.

Найпоширенішим типом є самі ***компоненти***. Вони розширюють мову HTML за рахунок створення тегів із власною назвою. (Приклад було наведено раніше, на рисунку 1.10). Назву тегу, що використовується в розмітці можна задати як текст селектору в декораторі @Component кожного компоненту (рисунок 1.9).

```
@Component({
  selector: 'side-menu',
  templateUrl: './side-menu.component.html',
  styleUrls: ['./side-menu.component.css']
})
export class SideMenuComponent implements OnInit {···
}
```

Рисунок 1.9 – Задання селектору компоненту для розширення HTML

Директиви атрибутів змінюють зовнішній вигляд або поведінку елемента. Ці директиви широко використовуються для прив’язки даних або стилів. Найбільш розповсюдженими є директиви є наступні директиви:

ngClass – додає або видаляє набір CSS класів;

ngStyle – додає або прибирає набір стилів HTML;

ngModel – використовується для двонаправленої прив’язки даних до елементу форми HTML.

Приклад задання CSS класу за допомогою директиви атрибутів наведено на рисунку 1.10. Колір фону елементу встановлюються в залежності від значення поля isSpecial в класі компоненту.

```
<div [ngClass]="isSpecial ? 'bg-red-500' : 'bg-gray-600'">
    My element
</div>
```

Рисунок 1.10 – Використання директиви атрибутів

Структурні директиви дозволяють керувати структурою DOM дерева: додавати або видаляти елементи тощо.

Серед структурних директив найбільш часто використовуються директиви ***ngIf** та ***ngFor**. Перша дозволяє приховати елемент, якщо він не існує або якщо заданий вираз повертає false, та показати елемент при значенні true. Друга дозволяє реалізувати цикл. Приклад використання *ngFor приведено нижче (рисунок 1.11).

```
<ul>
    <li *ngFor="let test of tests">{{ test.name }}</li>
</ul>
```

Рисунок 1.11 – Використання структурної директиви

На рисунку 1.11 наводиться код для виведення назви кожного тесту у масиву tests (визначеному у класі компоненту).

1.3.6 Service

Сервіси в Angular застосовуються для спільного використання декількома компонентами даних або певного функціоналу.

Сервіси допомагають забезпечити виконання двох важливих принципів об’єктно-орієнтованого програмування та дизайну:

- DRY (Don’t Repeat Yourself);
- SRP (Single Responsibility Principle).

Принцип DRY українською означає «не повторюйся». Часто компоненти повинні працювати із тими самими даними, наприклад, такими, що надходять з сервера. Якби не було сервісів, кожен компонент мав би виконувати однакові запити до сервера. Очевидно, що такий код був би складний в підтримці та мав би необґрунтовано великий обсяг через постійне дублювання.

Принцип SRP є одним з принципів SOLID. Він наголошує на необхідності єдиної відповідальності. Завданням компоненту має бути забезпечення логіки представлення, якби компонент мав ще й здійснювати запит до сервера, принцип SRP було б порушено. Сервісам делегуються завдання, які не мають виконуватися безпосередньо в компоненті.

Основною задачею сервісів є забезпечення організації та спільного використання бізнес-логіки, даних та функцій різними компонентами.

Сервіси є класами з декоратором @Injectible (рисунок 1.12). Цей декоратор сповіщає про те, що сервіс може бути впроваджений в інший клас у вигляді залежності. Кожен сервіс існує в єдиному екземплярі – таким чином реалізується паттерн проєктування Singleton. Об’єкт сервісу передається у компонент як параметр конструктора, тобто відбувається Dependency Injection. Після цього компонент може користуватися функціоналом, який надає сервіс.

```
@Injectable({
  providedIn: 'root'
})
export class UtilService {

  constructor(private service: TestsService) {
  }
}
```

Рисунок 1.12 – Оголошення та впровадження сервісу в якості залежності

Сервіси також можуть користуватися послугами інших сервісів, для цього їх треба так само передати в клас через конструктор завдяки механізму впровадження залежності. Рисунок 1.12 показує сервіс UtilService, який користується послугами TestsService, впровадженого в якості залежності.

1.3.7 Підходи до створення форм: Template-Driven та Reactive Forms

Ключовою складовою застосунку, що розробляється, є отримання даних від користувача. Angular підтримує два підходи до створення форм:

- Template-Driven Forms;
- Reactive Forms.

В Angular кожне поле введення у формі представляє об’єкт FormControl. Ці об’єкти об’єднуються у групи, представлені об’єктами FormGroup. Кожна форма представлена об’єктом FormGroup та кожен FormControl має належати до групи (рисунок 1.13).

Рисунок 1.13 – Об'єкти-складові форми в Angular

Об’єкти FormControl і FormGroup дозволяють відслідковувати стан кожного окремого поля введення та форми в цілому. Обидва класи наслідують клас AbstractControl та однаково надають набір властивостей, серед який наступні:

- value (значення поля або полів введення);
- touched (користувач взаємодіяв із полем або формою – клікнув на ньому або вводив інформацію);
- untouched (користувач не взаємодіяв із полем);
- dirty (дані в полі або формі були змінені користувачем – і ще не збережені на сервері);

- pristine (дані в полі або формі ще не були змінені користувачем) ;
- valid (дані коректні);
- errors (список помилок).

В різних ситуаціях корисно мати доступ до значень цих властивостей окремого елементу або форми в цілому. Наприклад, якщо потрібно зробити неактивною кнопку надсилання форми, якщо вона неправильно заповнена, доцільно використати властивість valid об’єкту FormGroup. Це позбавляє необхідності перебирати всі елементи форми та перевіряти їх стан. З іншого боку, якщо необхідно вивести помилки валідації для конкретного поля, варто скористатися властивостями об’єкту FormControl.

Існують два способи створити об’єкти, що представляють форму:

1. Використання спеціальних директив у шаблоні. В цьому випадку Angular автоматично створює об’єкти на основі вказаних директив. Форми, створені в такий спосіб, називають **Template-Driven** (тобто створені за шаблоном).
2. Створення об’єктів FormControl і FormGroup в явний спосіб в коді класу компонента. Такі форми називаються **Reactive Forms**.

Ці два види форм мають свої особливості, викладені у таблиці 1.3.

Таблиця 1.3 - Порівняння Template-Driven та Reactive Forms

Template-Driven Forms	**Reactive Forms**
Легкі в створенні, зрозумілі, потребують менше коду	Більше контролю над логікою валідації та структурою
Базова валідація	Можливість unit-тестування логіки валідації
Добре підходять для простих форм	Добре підходять для складних форм

1.4 Вибір засобів стилізації. Tailwind CSS

Існують три основні підходи до стилізації вебзастосунків:

– засобами чистого CSS;

– за допомогою компонентних фреймворків стилів;

– за допомогою функціональних фреймворків стилів (utility frameworks).

Кожен зі способів має свої особливості, переваги та недоліки.

Головною перевагою використання **чистого CSS** є повний контроль над зовнішнім виглядом елементів інтерфейсу. Однак вагомим недоліком є трудомісткість процесу задання стилів та необхідність написання великого обсягу коду. Це сповільнює процес розробки, тому все частіше перевага надається використанню фреймворків.

Нижче наведено код стилізації кнопки за допомогою CSS:

```
.button {
  background-color: #4CAF50;
  border: none;
  color: white;
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
}
```

Рисунок 1.14 – Кнопка, стилізована засобами CSS

Компонентні фреймворки стилів надають розробнику заздалегідь визначені компоненти, такі як кнопка, випадаючий список або навігаційна панель. На сьогоднішній день найпопулярнішим фреймворком цієї категорії є Bootstrap. Завдяки наявності великої кількості стандартних компонентів, фреймворк є ідеальним для швидкої розробки. Базова стилізація вимагає дуже невеликого обсягу коду. Проте суттєвим недоліком є те, що Bootstrap має обмежені можливості персоналізації вигляду елементів GUI. Сайти, що використовують Boostrap, часто мають дуже схожий вигляд.

Приклад стилізації кнопки за допомогою Bootstrap:

```
<button type="button" class="btn btn-primary">Primary</button>
```

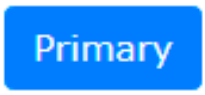

Рисунок 1.15 – Кнопка, стилізована засобами Boostrap

Фреймворки на основі utility класів не надають готових компонентів, а визначають перелік класів, кожен з яких надає певну властивість елементу розмітки (наприклад, його розміри, колір, розташування). Як правило класи є універсальними і не прив'язані до конкретного типу елементу. Завдяки цьому можливо створити персоналізований дизайн будь-якого компоненту, лише додаючи до нього класи в шаблоні HTML. Найбільш популярним фреймворком даного типу є ***Tailwind CSS***.

Використання untility фреймворку є компромісним варіантом між гнучкістю та швидкістю розробки, тому саме Tailwind CSS обрано для стилізації застосунку, що розробляється.

На рисунку 1.16 наведені приклади класів, що використовуються в Tailwind, він демонструє, що фреймворк значно скорочує синтаксис стилізації, чим пришвидшує розробку.

Class	Properties
p-0	padding: 0px;
px-0	padding-left: 0px; padding-right: 0px;
py-0	padding-top: 0px; padding-bottom: 0px;
pt-0	padding-top: 0px;

Рисунок 1.16 – Класи Tailwind CSS

Приклад стилізації кнопки за допомогою Tailwind:

```
<button class="bg-violet-500 rounded shadow-md shadow-violet-500/50
    text-white py-2 px-4">Button</button>
```

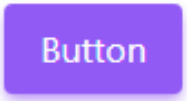

Рисунок 1.17 – Кнопка, стилізована засобами Tailwind CSS

Принцип роботи Tailwind наступний (рисунок 1.18): в проєкті створюється вхідний файл, в який імпортуються компоненти фреймворку. Також у цьому файлі можна писати власний код CSS. Після цього Tailwind оброблює цей вхідний файл та під час збірки генерує код CSS у вихідному файлі, з яким пов’язаний файл HTML.

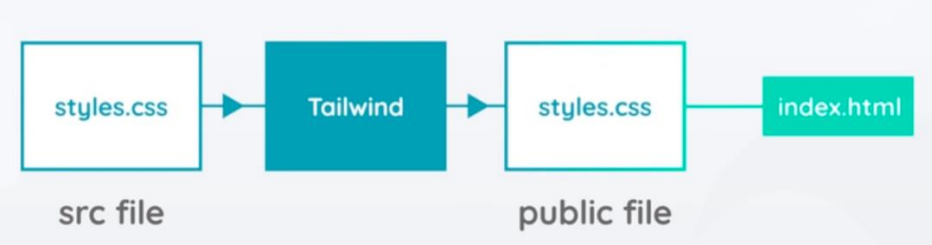

Рисунок 1.18 – Схема роботи Tailwind CSS

Tailwind відслідковує зміни в файлі HTML (додані utility класи) та оновлює вміст вихідного файлу стилів.

Tailwind легко інтегрується з Angular. Для більшої зручності роботи можна також встановити плагін Tailwind CSS Intellisense для редактора VS Code.

1.5 Angular Material

Angular Material є бібліотекою компонентів UI, створених за допомогою TypeScript та Angular.

Компоненти Angular Material мають наступні переваги:

- мають сучасний та естетично приємний дизайн;
- компоненти інтернаціоналізовані (доступні на різних мовах);
- мають простий та лаконічний API;
- ретельно протестовані (за допомогою unit та інтеграційного тестування);

– підтримують можливість персоналізації зовнішнього вигляду;

– швидкі;

– мають якісну документацію.

Компоненти були розроблені спеціально для фреймворку Angular, тому ці дві технології чудово працюють разом.

Angular Material не є вбудованим модулем в фреймворку, це окрема технологія, для використання якої треба спершу встановити її у проєкті. Кожен компонент міститься в окремому модулі. Для використання компонента в проєкті Angular необхідно:

1. імпортувати його у файлі app.module.ts;
2. додати його тег в HTML шаблон компонента;
3. за необхідності виконати прив’язку даних або подій.

Нижче наведено приклад використання компоненту Angular Material checkbox в шаблоні HTML:

```
<mat-checkbox [value]="" [checked]="isChecked"
    (change)="onCheckboxClick($event)">Check me!</mat-checkbox>
```

З коду видно, що в компоненти можна передавати дані за допомогою вхідних властивостей (input properties). Можна також додавати обробку подій, пов’язаних із компонентом (виклик функції при зміні значення).

Назви усіх компонентів Angular Material починаються із префіксу mat, завдяки чому їх легко впізнати у коді. На момент написання роботи Angular Material пропонує 36 різних типів компонентів. Серед них доступні зокрема: datepicker (елемент для вибору дат), paginator (елемент для розбиття на сторінки), випадаючий список, перемикач, tooltip (спливаюча підказка).

Серед компонентів є як базові, так і більш складні, реалізація котрих самостійно була б трудомісткою. Також Angular Material підтримує анімації для компонентів, які роблять інтерфейс більш цікавим та інтерактивним. Стилі компонентів в Angular Material задається згідно із темою (Theme). Є декілька

вбудованих варіантів поєднання кольорів. Тему можливо задати самостійно або налаштувати за допомогою змінних SASS.

