МІНІСТЕРСТВО ОСВІТИ І НАУКИ КРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ОНУФРІЄВ ОЛЕКСІЙ СЕРГІЙОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
док. техн. наук, доцент
_____Т.В. Нескородєва
«_____»_____2022 р.

ДОСЛІДЖЕННЯ АЛГОРИТМІВ ЕЛЕКТРОННОГО ГОЛОСУВАННЯ В УМОВАХ ПРЯМОЇ ДЕМОКРАТІЇ

Спеціальність 122 Комп'ютерні науки
Магістерська робота

Науковий керівник:
Потапова Надія Анатоліївна,
доцент кафедри інформаційних технологій
доц., к.е.н.
_____
(підпис)

Оцінка:____ /____ /_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2022

АНОТАЦІЯ

Онуфрієв О.С. Дослідження алгоритмів електронного голосування в умовах прямої демократії. Спеціальність 122 «Комп'ютерні науки». Освітня програма Комп'ютерна обробка даних (Data Science). Донецький національний університет імені Василя Стуса, Вінниця 2022.

У магістерській роботі представлена розробка платформи, призначення якої є створення середовища та критеріїв для оцінки процесів електронного голосування. Платформа має дизайнерську ідентичність, допомагає створити процеси голосувань та опитувань, має клієнт- та серверну частини. При розробці платформи була використана мова програмування JavaScript та TypeScript, яка є типізованою мовою JavaScript.

Магістерська робота складається з вступу, трьох розділів, висновків та додатків. У вступі обґрунтовується актуальність теми, описується мета роботи та постановка завдань дослідження. У першому розділі висвітлюються питання теоретико-методичних засад розвитку виборчих процесів та історія голосування. У другому розділі наведено результати формалізації та опис функціональних вимог до створення системи електронного голосування. У третій частині наведено результати розробки готового продукту – платформи електронного голосування.

Магістерська робота складається зі вступу, 3 розділів, висновків, списку літератури з 55 джерел, 51 рисунок. Загальний обсяг роботи становить 83 сторінки.

ANNOTATION

Onufriev O.S. Study of electronic voting algorithms in conditions of direct democracy. Specialty 122 "Computer Science". Educational program Computer data processing (Data Science). Vasyl Stus Donetsk National University, Vinnytsia 2022.

The master's thesis presents the development of a platform, the purpose of which is to create an environment and criteria for evaluating electronic voting processes. The platform has a designer identity, helps to create voting and survey processes, has client and server parts. The platform was developed using JavaScript and TypeScript, which is a typed JavaScript language.

The master's thesis consists of an introduction, three chapters, conclusions and appendices. The introduction substantiates the relevance of the topic, describes the purpose of the work and setting the research objectives. The first chapter covers the issues of theoretical and methodological foundations of the development of electoral processes and the history of voting. The second section presents the results of formalization and a description of the functional requirements for creating an electronic voting system. The third part presents the results of the development of the finished product - an electronic voting platform.

The master's thesis consists of an introduction, 3 chapters, conclusions, a list of references from 55 sources, 51 figures. The total volume of work is 83 pages.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DOM – (Document Object Model)

APP – додаток (Application)

SPA - односторінковий додаток (Single-page application)

SSR – візуалізація на стороні сервера (Server side rendering)

SSG – генерація статичного сайту (Static Site Generation)

CDN – мережа доставки вмісту (Content Delivery Network)

API – інтерфейс прикладного програмування (Application Programming Interface)

XML – розширювана мова розмітки (Extensible Markup Language)

AJAX – асинхронний JavaScript і XML (Asynchronous JavaScript and XML)

JSON – нотація об’єктів JavaScript (JavaScript Object Notation)

PWA – прогресивний веб-додаток (Progressive web app)

NODE.JS – JavaScript на стороні сервера (Server side JavaScript)

AWS – веб-сервіси Amazon (Amazon Web Services)

AZURE – хмарний сервіс Microsoft (Microsoft cloud service)

MVC – контролер перегляду моделі (Model View Controller)

CX – досвід роботи з клієнтом (Client Experience)

BX – досвід бренда (Brand Experience)

UX - досвід користувача (User Experience)

UI – інтерфейс користувача (User Interface)

БД – база даних (Data Base)

СУБД – система управління бази даних (System control data base)

JS - JavaScript

р.н.е. – рік нової ери

млрд. дол. США – мільярд доларів США

ЗМІ – засоби масової інформації

ВСТУП

Актуальність теми. З давніх часів голосування було одним із найважливіших процесів, результати якого забезпечували механізми та траєкторію подальшого існування та розвитку суспільства. Історично, проведення голосування мало складність самого процесу, як з боку його організації, так і з прозорістю та чесністю підрахунків голосів. Сучасний розвиток інформаційних технологій дав поштовх до появи нових форм голосування, що відтворюють механізм прямої демократії (вплив народу на вибір рішення), зокрема, електронного голосування. Такий підхід має ряд переваг у порівнянні зі стандартною формою організації, серед яких: дотримання відповідних правил захисту учасників, зменшення часу на підрахунок результатів та проведення голосувань, скорочення фінансових витрат на проведення заходу та ін. Поряд з цим, відбулись розробки методики для дослідження поведінкових елементів виборів з урахуванням свідомості учасників, а разом з цим зменшення негативних впливів та маніпуляцій з виборчим процесом. Проте, на сьогодні немає уніфікованого підходу щодо організації електронного голосування, відсутня чітка аналітика голосувань людей та ідентифікація критеріїв впливу на вибір. Тому, розробка нових онлайн платформ, за допомогою яких можливо організувати та реалізувати процес голосування залишається актуальним питанням, як зі сторони законодавчої ініціативи, так і зі сторони проектування та запровадження засобів сучасних інформаційних технологій.

Мета магістерської роботи. Метою магістерської роботи є формалізація та розробка онлайн платформи для відтворення процесів організації, проведення та аналізу результатів електронного голосування в умовах прямої демократії.

Завдання магістерської роботи. Для реалізації поставленої мети було поставлено ряд завдань:

- дослідити теоретичні засади функціонування виборчих процесів та голосування, з метою виявлення їх основних характеристик та індикаторів

оцінювання;

- визначити основні проблеми, що виникають в результаті проведення голосувань фізичним шляхом та реалізувати їх запобіжники під час електронного голосування;

- провести формалізацію механізму процедури голосування, з метою її подальшої реалізації при електронному голосуванні через використання онлайн-платформи;

- розробити та програмно реалізувати дизайн онлайн платформи для організації та проведення електронного голосування;

- спроєктувати та реалізувати програмно клієнт частину онлайн платформи для організації та проведення електронного голосування;

- спроєктувати та реалізувати програмно сервер частину онлайн платформи для організації та проведення електронного голосування.

Об'єктом дослідження є процес організації та проведення електронного голосування в умовах прямої демократії.

Предмет дослідження - алгоритми та методи електронного голосування в умовах прямої демократії.

Наукова новизна дослідження полягає в тому, що:

вперше отримано:

програмний продукт (онлайн платформу), шляхом використання якої є можливість реалізувати поєднання алгоритмів голосування та механізмів передвиборчих опитувань з урахуванням поведінкових технологій щодо усвідомлень учасників.

вдосконалено:

алгоритмічні підходи щодо реалізації механізму функціонування процесу електронного голосування в умовах прямої демократії.

Структура роботи. Магістерська робота складається зі вступу, трьох розділів, висновків та списку використаних джерел.

У першому розділі висвітлено теоретичні засади функціонування виборчих процесів та голосування, проаналізовано історію розвитку процесу голосувань,

розглянуто процес передвиборчого опитування як складової формування оцінки передвиборчих вподобань учасників.

У другому розділі наведено функціональні вимоги до проєктованої платформи, формальний опис структурних модулів та елементів.

У третьому розділі наведено результати розробки онлайн платформи для реалізації механізму електронного голосування, дизайн форм платформи, програмна реалізація клієнтської та серверної частин платформи.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ЗАСАДИ ФУНКЦІОНУВАННЯ ВИБОРЧИХ ПРОЦЕСІВ ТА ГОЛОСУВАННЯ

1.1 Роль голосувань в історії суспільного розвитку

Конституція України (ст. 5 ч. 2) гарантує народу право здійснення владних повноважень і визнає народ носієм суверенітету і єдиним джерелом влади [8]. Реалізація цього невід'ємного права відбувається шляхом запровадження різноманітних форм та механізмів прямої (безпосередньої) демократії. Такими формами визначено: вибори, референдум, громадські слухання, народна ініціатива, загальні збори громадян та інші не заборонені законом форми волевиявлення народу, виявлення громадської думки тощо, які характеризуються голосуванням, обговоренням, проведенням зборів. Одним із найважливіших об'єктів законодавчого реформування та предметом багатьох політичних дискусій є інститут референдуму. Зазначимо, що усі форми використовують процедуру голосування, яке має глибокі історичні корені та історію розвитку в становленні суспільної думки.

Від проведення голосування залежить майбутній розвиток траєкторії суспільства та вибір варіантів розв'язку проблемних рішень. Якість голосувань впливає на взаємовідносини в суспільстві, оскільки формує процедуру прийняття рішення як формальну групову дію, наслідком якої є результат вибору населенням одного або декількох рішень для подальшої діяльності. Окремою галуззю, що провадить дослідження результатів виборів та обробку статистичних виборчих даних є псефологія. Однією із проблем демократичних виборів є можливість їх спотворення махінаціями та просуваннями недостовірних результатів, вирішення якої потребує проведення виборчої реформи, з детальним алгоритмом процесів запровадження чесних виборчих систем там, де вони відсутні, підвищення справедливості чи ефективності існуючих систем.

Історично голосування було звичайною традицією, за допомогою якої

функціонувала сучасна традиційна демократія починаючи з 17 століття. Голосування можуть займати посади в законодавчій, іноді у виконавчій і судовій владі, а також у регіональних і місцевих органах влади. Цей процес також використовується в багатьох інших приватних і бізнес-організаціях, від клубів до добровільних асоціацій і корпорацій. [3, 12]

В теперішній час використання процедури голосування як рішення вибору людей, котрі були би чиновниками, дуже відрізняється від стародавніх часів древньої Греції, де у виборах був присутнім значний вплив олігархічних груп та більшість посад розподілялась шляхом жеребкування.

Голосування використовувалися в історії Стародавньої Греції та Стародавнього Риму, а також протягом Середньовічного періоду для обрання таких правителів, як імператор Священної Римської імперії і Папа. У ведичний період в Індії раджа ґани (племені) обирався ґаною. Раджа належав до класу воїнів і, як правило, був сином попереднього раджі. Однак останнє слово на його виборах мали члени ГАНА. Під час періоду Сайгам люди обирали своїх представників, віддаючи голоси, а урни для голосування (зазвичай горщик) були перев'язані мотузкою та запечатані. Після виборів голоси відбирали та підраховували.

Король Пала Гопала у ранній середньовічній Бенгалії був обраний групою феодальних вождів. Такі вибори були досить поширеними у тогочасних суспільствах регіону. В імперії Чола (920 р.н.е.), в Утірамерур (на території сучасного Таміл Наду) пальмове листя використовувалося для вибору членів сільських комітетів. Листя з написаними на них іменами кандидатів клали в глиняний горщик. Для вибору членів комісії, хлопця просили взяти стільки листків, скільки було доступних місць.

Перші зареєстровані всенародні вибори чиновників на державні посади більшістю голосів, де всі громадяни мали право як голосувати, так і обіймати державні посади, відносяться до ефорів Спарти (754 р.д.н.е.), за змішаного правління спартанської Конституції.

Афінські демократичні вибори, де всі громадяни могли обіймати державні

посади, не були введені ще 247 років, аж до реформ Клісфена. Усі афінські громадяни мали право голосу в народних зборах, з питань закону та політики, а також були присяжними, але лише три найвищі класи могли голосувати на виборах. Згідно з реформами Солона, найнижчі з чотирьох класів афінських громадян (що визначаються ступенем їх багатства та майна, а не походженням) не мали права обіймати державні посади.

Питання виборчого права, особливо виборчого права для груп меншин, домінували в історії виборів. Чоловіки, домінуюча культурна група в Північній Америці та Європі, часто домінували в електораті. На дострокових виборах у таких країнах, як Велика Британія та Сполучені Штати, домінували представники земельного чи правлячого класу. Проте, до 1920 року всі західноєвропейські та північноамериканські демократії мали загальне виборче право для дорослих чоловіків (за винятком Швейцарії), і багато країн почали розглядати можливість виборчого права для жінок. Незважаючи на законодавче загальне виборче право для дорослих чоловіків, іноді зводилися політичні бар'єри, щоб перешкодити справедливому доступу до виборів [4, 5].

Вибори проводяться в різних політичних, організаційних і корпоративних умовах. Багато країн проводять вибори, щоб обирати людей в уряди, але чимало організацій використовують вибори як внутрішній стимул для отримання варіанту рішення конкретної проблеми. Так, корпоративні вибори серед акціонерів формують склад та впливають на обрання членів ради директорів і можуть бути передбачені статутними корпоративними документами. Вибори в корпораціях та інших організаціях в більшості використовують процедури та правила, подібні до урядових виборів.

Одним із центральних питань в процедурі виборів є питання про тих осіб, що можуть брати участь у виборах. Електорат зазвичай не включає все населення. Так, багато країн забороняють голосувати особам, які не досягли повноліття (усі юрисдикції вимагають мінімального віку для голосування). В Австралії аборигени не мали права голосу до 1962 року, а в 2010 році федеральний уряд позбавив права голосувати ув'язнених, які відбували

покарання протягом 3 років і більше.

Виборче право зазвичай надається лише громадянам конкретної країни, хоча можуть бути встановлені додаткові обмеження. Проте, в Європейському Союзі можна голосувати на муніципальних виборах, якщо ви проживаєте в муніципалітеті та є громадянином ЄС; громадянство країни проживання не вимагається. У деяких країнах голосування вимагається законом. До тих, хто має право голосу, можуть бути застосовані такі заходи покарання, як штраф за непроголосування. У Західній Австралії покаранням для порушника, який вперше не проголосував, є штраф у розмірі 20 доларів США, який збільшується до 50 доларів США, якщо порушник раніше відмовився голосувати.

Історично, кількість виборців з правом голосу була невеликою за розміром груп або спільнот привілейованих людей (аристократи та жителі міст). Із зростанням числа буржуазії за містами та розширенням поняття громадянина, кількість виборців почала стрімко зростати (перевищувала тисячі). Вибори з сотнями тисяч виборців з'явилися в останні десятиліття Римської республіки. У Королівстві Великобританія в 1780 році було близько 214 000 виборців, що становило 3% від усього населення.

Виборче право реалізують через представницьку та пряму демократію. Представницька демократія вимагає процедури висування на політичну посаду. У багатьох випадках висування на посаду відбувається через процеси попереднього відбору в організованих політичних партіях. Позапартійні системи, як правило, відрізняються від партійних систем щодо висунення. У прямій демократії, може бути висунута будь-яка відповідна особа. Витоки виборів у сучасному світі лежать у поступовій появі представницького правління в Європі та Північній Америці, починаючи з 17 століття. У деяких системах висування не відбувається взагалі, а виборці можуть вільно обирати будь- яку особу під час голосування. У таких випадках не вимагається, щоб члени виборчого корпусу були знайомі з усіма особами, які мають на це право, хоча такі системи можуть включати непрямі вибори на більших географічних рівнях, щоб забезпечити певне знайомство серед потенційних обранців. В деяких

країнах можна висувати лише членів конкретної партії. [6]

Виборчі системи – це детальні конституційні механізми та системи голосування, які перетворюють голосування на політичне рішення. [9, 11]. Першим кроком є проголосування бюлетенів, які можуть бути простими бюлетенями з одним варіантом, але також можуть використовуватися інші типи: бюлетені з множинним вибором або ранжовані бюлетені. Потім відбувається підрахунок голосів, де можуть бути задіяними системи підрахунку. І потім система голосування визначає результат. Розрізняють виборчі системи такі, як: пропорційні, мажоритарні або змішані. Серед пропорційних систем використовуються системи пропорційного представництва за партійними списками (PR за списками), серед мажоритарних – мажоритарна виборча система (голосування за мажоритарного голосування за одного переможця) та різні методи мажоритарного голосування (наприклад, широкомасштабне голосування). Змішані системи поєднують елементи як пропорційного, так і мажоритарного методів, деякі з них дають результати, ближчі до першого (змішана пропорційна система) або іншого (паралельне голосування).

У багатьох країнах зростають рухи за реформування виборчого процесу. Відкритість і підзвітність виборчого процесу вважаються наріжними каменями демократичної системи. Таємне голосування вважається ключовим у більшості вільних і чесних виборів, оскільки обмежує ефективність залякування.

Виборча кампанія є елементом супроводу проведення виборів, шляхом проведення яких відбувається політичний вплив та боротьба за голоси виборців. Вартість найдорожчої виборчої кампанії становила 7 млрд. дол. США, витрачених на президентські вибори в Сполучених Штатах Америці у 2012 році.