Для стилізації деяких складних елементів інтерфейсу виключно за допомогою Tailwind CSS знадобиться занадто багато коду, що робить програму перевантаженою та менш читабельною. Отже, для створення певних елементів інтерфейсу, таких як випадаючі списки або datepicker, будуть використані компоненти Angular Material. Додатковою перевагою є їх легка інтеграція в проєкт та протестованість.

1.6 Створення Telegram-ботів

Ботів для Telegram можна створювати на багатьох мовах програмування: C#, C++, Java, Kotlin, PHP, Ruby та ін. Проте однією з найбільш популярних при розробці ботів залишається мова Python. Мова має наступні переваги:

- проста у вивченні;
- особливості синтаксису сприяють написанню лаконічного і читабельного коду;
- велика спільнота розробників;
- абсолютно все в Python є об’єктом в сенсі ООП, але об’єктно-орієнтований підхід не нав’язується програмісту.

Найбільшою перевагою Python є стислість синтаксису. Тоді як мови Java та C# для написання найбільш базової програми “Hello World” використовують 4-5 лінійок коду, в Python необхідна лише одна. Завдяки цій особливості на Python можна дуже швидко здійснювати розробку. Також мова підтримує велику кількість бібліотек для створення Telegram-ботів.

Серед найбільш популярних бібліотек для розробки ботів на мові Python:

- aiogram;
- python-telegram-bot;
- pyTelegramBotAPI;
- telethon.

Серед розробників найкращою вважається aiogram, особливо для великих проєктів із довгою підтримкою. Aiogram – популярний повністю асинхронний фреймворк для створення ботів. Основним його недоліком у порівнянні із конкурентами, зокрема python-telegram-bot, є відносна складність вивчення. Проте цей недолік компенсується високою функціональністю. Тому для написання боту обрано саме цю технологію.

Для створення будь-якого Telegram боту необхідно:

1. Знайти в месенджері Telegram бот BotFather – це офіційний бот для роздачі ключів API.
2. Виконати команду /newbot.
3. Ввести ім’я боту, необов’язково унікальне.
4. Обрати ім’я користувача для боту, обов’язково має бути унікальне та закінчуватися на *bot.*

Після цього бот створюється в системі і його вже можна знайти в месенджері. BotFather надає розробнику ключ API, який має бути використаний у коді для зв’язку з ботом.

Висновки до розділу 1

В ході написання першого розділу індивідуальної частини ККРБ було розглянуто підходи до створення front-end частини застосунку. Здійснено та обґрунтовано вибір технологій розробки.

Front-end застосунку буде написано на базі фреймворку Angular, при розробці сторінок буде задіяно деякі компоненти Angular Material. Для стилізації буде використано фреймворк стилів Tailwind CSS. Для розробки боту для месенджера Telegram буде використано мову Python та фреймворк aiogram.

У розділі описано основні поняття, структурні компоненти та принципи роботи технологій, задіяних у розробці програмного забезпечення для автоматизації тестування знань.

2 ПРОГРАМНА РЕАЛІЗАЦІЯ

2.1 Розробка графічних інтерфейсів користувача

2.1.1 Принципи вебдизайну

Дизайн користувацького інтерфейс має ефективно виконувати своє призначення: надавати користувачу зручний та зрозумілий спосіб взаємодії із логікою вебзастосунку, а також зацікавлювати користувача та втримувати його увагу під час роботи із сайтом. Існують кілька ключових факторів в дизайні інтерфейсів, які впливають на те, як він буде сприйнятий. Забезпечення доброго користувацького досвіду передбачає оптимізацію дизайну для зручності використання, що включає спосіб організації елементів в інтерфейсі та його естетику. Нижче розглянемо основні аспекти та принципи побудови GUI.

Відповідність цілям та призначенню. Застосунок має відповідати потребам користувачів. При дизайні кожної сторінки та вебсайту в цілому важливо чітко визначити призначення кожної сторінки та елементу. Це допоможе відповідним чином розташувати функціонал на сторінках та обрати найбільш оптимальну структуру сайту. Логічність структури дозволяє користувачу легко зорієнтуватися на сайті, запобігає наявності зайвих елементів.

Правильний добір кольорів. Кольори мають змогу викликати емоційні реакції у користувача та сильно впливають на його сприйняття застосунку. Важливо дібрати палітру, що відповідає бренду. Варто обмежити вибір кольорів до п’яти. Також треба подбати про достатню контрастність між текстом та кольором фону. Колір фону зазвичай обирають світлих або темних тонів, а для елементів інтерфейсу, таких, як кнопки, обирають більш контрастні або яскраві кольори. Для створення акцентів та привернення уваги користувача добре працюють комплементарні кольори, тобто такі, що знаходяться на навпроти одне одного на колірному колі [2].

Існують програмні продукти, які полегшують добір кольорової палітри. Прикладами є сайт FlatUIColors, що пропонує вручну підібрані варіанти

поєднання кольорів, та сервіс HtmlColorCodes, де можна обрати колір з бібліотеки, декількох палітр або довільний колір спектру та одразу отримати його код для використання у програмі.

Приємні поєднання кольорів покращують настрій користувача та позитивно впливають на його відношення до програмного продукту.

Шрифти. Важливу роль у сприйнятті інформації має шрифт. Він має бути розбірливим, щоб не ускладнювати розуміння, літери повинні бути достатнього розміру. На вебсайті не варто використовувати більше трьох різних шрифтів.

Зображення. Якщо на сайт використовуються графічні елементи, такі як фотографіїї та ілюстрації, потрібно подбати про їх якість, відповідність бренду та загальній кольоровій гамі сайту. Високоякісні зображення формують у користувача враження професіоналізму та сприяють довірі до бренду. У випадку іконок важливо забезпечити їх розбірливість, простоту та інтуїтивну зрозумілість.

Навігація. Якісна та зрозуміла навігація на сайті є ключем для утримання користувачів. Якщо навігація є незрозумілою, користувачі з великою вірогідністю вийдуть та знайдуть необхідну їм інформацію чи послуги на іншому сайті. Навігація має бути простою, інтуїтивно зрозумілою та послідовною на кожній сторінці сайту.

F-подібна організація інтерфейсу. Найпоширеніший спосіб, яким користувачі сканують графічний інтерфейс нагадує формою літеру F. Дослідження з відстеження руху очей [4] показують, що найбільше уваги люди звертають на те, що розташоване на у верхній та лівій частинах екрана. Такий розподіл уваги відповідає західній моделі читання (зліва направо та зверху вниз). Тобто при розробці інтерфейсу варто розташувати найбільш важливі елементи (панель навігації, назва компанії тощо) саме в цих областях екрана.

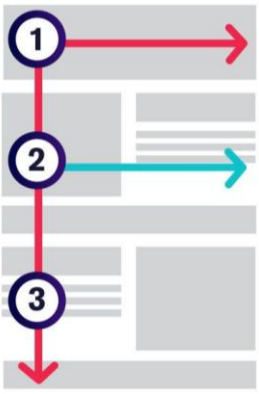

Рисунок 2.1 – F-подібна схема сканування користувачем інтерфейсу

Проте останні дослідження виявили, що добрий візуальний дизайн може змінити паттерн сприйняття [5]. Варто лише пам’ятати про те, що люди мають тенденцію звертати менше уваги на елементи справа та внизу екрану, тому при такому розташуванні їх потрібно максимально виділити за допомогою кольору або других засобів.

Візуальна ієрархія. Для полегшення орієнтування в інтерфейсі елементи слід виділяти та розташовувати відповідно до їх важливості. Це можна забезпечити за рахунок величини, кольору, контрасту, візуальних елементів (іконки, зображення), відступів, шрифтів тощо. Ступінь привернення уваги елементом має бути пропорційним його важливості для користувача. Доцільним є також використання візуальних ефектів для позначення інтерактивних елементів інтерфейсу, таких як кнопки, гіперпосилання, поля введення тощо. Це можна реалізувати зокрема за допомогою зміни кольору або величини, появи підказок при наведенні курсора на елемент. Подібний «відгук» елементів інтерфейсу допомагає користувачу швидко зрозуміти, як взаємодіяти з ними.

Представлення на основі Grid. Використання Grid допомагає структурувати елементи на сторінці за допомогою розподілу елементів у сітці з колонками та чітким поділом на секції. Grid є гнучким способом організації інтерфейсів та дозволяє легку адаптацію під пристрої із різними розмірами екрану. За допомогою структури Grid можна створити естетично привабливі інтерфейси, що мають збалансований та впорядкований вигляд.

Адаптивність інтерфейсу. На сьогоднішній день більшість користувачів переглядають вебсторінки зі смартфонів та інших девайсів [1] Тому важливо приділити увагу коректному відображенню інтерфейсу на пристроях із різною величиною екранів.

2.1.2 Макети графічного інтерфейсу

На основі викладених принципів вебдизайну побудовані макети інтерфейсів для застосунку, що розроблюється. Макети створені за допомогою програмного продукту Figma, який дозволяє створювати детальні та реалістичні ескізи інтерфейсів.

Фон виконано темних тонах. Різні відтінки кольору допомагають візуально розділити інтерфейс на бокову панель навігації та робочу область (рисунок 2.2). Розташування елементів відповідає принципу літери F: навігаційна панель знаходиться зліва, назва застосунку – у лівому верхньому куті. Тобто найважливіші елементи розміщенні в області найбільшого фокусу уваги користувача. Нижче наведено декілька найбільш цікавих макетів, використаних при проєктуванні.

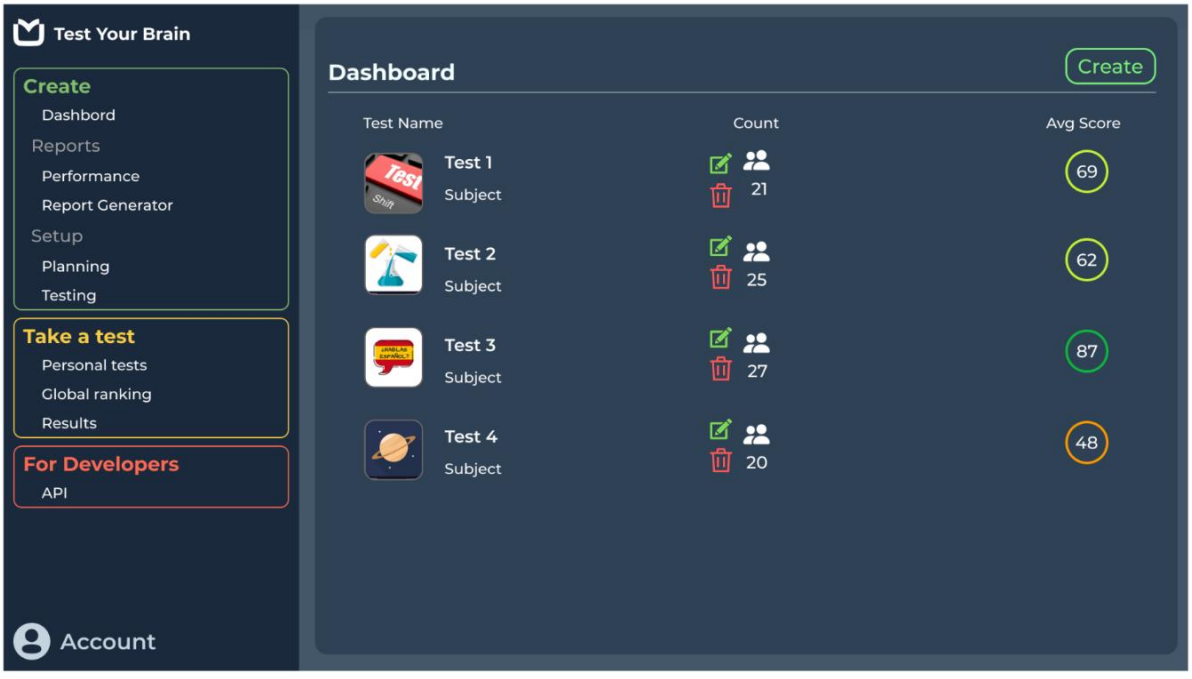

Рисунок 2.2 – Макет панелі керування тестами

Ієрархія важливості забезпечується використанням тексту різного розміру: найбільшою є назва застосунку, меншою – назва поточної сторінки.

Панель навігації розділена на три основні секції: функціонал створення тестів, проходження тестів, секція для розробників. Області відокремлені візуально за допомогою кольорів. Нижче знаходиться секція Account, за допомогою якої можна перейти на сторінку з обліковим записом. Секція розміщена нижче та відрізняється за стилем від решти навігаційних посилань, щоб логічно відокремити її від основного функціоналу застосунку.

В панелі керування можна переглянути список тестів, кількість людей, що пройшли тест та середню оцінку за тест (рисунок 2.3). Колір кола, в якому відображається оцінка, залежить від балу: зелені відтінки для більш високих результатів та оранжеві – для низьких. Використання кольорів забезпечує візуалізацію результату та робить GUI більш цікавим.

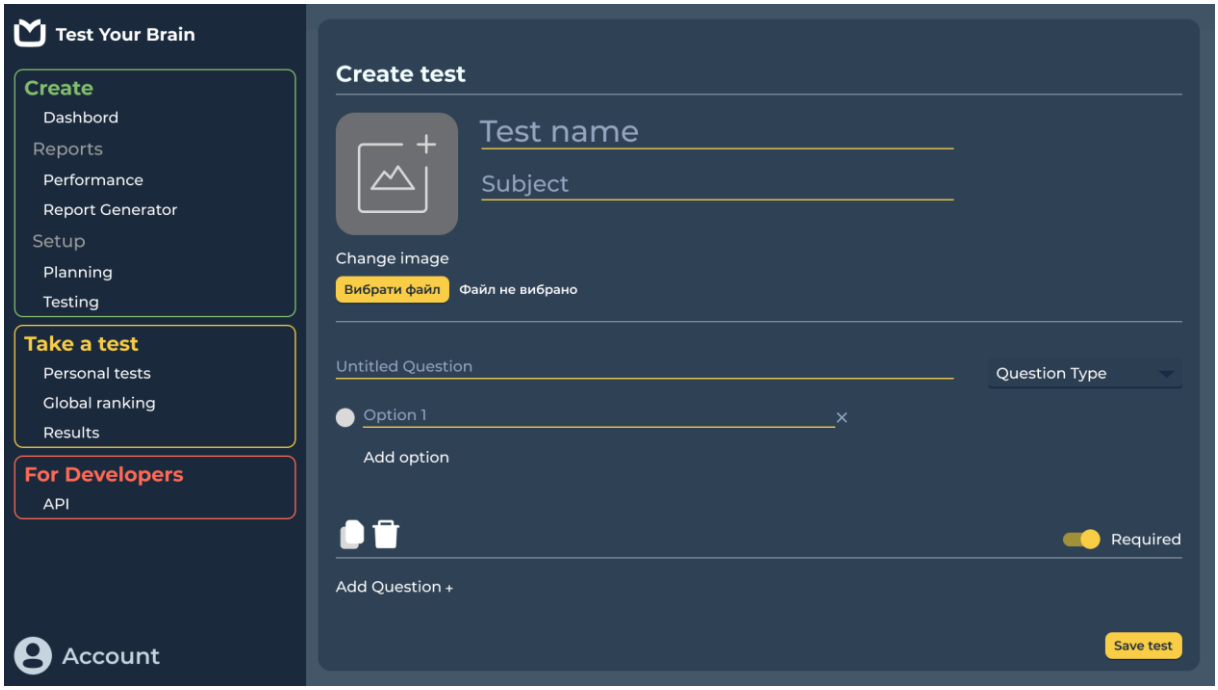

Рисунок 2.3 – Макет сторінки створення тесту

На сторінках використовуються загальноприйняті іконки на позначення дій та сутностей. Це позбавляє інтерфейс зайвого тексту, робить його більш візуально привабливим.

Рисунок 2.4 – Макет сторінки з рейтингом користувачів

Отже, макети розроблені у відповідності до розглянутих принципів вебдизайну. Макети використовуються як орієнтир для подальшого створення інтерфейсу. Зовнішній вигляд інтерфейсів може зазнати незначних змін, якщо це вважатиметься за необхідне для покращення користувацького досвіду.

2.2 Структура застосунку та опис класів

Застосунок є односторінковим. Умовно його можна розділити на бокову панель навігації, яка постійно знаходиться на екрані та на робочу область, в якій відображається вміст усіх інших компонентів, із якими взаємодіє користувач. За кожну сторінку, що відображається в робочій області, відповідає окремий компонент. У таблиці 2.1 наведено опис компонентів, реалізованих у застосунку.

Таблиця 2.1 – Опис основних компонентів системи

Компонент	**Призначення**
app	Основний компонент односторінкового застосунка. В ньому відображаються інші компоненти в залежності від дій користувача.

Продовження таблиці 2.1

Компонент	**Призначення**
side-menu	Бокове меню застосунка. Завжди присутнє на сторінці, слугує для навігації на сайті.
dashboard	Панель керування. Слугує для перегляду списку створених користувачем тестів, містить інформацію про кількість проходжень тестів та середній бал. Дозволяє перейти до створення або редагування тесту, видалити тест.
create	Форма створення нового тесту.
edit	Форма редагування тесту.
performance	Відображення статистики про проходження тестів у вигляді графіка з динамікою зміни середнього балу. Можливо обрати, для яких тестів висвітлюється графік та за який часовий проміжок.
report-generator	Надання звітів. Вибір тесту зі створених поточним користувачем та перегляд результатів інших користувачів, що пройшли цей тест.
planning	Налаштування та планування тестів. Керування статусом: тест загальнодоступний або з обмеженим доступом. Вибір користувачів, яким доступний тест. Задання часу, коли тест доступний.
preview	Попередній перегляд тестів. Проходження власних тестів без збереження результату для перевірки коректності заповнення даних.
personal-tests	Список тестів, доступ до яких надано особисто користувачу. Зі сторінки можна перейти до проходження тесту.
pass	Проходження тесту.
testing-result	Результат, що відображається після завершення тестування.

Кінець таблиці 2.1

Компонент	**Призначення**
global-ranking	Загальний рейтинг тестів та користувачів.
results	Результати та дати проходження тестів користувачем.
dev-api	Інформація для розробників.
account	Обліковий запис коритувача: перегляд та керування власними даними.
change-password	Зміна паролю.
login	Вхід у систему.
register	Створення облікового запису.

Також в застосунку є перелік допоміжних компонентів, що використовуються для уникнення дублювання коду та покращення читабельності. В ці компоненти винесена логіка, стилі та розмітка окремих елементів, що використовуються у декількох інших компонентах. Таким чином виконується принцип DRY. Опис компонентів наведений у таблиці 2.2.

Таблиця 2.2 – Допоміжні reusable компоненти

Компонент	**Призначення**	**Місце використання**
page-title	Відображення заголовку сторінок у робочій області.	в усіх компонентах сторінок робочої області, окрім testing-result
count	Відображення кількості користувачів в елементі з іконкою.	в dashboard, global-ranking

Кінець таблиці 2.2

Компонент	**Призначення**	**Місце використання**
score	Відображення балів за тест, вибір кольору елементу відповідно до висоти результату.	в dashboard, report-generator, global-ranking, results, testing-results
test-count	Відображення кількості тестів в елементі з іконкою.	в global-ranking
test-details	Відображення основних даних про тест: назви, теми, зображення.	в dashboard, planning, preview, personal-tests, global-ranking, results
user-details	Відображення даних про користувача: ім’я в системі, ім’я та прізвище, аватар	в report-generator, global-ranking

Деякий функціонал винесено в сервіси, що знаходяться у папці services та можуть використовуватися декількома компонентами одразу. Сервіс **tests** відповідає за обмін даними про тести із сервером та реалізацію CRUD-операцій за рахунок http-запитів типу get, post, put та delete. Аналогічно, сервіс **users** відповідає за обмін даними про користувачів системи, а сервіс **stats** – статистичними даними про проходження тестів. Оскільки в декількох місцях застосунку необхідно обробляти завантаження зображення, цей функціонал винесено у сервіс **image**. Сервіс слугує для конвертації зображення в base-64 (в цьому форматі зображення потім передається на back-end та зберігається у БД).

Для створення динамічних форм за допомогою підходу Reactive Forms використовується сервіс **util**. У ньому програмно створюються моделі форм за допомогою FormBuilder, впровадженого в якості залежності. Готові форми передаються в CreateComponent та EditComponent. Це допомагає відокремити досить обширну логіку створення форми від її використання безпосередньо в

компоненті. Код генерації динамічної форми редагування тесту з декількома рівнями вкладеності заслуговує особливої уваги і наведений у додатку А.

Контроль доступу до сторінок реалізується за допомогою **auth-guard** – сервісу, що реалізує інтерфейс canActivate. Якщо користувач авторизований, він має доступ до усіх сторінок застосунку, якщо ні – лише до деяких. При спробі перейти на недоступну сторінку відбувається перенаправлення до форми авторизації.

2.3 Графічний інтерфейс користувача

Авторизація

Для входу в систему використовуються форми авторизації та реєстрації користувача. При реєстрації користувач повинен надати адресу електронної пошти, ввести та підтвердити пароль, а також придумати собі псевдонім у системі. При авторизації достатньо лише ввести електронну адресу та пароль. Опція «Запам’ятати мене» призначена для легшого входу в обліковий запис.

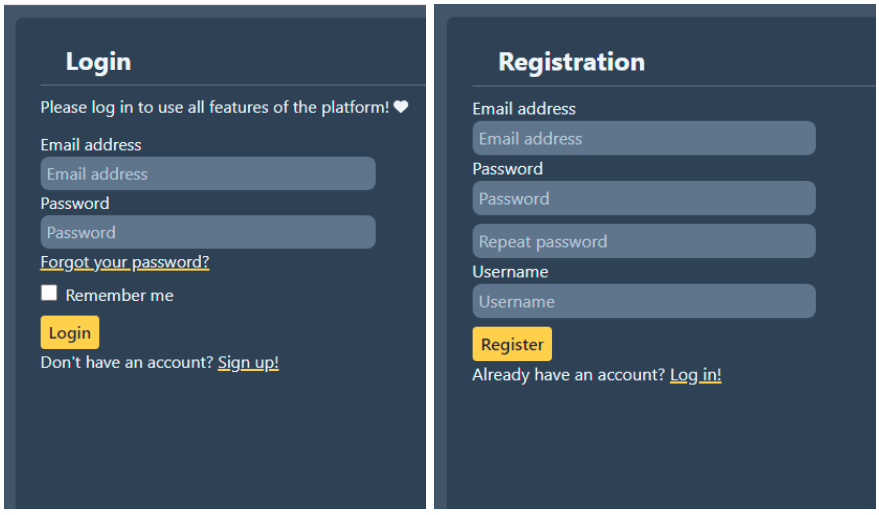

Рисунок 2.5 – Форми авторизації та реєстрації

Форма авторизації вміщує посилання для переходу до форми реєстрації для нових користувачів. Аналогічно можна перейти від форми реєстрації до авторизації, якщо користувач вже має обліковий запис.

Обліковий запис користувача

Після реєстрації користувач системи має змогу надати додаткову інформацію про себе на сторінці свого облікового запису: обрати країну, ввести ім’я та прізвище. Ця інформація може бути корисною для користувачів, яким важливо знати, хто проходив тест (наприклад, якщо система використовується вчителем для перевірки знань своїх учнів).

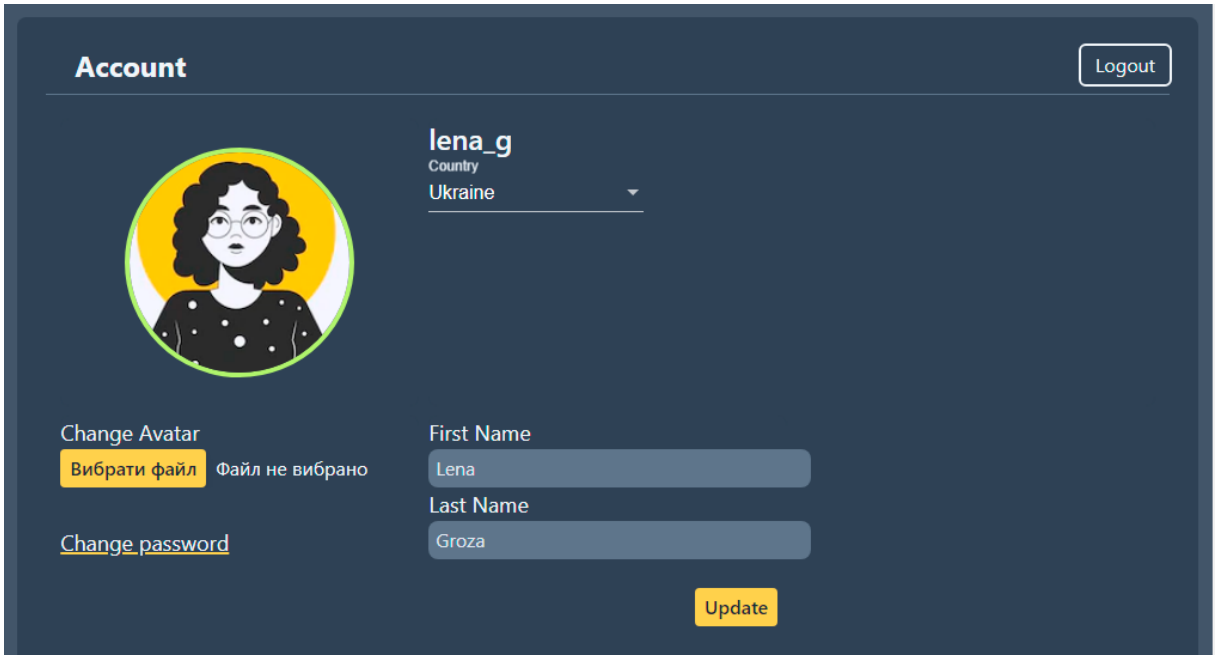

Рисунок 2.6 – Сторінка облікового запису користувача

Окрім цього користувач може додати аватар, змінити пароль або свій псевдонім у системі. За допомогою кнопки у правому верхньому куті можна вийти з облікового запису.

Панель навігації

Після входу в систему користувач має доступ до усіх її функцій через навігаційну панель в лівій частині екрану (рисунок 2.6).

Панель має три секції:

1. Створення тестів. Через сторінку «Dashboard» можна створювати тести, редагувати та видаляти їх, переглядати список створених тестів. Підрозділ «Reports» надає інформацію про результати проходження тестів користувачами

та дозволяє згенерувати звіт. Підрозділ «Setup» відповідає за статус тесту, дозволяє запланувати час, коли тест буде доступний для проходження, та переглянути створений тест для визначення зручності проходження.