Природа демократії обумовлює підзвітність виборних посадових осіб народу. Таким чином, вони повинні звертатись до виборців через встановлені проміжки часу, щоб отримати свій мандат, щоб продовжити свою посаду. З цієї причини більшість демократичних конституцій передбачають часову регулярність проведення виборів. У Сполучених Штатах вибори на державні посади проводяться кожні два-шість років у більшості штатів і на федеральному

рівні, за винятком виборних суддів, які можуть мати більший термін повноважень. Існують різноманітні розклади, наприклад президенти: президент Ірландії обирається кожні сім років, президент Фінляндії кожні шість років, президент Франції кожні п'ять років, президент США кожні чотири роки.

Недемократичність виборів пов'язують із невідповідністю їх міжнародним стандартам «вільності та чесності», що є притаманним країнам зі слабким верховенством права. При цьому диктатори можуть використовувати повноваження виконавчої влади (поліція, воєнний стан, цензура, фізична реалізація виборчого механізму тощо), щоб залишатися при владі, незважаючи на загальну думку при їхнє усунення від влади. Члени певної фракції в законодавчому органі можуть використовувати владу більшості або надбільшості (приймаючи кримінальні закони, визначаючи виборчі механізми, включно з правомочністю та межами округів), щоб запобігти переміщенню балансу влади в органі до конкуруючої фракції через вибори. Неурядові організації також можуть втручатися у вибори шляхом застосування фізичної сили, словесного залякування або шахрайства, що може призвести до неналежного підрахунку голосів. Перевірка та захист від фальсифікації голосувань на виборах – є постійне завдання демократичних країн з традиціями вільних і чесних виборів.

Проблеми, які заважають виборам бути «вільними та чесними», мають різні форми:

1. Відсутність відкритих політичних дебатів чи поінформованого електорату. Електорат може бути погано поінформований про проблеми чи кандидатів через відсутність свободи преси, відсутність об'єктивності в пресі через державний чи корпоративний контроль або відсутність доступу до новин і політичних ЗМІ.

2. Несправедливі правила. Недопущення кандидатів від опозиції до кандидатів, непотрібно високі обмеження щодо того, хто може бути кандидатом, правила доступу до бюлетенів, і маніпулювання порогами успіху на виборах – це лише деякі способи зміни структури виборів на користь певної фракції.

3. Втручання в передвиборчі кампанії. Ті, хто перебуває при владі, можуть заарештовувати кандидатів, придушувати або криміналізувати передвиборчу кампанію, закривати виборчі штаби, переслідувати або бити працівників кампанії або залякувати виборців насильством.

4. Втручання у виборчий механізм. Це може включати фальсифікацію інструкцій виборців, порушення таємного голосування, вкидання бюлетенів, фальсифікацію машин для голосування, знищення законно поданих бюлетенів, придушення виборців, шахрайство з реєстрацією виборців, непідтвердження місця проживання виборців, шахрайство відображення результатів, застосування фізичної сили чи словесних закликів на виборчих дільницях.

5. Фальшиві вибори. Фіктивні вибори – це вибори, які проводяться виключно для показу, тобто без будь-якого значного політичного вибору чи реального впливу на результати виборів. Фальшиві вибори є звичною подією в диктаторських режимах, які відчувають потребу вдавати видимість публічної легітимності. Іноді лише одному затвердженому урядом кандидату дозволяється брати участь у фіктивних виборах без участі інших кандидатів що представляють опозицію, також інших кандидатів заарештовують з несправжніми претензіями перед виборами, щоб запобігти їх балотуванню. Бюлетені можуть містити лише один варіант «так», або у випадку простого запитання «так чи ні» силовики часто переслідують людей, які вибирають «ні», таким чином заохочуючи їх вибрати варіант «так». У деяких випадках фіктивні вибори можуть мати зворотний ефект проти правлячої партії, особливо якщо режим вважає, що вони достатньо популярні, щоб перемогти без примусу чи шахрайства.

1.2. Сутність тестувань та опитувань при процедурі голосування

Нині традиційний референдум зі стандартними скриньками для голосування та паперовими бюлетенями можна вважати лише одним із можливих варіантів реалізації принципу народовладдя, який, до того ж, має ряд суттєвих недоліків: ситуативність застосування, технічна складність організації, значні матеріально-фінансові витрати, зайва заполітизованість процесу

народного волевиявлення.

Опитування громадської думки – це опитування громадської думки людиною за певною сформованою вибіркою. Опитування призначені для представлення думок населення шляхом проведення серії запитань і подальшої екстраполяції загальних ознак у співвідношенні або в межах довірчих інтервалів. Людина, яка проводить опитування, називається опитувальником.

Першим відомим опитуванням громадської думки був підрахунок уподобань виборців, повідомлених у Telegram Messenger до президентських виборів 1824 року, які повідомляли, що Ендрю Джексон випередив Джона Квінсі на 335 голосів у боротьбі за пост президента Сполучених Штатів. У 1916 році «Літературний дайджест» розпочав національне опитування і передбачив обрання Вудро Вільсона президентом. Розіслано мільйони поштових листівок і просто підраховано повернення. У 1936 році, опитування 2,3 мільйона виборців показало, що Альф Лендон переможе на президентських виборах, але натомість Рузвельт був переобраний із переконливою перемогою. Дослідження Джорджа Геллапа показало, що помилка в основному спричинена упередженням участі, кому подобався Лендон то вони більше хотіли забрати свою листівку. До середині 20-го століття опитування з'явились маже у всіх демократичних країнах. В 30-ті роки 20-го століття відчувається потужній вплив на думку людей реклами. Багато підприємств зазнало фінансових проблем під час великої депресії, тому розвиток рекламних акцій та опитувань став для людей одним із джерел доходів.

Методологічну складову формування опитувань становлять механізми проведення опитувань та визначення вибірок. Протягом багатьох років опитування думки людей робили через телебачення або при особистої зустрічі на вулиці. З часом, при зміні технології змінювалися підходи. Так, через інтернет вдалося пришвидшити процес опитування великої кількості людей. Інтернет дозволяв організаторам опитувань та і тим, кого опитують, заощаджувати час, та підвищити рівень відкритості. З'явились програми для опитувань, які стали популярними. Серед них такі програми, як: YouGov, Angus Reid Public Opinion,

та Zogby. Існує можливість фільтрації учасників опитувань за різними критеріями, що дозволяє моніторити вподобання окремих слоїв населення. Проте, проблемний залишається питання залучення екпертів окремих напрямів, по яких проводять опитування. Нещодавно були запропоновані статистичні методи навчання для використання вмісту соціальних медіа (наприклад, повідомлень на платформі мікро блогів Twitter) для моделювання та прогнозування опитувань про намір голосувати.

Активно використовують опитування в сфері зв'язків з громадськістю. На початку 1920-х років експерти зі зв’язків з громадськістю описували свою роботу як вулицю з двостороннім рухом. Їх завдання полягало в тому, щоб представити громадськості неправильно тлумачені інтереси великих установ. Вони також оцінюють інтереси громадськості за допомогою опитувань.

Одним із підходів є проведення порівняльних опитувань за принципом оцінки з еталоном. Еталонне опитування є першим опитуванням у кампанії. Метою такого опитування є першочергова заявка кандидата на виборчу посаду. Це коротке та просте опитування ймовірних виборців. Порівняльне опитування служить для кількох цілей кампанії:

1. Надання кандидату уявлення про його становище на думку виборців до початку будь-якої кампанії. Якщо опитування проводиться до оголошення кандидата на посаду, кандидат може використати опитування, щоб вирішити, чи варто балотуватися на посаду.

2. Характеризує слабкі та сильні сторони сферах: вподобання електорату та ідеї виборів. Стосовно електорату, еталонне опитування показує, який тип виборців можуть найбільше залучити до виграшу і навпаки – до програшу. По-друге, надає уявлення про те, які меседжі, ідеї чи гасла є найсильнішими серед електорату.

Опитування Brushfire – це опитування, які проведені в період між опитуванням порівняльного рівня та опитуваннями відстеження. Кількість опитувань, проведених кампанією, залежить від того, наскільки конкурентоспроможними є виборчі перегони та який бюджет кампанії. Дані

опитування зосереджуються на вірогідних виборцях, а тривалість опитування залежить від кількості перевірених повідомлень. Brushfire опитування використовуються для багатьох цілей:

1. Можливість отримати інформацію для кандидата про прогрес на виборах та ознаки демографічних показників, де він досягав або втрачав позиції.

2. Перевірка різноманітних меседжів на собі та опонентах, що дозволяє кампанії знати, які повідомлення найкраще працюють з певною демографією, а яких повідомлень слід уникати. Такі опитування використовують, для перевірки можливих повідомлень про атаку опоненту та потенційну відповідь на них.

3. Метод переконати основних претендентів для виходу з перегонів і підтримки сильнішого кандидата.

Опитування з відстеженням (постійне опитування) – це опитування, в якому відповіді отримують протягом декількох послідовних періодів, а потім результати обчислюються за допомогою ковзного середнього по відповідях, які були зібрані за фіксовану кількість останніх періодів. Однак, ці опитування іноді зазнають різких коливань. Прикладом опитування відстеження, яке викликало суперечки неточності, є опитування, проведене організацією Gallup під час президентських виборів у США в 2000 році. Результати за один день показали, що кандидат від Демократичної партії Ел Гор випереджає кандидата від Республіканської партії Джорджа Буша на одинадцять балів. Тоді наступне опитування, проведене лише через два дні, показало, що Буш випереджає Гора на сім пунктів. Невдовзі було встановлено, що мінливість результатів була спричинена нерівномірним розподілом виборців у вибірках, пов'язаних з Демократичною та Республіканською партіями.

Для уникнення та пояснення помилкових результатів опитувань були запропоновані низка теорій і механізмів, деякі з яких відображають помилки опитувальників. Опитування, що ґрунтуються на вибірках населення, є джерелом виникнення помилок вибірки, що відображає вплив випадковості та невизначеності в процесі її формування. Вибіркові опитування покладаються на закон великих чисел для вимірювання думок усієї сукупності лише на основі

підмножини, і для цієї мети важливий абсолютний розмір вибірки, але відсоток усієї сукупності не важливий. Можлива різниця між вибіркою та всією генеральною сукупністю часто ототожнюється з межею похибки – визначається як радіус 95% довірчого інтервалу для конкретної статистики. Коли для опитування повідомляється про єдину глобальну похибку, це стосується максимальної похибки для всіх відсоткових показників, заявлених із використанням повної вибірки з опитування. Якщо статистикою є відсоток, цю максимальну похибку можна розрахувати як радіус довірчого інтервалу для зареєстрованого відсотка 50%. Інші припускають, що опитування з випадковою вибіркою з 1000 осіб має похибку вибірки ±3% для розрахункового відсотка всього населення. Похибка 3% означає, що якщо одна й та сама процедура використовується багато разів, у 95% випадків справжнє середнє значення сукупності буде в межах оцінки вибірки плюс-мінус 3%. Похибку можна зменшити, використовуючи більшу вибірку, однак, якщо опитувальник бажає зменшити похибку до 1%, йому знадобиться вибірка приблизно з 10 000 осіб. На практиці опитувачам необхідно збалансувати вартість великої вибірки та зменшення помилок вибірки, а розмір вибірки приблизно 500 – 1000 є типовим компромісом для політичних опитувань. [10]

Один із способів зменшення похибки – орієнтація на середні результати опитування. Це робить припущення, що процедура досить подібна між багатьма різними опитуваннями, і використовує розмір вибірки кожного опитування для створення середнього опитування. Іншим джерелом помилок є хибні демографічні моделі опитувальників, які зважують свої вибірки за певними змінними, такими як ідентифікація партії на виборах.

Вибірки опитування можуть не бути репрезентативними через упередженість відсутності відповідей. Рівень відповідей знижувався і впав приблизно до 10% в останні роки. Через таку упередженість відбору характеристики тих, хто погоджується на інтерв'ю, можуть помітно відрізнятися від тих, хто відмовляється. Тобто фактична вибірка є упередженою. Упередження вносить нові помилки, які є доповненням до помилок, які

викликані розміром вибірки. Помилка, спричинена упередженням, не зменшується зі збільшенням розміру вибірки, оскільки вибірка більшого розміру просто повторює ту саму помилку у більшому масштабі. Якщо люди, які відмовляються відповідати, мають ті самі характеристики, що й люди, які відповідають, тоді остаточні результати мають бути неупередженими. Якщо люди, які не дають відповіді, мають різні думки, то в результатах є упередженість. Що стосується опитувань виборів, дослідження показують, що ефект упередженості невеликий, але кожна фірма, що займається опитуваннями, має власні методи коригування вагових коефіцієнтів, щоб мінімізувати упередження відбору. [2]

На результати опитування може вплинути упередженість відповідей, коли відповіді респондентів не відображають їхні справжні переконання. Респонденти можуть навмисно намагатися маніпулювати результатами опитування. Респонденти також можуть відчувати соціальний тиск, щоб не давати непопулярну відповідь. Так, респонденти можуть не бажати визнавати такі непопулярні погляди, як расизм або сексизм, і тому опитування можуть не відображати справжню поширеність цих поглядів серед населення (ефект Бредлі). Використання системи голосування за мажоритарністю (один кандидат) під час опитування створює ненавмисне упередження в опитуванні, оскільки люди, які віддають перевагу більш ніж одному кандидату, не можуть цього вказати. Той факт, що вони повинні вибрати лише одного кандидата, упереджує опитування, змушуючи його віддавати перевагу кандидату, який найбільше відрізняється від інших, і віддавати перевагу кандидатам, схожим на інших кандидатів.

Серед факторів, які впливають на результати соціологічних опитувань, є чіткий вибір та порядок які є запитання, що ставить опитувач. Питання, які навмисно впливають на відповідь респондента, називаються навідними. Окремі особи та групи використовують такі типи запитань в опитуваннях, щоб отримати відповіді, які відповідають їхнім інтересам. Навідні запитання часто містять або не містять певних фактів, які можуть вплинути на відповідь респондента.

Головні запитання, що мають аргументи то вони можуть вплинути на результат опитування. Ці різні варіанти запитань, залежно від їх характеру, позитивні чи негативні, можуть змінити думку та вибір за що голосувати, щоб відобразити тон запитання і започаткувати певну відповідь чи реакцію, а не оцінювати настрої неупередженим чином. В опитуваннях існують «завантажені питання», інакше відомі як «підступні питання». Цей тип навідного запитання може стосуватися для опису незручної або суперечливої теми та автоматично припускати, що тема запитання пов'язана з респондентом. Подібним чином питання формуються таким чином, що обмежують можливі відповіді, так чи ні. Іншим типом запитань, які можуть дати неточні результати, є «подвійно-негативні запитання». Це частіше результат людської помилки, а не навмисної маніпуляції. Тому порівняння опитувань часто зводяться до формулювання питання. У деяких питаннях формулювання запитань може призвести до досить виразних відмінностей між опитуваннями. Однак це також може бути результатом законно суперечливих почуттів або зміни ставлення, а не погано побудованого опитування. Загальною технікою контролю за упередженням є чергування порядку постановки запитань. Можливим є поділ вибірки, що передбачає наявність двох різних версій запитання, кожна з яких представлена половині респондентів. Найефективнішими засобами контролю запитань є:

1. Задавати достатню кількість запитань, щоб дозволити охопити всі аспекти проблеми та контролювати вплив через форму запитання (наприклад, позитивне чи негативне формулювання), адекватність кількості, встановленої кількісно за допомогою психометричних показників, таких як коефіцієнти надійності.

2. Аналіз результатів за допомогою психометричних методів, які синтезують відповіді в кілька надійних балів і виявляють неефективні питання. Ці засоби контролю не використовуються широко в галузі опитувань. Емпіричні тести дають зрозуміти якість анкети, деякі можуть бути складнішими за інші. Так, тестування анкети можна виконати за допомогою:

1. Проведення когнітивного інтерв'ю.

2. Проведення невеликого попереднього тестування анкети з використанням малої підгрупи цільових респондентів. Результати можуть повідомити дослідника про помилки (відсутність питань, логічні та процедурні помилки).

3. Оцінка якості вимірювання питань. Проводиться за допомогою моделей типу «тест – повторний тест», квазісимплекс або багатометодних моделей.

4. Прогнозування якості вимірювання питання. Проводиться за допомогою програмного забезпечення Survey Quality Predictor (SQP).

Джерелом помилок також є використання вибірок, які не є репрезентативними для населення внаслідок методології, яка використовується. Така помилка може з'являтись в телефонних опитуваннях. Подекуди багато людей мають лише мобільні телефони. Оскільки не може бути використаний автоматичний дозвін (для дзвінків на мобільні телефони), то такі особи виключаються з вибірки опитувань. Також може бути різниця між вибірками опитуваних. Потенційними джерелами упередженості в телефонних опитуваннях є:

1. Наявність мобільних телефонів і відсутність стаціонарного зв'язку. Це включає меншини та молодих виборців і частіше зустрічається в мегаполісах. Чоловіки частіше користуються лише мобільним телефоном, ніж жінки.

2. Невизначеність термінів встановлення зв'язку по стаціонарному телефону. З деякими людьми неможливо зв'язатися по стаціонарному телефону з понеділка по п'ятницю, а лише по мобільному телефону.