2. Проходження тестів. Сторінка «Personal tests» показує тести, доступ до яких був наданий користувачеві. Сторінка «Results» надає інформацію про результати пройдених користувачем тестів. Сторінка «Global Ranking» показує інформацію про найбільш успішних користувачів системи (за кількістю та результатом пройдених тестів).

3. Для розробників. Ця секція вміщує API для розробників для можливості налаштовувати тести згідно зі своїми потребами.

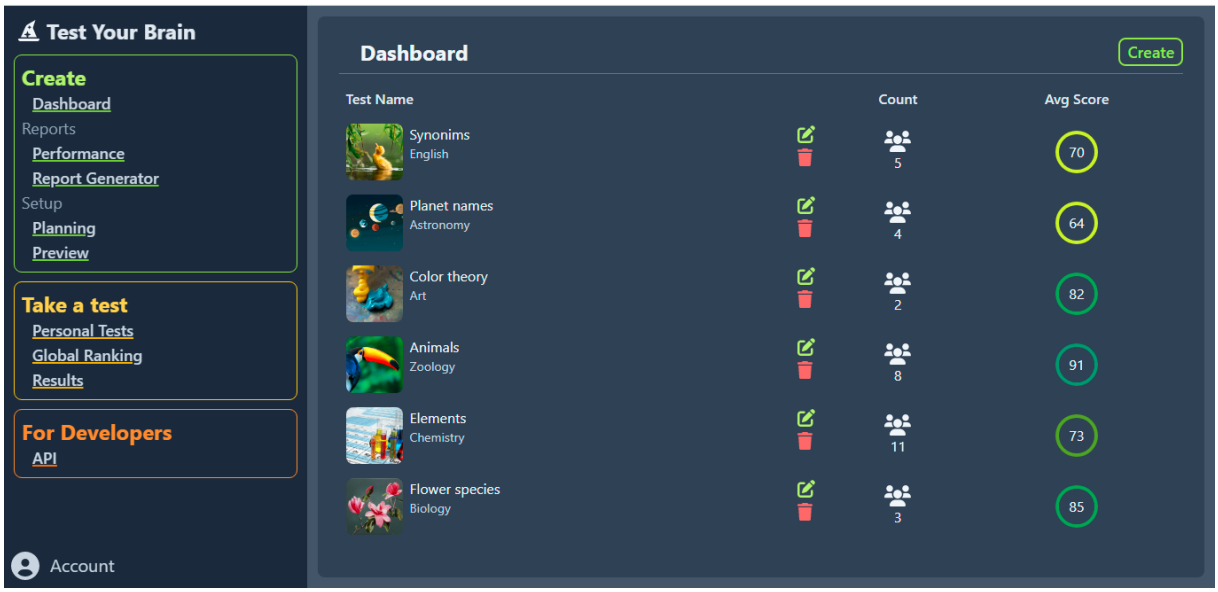

Рисунок 2.7 – Головний екран (бокове меню та панель керування)

Панель управління даними про тести

Ця панель використовується для перегляду даних про створені тести та керування ними: створення, редагування та видалення. Також ця сторінка надає інформацію про кількість людей, що пройшли даний тест та їх середню оцінку за нього (рисунок 2.7). Інформація про тести представлена у вигляді таблиці. Для відображення іконок на сторінці використано FontAwesome. Нижче наведено код сервісу, що відповідає за CRUD-операції над тестами:

```
import { Injectable } from '@angular/core';
```

```
import { HttpClient } from '@angular/common/http';
import { Test } from '../entities/test.model';
@Injectable({
  providedIn: 'root'
})
export class TestsService {
  private testsUrl = "http://localhost:3000/tests"; //base URL
  headers = { 'content-type': 'application/json'};
  constructor(private http: HttpClient) { }
  getTests(){
    return this.http.get<Test[]>(this.testsUrl); }
  getTest(id: number){
    return this.http.get<Test>(this.testsUrl+"/"+id);
  }
  addTest(test: Test){
    const body=JSON.stringify(test);
    this.http.post(this.testsUrl, body,{'headers':this.headers})
      .subscribe(response => console.log(response));
  }
  deleteTest(id: number){
    return this.http.delete(this.testsUrl + '/' + id);
  }
  updateTest(test: Test){
    const body=JSON.stringify(test);
    this.http.put(this.testsUrl +"/"+test.id, body,{'headers':this.headers})
      .subscribe(response => console.log(response));
  }
}
```

Сторінки створення та редагування тестів

Створення тесту відбувається через наступну форму (рисунок 2.8).

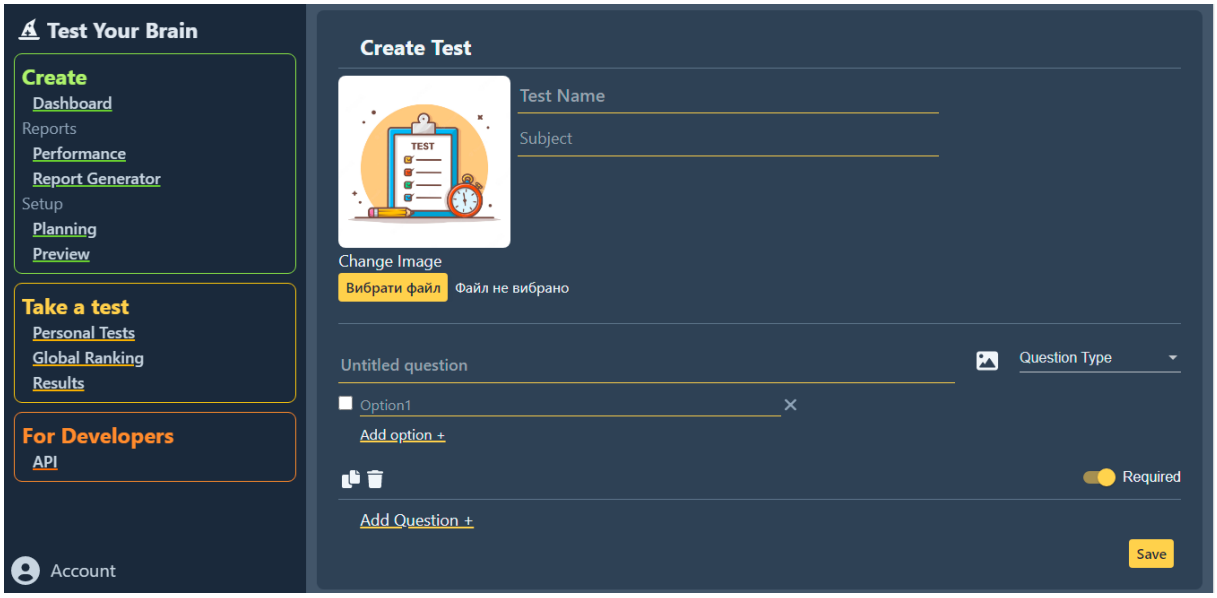

Рисунок 2.8 – Сторінка створення тестів

Тест має наступні атрибути: назва, тема, зображення-іконка та список питань. Кожне питання має власні властивості: може бути обов'язковим або ні (встановлюється за допомогою перемикача «Required»), мати одну або більше правильних відповідей. Питання та відповіді у формах динамічно додаються та видаляються. Замість зображення за замовчуванням можна завантажити власне.

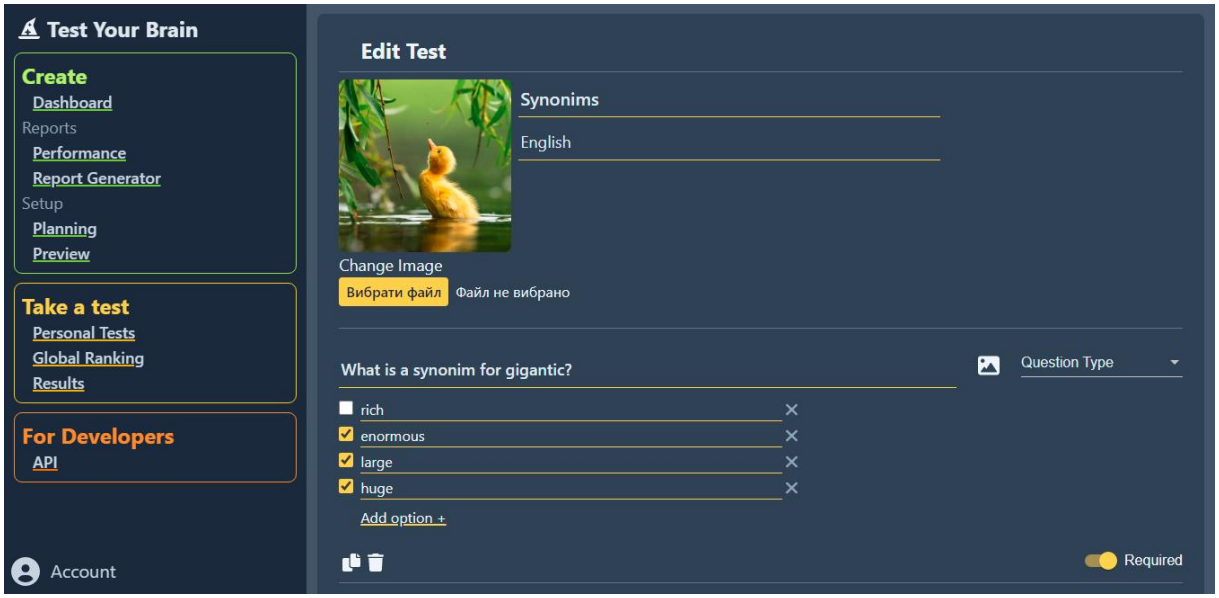

Рисунок 2.9 – Сторінка редагування тестів

Звіти

Сторінка Performance представляє статистику результатів тих, хто пройшов тест, в графічному вигляді (рисунок 2.10). В цій секції проводиться аналіз ефективності створених тестів. Можлива фільтрація за часом, перегляд інформації про окремий тест та всі тести загалом. Будується графік динаміки зміни середньої оцінки за кожен з тестів для порівняння з іншими створеними тестами.

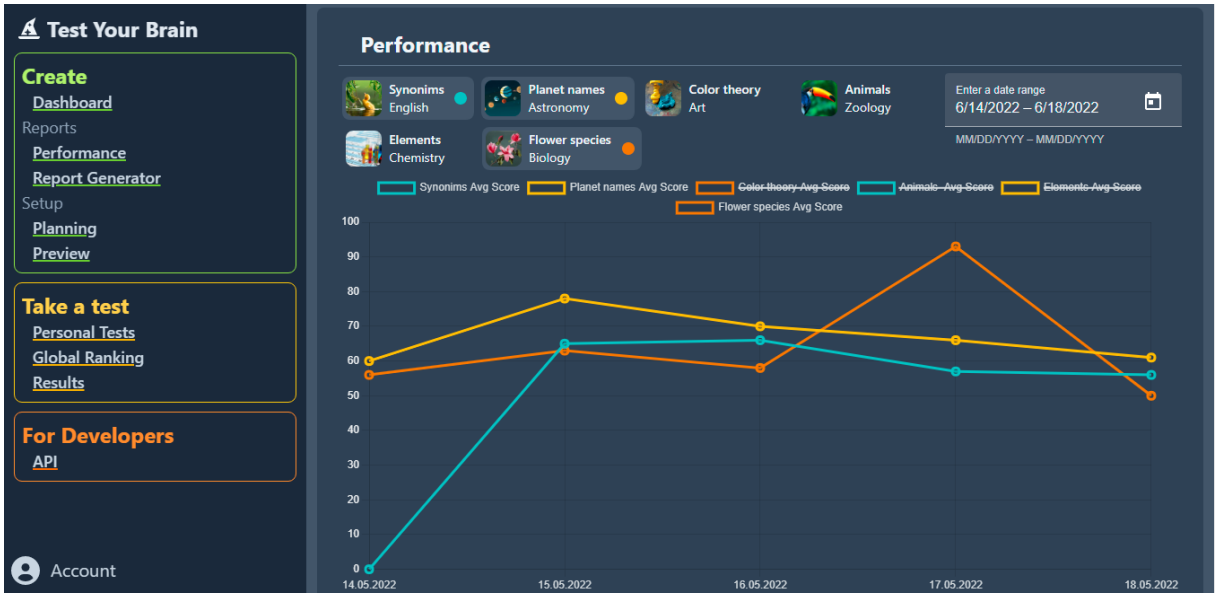

Рисунок 2.10 – Секція звітів

Для побудови графіку використано Chart.js – одну з найпопулярніших бібліотек для створення графіків та діаграм у застосунках JavaScript. Нижче наведена функція для налаштування і побудови графіку на сторінці Performance:

```
drawChart(){ // 1. create and configure a chart object; 2.render it on the canvas
    let testsDataSet :DataItem[] = [];  // generate the datasets Array
    const labels = this.getDateStrings();
    this.decoratedTests.forEach( (t) =>{
        let dataItem :DataItem = this.testToDataItem(t);
        testsDataSet.push(dataItem);
    });
    const performanceData = {
      labels: labels,
      datasets: testsDataSet,  };
    const chartOptions = {
      elements: {
        point: {
```

```
          radius: 4,
          borderWidth: 3,
          hoverRadius: 5,
          hoverBorderWidth: 4
        }
      },
      plugins: {  // legend
        legend: { labels: { color: 'rgb(214, 219, 223)', } }
      },
      scales: { // axes
        x: { ticks: { color: 'rgb(214, 219, 223)' } },
        y: { ticks: { color: 'rgb(235, 237, 239)' } }
      },
    };
    this.chartCanvas = new Chart("chartCanvas", {
      type: 'line',
      data: performanceData,
      options: chartOptions,
    });
  }
```

Сторінка Report Generator дозволяє переглянути результати тестування користувачів, що проходили тести, створені поточним користувачем. Таким чином вчитель може побачити бали своїх учнів. При виборі тесту з випадаючого списку на сторінці з’являються дані про користувачів та їх результати.