3. Використання стаціонарних телефонів для вузької мети, наприклад, лише для доступу до Інтернету та відповіді на дзвінки проводяться лише на мобільний телефон.

Прикладом опитувань громадської думки, що пов'язані з помилками, є опитування під час загальних виборів у Великобританії 1992 року. Так, усі опитування, проведені перед голосуванням, показали перевагу опозиційної Лейбористської партії, тоді як фактичне голосування забезпечило явну перемогу

правлячій Консервативній партії. Після обговорення було висунуто кілька ідей для пояснення помилок, зокрема:

1. Пізнє коливання: виборці, які передумали незадовго до голосування, віддавали перевагу консерваторам, тому помилка була не такою великою, як здавалося спочатку.

2. Упередженість відсутності відповіді: виборці-консерватори рідше брали участь в опитуваннях, ніж у минулому, і тому були недостатньо представлені.

3. «Фактор сором'язливих торі»: консерватори пережили тривалий період непопулярності внаслідок економічних труднощів і серії незначних скандалів, що призвело до спіралі мовчання, під час якої деякі прихильники консерваторів не бажали розкривати свої щирі наміри при опитуванні.

На сьогодні соціальні мережі можуть виступати як джерело та індикатор думки про кандидатів при опитуваннях. Деякі дослідження показали, що прогнози, зроблені за допомогою сигналів соціальних мереж, можуть збігатися з традиційними опитуваннями громадської думки. Докази показують, що соціальні медіа відіграють велику роль у постачанні новин. Цей факт робить проблему фейкових новин у соціальних мережах більш актуальною. Інші дані свідчать про те, що найпопулярніші фейкові новини ширше поширювалися у Facebook, ніж найпопулярніші основні новини.

Надаючи інформацію про наміри голосування, опитування громадської думки іноді можуть впливати на поведінку виборців. Опитування громадської думки є засобом впливу на громадську думку. Різноманітні теорії про те, як це відбувається, можна розділити на дві групи: ефекти перемоги/аутсайдера та стратегічне (тактичне) голосування. Ефект перемоги виникає, коли опитування спонукає виборців підтримати кандидата, який перемагає в опитуванні. Ідея про те, що виборці сприймають такі впливи, є старою, вона походить з 1884 року. Протилежним до ефекту перемоги є ефект аутсайдера. Він спостерігається, коли люди голосують із симпатії за партію, яку вважають «програшною» на виборах.

Друга категорія теорій про те, як опитування безпосередньо впливають на голосування, називається стратегічним або тактичним голосуванням. Ця теорія

базується на ідеї, що виборці розглядають акт голосування як засіб обрання уряду. Таким чином, іноді вони обиратимуть не кандидата, якому віддають перевагу через ідеологію чи симпатії, а іншого, менш бажаного кандидата зі стратегічних міркувань. Виділяють три додаткові «поведінкові» відповіді, які виборці можуть продемонструвати, стикаючись з даними опитувань [1]:

1. Ефект «підказки» – дані опитування використовуються як «проксі» для інформації про кандидатів або партії. Взяття репліки «базується на психологічному феномені використання евристики для спрощення складного рішення».

2. Теорія «когнітивної реакції». Ця теорія стверджує, що відповідь виборця на опитування може не відповідати його початковому уявленню про виборчу реальність. У відповідь виборець створить «розумовий список», у якому вони створять причини програшу чи перемоги партії на виборах. Це може посилити або змінити їхню думку про кандидата і таким чином вплинути на поведінку при голосуванні.

3. «Поведінкова реакція» – відмінність від когнітивної реакції полягає в тому, що виборець буде шукати нову інформацію для формування свого «ментального списку», таким чином стаючи більш поінформованим про вибори. Ці ефекти показують, як опитування громадської думки можуть безпосередньо впливати на політичний вибір електорату. Але прямо чи опосередковано інші впливи можна вивчити й проаналізувати для всіх політичних партій. Слід також враховувати форму оформлення ЗМІ та зміни партійної ідеології. У деяких випадках опитування громадської думки є мірою когнітивного упередження, яке по-різному враховується та обробляється належним чином у різних застосуваннях.

Деякі юрисдикції по всьому світу обмежують публікацію результатів опитувань громадської думки, особливо в період перед виборами, щоб запобігти впливу помилкових результатів на рішення виборців. Наприклад, у Канаді заборонено публікувати результати опитувань громадської думки, які б ідентифікували конкретні політичні партії чи кандидатів, протягом останніх

трьох днів до закриття голосування. Однак більшість західних демократичних країн не підтримують повну заборону на публікацію передвиборчих опитувань громадської думки; більшість із них не мають жодного регулювання, а деякі забороняють це лише в останні дні або години до закриття відповідного голосування.

1.3. Електронне голосування та сучасні рішення його реалізації

Електронне голосування – є участю в публічних опитуваннях, виборах, референдумах, що передбачає використання електронних засобів для ідентифікації та підрахунку голосів. Електронне голосування у віддаленому режимі надає однаковий доступ до виборів всім учасникам незалежно від місця їх перебування. Однак запровадження електронного голосування в існуюче законодавство є складним завданням, на виконання якого потрібно час і проведення певних реформ. [13]

Ресурс електронного голосування вимагає новітніх інформаційних технологій та програмного забезпечення, що гарантують надійність передачі даних в Інтернеті як виборців, так і виборчих комісій. В якості такої технології пропонується використання під час голосування протоколу блокчейну. Блокчейн – це вибудований за певними правилами безперервний послідовний ланцюг, що містить інформацію про блоки. Кожен наступний блок послідовно перевіряє формат попереднього блоку, а сам ланцюг одночасно зберігається на багатьох комп’ютерах, незалежно один від одного. За таких умов технологія блокчейну робить неможливою фальсифікацію результатів голосування, оскільки всі несанкціоновані зміни даних будуть відомі всім користувачам. [45].

Реалізація електронного голосування можлива шляхом використання багатьох інформаційних ресурсів та платформ. Google Forms є потужним інструментом для збору та інформації (рис. 1.1).

На рисунку 1.1 наведено головну сторінку гугл форми, яка є адаптованою на всі мобільні пристрої, чим менше екран пристрою, тим буде менше контент, також шрифт автоматично масштабується при змінні екрану.

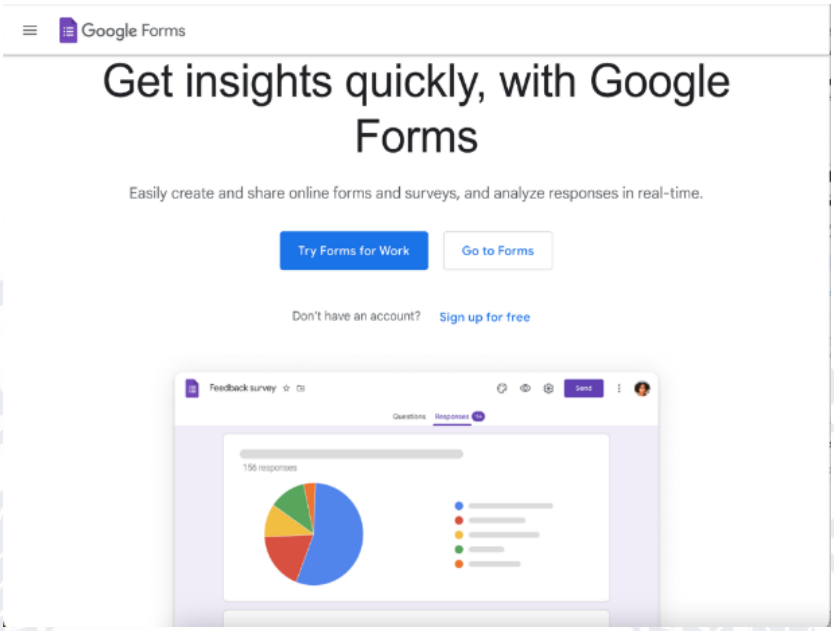


Рисунок 1.1 – Головна сторінка гугл форми

Приклад створювання голосування онлайн наведено на рисунку 1.2.

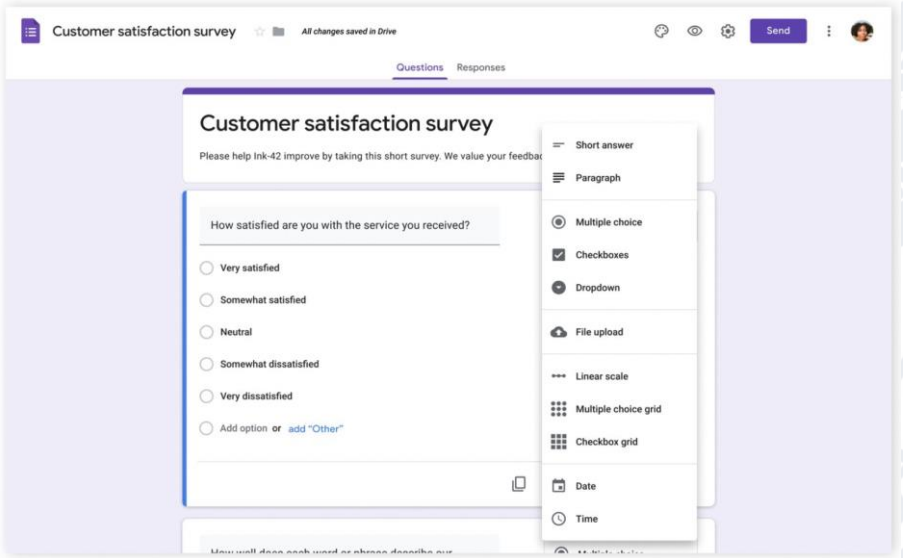


Рисунок 1.2 – Приклад створювання голосування онлайн

На рисунку 1.2 наведено результат, вибору із кількох типів запитань, для яких можливо змінити порядок через перетягування. Можливо змінювати вид голосування (рис. 1.3).

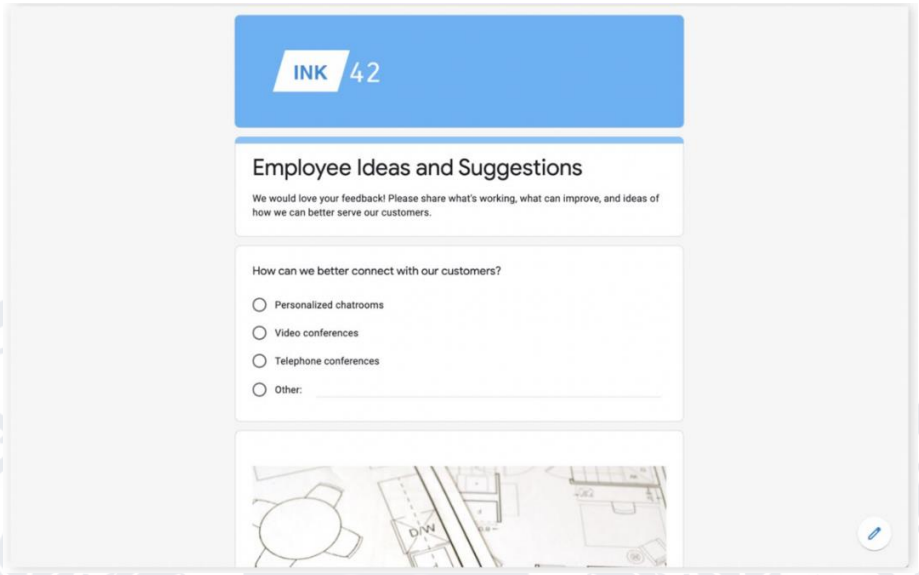


Рисунок 1.3 – Змінений документ для голосування

Гугл дозволяє змінювати вид голосування та опитувань на свій смак, тобто можна налаштувати кольори, зображення та шрифти, для зміни та формування зовнішнього вигляду або брендингу організації. Також можливо додати спеціальну логіку, яка показує запитання на основі відповідей (рис. 1.3).

Аналіз відповідей при використанні гугл форм наведено на рис. 1.4.

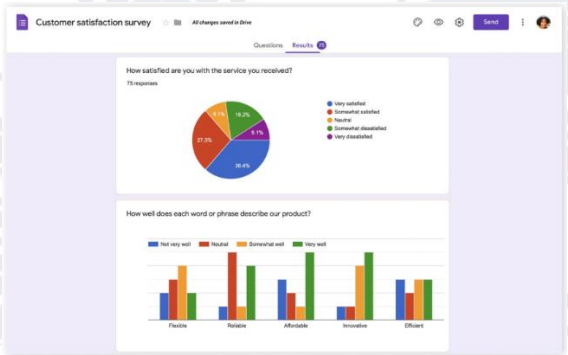


Рисунок 1.4 – Аналіз відповідей за допомоги автоматичних підсумків

Додаток дозволяє переглядати діаграми з оновленням даних відповідей у режимі реального часу або відкривати необроблені дані за допомогою Google Таблиць для глибшого аналізу чи автоматизації. Налаштування адаптації до

мобільного пристрою наведено на рис. 1.5.

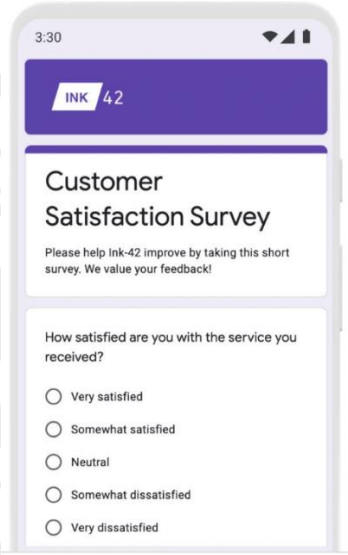


Рисунок 1.5 – Мобільна адаптація

Платформа дає можливість створювати голосування та опитування для досягнення кінцевої мети – отримання відповідей. та відповідайте на них будь-де це була кінцева мета Гугл форм. Власники, що створюють голосування, адаптують його до різних розмірів екранів, і що в кінцевому випадку є доступним з будь-якого мобільного пристрою, планшета чи комп’ютера.

ВИСНОВКИ ДО РОЗДІЛУ 1

Пряма демократія визначена як організаційний механізм державної влади, при якій базові (основні) рішення в суспільстві приймаються усіма громадянами через її основні форми. Вона може існувати як окремий дієвий механізм, так і частково вбудовані її елементи. Основними формами прямої демократії визначають: демократію участі та дорадчу демократію. Найбільш виразним проявом вибору народної думки є референдум. На сьогодні її розвиток досяг електронної демократії, сформувавши витоки концепції інформаційної теорії демократії.

Проведений аналіз історичних подій пов'язаних з процесами голосувань, підтвердив тезис, що голосування використовувалось спільнотами з давніх часів та було проявом волевиявлення людей в процесі прийняття важких рішень при виборах на посади та вирішенні суспільних проблем. Голосування може проводитись як на національному (державному) рівні, так і на рівні окремих підприємств та корпорацій. Для підприємств голосування має одне із базових застосувань в технологіях менеджменту, оскільки дозволяє залучити в процес вибору більшість працівників.

Значну роль в організації виборчих процесів та голосувань відіграють передвиборчі вподобання, або попередні оціночні прогнози подій (кандидатів). В такий спосіб попередню оцінку можливо проводити на основі організацій суспільних (експертних) опитувань, результати яких можуть бути оприлюдненими. Визнається неоднозначний позитивний вплив опитувань на результати голосувань.

В теперішній час діяльність людей використовує інформаційні технології, а процес презентацій особистих надбань проходить через цифрові платформи. Електронне голосування, дозволяє залучити в даний процес велику кількість учасників, зменшити час на обробку результатів, а також знизити кошти на проведення голосування в довгостроковій перспективі.

РОЗДІЛ 2
ФОРМАЛІЗАЦІЯ ТА ВИМОГИ ДО СТВОРЕННЯ ПЛАТФОРМИ ДЛЯ ГОЛОСУВАННЯ

2.1 Особливості проектування веб платформ і додатків

Архітектура веб-додатків – це структура високого рівня, яка визначає спосіб роботи, ефективності та масштабування продукту. На етапі вибору архітектури веб-додатків стоїть вибір серед різноманітних варіантів, доступних на ринку розробки програмного забезпечення. Чим більше з'являється нових імен і трендів, тим важче стає прийняти рішення. Ізоморфна, прогресивна, SPA чи SSR – яка архітектура сучасної веб-програми найкращі та які критерії використовувати для оцінки. Розглянемо основні типи зовнішніх архітектур, доступних для Інтернету, і пояснимо особливості їх реалізації.

Розглянемо, в чому полягає різниця між веб-додатком та веб-сайтом. Веб-додаток – це клієнт-серверна програма, де є браузер (клієнт) і веб-сервер. Логіка веб-додатку розподілена між сервером і клієнтом, є канал для обміну інформацією і сховище даних, розташоване локально або в хмарі. Веб-додатки з'явилися на етапі еволюції веб-сайтів і мають багато спільного. Факторами, які відрізняють веб-сайт від веб-додатку, є інтерактивність, інтеграція та автентифікація. Іншими словами, чим більш настроюваний, інтерактивний і функціональний веб-сайт, тим ближче його називають веб-додатком. Різницю в роботі веб-додатку та веб-сайту можна побачити на рис. 2.1 та рис. 2.2.

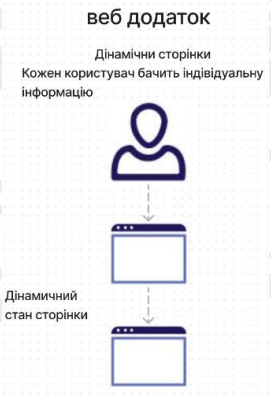


Рисунок 2.1 –Схема роботи веб-додатку

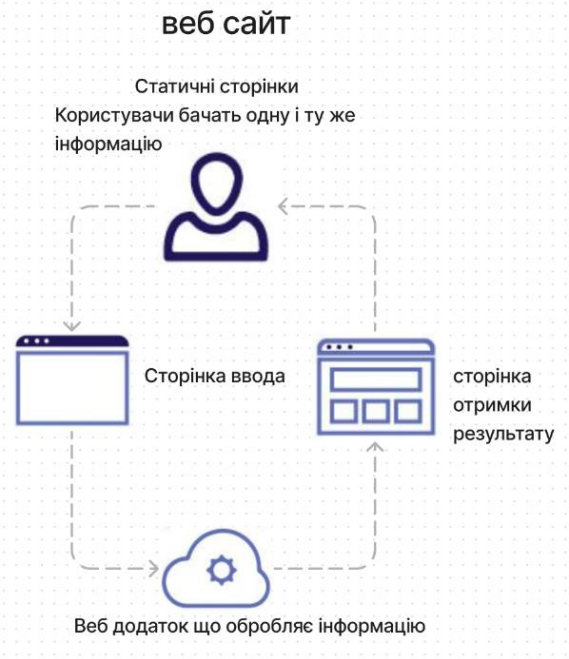


Рисунок 2.2 – Схема роботи веб-сайту

Розглянемо особливості трирівневої архітектури у веб-розробці. Сучасні веб-програми використовують концепцію 3-рівневої архітектури, що розділяє програми на рівень презентації, рівень програми та рівень даних (рис. 2.3).

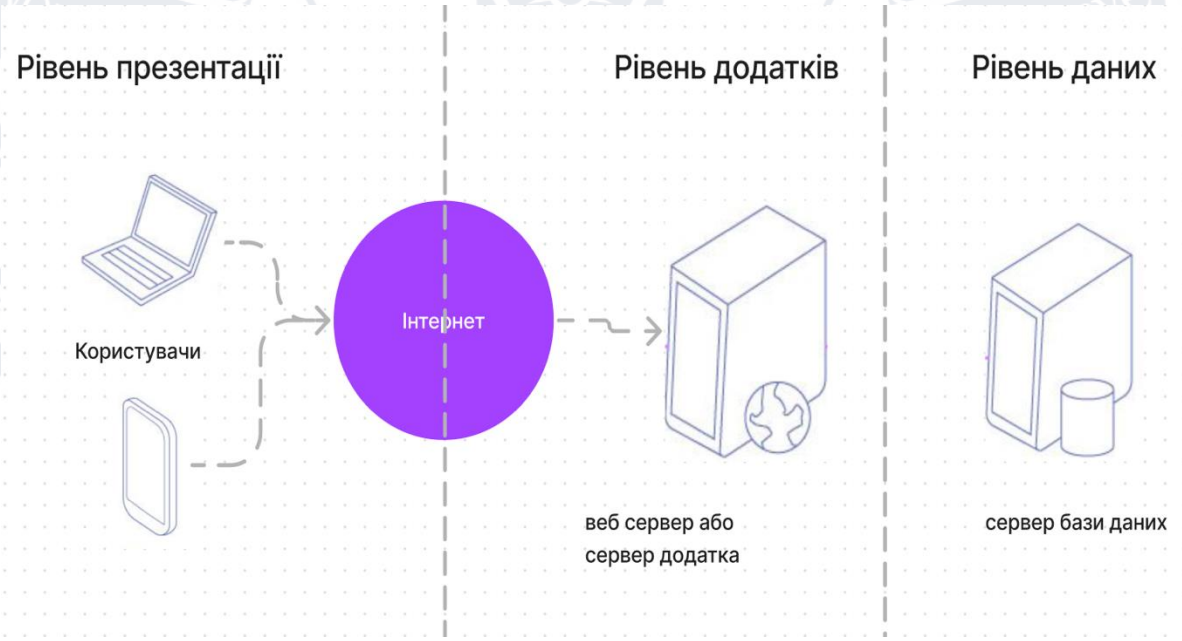


Рисунок 2.3 – Трьохрівнева архітектура веб-додатку

У 3-рівневій архітектурі веб-додатків кожен рівень працює на власній інфраструктурі та може розроблятися паралельно різними командами. Така структура дозволяє оновлювати та масштабувати кожен рівень за потреби, не впливаючи на інші рівні. [45]

Розглянемо, які основні типи архітектури веб-додатків. Щоб зрозуміти, чи дійсно запропонований підхід до вашого продукту відповідає потребам конкретного бізнесу, оцінюють сучасні типи архітектури веб-додатків відповідно до критеріїв, які вважають найважливішими: продуктивність, інтерфейс користувача, пошукова оптимізація, можливість зв'язування та швидкість реалізації. Схема роботи SSR наведена на рис. 2.4.

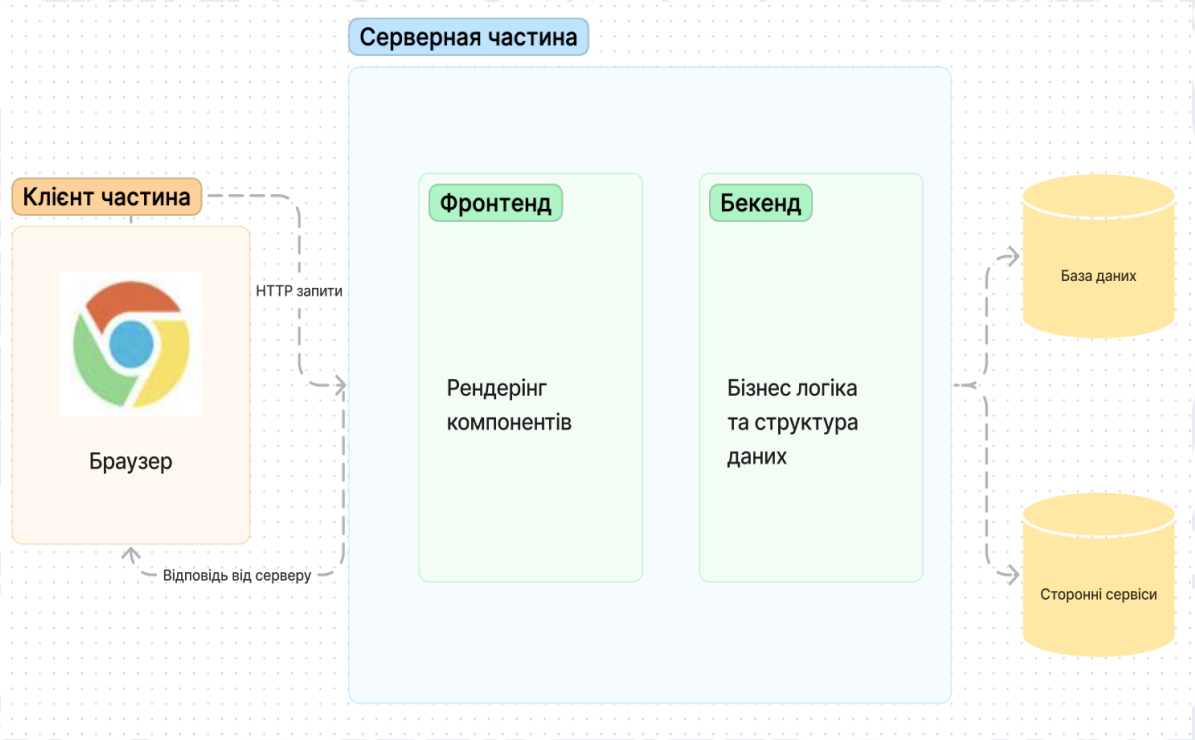


Рисунок 2.4 – Схема роботи SSR

Говорячи про основні веб-принципи, ми маємо на увазі архітектуру клієнт-сервер. Клієнт запитує вміст із сервера, де розташована бізнес-логіка та база даних. Використовуючи простий JavaScript, статична веб-сторінка надсилає запит до служби (можливо, API). Служба повертає дані та відображає клієнту

HTML-сторінку. Якщо програма відображується на стороні сервера, вміст отримується із сервера та передається в браузер для відображення користувачеві. Якщо сторінка HTML відображається на стороні сервера, користувач повинен перейти до сторінки, перш ніж браузер отримає сторінку з сервера. Це означає, що для відображення вмісту користувачеві потрібно більше часу. Для кешування вмісту сторінки ця схема часто отримується з Nginx, веб-сервером, який також можна використовувати як поштовий проксі та балансувальник навантаження. Той факт, що HTML відтворюється на сервері, забезпечує низку переваг, таких як SEO, можливість зв'язування та миттєве завантаження першої сторінки. Візуалізація на стороні сервера працює, коли JS вимкнено в браузері. Оскільки код обробляється на сервері, ніяких особливих вимог до браузера не висувається, що дозволяє миттєво виявляти помилки. Однак SSR не може обробляти великі запити до сервера (повторювані HTML, CSS), що призводить до повільного рендерингу під час завантаження сервера або повного перезавантаження сторінки. Проте, головним мінусом цього базового типу архітектури веб-додатків є погана взаємодія з кінцевим користувачем і неможливість створити повноцінний інтерфейс користувача. Іншими словами, SSR є простим і економічно ефективним шляхом, якщо потрібно створити простий веб-сайт. Реалізація цього типу архітектури можлива з будь-якою мовою програмування та серверною частиною.

Статична генерація сайту (SSG). У процесі генерації статичного сайту використовується генератор, який автоматизує кодування окремих HTML-сторінок, створюючи їх із шаблонів. Вибираючи статичну генерацію сайту, отримується простий статичний веб-сайт, розташований на CDN або будь-якому сервері, який містить уже згенеровану HTML-сторінку, яка буде надана користувачам за запитом [46]. Тому не потрібно генерувати його щоразу, коли хтось відвідує ваш веб-сайт – сервер просто надсилає вже наявні дані через API (рис. 2.5).

Цей підхід також має свої переваги та недоліки. Перш за все, цей підхід підходить лише для веб-сайтів. Крім того, вміст згенерованих сторінок веб-сайту

не змінюється, якщо не додавати нові дані чи компоненти. Це означає, що доведеться повністю генерувати веб-сайт, як тільки потрібно додати новий вміст. Це один є одним із головних недоліків, який серйозно обмежує використання бізнес-випадків, до яких він застосовується. Серед переваг, є висока швидкість статичного вмісту, який доставляється через CDN. Також у SSG усі серверні операції та робота з базою даних реалізуються через незалежний від сайту API. Цей варіант простий і тому виключно доступний для реалізації. Прикладами простих генераторів статичних сайтів є Jekyll і Hugo, тоді як Gatsby і VuePress підходять для реалізації більш складних рішень. [47]

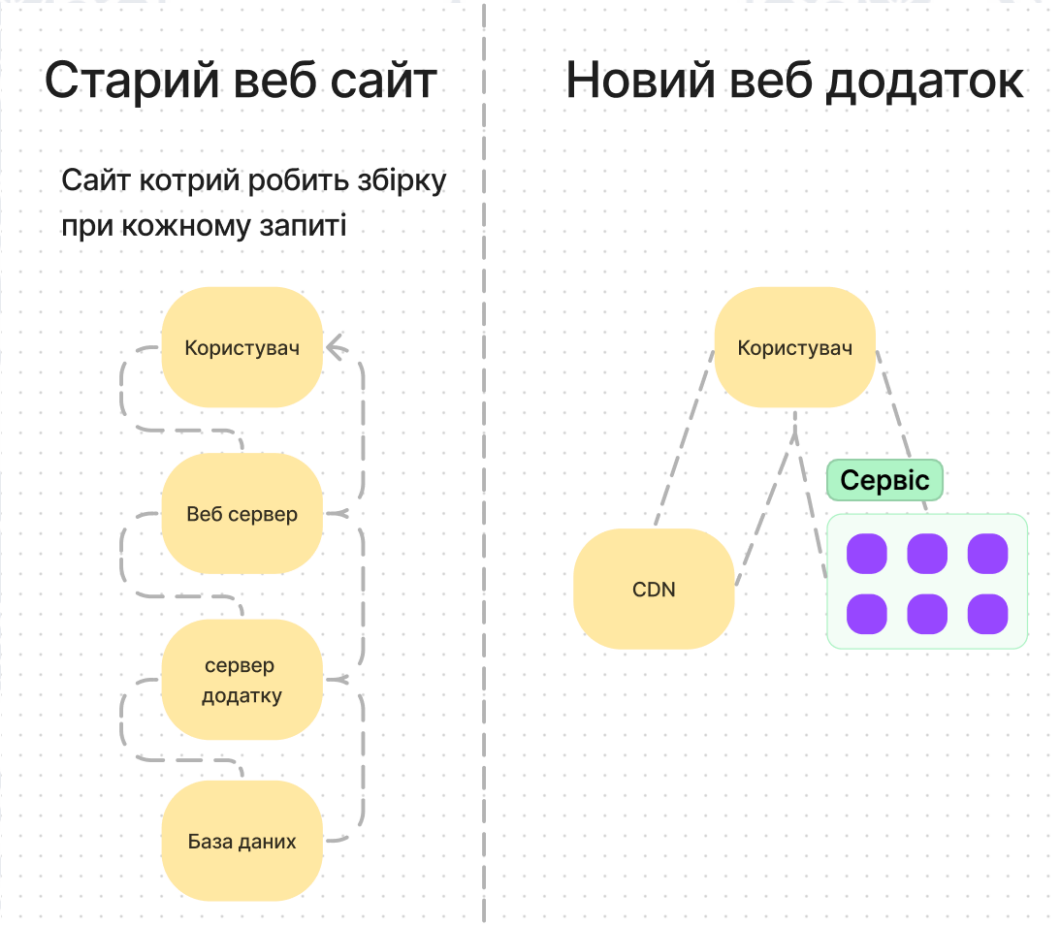


Рисунок 2.5 – Стара та нова версія веб платформ

Розглянемо сутність односторінкового додатку (SPA). SPA – це тип веб-програми, яка працює в браузері (рис. 2.6). Не потрібно перезавантажувати

сторінку, коли потрібно відобразити нові дані. Цей тип архітектури веб-додатків широко використовується в нашому повсякденному житті: Facebook, Gmail, Google Maps, GitHub і Twitter. На відміну від SSR і SSG, SPA дозволяє створити інтерактивну веб-програму. Він використовує API для зв'язку із сервером. Ця архітектура хороша для легкого масштабування продукту. Також, якщо потрібен мобільний додаток, не потрібно докладати додаткових зусиль для розробки API – мобільний додаток може використовувати той самий API, що й Web.

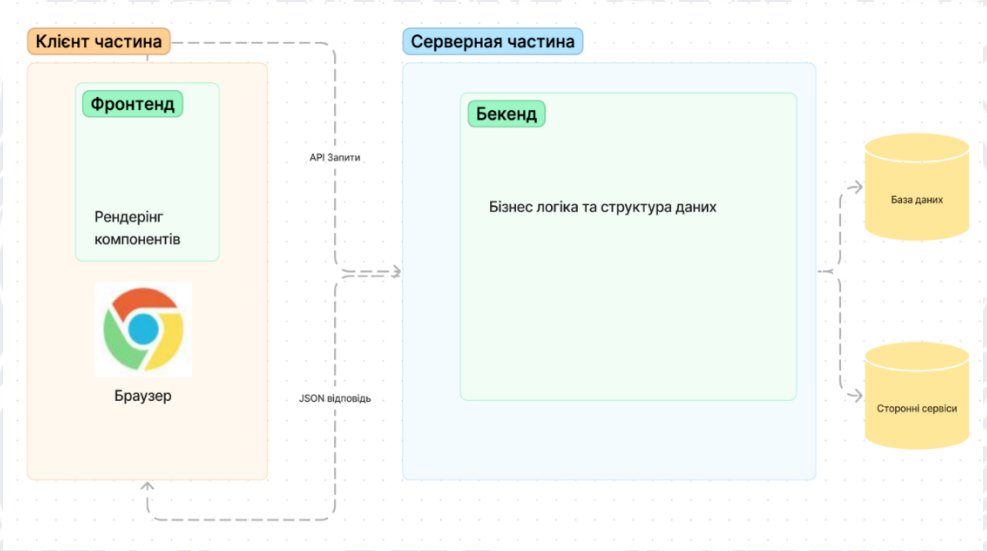


Рисунок 2.6 – Схема роботи SPA

SPA забезпечує швидку візуалізацію після повного завантажування програми у браузері та створює програмне забезпечення для кінцевого користувача з високою чутливістю. У той же час це «вбиває» ваше SEO і обмежує зв'язуваність, оскільки реалізація такого функціоналу потребуватиме додаткових зусиль. Серед інших недоліків: тривалий час, необхідний для першого завантаження, погана маршрутизація та обмежена підтримка застарілих браузерів. Будучи досить дорогим типом веб-архітектури, SPA підходить для створення адаптивного інтерфейсу користувача для користувачів B2C.