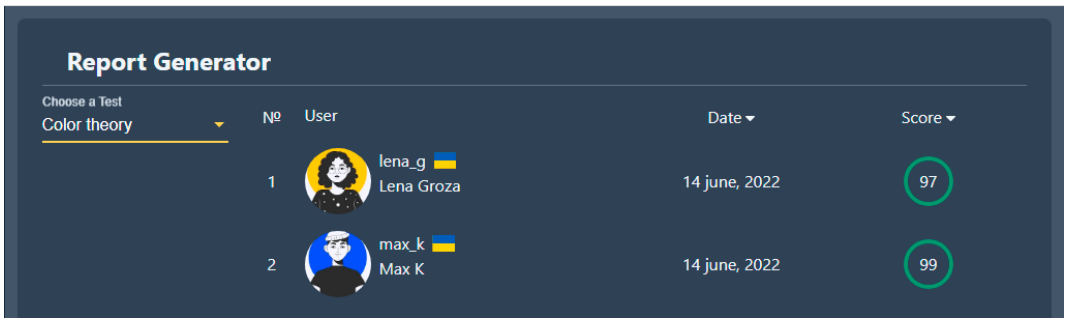

Рисунок 2.11 – Сторінка Report Generator

Для відображення прапорів країн використано бібліотеку flag-icon-css. Іконка прапору України має наступний код:

```
<span class="flag-icon flag-icon-ua ml-2"></span>
```

Для виведення прапору довільної країни використовується одностороння прив’язка даних:

```
<span class="flag-icon flag-icon-{{ countryCode }} ml-2"></span>
```

Секція налагодження

Ця секція надає функціонал для зміни статусу тесту, а також попереднього перегляду тесту. Сторінка планування має наступний вигляд (рисунок 2.12).

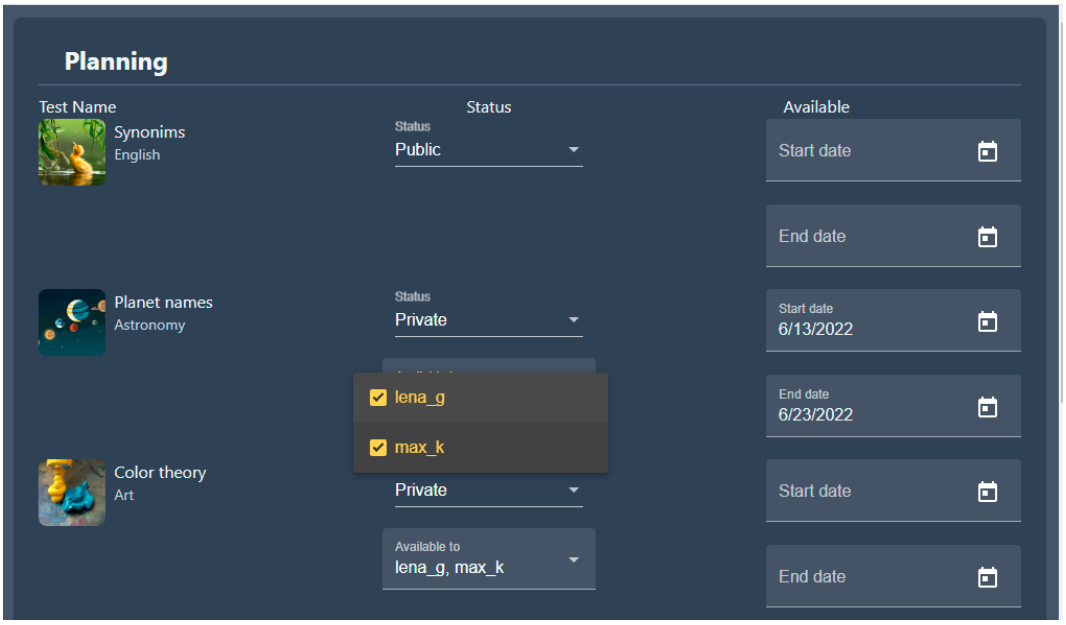

Рисунок 2.12 – Сторінка планування

На сторінці наведений список тестів з їх статусами: загальнодоступний або приватний. Загальнодоступні тести відкриті для всіх користувачів, приватні – лише для тих, кому до них надано доступ. Для кожного тесту визначається період часу, коли цей тест доступний.

Сторінка Preview призначена для перевірки якості створеного тесту через його проходження автором.

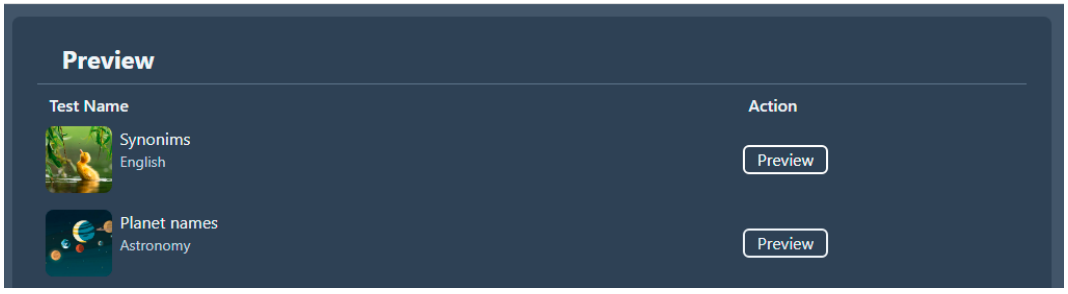

Рисунок 2.13 – Сторінка Preview

При натисненні кнопки Preview користувач переходить до форми проходження тесту.

Рисунок 2.14 – Проходження тесту

Ця сторінка виглядає так само, як сторінка звичайного проходження тесту, але не зберігає результат. Після надсилання відповідей виводиться результат тестування (рисунок 2.15).

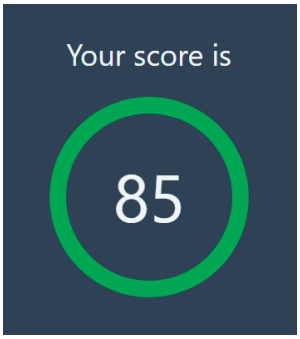

Рисунок 2.15 – Висвітлення результату

Появу результату супроводжує анімація типу fade-in. Вона створена за допомогою модуля @angular/animations. Налаштування анімації здійснюється в декораторі компоненту, в масиві animations. Код реалізації наведено нижче:

```
@Component({
  selector: 'app-testing-result',
  templateUrl: './testing-result.component.html',
  styleUrls: ['./testing-result.component.css'],
  animations: [
```

```
    trigger('fadeIn', [
      transition('void => *', [
        style({opacity: 0}),
        animate(1500)  // to default state
      ])
    ])
  ]
})
export class TestingResultComponent implements OnInit { … }
```

Шаблон компонету (testing-result.component.html):

```
<div class="text-4xl w-full text-center mt-20">
    Your score is
</div>
<div @fadeIn class="w-full flex justify-center items-center mt-32">
    <score  [scoreValue] ="score" [isScaledUp]="true"></score>
</div>
```

Сторінка власних тестів

Сторінка відображає список тестів, доступних для проходження даному користувачу. Користувач бачить як вже доступні тести, так і ті, доступ до яких відкриється у майбутньому. При натисненні на кнопку «Pass» користувач може перейти до проходження тесту.

Рисунок 2.16 – Сторінка власних тестів

Сторінка результатів

Ця сторінка надає користувачеві інформацію про пройдені ним тести (рисунок 2.17). Тут зібрана інформація про дату проходження та відсоток правильних відповідей до кожного тесту. Є можливість сортування за назвою тесту, датою та оцінкою.

Рисунок 2.17 – Сторінка результатів

Як видно, в декількох місцях застосунка використовуються однакові елементи, наприклад, кольорове коло для представлення результату. Код таких елементів винесений в окремі компоненти, дані в них передаються за допомогою механізму input properties.

Сторінка загального рейтингу

Ця сторінка показує інформацію про найуспішніших користувачів системи та найбільш популярні тести. Сторінка включає два таби (рисунок 2.18).

Таб Top Users показує інформацію про користувачів, що пройшли найбільшу кількість тестів з максимальною кількістю правильних відповідей.

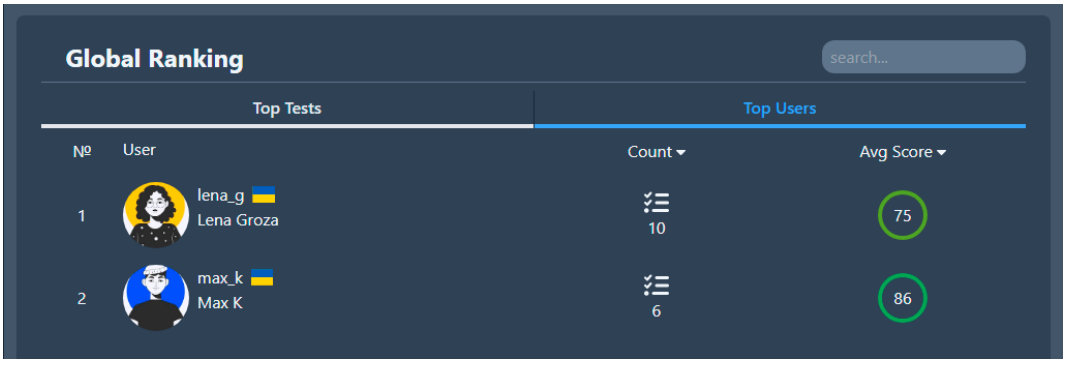

Рисунок 2.18 – Таб Top Users

Таб Top Tests показує інформацію про тести, що були пройдені найбільшу кількість разів та середній відсоток правильних відповідей для кожного тесту. Дані можна сортувати за кількістю тестів або за середньою оцінкою. Таким чином можливо побачити своє місце серед інших користувачів та наскільки популярним є тест, створений певним користувачем, серед решти користувачів.

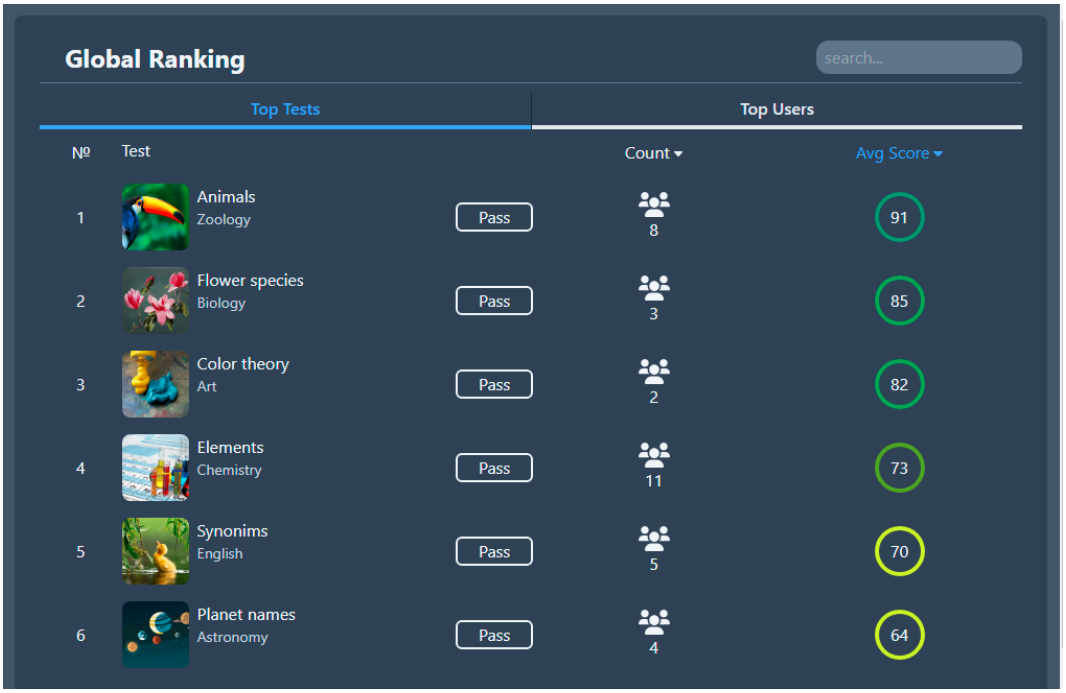

Рисунок 2.19 – Таб Top Tests

Пошук реалізований за допомогою механізму pipe фреймворку Angular. Використання pipe типу filter для пошуку тестів наведено нижче:

```
<div class="flex flex-row  basis-1 py-2" *ngFor="let t of dashboardTests | 
filter:testSearchText; let idx = index">
```

Поле пошуку має двосторонню прив’язку до змінної testSearchText компоненту:

```
<input *ngIf="tabId === 'topTests'" type="text" placeholder="search..." 
[(ngModel)]="testSearchText" class="bg-slate-500 rounded-xl px-2 py-1" >
```

Для здійснення сортування за середнім балом та кількістю проходжень тестів реалізовано функцію порівняння об’єктів:

```
  sortTestsByScore(){
    this.dashboardTests.sort((obj1, obj2) => {
      if (obj1.avgScore > obj2.avgScore) {
          return -1; // in reverse order: highest to lowest }
      if (obj1.avgScore < obj2.avgScore) {
          return 1;}
      return 0;
  });
```

Функція викликається при кліку на заголовок колонки з результатами:

```
<div (click)="sortTestsByScore()" class="basis-3/12 table-heading-elem flex 
justify-center items-center hover:text-blue-400">
```

```
    <p>Avg Score</p>
    <div class="pb-1 pl-1"><fa-icon [icon]="faSort" class="fa-sm"></fa-icon></div>
</div>
```

Також зі сторінки можна перейти безпосередньо до проходження тесту, який видався цікавим користувачу.