Прогресивний веб-додаток (PWA) – останні кілька років усі говорять про PWA [48]. Розглянемо його особливості і основні питання, що вирішуються при

розробки веб-додатків засобами даної архітектури. Прогресивна архітектура веб-програми використовує логіку односторінкової веб-програми з деякими службами, які запускаються далі у браузері. Це означає, що основним моментом, який слід взяти до уваги, є те, що і браузер, і ОС повинні підтримувати цей набір стандартів. Для кінцевого користувача прогресивна веб-програма фізично означає спливаюче вікно з пропозицією додати програму на екрані запуску (не в браузері, а на екрані операційної системи), коли він відвідує веб-сайт. Якщо користувач погоджується, програма автоматично додається на пристрій (рис. 2.7).

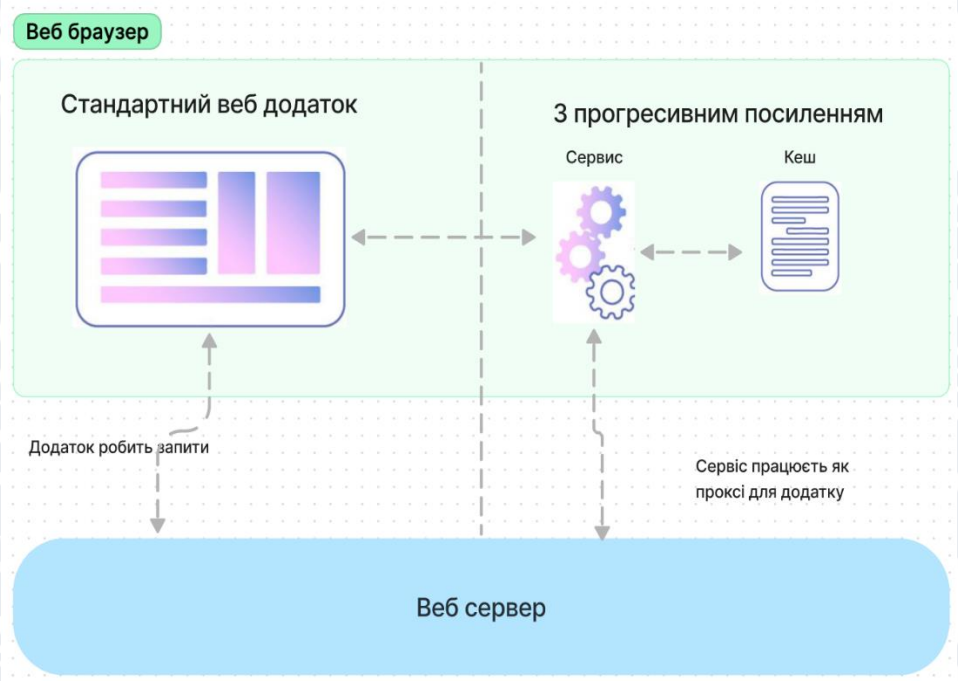


Рисунок 2.7 – Схема роботи PWA

Впровадження PWA [49] дозволяє веб-програмі підтримувати роботу в автономному режимі, фонову синхронізацію та push-сповіщення. Це відкриває доступ до функціональних можливостей, які раніше потребували рідної програми. При цьому, вибираючи архітектуру PWA для власного проекту, необхідно враховувати, що більшість функцій недоступні на iOS. Тобто, потрібно аналізувати кожен конкретний бізнес-кейс. Прогресивна архітектура

веб-додатків підтримується Windows, Android і iOS (проте для iOS офлайн-режим вимкнено). Розробники можуть додавати оновлення до веб-програми віддалено. PWA є безпечним, оскільки використовує HTTPS. У той же час кінцеві користувачі можуть встановити PWA, навіть не відвідуючи Play Market або App Store. Серед недоліків цього типу архітектури – необхідність вибору браузера та ОС, які його повністю підтримують.

Ще одним із видів сучасної архітектури веб-додатків є ізоморфна архітектура. Це тип програми JavaScript, яка може працювати як на стороні клієнта, так і на стороні сервера. Спочатку клієнт завантажує HTML, де програма JavaScript завантажується у браузер, а потім програма починає працювати як SPA (рис. 2.8).

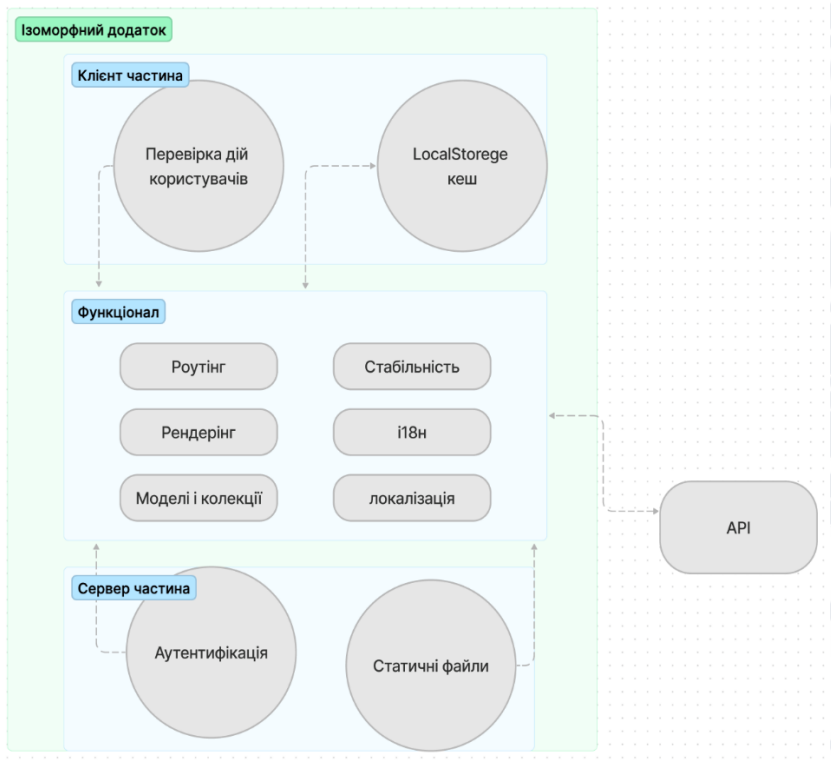


Рисунок 2.8 – Схема роботи ізоморфного додатку

На відміну від рендерингу на стороні сервера, ізоморфна веб-архітектура забезпечує швидке оновлення даних, швидкість реагування та численні параметри UI/UX. Це забезпечує швидший рендеринг під час завантаження

сервера, оскільки оброблений код передається клієнту. На відміну від візуалізації на стороні клієнта, це забезпечує миттєве відображення у веб-переглядачі, зручну маршрутизацію, пошукову оптимізацію та можливість зв'язування. Єдиним недоліком цього типу архітектури веб-додатків є те, що він повністю підтримується лише JavaScript. Найчастіше це означає, що технологічний стек для вибору обмежений фреймворками та інструментами на основі JS.

Серед інших принципів дизайну веб-додатків виокремлюють мікроінтерфейс – підхід, який базується на декомпозиції зовнішнього додатка на окремі «мікропрограми», що працюють разом. Для кінцевого користувача всі вони розташовані на одній сторінці. Схема роботи мікрофронтенду наведена на рис. 2.9.

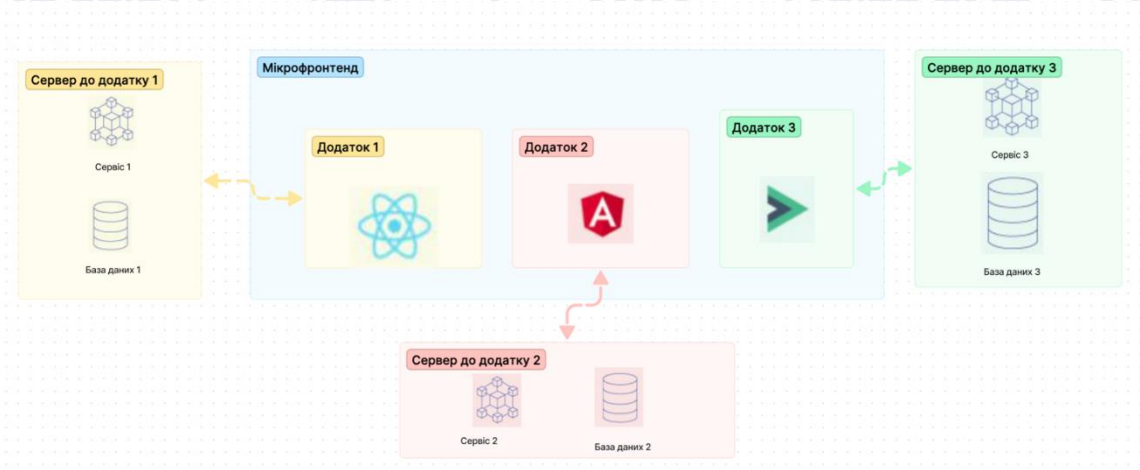


Рисунок 2. 9 – Схема роботи мікрофронтенду

Цей тип архітектури веб-програм є модульним. Це означає, що сторінки та віджети є повністю незалежними програмами. При такому підході розробка та розгортання йдуть паралельно. Але в той же час структура ускладнює додаток і спричиняє дублювання коду. Розглянемо концепцію Node.JS і новий веб-фронтенд. Ця концепція була вперше сформульована в 2013 році Ніколасом С. Закасом і зараз використовується для складних веб-рішень. «Відокремлення внутрішнього рівня інтерфейсу користувача від внутрішньої бізнес-логіки має

сенс у більшій веб-архітектурі. Чому інженерам інтерфейсу має бути важливо, яка мова на стороні сервера потрібна для виконання критично важливих для бізнесу функцій? Чому це рішення має витікати на рівень внутрішнього інтерфейсу користувача? Потреби фронтенду принципово відрізняються від потреб бекенда» – так Н.С. Закас пояснює головний принцип розробки цього типу веб-додатків [50]. Цей тип архітектури веб-програм складається з сервера на основі Node.js і рівня інтерфейсу користувача. Крім того, сервер бізнес-логіки може бути написаний будь-якою мовою (розглянемо PHP як приклад) і використовувати API для зв'язку з сервером.(рис. 2.10)

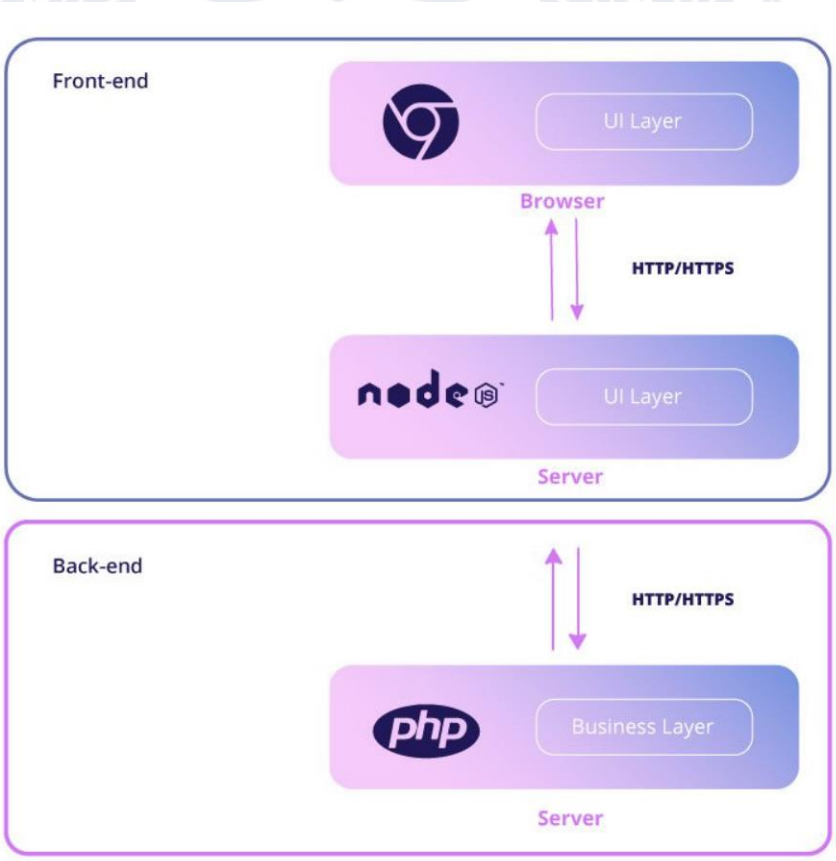


Рисунок 2.10 – Схема роботи нового виду веб додатка

Насправді новий підхід до веб-інтерфейсу дозволяє використовувати всі переваги типу ізоморфної архітектури, SSR або SPA, API для мобільних пристроїв і можливості зв'язування. На відміну від ізоморфних веб-програм, немає обмежень щодо мови чи платформи бізнес-логіки. Проте слід зазначити,

що розробка веб-програми, яка використовує нову логіку веб-інтерфейсу, займає більше часу, ніж розробка SSG або SSR. Щоб заощадити час на розробку, ми пропонуємо використовувати фреймворки Next.js і Nuxt.js. [51] Концепція нового веб-інтерфейсу підходить для рішень, коли йдеться про розробку ізоморфної веб-програми, де вже існує сервер API.

Хмарна архітектура для розробки веб-додатків. Говорячи про архітектуру веб-додатку, розглядають його серверну частину. Хмарна архітектура передбачає, що серверами керує хмарний провайдер, наприклад Amazon AWS, Azure або Google Cloud. Крім розміщення серверів на своїй стороні, ці провайдери пропонують комплекс послуг, які дозволяють створювати веб-додатки, розміщені та керовані в хмарі. Безсерверні послуги AWS є одним із найпопулярніших хмарних рішень, які використовуються для реалізації популярних шаблонів, таких як мікросервіси, мобільні серверні програми та односторінкові програми. Наведена на рисунку 2.11 схема дає можливість зрозуміти те, як веб-сервіси AWS використовують для створення веб-додатку за допомогою логіки 3-рівневої архітектури.

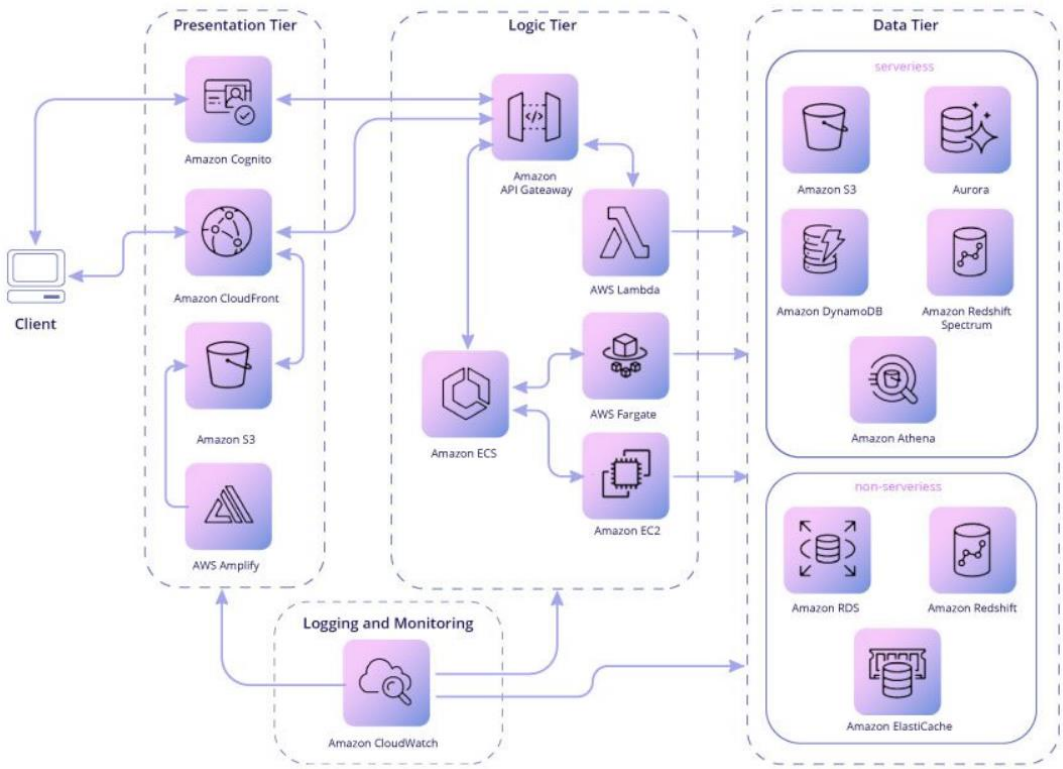


Рисунок 2.11 – Схема роботи архітектури AWS

Архітектура веб-додатку AZURE. Azure надає різноманітні служби хмарних обчислень, які дозволяють створювати базові та безпечні безсерверні веб-програми. Основна веб-програма створена за допомогою служби додатків Azure та бази даних SQL Azure. Поєднання Web App, Front Door, Function App, Azure DNS, Azure Cognitive Search і Azure DNS допомагає покращити масштабованість і продуктивність у веб-програмі Azure App Service. Безсерверна веб-програма, розроблена за допомогою Azure, обслуговує статичний вміст із Azure Blob Storage та реалізує API за допомогою функцій Azure. API зчитує дані з Cosmos DB і повертає результати до веб-програми.