Telegram-бот

Користувач керує ботом за допомогою кнопок та текстових повідомлень у месенджері. На рисунку представлений процес авторизації користувача та отримання статистики про результати тестування.

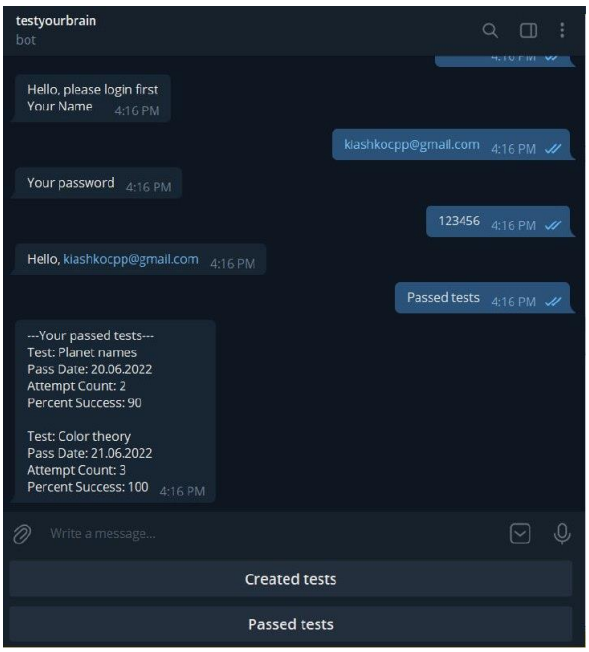

Рисунок 2.20 – Інтерфейс Telegram бота

Висновки до розділу 2

У другому розділі описано процес розробки графічних інтерфейсів застосунку, спираючись на основні принципи вебдизайну. В результаті були створені макети інтерфейсів, на основі яких здійснена програмна реалізація frontend частини системи. В розділі описано створені інтерфейси, зазначено використані технології та наведено фрагменти коду реалізації функцій системи. Результатом є готовий застосунок для автоматизації процесу тестування знань.

ВИСНОВКИ

Сучасні технології здатні значно прискорити та зробити зручнішим тестування знань в ході навчального процесу. Ключову роль у сприйнятті програми користувачем відіграє клієнтська частина системи, зовнішній вигляд та логіка взаємодії графічного інтерфейсу із користувачем.

Дана частина роботи була присвячена розробці front-end частини застосунку для автоматизації тестування знань.

В ході написання тому було виконано наступні завдання:

- проаналізовано технології розробки клієнтської частини застосунку та стилізації графічного інтерфейсу користувача, визначено їх переваги та недоліки;
- обрано стек технологій розробки та обґруновано здійснений вибір;
- наведено опис основних технологій, а саме front-end фреймворку Angular, фреймворку стилів Tailwind CSS, технології створення ботів для месенджера Telegram;
- наведено принципи дизайну графічних інтерфейсів з метою покращення користувацького досвіду при взаємодії із застосунком;
- створено макети інтерфейсів згідно відповідно до розглянутих принципів;
- реалізовано та протестовано програмну частину застосунку, наведено опис інтерфейсів та деталі реалізації.

Результатом роботи є готове програмного забезпечення автоматизації тестування знань. Застосунок відповідає поставленим вимогам та успішно досягає мети підвищення ефективності перевірки знань за рахунок автоматизації процесу тестування. Інтерфейс застосунку є простим, інтуїтивно зрозумілим та сприяє зручній взаємодії користувача із системою.

Отже, під час написання кваліфікаційної роботи бакалавра були успішно виконані усі поставлені завдання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mobile percentage of website traffic 2021 | Statista. URL: https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/ (accessed 10/06/2022).
2. Beaird J., Walker A., George J. The principles of beautiful web design. Sitepoint, 2020.
3. Dresher T., Zuker A., Friedman S. Hands-On Full-Stack Web Development with ASP. NET Core: Learn end-to-end web development with leading front-end frameworks, such as Angular, React, and Vue. Packt Publishing Ltd, 2018.
4. F-Shaped Pattern For Reading Web Content (original eyetracking research). URL: https://www.nngroup.com/articles/f-shaped-pattern-reading-web-content-discovered/ (accessed 10/06/2022).
5. F-Shaped Pattern of Reading on the Web: Misunderstood, But Still Relevant (Even on Mobile). URL: https://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/ (accessed 10/06/2022).
6. Flavian C., Gurrea R., Orus C. Web design: a key factor for the website success. *Journal of Systems and Information Technology*. 2009.
7. Is there a demand for front-end developers in 2022? URL: https://www.front-endplanet.com/are-front-end-developers-in-demand/ (accessed 12/06/2022).
8. Moiseev A., Fain Y. Angular Development with TypeScript. Simon and Schuster, 2018.
9. Shahzad F. Modern and responsive mobile-enabled web applications. *Procedia Computer Science*. Vol. 110, 2017. P. 410–415.

ДОДАТОК А
Динамічні форми для створення та редагування тестів

Файл util.service.ts:

```
import { FormBuilder } from '@angular/forms';
import { TestsService } from './tests.service';
import { Injectable } from '@angular/core';
import { map } from 'rxjs/operators';
import { Answer, Question, Test } from '../entities/test.model';

@Injectable({
  providedIn: 'root'
})
export class UtilService {

  constructor(private service: TestsService, private fb: FormBuilder) {
  }

  // this service builds, fills with data (from test service) and returns a form
for editing a test

  // for CREATE test:
  getEmptyTestForm(){
    return this.fb.group({
      id: 0,
      caption: "",
      subject: "",
      icon: "",
      open: false,
      questions: this.fb.array([
         //generate a form for each question and add here
        this.generateEmptyQuestionForm(0) // brand new question -> id = 0
      ])// end of questions array
    }
    );
  }

  // for EDIT test:
  getTestForm(testId: number){
    this.service.getTest(testId).subscribe(response =>console.log(response));
    return this.service.getTest(testId).pipe(
      map((response: any)=>this.fb.group({
        id: response.id,
        caption: response.caption,
        subject: response.subject,
        icon: response.icon,
        open: response.open,
```

```
        questions: this.fb.array(
           //generate a form for each question and add here
          response.questions.map((question: any) =>
this.generateQuestionForm(question))
        )// end of questions array
      })
      )//end of map
    );//end of pipe
  }
  generateQuestionForm(question :Question){
    const questionForm = this.fb.group({
      id: question.id,
      caption: question.caption,
      required: question.required,
      answers: this.fb.array(
        question.answers.map((answer : Answer) =>
          this.generateAnswerForm(answer)
        )
      ) //end of answers array
    });
    return questionForm;
  }
  //Ids are assigned at once bc we need to know which question to add options to
  generateEmptyQuestionForm(newId: number){
    const questionForm = this.fb.group({
      id: newId,
      caption: "",
      required: true,
      answers: this.fb.array([
        this.generateEmptyAnswerForm(1) //brand new question, id of 1st answer = 1
      ]
      ) //end of answers array
    });
    return questionForm;
  }
  generateAnswerForm(answer: Answer){
    const answerForm = this.fb.group({
      id: answer.id,
      text: answer.text,
      right: answer.right,
    })
    return answerForm;
  }

  generateEmptyAnswerForm(newId: number){
    const answerForm = this.fb.group({
      id: newId,
      text: "",
```

```
      right: false,
    })
    return answerForm;
  }
}
```

Файл edit.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { UtilService } from '../services/util.service';
import { FormArray, FormGroup, FormBuilder, FormControl } from '@angular/forms';
import { Observable } from 'rxjs';
import { TestsService } from '../services/tests.service';
import { SCROLL_THROTTLE_MS } from '@angular/material/tooltip';
import {  faTrash, faCopy, faImage, faXmark } from '@fortawesome/free-solid-svg-
icons';
import { ActivatedRoute, Router } from '@angular/router';
import { ConditionalExpr } from '@angular/compiler';
import { ImageService } from '../services/image.service';


@Component({
  selector: 'app-edit',
  templateUrl: './edit.component.html',
  styleUrls: ['./edit.component.css']
})
export class EditComponent implements OnInit {

  testForm!: FormGroup;
  faDelete = faTrash;
  faCopy = faCopy;
  faUploadImage = faImage;
  faXmark = faXmark;

  testIcon : string = '';

  constructor(private readonly util: UtilService,
              private service: TestsService,
              private route: ActivatedRoute,
              private imgService: ImageService,
              private router: Router) {
  }
  ngOnInit(): void {
    this.route.paramMap
      .subscribe(params=>{
          let testId =  +params.get('testId')!;
          console.log(testId);
          //retrieve test data:
```

```
        this.util.getTestForm(testId).subscribe(
          testForm => this.testForm = testForm
        );
    })

}
get questions(){
  return this.testForm.get('questions') as FormArray;
}
get icon(){
  return this.testForm.get('icon') as FormControl;
}

getAnswers(qIdx: number){
  return this.questions.at(qIdx).get('answers') as FormArray;
}

addQuestion(){
  let newId = this.assignQuestionId();
  this.questions.push(this.util.generateEmptyQuestionForm(newId));
}
OnChange(fileInput: HTMLInputElement){
  if(!fileInput.files) return;
  let imageFile:File = fileInput.files[0];

  this.imgService.toBase64(imageFile)
    .subscribe((data)=>{
      this.testIcon = data;
      this.testForm.patchValue({
        icon: this.testIcon
      });
    });

}


// this finds max question ID for this test and assigns returns id++
assignQuestionId(){ //helper function
  let maxId = 0;
  this.questions.controls.forEach(cg => {
    let id = cg.get("id")?.value;
    if(id >= maxId){
      maxId = id;
    }
  });
 return maxId+1;
}
```

```
  assignAnswerId(qIdx: number){ //helper function
    let maxId = 0;
    this.getAnswers(qIdx).controls.forEach(cg => {
      let id = cg.get("id")?.value;
      if(id >= maxId){
        maxId = id;
      }
    });
   return maxId+1;
  }

  removeQuestion(qIdx: number){
    this.questions.removeAt(qIdx);
  }

  addAnswer(qIdx: number){
    const newId = this.assignAnswerId(qIdx);
    this.getAnswers(qIdx).push(this.util.generateEmptyAnswerForm(newId));
  }
  removeAnswer(qIdx: number, ansIdx:number){
    this.getAnswers(qIdx).removeAt(ansIdx);
  }

  save(){
    if (this.testForm.invalid) {
      // stop here if it's invalid
      alert('Invalid input');
      return;
    }
    this.service.updateTest(this.testForm.getRawValue())
    this.router.navigate(['/dashboard-component']);
  }