2.2 Інструменти та програмні засоби для проектування

Одним із потужних інструментів проектування є Figma. Figma – це хмарний інструмент для проектування, схожий на Sketch за функціями та функціями, але з великими відмінностями, які роблять Figma кращим для командної співпраці. Розглянемо яким чином Figma спрощує процес проектування та ефективніше, ніж інші програми, допомагає дизайнерам і командам працювати разом. Проект в Figma наведено на рисунку 2.12.

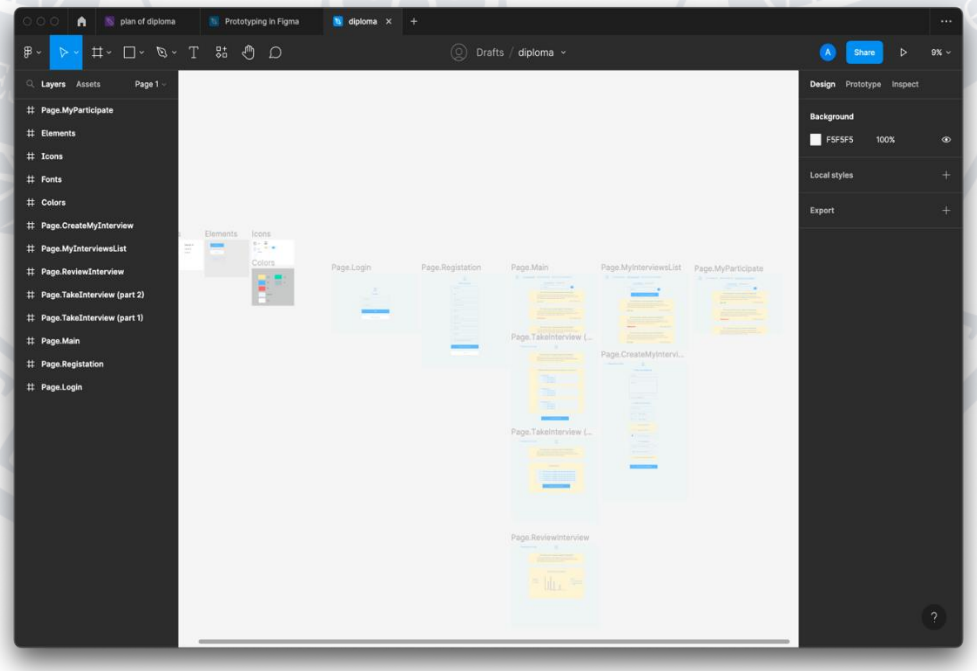


Рисунок 2.12 – Відкритий проект в Figma

Figma працює на будь-якій операційній системі, яка запускає веб-браузер. З Figma можна використовувати комп’ютери Mac, комп’ютери Windows, Linux і навіть Chromebook. У багатьох організаціях дизайнери використовують Mac, а розробники – ПК з Windows. Figma допомагає об’єднати ці групи. Figma відрізняється універсальністю використання. У Figma немає потреби в механізмі посередництва, щоб зробити роботу над дизайном доступною для всіх. Оскільки Figma базується на браузері, команди можуть співпрацювати так само, як і в Документах Google. Люди, які переглядають і редагують файл, відображаються у верхній частині програми як круглі аватари. Кожна особа також має іменований курсор, тому відстежувати, хто що робить, легко. Натискання на аватар іншої людини збільшує масштаб до того, що вони переглядають у цей час.

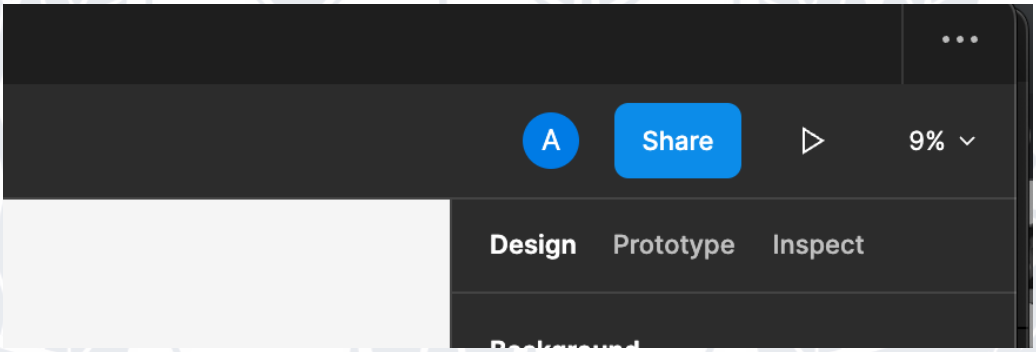


Рисунок 2.13 – Відображення людей котрі працюють в проекті

Співпраця над файлами в режимі реального часу допомагає пом’якшити «відхилення дизайну», яке визначається як неправильне тлумачення або відхилення від узгодженого дизайну. Дрейф дизайну зазвичай відбувається, коли ідея задумується та швидко реалізується під час виконання проекту. Використовуючи Figma, провідний дизайнер може перевірити, що проектує команда в реальному часі, просто відкривши спільний файл. Якщо дизайнер якимось чином неправильно інтерпретує бриф або історію користувача, ця функція дозволяє провідному дизайнеру втрутитися, виправити курс і заощадити незліченні години, які інакше були б витрачені даремно. Figma використовує Slack як канал зв'язку. Коли канал Figma створюється в Slack, будь-які коментарі

або редагування дизайну, внесені в Figma, передаються команді (рис. 2.14). Ця функція має вирішальне значення під час проектування в реальному часі, оскільки зміни у файлі Figma оновлять усі інші екземпляри, у які вбудовано файл. Перевіряються зміни в макеті, а канал зворотного зв'язку працює в реальному часі.

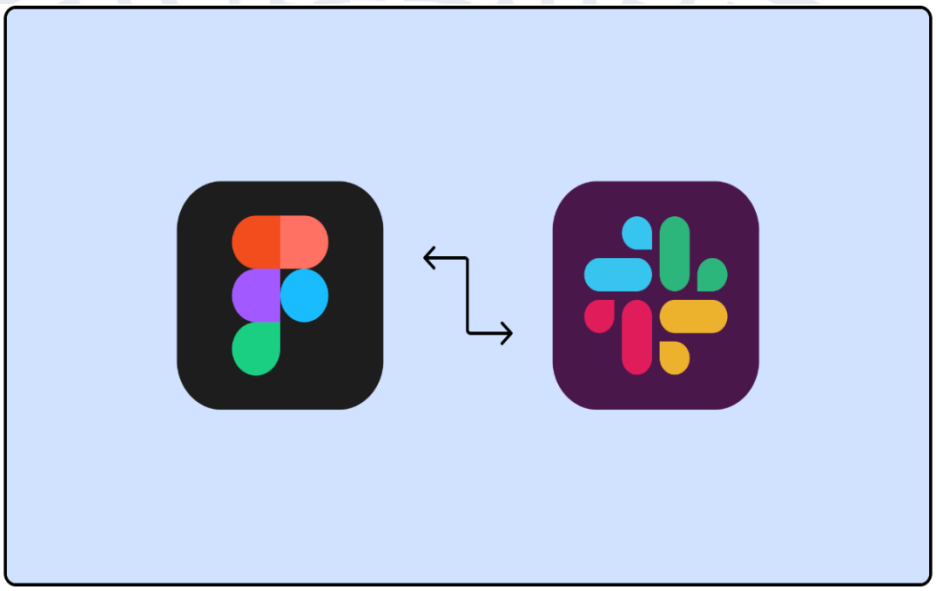


Рисунок 2.14 – Figma та Slack

Вбудовані файли Figma забезпечують оновлення в реальному часі. Figma також ділиться фрагментами вбудованого коду для вставки iFrame у інструменти сторонніх розробників. Так, якщо Confluence використовується для відображення вбудованих файлів макетів, ці файли не «оновлюються» шляхом збереження файлу Figma – ці вбудовані файли є файлом Figma.

Figma відображає фрагменти коду на будь-якому вибраному кадрі чи об'єкті у форматах CSS, iOS або Android для використання розробниками під час перегляду файлу дизайну. Будь-який розробник може перевірити компоненти дизайну в будь-якому файлі, який він може переглянути. Щоб отримати інформацію, не потрібно використовувати інструмент сторонніх розробників. Figma має повну інтеграцію з Zeplin, якщо команди хочуть робити більше, ніж просто вимірювання та відображення CSS.

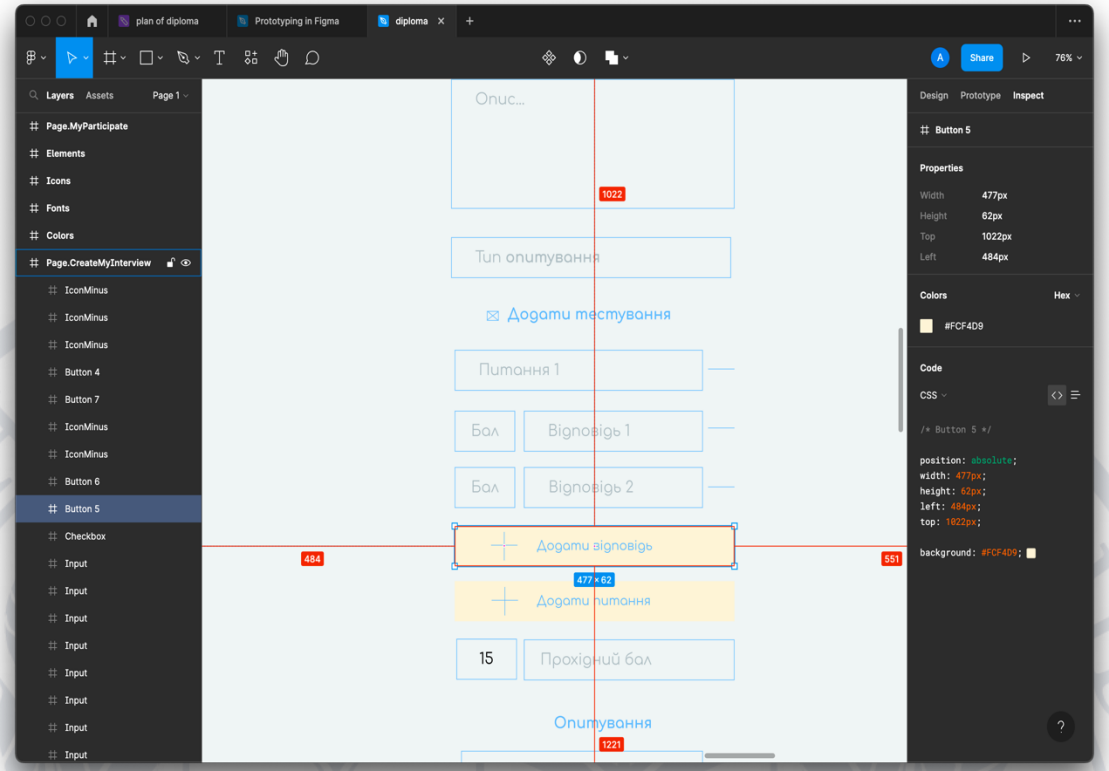


Рисунок 2.15 – Figma генерує готовий код

Файли проекту Figma зберігаються в одному місці онлайн. Оскільки Figma є онлайн-програмою, вона керує організацією файлів, відображаючи проекти та їх файли у спеціальному поданні. Figma також підтримує кілька сторінок у файлі, як Sketch.

2.3 Формалізація та структурні вимоги до платформи електронного голосування

Опираючись на можливі архітектурні рішення та предметну область голосування, необхідно визначити структуру проекту, тобто головні компоненти та додаткові елементи проекту. Наведемо списки функцій та опис оцінки проведених голосувань та опитувань.

Вимога структури веб-додатку на стороні клієнта:

1. Сторінка логін:

a. Ввід пошти.
b. Ввід пароля.
c. Кнопка входу.
d. Кнопка для переходу в режим створення аккаунту.

2. Сторінка створити аккаунт:
    a. Ввід пошти.
    b. Ввід пароля .
    c. Ввід дати народження.
    d. Ввід компанії.
    e. Ввід посади.
    f. Ввід освіти.
    g. Ввід регіону
    h. Кнопка створити аккаунт.
    i. Кнопка для переходу в логін.
3. Сторінка списків голосувань:
    a. Список всіх голосувань.
    b. Список всіх опитувань.
    c. Список всіх тестувань.
    d. Сортування:
        - По даті: нові голосування та старі голосування, ті що скоро завершаються.
        - По кількості учасників.
        - По регіону.
4. Сторінка проходження голосування.
5. Сторінка проходження опитування.
6. Сторінка проходження тестування.
7. Сторінка аналітика голосування.
8. Сторінка аналітика опитування.
9. Сторінка аналітика тестування.
10. Сторінка створення голосування:

a. Назва голосування.

b. Опис голосування.

c. Список варіантів, які можна обрати.

11.Сторінка створення опитування:

a. Назва голосування.

b. Опис голосування.

c. Список варіантів котрі можна обрати.

12.Сторінка створення тесту:

a. Назва голосування.

b. Опис голосування.

c. Список питань.

d. Список відповідей.

e. Бал на кожну відповідь.

f. Загальний Бал.

g. Прохідний Бал.

Вимога розмірів для адаптивності. Значення, по яким буде працювати адаптивність додатку під різні екрани пристроїв, буде три точки розмірів, які використовуватимуться різними стилями та розмірами компонентів і елементів для різних пристроїв:

1. m - Мобільна версія працює за розміром нижче 550 пікселів широти, $m < 550$.

2. t – Планшетна версія працює за розміром вище 550 пікселів та нижча 1200 пікселів, $550 < t < 1200$.

3. d – Комп’ютерна версія працює за розміром вище 200 пікселів, $d > 1200$.

Вимога структури веб-додатку на стороні клієнта, включає основні елементи роботи з учасниками голосування:

1. Логін.

2. Створення аккаунту.

3. Робота з юзерами:

a. Запит на список юзерів.

4. Робота с голосуванням:
    a. Запит на список голосувань.
    b. Запит на список голосувань окремої людини.
    c. Запит на окреме голосування.
    d. Запит на залежності для окремого голосування.
5. Робота с опитуванням:
    a. Запит на список опитувань.
    b. Запит на список опитувань окремої людини.
    c. Запит на окреме опитування.
    d. Запит на залежності для окремого опитування.
6. Робота с тестуванням:
    a. Запит на список тестувань.
    b. Запит на список тестувань окремої людини.
    c. Запит на окреме тестування.
    d. Запит на залежності для окремого тестування.

Для роботи платформи будемо використовувати архітектуру MVC (Model-View-Controller). Скорочено, MVC – це широко використовуваний шаблон проектування для розробки програмних додатків (рис. 2.16). Патерн спочатку був розроблений Трюгве Рейнскаугом під час його роботи над Smalltalk-80 у 1979 році [52]. За роль View буде відповідати бібліотека ReactJS [53], а за ролі Controller та Model буде відповідати бібліотека ReduxJS [54].

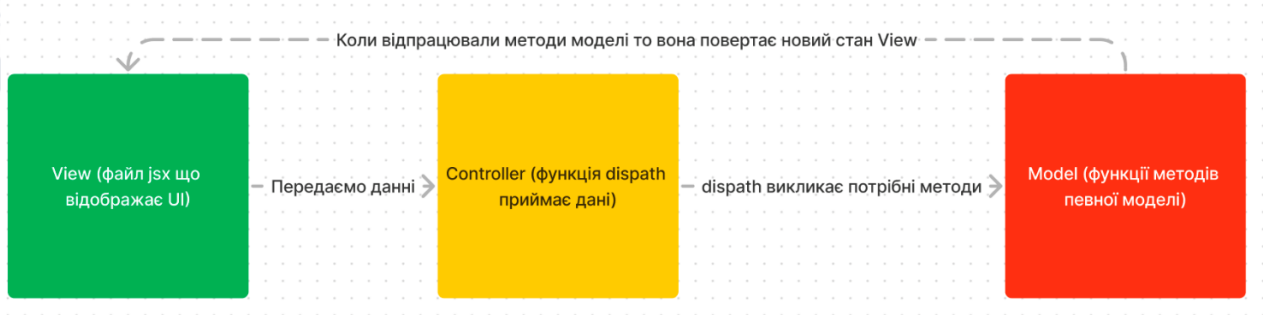


Рисунок 2.16 – Архітектура MVC

Частини архітектурного партерну MVC мають свої характеристики та обсяг завдань.

View – компонент має два завдання: представлення даних користувачу та обробка взаємодії з користувачем [52]. В розробленій платформі за цю роль відповідають компоненти бібліотеки ReactJS [53]. Використовуються компоненти формату JSX, що дозволяє записати UI (User Interface) в HTML розмітці, що потім конвертується в об’єкти JavaScript і відображає контент. Після вводу даних користувачем дані надсилаються контролеру та підписуємося на оновлення та інші стани платформи на функції useSelector від бібліотеки ReduxJS [54]. У разі отримання нових станів платформи – перерисовується UI.

Controller - шар перегляду та шар моделі склеюються разом одним або кількома контролерами [52]. У нашому випадку контролером виступає функція Dispatch від бібліотеки ReduxJS [54], яка в свою чергу буде приймати данні від користувача платформи та потім відправляти моделі.

Model – відповідає за бізнес-логіку програми та керує станом програми. В модулі передбачено читання та запис даних, збереження стану додатка, і може включати завдання, пов’язані з керуванням даними, як робота в мережі та перевірка даних [52]. У нашому випадку моделлю є функція, яка використовує методи з createSlice бібліотеки ReduxJS [54]. Вона приймає від контролера дані від користувача платформи та потім може відправляти дані на сервер, отримувати результат та робити зміни в сховище, що потім надає можливість знати компоненту ознаку відображення контенту: чи перемалювати старий стан на новий.

Психологія поведінки користувача покладена в основу виявлення функцій обробки результатів опитувань та розуміння свідомості користувача під час процесу голосування (опитування). Людина, яка спирається на інтуїцію і несвідоме, будує свої міркування насамперед на почуттях, а не на фактах. Інтуїція тут домінує. Опора на віру відіграє першорядну роль при прийнятті рішень [55]. З різним настроєм та з різними почуттями користувач може вибрати різні варіанти опитування, що не дає чіткого розуміння для аналітики. Перед

опитуванням запроваджують тестування почуттів, щоб виявити стан учасника перед опитуванням чи голосуванням.

В основу побудови програми покладено архітектурний підхід MVC та додано тестування учасника опитувань. На рисунку 2.17 зображено схему роботи веб-платформи.

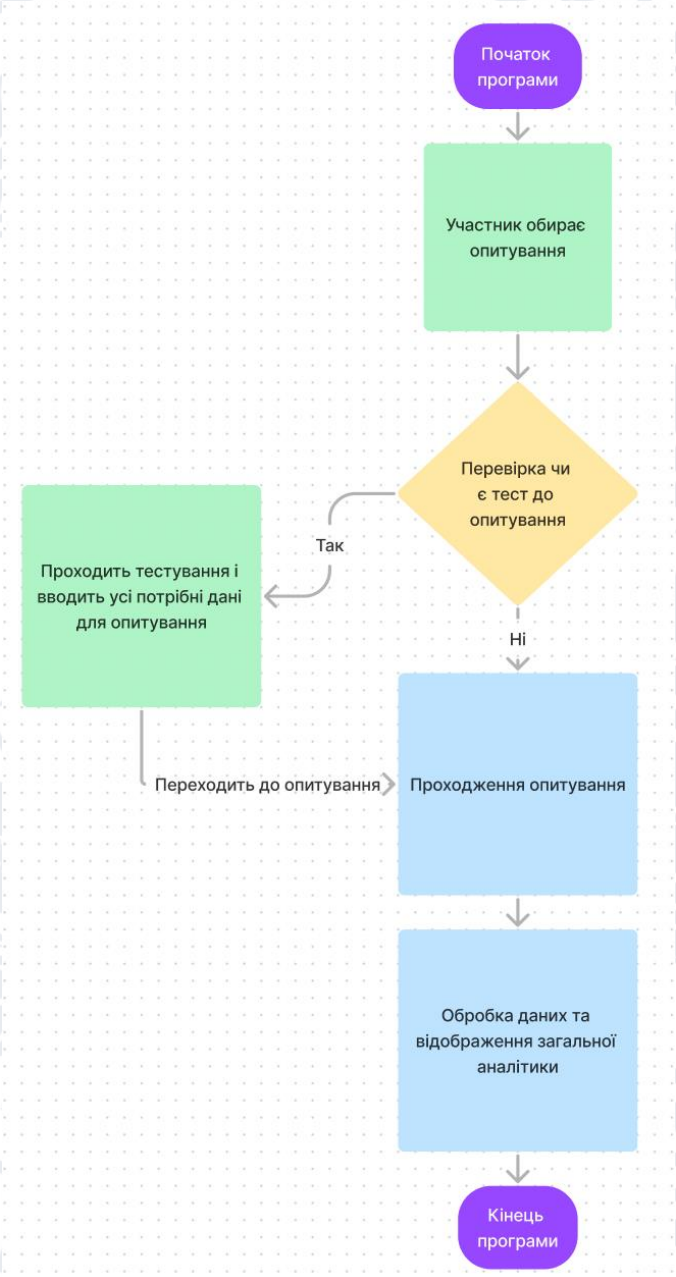


Рисунок 2.17 – Схема роботи веб-платформи опитування

ВИСНОВКИ ДО РОЗДІЛУ 2

В результаті дослідження формалізації та дослідження вимог до створення веб-платформи визначено основні особливості архітектурних рішень при розробці веб-платформ та веб-сайтів. Встановлено, що трьохрівнева архітектура сучасних веб-програм є найкращим підходом для відтворення незалежних процесів оновлення та масштабування.

Однією із основних вимог функціонування крос платформної розробки є адаптація до пристроїв, зокрема, інтерфейс та дизайн на може змінюватись при роботі на екранах різної роздільної здатності. Досліджено особливості вибору сучасних типів архітектур веб-додатків при відповідності критеріям: продуктивність, інтерфейс користувача, пошукова оптимізація, можливість зв'язування та швидкість реалізації.

З урахуванням сутності процесів голосування структура проектної розробки платформи та алгоритми організації проведення голосування, визначено структуру веб-додатку на стороні клієнта та вимоги розмірів для адаптивності. Обрано архітектуру MVC (Model-View-Controller). Алгоритм проведення голосування включає поетапність виконання процесів: реєстрація; вибір процедури (голосування, опитування); перевірка можливості та проходження попереднього тестування учасників; проходження процедури голосування (опитування); обробка даних та відображення аналітики.

РОЗДІЛ 3
РОЗРОБКА ПЛАТФОРМИ ЕЛЕКТРОННОГО ГОЛОСУВАННЯ

3.1 Розробка дизайну

Основою проекту є аналіз і розробка дизайну. Даний елемент розробки, задає вектор руху проєкту. Взаємодія елементів роботи з проєктом платформи CX, BX, UX та UI наведена на рис. 3.1.

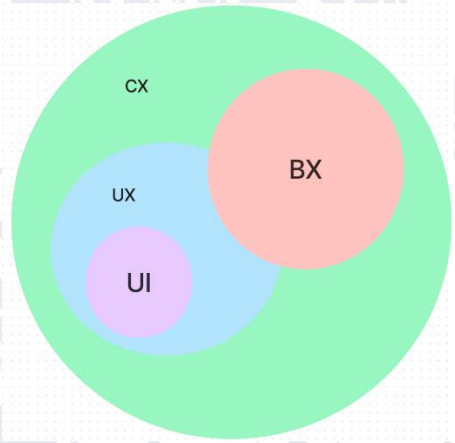


Рисунок 3.1 – Взаємодія CX, BX, UX та UI

CX, CX (customer experience) – реалізація ідеї проекту, на базі даного досвіду вивчається напрямок продукту, наприклад, потрібно зрозуміти вподобання людей. У розробленій платформі вирішується питання, про найбільш зручне проведення процесу голосування та полегшення використання самою платформою учасниками.

BX (brand experience) – проведення роботи с брендом, метою якого є надання продукту бренду з визнанням користувачами відмінності від інших платформ, які проводять голосування. Організація роботи з BX, полягає в можливості представлення продукту та його поширення на ринках програмних продуктів.

UX (user experience) – це робота по розробці логіки функціонування продукту так, щоб він був зрозумілий користувачами при використанні. Також це розрахунок архітектурних рішень продукту з метою отримати його найбільш оптимальну та зручну форму.

UI (user interface) – це кінцева розробка продукту, коли отримано бачення продукту та уявлення проблематики від CX, коли відомо коло прогнозованого сегменту ринку для поширення продукту від BX та коли визначено архітектурі рішення проєкту від UX.

В даному досліджені для розробки UI використано програму Figma та підхід компонентного дизайну, який визначає практику поділу інтерфейсу користувача на менші, більш керовані частини з чіткими назвами. Кожна з цих частин належить до однієї з шести окремих груп.

Першою з цих шести груп є ідентичність. Тут визначено ключові елементи бренду проєкту. Зокрема, визначено гарнітури, типографіка, основні та додаткові кольори, які використовуються протягом усього проекту. Обрані кольори наведено на рис. 3.2.

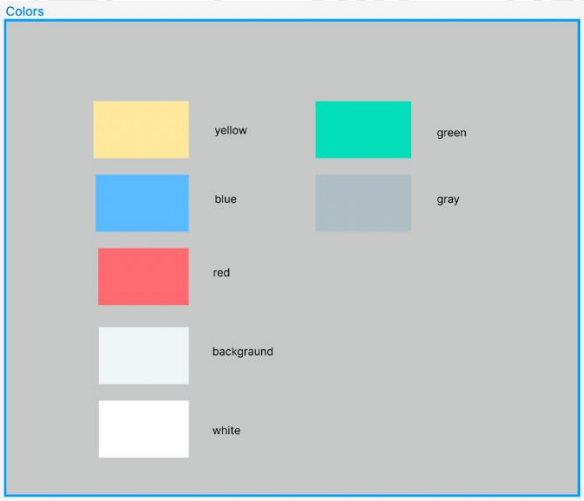


Рисунок 3.2 – Використані кольори

Друга група визначає найменші багаторазові частини проекту: елементи. До елементів вілносять: кнопки, посилання, вхідні елементи, спадні списки тощо. Кожен із них визначений разом із усіма їхніми станами; наприклад кнопки наведення, фокусування та вимкнені кнопки. Елемент визначений один раз, використовується повторно протягом усього проекту. На рисунку 3.3 можна побачити розробку головних елементів та іконок, які мають повторно використовуватись по усьому проєкту.

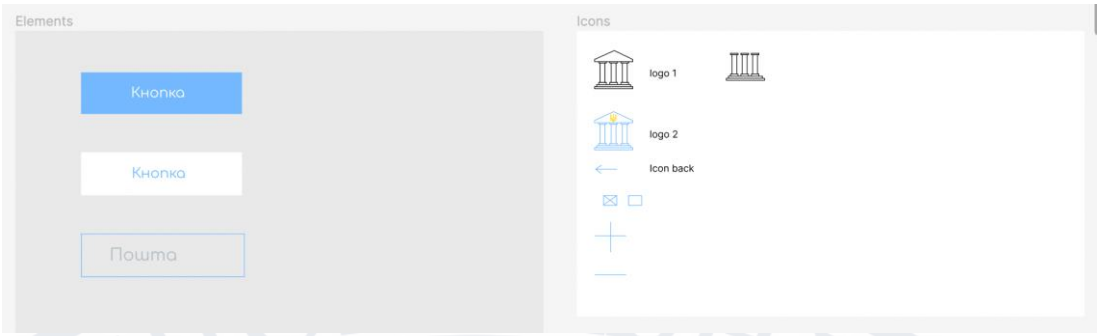


Рисунок 3.3 – Розроблені елементи на іконки

Розвиваючись у масштабі, третя група – це компоненти (рис. 3.4). Під час роботи над розробкою програм і веб-дизайну більшість блоків на екрані є компонентами. Компонентом може бути все, що використовує кілька елементів. Такі речі, як картки, герої та навігаційні меню, є традиційними прикладами компонентів. Однак вони не обов'язково повинні виглядати модульними.

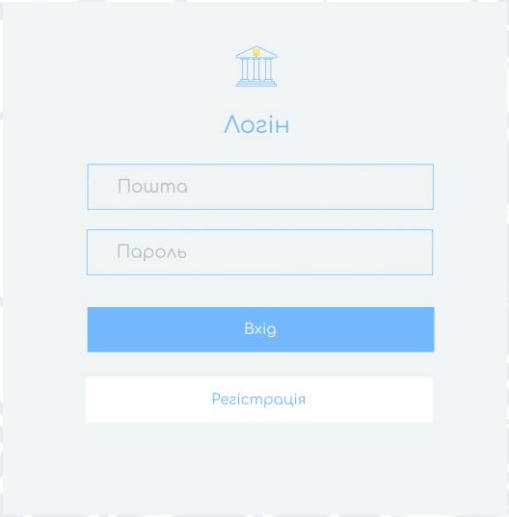


Рисунок 3.4 – Компонент реєстрації учасника

Розробляючи компоненти, створюються їх версії для кожного адаптивного розміру (або точки зупину) проекту.

Четвертою групою є композиції. Композиція – це частина, усередині якої є кілька компонентів (рис. 3.5). Вони визначають, як повинні поводитися

компоненти всередині нього. Нижче наведено приклад простого макета стовпців. Це дуже проста композиція. Він лише визначає деякий простір навколо композиції, заголовка та того, як компоненти всередині мають зациклюватися.

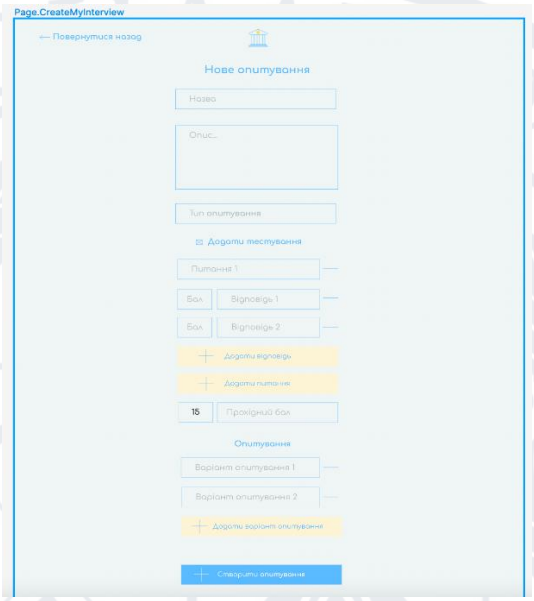


Рисунок 3.5 – Композиція створення нового опитування

П'ята група, Layout, є більш абстрактним набором принципів дизайну (рис. 3.6). Тут визначено кількість пробілів, сіток і обгорток. Визначаючи їх таким чином, іншим дизайнерам легко зайти в проект і використовувати існуючі стилі та рекомендації.

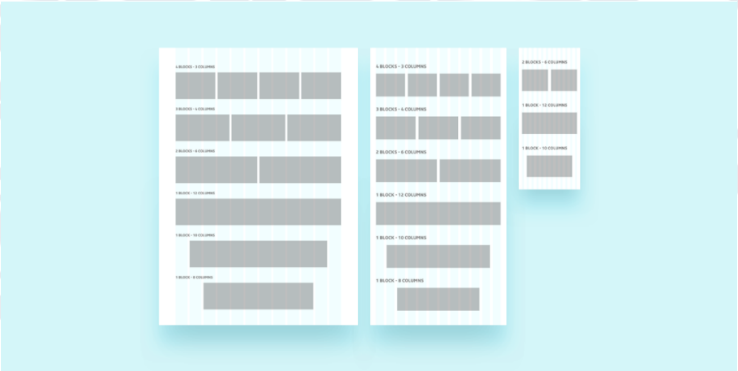


Рисунок 3.6 – Приклад макета

Остання група – це фактичні сторінки (або екрани) проєкту. Кожна сторінка складається з композицій і компонентів. Сторінки можуть повторно використовувати деякі компоненти, але в різних містах вони будуть робити різні речі це як копія одного компонента що створюють нові інші. Також важний фактор це послідовність сторінок, щоб послідовність сторінок тримала сенс та було зручно передавати дані при переході, та щоб процеси що почали робити в одній сторінці продовжувались в інших сторінок, щоб була комунікація між сторінками.