}
```

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

КОМПЛЕКСНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ ЗНАНЬ

СПЕЦІАЛЬНА ЧАСТИНА З ОХОРОНИ ПРАЦІ

ПИТАННЯ ОХОРОНИ ПРАЦІ ПРИ РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Спеціальність «Інженерія програмного забезпечення»

121 – ККРБ.1 – 409.21810910

Студент _____О. А. Гроза

підпис

«___» _____ 2022 р.

Консультант канд. техн. наук, доцент кафедри екології

___ _____А. О. Алексєєва

підпис

«___» _____ 2022 р.

Миколаїв – 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ДСанПіН – державні санітарні правила і норми

ДСН – державні санітарні норми

ІТ – інформаційні технології

ОП – охорона праці

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

ПУЕ – правила улаштування електроустановок

ВСТУП

В наш час все більша частка професійної діяльності припадає на роботу з комп'ютером. З розвитком техніки оновлюються стандарти охорони праці, що беруть до уваги особливості сучасних пристроїв та їх вплив на здоров'я людини. Діяльність спеціалістів сфери IT пов'язана з тривалим перебуванням за комп'ютером, тому для них особливо важливо знати вимоги до роботи з екранними пристроями, а також рекомендований режим праці та відпочинку.

Компетентність в питаннях охорони праці є актуальною для усіх фахівців комп'ютерної галузі, незалежно від того, чи хочуть вони заснувати свою фірму, чи працювати в чужій. Роботодавець повинен забезпечити умови праці для співробітників фірми та несе безпосередню відповідальність за дотримання державних стандартів [6]. Створення оптимальних умов праці сприяє продуктивності роботи співробітників, запобігає розвитку професійних хвороб та зменшує ризик нещасних випадків на робочому місці. Водночас знання в області охорони праці дозволяє працівнику захищати свої права щодо створення працедавцем належних умов трудової діяльності.

Особливої уваги також заслуговують питання безпеки життєдіяльності, адже вміння оперативно та грамотно зреагувати на виникнення надзвичайної ситуації може врятувати життя та здоров'я працівників.

Метою розділу є створення безпечних та комфортних умов праці на робочому місці програміста. Для досягнення цієї мети розглядаються вимоги до роботи з комп'ютером та робочого місця, правила електробезпеки, вимоги до відпочинку. Аналізуються умови в приміщенні, де відбувається трудова діяльність (офісі), такі як освітленість та параметри мікроклімату. Наводяться правила дій при пожежі та заходи щодо запобігання пожежам в офісі.

1 Охорона праці при роботі з комп’ютером

1.1 Вимоги до роботи із екранними пристроями

Робота програміста нерозривно пов’язана із використанням екранних пристроїв. Для забезпечення безпеки під час роботи з ними необхідно додержуватися наступних правил [3]:

– кожного дня перед початком роботи пристрої слід очищати від пилу та інших забруднень;
– після роботи з екранними пристроями їх треба відключити від електричної мережі;
– при виникненні аварійної ситуацій пристрій слід негайно відключити від мережі.

Забороняється:

– ремонтувати або полагоджувати екранні пристрої безпосередньо на робочому місці працівника під час роботи з ними;
– продовжувати працювати з екранними пристроями, якщо під час взаємодії з ними виникають нехарактерні сигнали, якщо зображення на екрані є нестабільним або наявні інші несправності;
– самочинно здійснювати зміни в конструкції та складі екранних пристроїв.

Екранні пристрої мають задовольняти визначені законодавством вимоги до них:

– випромінювання екранів, за виключенням видимої частини спектра, має буди зведене до мінімуму для збереження здоров’я працівників;
– символи на екрані мають бути чіткими, достатнього розміру та мати достатню контрастність, між символами та рядками має бути належна відстань;
– не повинно бути помітного миготіння екрану, зображення на ньому має бути чітким і стабільним;

- яскравість екрану має легко регулюватися працівником, адаптуватися до його потреб та навколишніх умов;
- при виборі екранів слід надавати перевагу таким, що легко повертаються та нахиляються;
- джерела світла мають бути встановлені так по відношенню до екрана, щоб запобігти виблискуванню та відбиванню світла, яке справляю дискомфорт працівнику підчас роботи з екранним пристроєм;
- слід віддавати перевагу клавіатурам, що є відділеними від екрану, щоб працівник міг переміщувати її та обрати зручну для себе робочу позу;
- поверхня клавіатури повинна бути матовою та не віддзеркалювати світло, символи на клавішах мають бути достатньо контрастними та розбірливими
- обладнання не має виділяти надлишкового тепла, що може спричинити дискомфорт працівника
- використовуване програмне забезпечення має відповідати поставленим завданням, а також рівню досвіду і знань працівника, бути зручним у використанні.

Робоче місце для праці з екранними пристроями має мати достатній розмір, дозволяти працівнику змінювати робочу позу відповідно до своїх потреб. Організація робочого місця та розміщення обладнання мають відповідати правилам ергономіки та характеру виконуваної праці. Освітлення має створювати достатній контраст між екраном і навколишнім середовищем, додержуватись вимог ДСанПІН 3.3.2.007-98 [1]. Мікроклімат має бути стабільним, його параметри мають задовольняти вимогам санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [8]. Поверхня столу повинна бути достатнього розміру та мати низьку відбивну здатність. Крісло повинно бути стійким та дозволяти працівнику легко регулювати своє робоче положення.

1.2 Правила електробезпеки при роботі в офісі

Електробезпека – це сукупність заходів та засобів для забезпечення захисту життя та здоров’я людини від шкідливого впливу електричного струму, електричної дуги, статичної електрики та електромагнітного поля.

Правила електробезпеки слід брати до уваги під час розгортання електромережі. Необхідно подбати про правильний розподіл навантаження між усіма приміщеннями офісу, а також розподіл електромережі за призначенням (наприклад, окремо освітлення і окремо – основна робоча група). Усі елементи електромережі, такі як розетки, кабелі та перемикачі, мають бути належної якості та відповідати стандартам. При плануванні варто брати до уваги можливість розширення мережі та збільшення навантаження (наприклад, пізніше може виникнути необхідність збільшення кількості робочих місць).

Задля уникнення ризику електронебезпеки при використанні електропристроїв усе обладнання в офісі має використовуватись за призначенням та згідно із рекомендаціями виробника [5]. Не можна вмикати у електромережу прилади, що мають шнури із пошкодженою ізоляцією. Заборонено користування пошкодженими розетками, вимикачами, саморобними подовжувачами або іншими елементами електромережі, що не відповідають правилам ПУЕ. Не дозволяється використання нестандартних або саморобних електропристроїв для обігріву приміщення. Варто уникати доторкання до металевих елементів увімкнених в розетку електропристроїв. При прибиранні пилу з обладнання, митті підлоги тощо необхідно вимикати електроспоживачі з розетки для запобігання ураження електричним струмом. При відключенні пристрої від мережі, вилку слід тримати за корпус, не можна висмикувати її, тримаючись за шнур живлення.

1.3 Вимоги до режиму праці і відпочинку при роботі з комп’ютером

Для збереження здоров’я працівників при роботі з ПК, запобігання професіональним захворюванням та підтримки продуктивності праці мають бути передбачені перерви для відпочинку.

Згідно із п. 5.3 ДСанПіН 3.3.2.007-98 протягом робочого дня передбачаються наступні типи перерв [1]:

– обідні перерви;
– перерви для відпочинку та особистих потреб;
– додаткові перерви (для окремих професій із врахуванням особливостей виконуваної праці).

В пункті 5.8 ДСанПіН 3.3.2.007-98 рекомендуються наступні режими праці і відпочинку в залежності від виду діяльності, пов’язаної з ПК:

– для розробників ПЗ передбачається регламентована 15-хвилинна перерва для відпочинку через кожну годину праці за комп’ютером;
– для операторів ПЗ – 15-хвилинна перерва через кожні дві години роботи;
– для операторів комп’ютерного набору передбачена перерва тривалістю 10 хвилин через кожну годину праці.

У випадках, коли через виробничі обставини неможливо додержуватись рекомендованого режиму відпочинку, тривалість роботи за ПК не може перевищувати 4 години.

Задля зменшення негативного впливу монотонної праці рекомендується чергувати операції з обробки тексту та числових даних, за можливості змінювати характер діяльності. Для збереження здоров’я, покращення кровообігу та зняття напруження доцільно під час перерв пройтися або виконати легкі фізичні вправи.

2 Умови праці на робочому місці розробника ПЗ

2.1 Освітлення приміщень з використанням комп’ютерів

Світло є одним з головних чинників зовнішнього середовища, що впливають на людину під час ведення трудової діяльності. Втомлюваність очей під час праці залежить насамперед від напруженості процесів, пов’язаних із зоровим сприйняттям. Під час роботи за комп’ютером напруження зору є значним, а отже вкрай важливим є забезпечення відповідних умов освітленості робочого місця.

Існує низка професійних захворювань, пов’язаних із поганим освітленням, серед яких короткозорість, робоча міопія, спазм акомодації. Рівень освітлення впливає не лише на зір, але й на діяльність людини загалом. При невідповідній якості освітлення зменшується продуктивність працівника, збільшується втомлюваність та ризик помилкових дій.

Кількісно охарактеризувати рівень освітлення приміщення можна за допомогою наступних показників:

Світловий потік (Ф) – фізична величина, що описує енергію світлового потоку, який проходить через поверхню за одиницю часу. Одиниця вимірювання – люмен (лм).

Сила світла (I) - відношення світлового потоку, до тілесного кута, в межах якого проходить цей потік. Одиниця вимірювання – кандела (кд).

Освітленість (Е) – величина, що чисельно дорівнює відношенню світлового потоку до площі поверхні, на яку він падає. Одиниця вимірювання – люкс (лк).

Яскравість (В) – відношення сили світла, що випромінюється поверхнею, до площі цієї поверхні (кд/м^2).

Освітлення приміщень, в яких відбувається праця, має відповідати наступним основним вимогам:

– забезпечувати освітленість робочих поверхонь, що відповідає нормам для даного типу зорової роботи;
– бути достатньо рівномірним та постійним;
– не чинити засліплювальної дії (від самих джерел світла та інших предметів, від яких світло може відбиватися);
– не створювати глибоких тіней на робочій поверхні.

Важливим фактором є також колір, в який фарбується елементи приміщення та обладнання. Вибір кольорів, що оточують працюючих, впливає на їх психологічний стан та схильність до втомлюваності. Класифікація кольорів наведена в таблиці 1.

Таблиця 1 – Класифікація кольорів

Типи кольорів	**Ділянки спектру**	**Довжина хвилі**	**Приклади**
Ахроматичні	-	-	Чорний, білий, відтінки сірого
Хроматичні	Довгохвильові	760-590 нм	Відтінки червоного та помаранчевого
	Середньохвильові	590-500 нм	Відтінки жовтого та зеленого
	Короткохвильові	500-380 нм	Відтінки синього та фіолетового

Кольори різних типів чинять різний вплив на стан людини. Довгохвильові кольори призводять до емоційного збудження та підвищеної рухливості, але через це сприяють швидшому втомленню. Короткохвильові кольори, навпаки, сприяють заспокоєнню. Оптимально впливають на людину кольори середньої ділянки спектру, вони є приємними для ока та знижують втомлюваність. Також слід уникати поєднань яскравих контрастних кольорів (наприклад, синього з помаранчевим або червоного з фіолетовим) – вони викликають роздратування і швидку втому.

При виборі кольору слід також брати до уваги такі його характеристики як насиченість (ступінь наближеності кольору до чистого спектрального тону) та яскравість (характеризується коефіцієнтом відбивання).

Варто пам’ятати не лише про колірну гамму інтер’єру, а й про відтінок освітлення, що характеризується колірною температурою. Для робочих приміщень колір освітлення може коливатися від нейтрального до теплого білого. Для зон, в яких необхідна гранична концентрація уваги, найкраще підійде поєднання нейтральних та холодних відтінків світла. А для створення затишної та комфортної атмосфери у зоні відпочинку найбільш відповідними є теплі відтінки.

Для перевірки рівня освітленості у приміщеннях використовують прилад люксометр.

В наступній таблиці наведені норми освітленості для офісу.

Таблиця 2 – Норми освітленості

Тип приміщення в залежності від виконуваних робіт	**Норма освітленості, лк**
Великий офіс з вільним плануванням	400
Офіс з комп’ютерами	200-300
Конференц зал	200
Сходові прольоти, ескалатори, хол	75-100

Для розрахунку світлового потоку та визначення необхідної кількості світильників в офісному приміщенні можна скористатися наступною формулою:

Світловий потік = Норма освітленості * Площа * Коефіцієнт висоти стелі

Значення коефіцієнтів висоти стелі наведені в таблиці 3.

Таблиця 3 – Коефіцієнти висоти стелі для розрахунку світлового потоку

Висота стелі, м	**Коефіцієнт**
2,5-2,7	1
2,7-3	1,2
3-3,5	1,5
3,5-4,5	2

Розрахуємо значення для офісу площею 40 м^2 та з висотою стелі 3 м. Світловий потік:

$$Ф = 300 \cdot 40 \cdot 1{,}2 = 14400(Лм)$$

Кількість необхідних лампочок залежить від їх типу та потужності. Приблизна кількість світильників для даного приміщення обрахована у наступній таблиці.

Таблиця 4 – Розрахунок кількості лампочок для освітлення офісу

Тип лампочки	**Потужність, Вт**	**Світловий потік, Лм**	**Кількість лампочок, шт.**
Розжарювання	40	470	14 400/470 = 31
Галогенна	32	470	14 400/470 = 31
Енергоощадна	15	700	14 400/700 = 21
Світлодіодна	10	750	14 400/750 = 20

Варто пам’ятати, що колір інтер’єру вливає на яскравість освітлення. Світліший інтер’єр вимагає менш яскравого освітлення, в той час як для темних інтер’єрів яскравість потужність світильників збільшують у 1,5-2 рази.

2.2 Мікроклімат в приміщеннях з використанням комп’ютерів

Мікроклімат – це сукупність метеорологічних умов внутрішнього середовища виробничого приміщення [2]. До параметрів мікроклімату відносяться температура, відносна вологість, швидкість руху повітря, теплове випромінювання нагрітих поверхонь.

Працедавець зобов’язаний створити оптимальні, або принаймні допустимі мікрокліматичні умови на місці праці.

Оптимальними умовами вважаються такі, при яких не відбувається напруження механізмів терморегуляції; такі умови є найбільш сприятливими та при тривалій дії на людину не викликають відчуття дискомфорту.

Допустимими вважаються умови, при довготривалій дії яких у людини можуть виникати дискомфортні відчуття, проте вони швидко минають і стан людини нормалізується за рахунок напруження механізмів терморегуляції та пристосування організму до умов середовища.

Норми параметрів мікроклімату залежать від наступних факторів:

– період року;

– категорія важкості роботи з огляду на фізичне навантаження;

– вид робочого місця.

Періоди року поділяється на:

– теплий (середньодобова температура навколишнього середовища вища за 10°С);
– холодний (середньодобова температура навколишнього середовища нижча за 10°С).

Таблиця 5 – класифікація роботи за рівнем фізичного навантаження

Категорія	**Характеристика роботи**	**Енерговитрати, Дж/с**
Легкі (I,а та I,б)	Робота виконується сидячи, стоячи або пов'язана з ходьбою, не вимагає систематичної фізичної напруги, підняття або переносу тягаря	До 139 140-174
Середньої важкості (II, а)	Робота пов'язана з постійною ходьбою, виконується стоячи або сидячи, але не вимагає переносу тягарів	175-232
Середньої важкості (II, б)	Робота пов'язана з ходьбою і переносом невеликих тягарів (до 10 кг)	233-290
Важкі	Робота пов'язана з постійним переносом або переміщенням значних тягарів (більше 10 кг)	Більше 290

Вид робочого місця:

– постійне;
– непостійне.

Вологість повітря суттєво впливає на працездатність та самопочуття людей. Занадто висока вологість погіршує віддачу тепла організмом за рахунок випаровування, тобто перешкоджає процесу терморегуляції. Низька вологість призводить до пересихання слизових оболонок дихальних шляхів.

Параметри вологості нормуються законодавством. Згідно з ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень», оптимальна вологість повітря становить 40-60%, при цьому вважається допустимою вологість не більше 75%. Температура повітря в офісному

приміщенні має виносити 21-25°С. Швидкість руху повітря не повинна перевищувати 0,1 м/с [8].

Таблиця 6 – Норми температури в залежності від пори року та виду праці

Період року	**Категорія робіт**	**Температура, °С**
Холодний	Легка (І а)	22-24
	Легка (І б)	21-23
Теплий	Легка (І а)	23-25
	Легка (І б)	22-24

В таблиці 6 наведені оптимальні показники температури відповідно до періоду року для легкої за фізичним навантаженням категорії робіт, до яких відноситься праця за комп’ютером у офісі.

2.3 Рівень шуму при роботі з ПК

Вагомий влив на концентрацію робітника справляє рівень шуму на робочому місці. Для нормування рівня шуму існують два методи:

– нормування по граничному спектру шуму;

– нормування рівня шуму.

В першому методі нормуються рівні припустимого звукового тиску для кожної з дев’яти октавних смуг [7], тобто береться до уваги розподіл шуму за висотою. Октавою є смуга частот, для яких виконується співвідношення $f_1/f_2=2$. Сукупність нормативних рівнів для усіх дев’яти смуг називається граничним спектром.

В другому методі розраховується загальний рівень звуку, та використовується для приблизної оцінки постійного і непостійного шуму на робочому місці та його відповідності нормам для даного виду трудової діяльності.

Для розрахунку рівню шуму в приміщенні під час роботи програміста скористаємося вхідними даними, наведеними у таблиці 7.

Таблиця 7 – Вхідні дані для розрахунку рівня шуму

Призначення приміщення	Джерела шуму		
	Шумові характеристики, дБА		
Офіс	Комп'ютер	Кондиціонер	Інше обладнання
	30	42	40

Загальний рівень шумового тиску визначається за наступною формулою:

$$L = 10lg \sum_{i=1}^{n} 10^{0,1*L_i}$$

де

n – кількість джерел шуму,

L_i – рівень шумового тиску і-го джерела звуку.

Після підстановки даних про звуковий тиск обладнання у формулу, отримуємо:

$$L = 10\lg(10^{0,1\cdot 30} + 10^{0,1\cdot 42} + 10^{0,1\cdot 40}) = 44{,}3 \text{ (дБА)}$$

Розрахований рівень шуму порівнюємо із нормами, наведеними у таблиці 8.

Для роботи програміста допустимим є рівень шуму не вище 50 дБА. А отже розраховане значення 44,3 дБА вкладається в норму та відповідає вимогам ОП.

Таблиця 8 – Вимоги до рівня шуму відповідно до виду діяльності

№	Вид трудової діяльності	Рівні звуку, дБА
1	Творча діяльність, керівна діяльність із підвищеними вимогами, наукова діяльність, конструювання й проектування, програмування, викладання	50
2	Висококваліфікована робота, що адміністративно-керівна діяльність, вимірювальні й аналітичні роботи в лабораторії	60

Кінець таблиці 8

№	Вид трудової діяльності	Рівні звуку, дБА
3	Робота із часто одержуваними вказівками й акустичними сигналами, робота, що вимагає постійного слухового контролю, операторська робота, диспетчерська робота	65
4	Робота, що вимагає зосередження, робота з підвищеними вимогами до процесів спостереження й дистанційного керування	75
5	Виконання всіх видів робіт (крім перерахованих у п. п. 1-4 та аналогічних їм) на постійних робочих місцях у виробничих приміщеннях і на території підприємств	80

2.4 Санітарно-гігієнічні норми для приміщень з використанням комп'ютерів

Серед основних санітарно-гігієнічних вимог для офісних приміщень виділяють наступні:

- на одне робоче місце повинно припадати не менше ніж 6 м^2;
- відстань від робочого місця до стіни з вікном має бути не меншою за 1 м, а від звичайної стіни – не меншою за 1,4 м;
- відстань між сусідніми комп'ютерами має бути не менше ніж 1,2 м, рахуючи між боковими поверхнями, та не менше ніж 2,5 м між тильною частиною одного та екраном іншого;
- офісні приміщення заборонено розміщувати у підвальних та цокольних приміщеннях будинків;
- підлога в офісі має бути рівною, із неслизьким матовим покриттям;
- забороняється використання матеріалів, які виділяють у повітря шкідливі речовини;
- в офісних приміщеннях, де відбувається робота з комп'ютерами, кожного дня має проводитися вологе прибирання;

– окрім робочої зони, повинна бути обладнана кімната або зона для відпочинку і психологічного розвантаження;
– робочий стіл та крісло повинні забезпечувати оптимальну робочу позу при роботі із комп’ютером та іншим обладнанням;

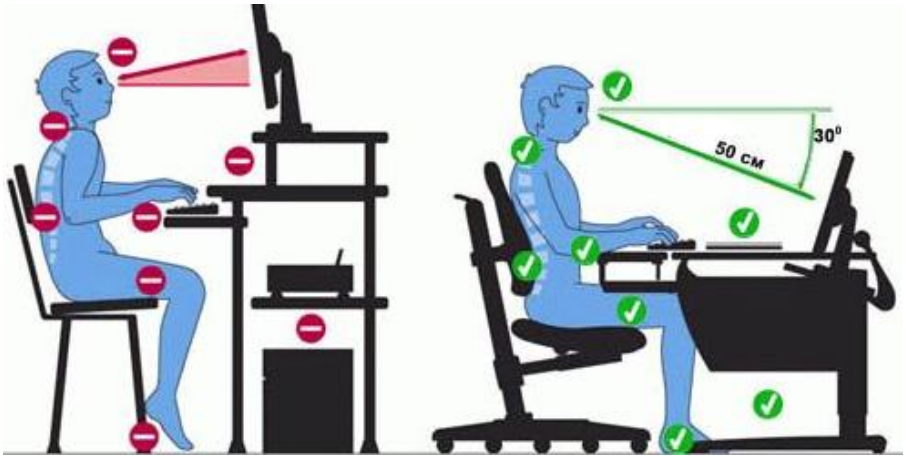

Рисунок 1 – Робоча поза при праці за комп’ютером

– приміщення мають бути обладнані системою опалення, вентиляцією, системою кондиціонування повітря для підтримки відповідних умов мікроклімату;
– коефіцієнт природної освітленості (КПО) в офісі має бути не меншим за 1,5%. Рекомендовано використовувати жалюзі для регулювання рівня природного освітлення у приміщенні;
– комп’ютери мають бути розташовані так, щоб уникати прямого попадання світла в очі та засвітлення екранів. Забороняється використання світильників без розсіювачів;
– освітленість на робочих поверхнях має бути не меншою за 300 лк;
– в приміщенні мають додержуватися норми рівня звуку відповідно до виду діяльності працівників. Для програмістів це 50 дБА, для операторів комп’ютерного набору та обробки інформації на ПК – 65 дБА.

3 Пожежна безпека в офісі

Відповідно до статистики, пожежі в офісах найчастіше виникають через куріння працівників, несправність опалювальних пристроїв, загоряння електропроводки.

Для вчасного реагування на надзвичайну ситуацію в офісах рекомендується встановлювати пожежні сповіщувачі. Вони бувають різних типів, зокрема:

– димові (реагують на дим у повітрі, потребують регулярного чищення для запобігання хибним спрацюванням);
– теплові (реагують на температуру, дозволяють виявити пожежу, якщо дим поширюється не безпосередньо в приміщенні, а наприклад, за підвісною стелею);
– сповіщувачі полум’я (реагують на інтенсивне ультрафіолетове та інфрачервоне випромінювання, що супроводжують полум’я).

Для попередження людей про небезпеку використовуються звукові системи сигналізації (динаміки, сирени) та світлові сигнали (миготливий оповіщувач, табло тощо).

Існують також системи автоматичного гасіння пожеж, які не лише сповіщають про їх появу, а й сприяють ліквідації вогню. Такі системи випускають спеціальну речовину, таку як воду, піну, газ чи аерозоль, у місці займання. Це дозволяє швидко зупинити поширення вогню та зменшити шкоду від пожежі. В сучасних офісі зазвичай віддають перевагу порошковій системі, адже гасіння водою може значно пошкодити документи та обладнання.

Сучасні системи здатні автоматично повідомити про пожежу та викликати рятівників, відключити електрику в офісних приміщеннях, оповістити працівників та вказати їм шлях евакуації.

Усі працівники офісу мають бути попередньо ознайомлені з планом евакуації та розташуванням аварійних виходів [4].

У разі виникнення пожежі слід:

1. Оцінити обстановку, зрозуміти, звідки надходить небезпека. Терміново повідомити про пожежу рятувальників: назвати своє ім’я та прізвище, адресу офісу, місце виникнення пожежі (коридор, кабінет тощо), повідомити про наявність людей в будинку та загрози їх життю, вказати найкоротший шлях під’їзду до осередку пожежі. Якщо не звучить сирена, необхідно сповістити інших людей в офісі про займання.
2. Якщо площа пожежі не перевищує 1 м^2, можна використати наявні первинні засоби пожежогасіння (наприклад, найближчий вогнегасник). Важливо пам’ятати, що гасіння водою електроприладів, увімкнених в мережу, небезпечне для життя. Для запобігання ураження струмом слід вимкнути електрику в мережі.

   Якщо площа пожежі більша за 1 м^2 та самостійно загасити її неможливо або існує загроза життю, слід негайно покинути приміщення та щільно зачинити за собою двері, щоб запобігти поширенню вогню.
3. Працівники мають евакуюватися з приміщення згідно із планом евакуації та дочекатися прибуття пожежної бригади.

ВИСНОВОК

В процесі написання спеціальної частини кваліфікаційної роботи бакалавра були вивчені чинні норми та вимоги до охорони праці програмістів, проаналізовані умови організації робочого місця розробника програмного забезпечення. Отже, поставлена мета досягнута.

Такі фактори, як освітлення та рівень шуму можуть сильно впливати на якість роботи розробника, адже для цієї праці характерна значна напруга зору при постійній роботі перед монітором та необхідність високого рівня концентрації. Тому у розділі наведено способи розрахунку рівня освітленості та звукового тиску на робочому місці для забезпечення дотримання державних норм. Також шкідливий вплив має тривале сидіння в одному положенні, тому розглянуті вимоги до організації робочого місця та відпочинку.

В рамах питання безпеки життєдіяльості працівників розглянуто дії при пожежі та заходи, спрямовані на запобігання пожежам в офісних приміщеннях.

Отже, було розглянуто основні аспекти охорони праці та безпеки життєдіяльності, пов’язані з працею розробника програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСаНПіН 3.3.2.007-98. ДСанПіН 3.3.2.007-98 Державні санітарні правила та норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. (40939). URL: https://dnaop.com/html/40939/doc-ДСаНПіН_3.3.2.007-98 (accessed 06/06/2022).
2. Л.Е. В., М.В. В., М.В. Г. Основи охорони праці. 2001. 192 р.
3. Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями | від 14.02.2018 № 207. URL: https://zakon.rada.gov.ua/laws/show/z0508-18#Text (accessed 31/05/2022).
4. Про затвердження пожежної безпеки в Україні | від 30.12.2014 № 1417. URL: https://zakon.rada.gov.ua/laws/show/z0252-15#Text (accessed 31/05/2022).
5. Про затвердження технічної експлуатації електроустановок споживачів | від 25.07.2006 № 258 | від 25.07.2006 № 258. URL: https://zakon.rada.gov.ua/laws/show/z1143-06#Text (accessed 31/05/2022).
6. Про охорону праці | від 14.10.1992 № 2694-XII. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (accessed 31/05/2022).
7. Санітарні норми виробничого шуму ультразвуку та інфразвуку | від 01.12.1999 № 37. URL: https://zakon.rada.gov.ua/rada/show/va037282-99#Text (accessed 31/05/2022).
8. Санітарні норми мікроклімату виробничих приміщень | від 01.12.1999 № 42. URL: https://zakon.rada.gov.ua/rada/show/va042282-99#Text (accessed 31/05/2022).