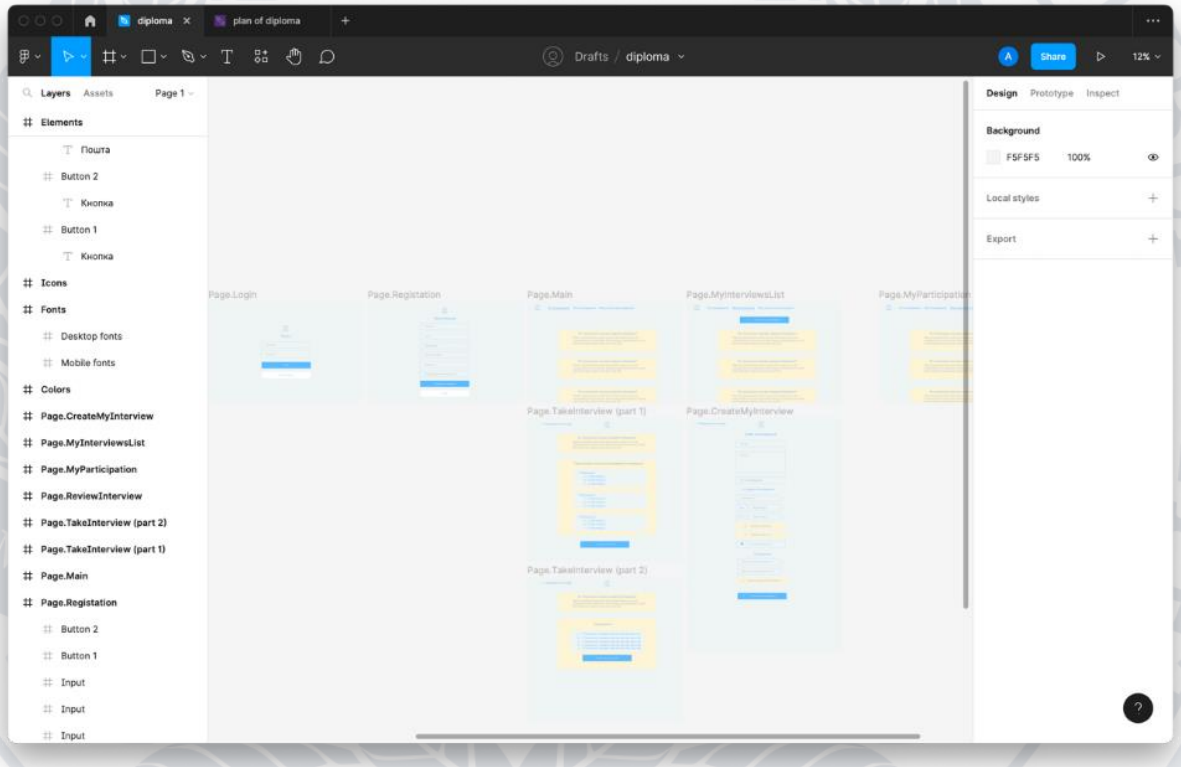


Рисунок 3.7 – Дизайн сторінок

Результат масштабування компонентів, можна розглянути а наступному прикладі. Розроблено компонент «Карточку, яка призначена для виводу інформації про голосування». Існує список де багато голосувань, але окреме голосування це окреме подія, але використовує один компонент.

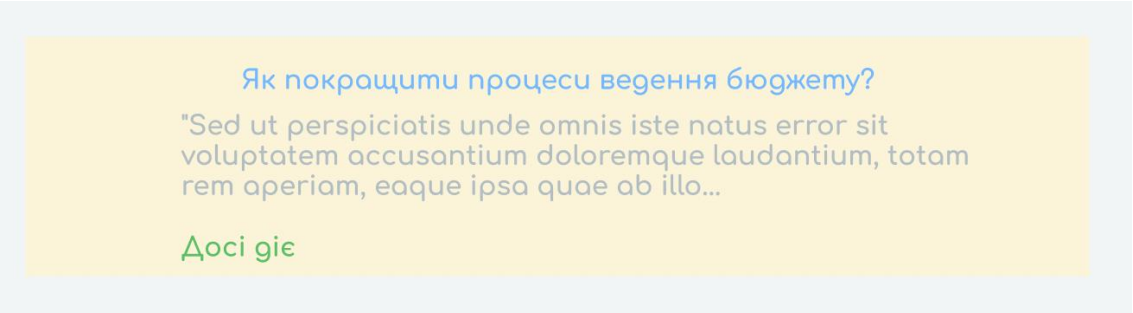


Рисунок 3.8 – Приклад компонента

У подальшому виникає необхідність розмістити три голосування на нашій домашній сторінці у простому макеті з трьох рядків. Вирішується питання про розробку композиції: композиція голосувань. Ця композиція вказує на те, що кожен із компонентів голосування повинен мати деяку відстань між ними та назву.

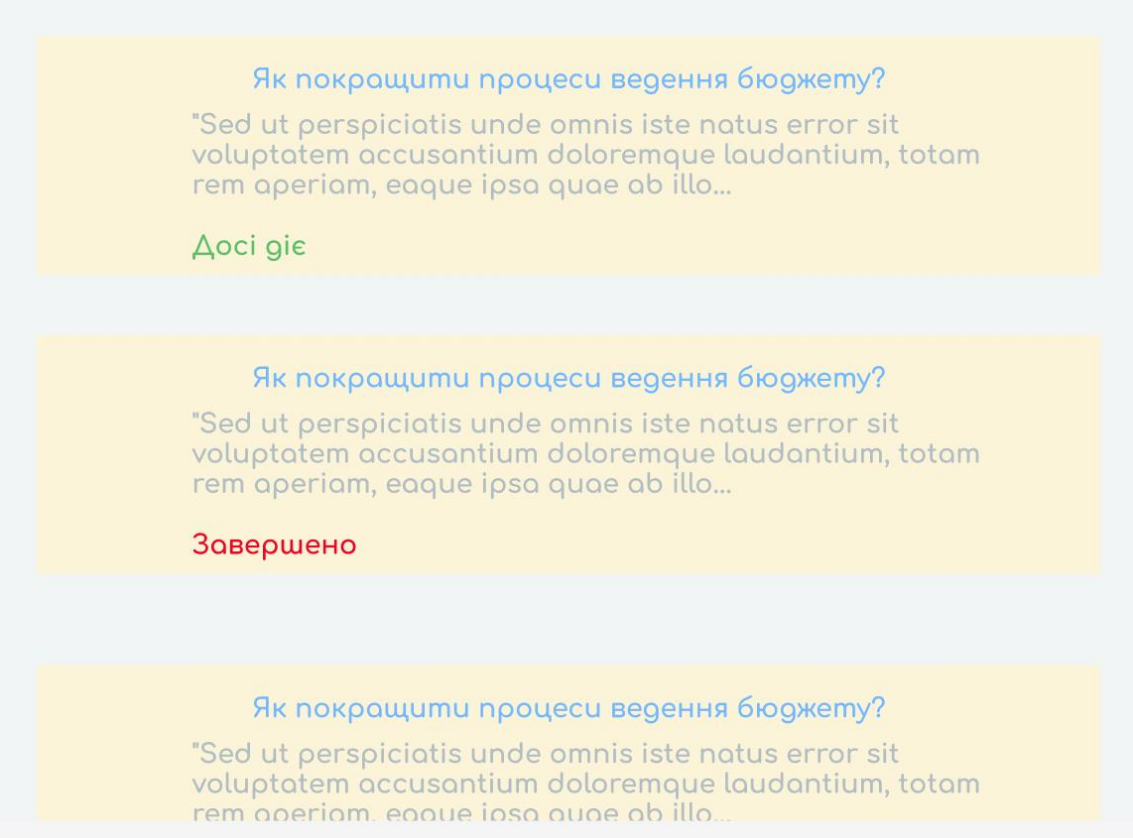


Рисунок 3.9 – Приклад композиції голосувань

У разі необхідності додати нову функцію, де буде відображатись активна фаза або завершення голосування, буде дописана логіка в одному компоненті і розпочинається процес по всій композиції.

Використання Figma нас сьогодні є популярним на ринках програмних продуктів для дизайну та інтерфейсу та UX. Використання стилів тексту, символів і монтажних областей Figma дозволило нам значно налаштувати робочий процес компонентного дизайну.

3.2 Розробка клієнт частини платформи

Розробка клієнт частини продукту пов'язана із використанням бібліотеки, ReactJs. Ця бібліотека була розроблена компанією facebook. Бібліотека добре робить декомпозицію складних композицій. Раніше для роботи з ReactJS потребувалось опис компонентів з використанням класових компонентів, для використання можливостей життєвого циклу компонентів, оскільки в функціональних компонентах не можна було використовувати життєвий цикл. Життєвий цикл React.js – це великий інструмент, який можна використовувати для створення незабутнього досвіду в Інтернеті. Це один із найнадійніших шляхів у світі розробки, який дозволяє створювати програми, орієнтовані на інтерфейс користувача. Ці програми можна запустити в Інтернеті для більшої аудиторії. Незважаючи на те, що React.js регулярно оновлюється, основні функції дотримуються життєвого циклу. Компоненти React.js виникають і породжують інші у форматі циклу. Це робить весь процес набагато ефективнішим і економнішим. Життєвий цикл React.js розділений на чотири основні етапи. Життєвий цикл React.js – ініціалізація, монтування, оновлення та демонтування (рис. 3.10).

Кожен етап служить унікальній меті в циклі розробки та має широкий спектр операцій для виконання. Розробник може кодувати певні команди, щоб забезпечити виняткові результати, коли користувач взаємодіє з програмою. Користувачі можуть експериментувати з додатком і бачити, що він працює (як запрограмовано) кожного разу, оскільки розробники React.js дотримуються

протоколу життєвого циклу та створюють для них унікальну функціональну програму. Незалежно від того, чи це онлайн-калькулятор, чи довідковий посібник для бібліотеки, інтерфейс користувача точно реагує на команду протягом життєвого циклу.

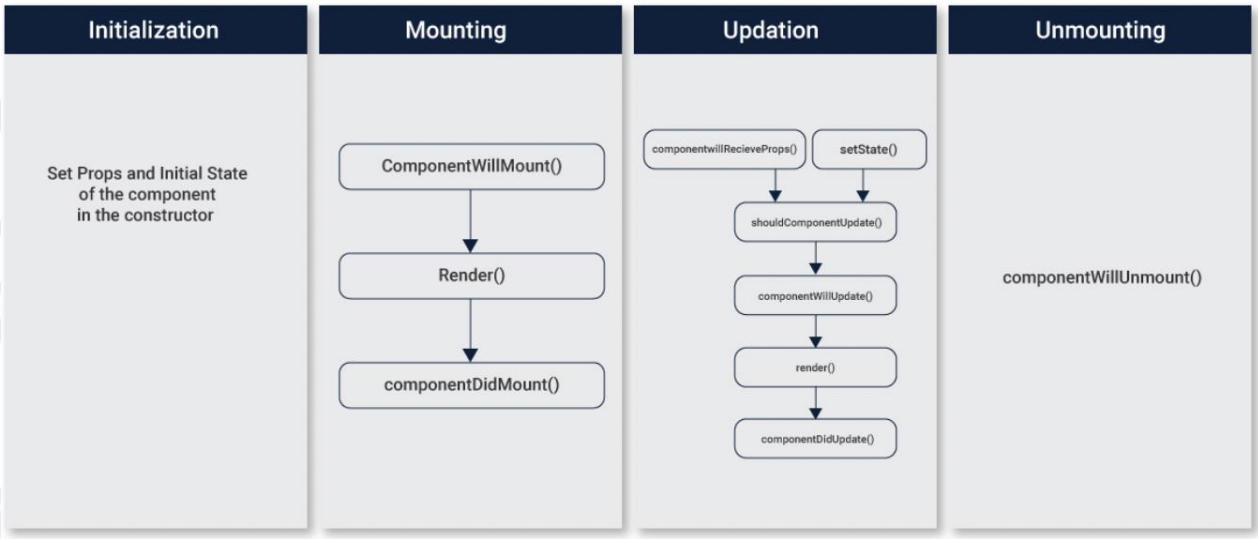


Рисунок 3.10 – Життєвий цикл React.js

React.js – є бібліотекою JavaScript, яку можна використовувати для розробки привабливих інтерфейсів користувача. Основна технологія, що лежить в основі React.js, дозволяє користувачам безперешкодно взаємодіяти з Інтернетом. Немає оновлення коду або нових функцій, інтегрованих через хмару. Обробка виконується на місці, і існує низька затримка до остаточного обчислення. React.js є має кілька функцій і функцій, які можна використовувати з часом. React.js може працювати з штучним інтелектом і системами машинного навчання для створення надійніших кінцевих рішень. React.js найкраще працює під час свого життєвого циклу розгортання, коли середовище кодування спрощене. Ось чому багато проектів React.js вимагають, щоб кодери працювали в межах параметрів згаданого життєвого циклу.

Віртуальний DOM забезпечує чудове з’єднання між браузерами та веб-додатками.

Initialization	Factor	Angular
2013	Release Date	2016
Facebook	Support from	Google
Javascript	Programming Language	Typescript
React 16.7.0	Latest Stable Version	Angular 7.2.0
Moderate	Learning Curve	Steep
Component Based	Architecture	Component Based
Relatively Small	App Size	Relatively Small
Virtual DOM	DOM	Real DOM
Client/Server Side	Rendering	Client/Server Side
JSX + JS (Es5 and Beyond)	Template	HTML+Typescript
Unidirectional(One-way)	Data Binding	Bidirectional(Two-way)
Mediun	Abstraction	Strong

Рисунок 3.11 – Порівняння плюсів та мінусів бібліотек

Однією з найважливіших переваг використання React.js є той факт, що він є гнучким як каркас. Це універсальна бібліотека, яка дозволяє низхідну авторитетну інформацію. Звіт гарантує, що доступна батьківська інформація не впливає на нові компоненти. Це дає змогу розробникам React.js кодувати унікальні функції протягом більш тривалого часу. Можливість адаптації вписується в загальну архітектуру, водночас надаючи більш значні переваги.

Мережа, побудована в React.js, неймовірно динамічна, завдяки чому весь вміст відображається як оновлений. Оскільки вміст оновлюється динамічно, існує більш одночасний характер, який досліджується в рамках JavaScript. Це дозволяє зберігати важливу інформацію в рамках React.js і відтворювати її в іншому контексті. Єдина проблема, яка є досить серйозною, пов’язана з інтерфейсом користувача та структурою MVC. Іноді оновлення тривають довше, ніж очікувалося, і доступна документація обмежена, тому спільнота React.js надає краще розуміння домену. Він також може створити більш надійну бібліотеку, де кодери можуть додавати або віднімати. Вони можуть

використовувати технологію, щоб відповідати власним унікальним потребам зі схеми кодування. React.js надійніший і кращий у використанні, ніж будь-яка інша альтернатива на ринку. Перехід від Angular до React.js також є чистішим кроком, оскільки є більше гнучкості у використанні бібліотеки React.js. Завдяки тому, що базується на JavaScript і JSX (розширенні PHP), результатом є багаторазово використовуваний кластер елементів для роботи в Інтернеті. З цієї причини його популярність зростає, і все більше компаній використовують програмістів React.js. Він стає очевидним переможцем у змаганні за найгнучкішу мову для створення інтерфейсу. Усі, починаючи від Instagram і закінчуючи Uber, використовують React.js для покращення взаємодії з користувачами. Технологія використовується в багатьох програмах, що сприяє зростанню кількості користувачів. Завдяки масштабу та гнучкості React.js також важливо вивчити його життєвий цикл.

Незважаючи на те, що React.js можна використовувати в IoT, AI та веб-додатках, необхідно слідувати певним шляхом для створення власних форм. Найкраще це досягається за допомогою життєвого циклу React.js, починаючи від ініціалізації до демонтування. Це робить React.js найбільш масштабованим додатком, який покладається на кращу обробку та ефективнішу архітектуру кодування.

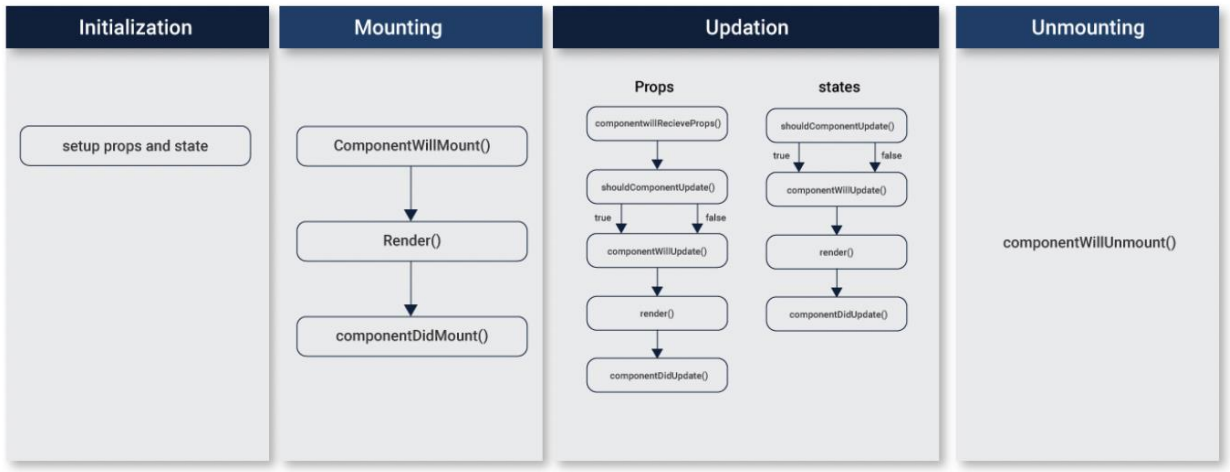


Рисунок 3.12 – Етапи життєвого циклу React.js

Перший етап життєвого циклу React.js – ініціалізація, який дуже важливий для розуміння та ретельного вивчення. Це етап, на якому компонент створюється з наданими властивостями та станом за замовчуванням і проводиться конструкторі класу компонентів. Необхідно визначити ідеальні властивості (властивості) і початковий стан компонента. Довідковий приклад наведено на рис. 3.13.

```
class Clock extends React.Component {
    constructor(props)
    {
        // Calling the constructor of
        // Parent Class React.Component
        super(props);

        // Setting the initial state
        this.state = { date : new Date() };
    }
}
```

Рисунок 3.13 – Перший код, який спрацьовує при ініціалізації

Важливим є розуміння проведення ініціалізації компонентів при заповненні їх стану або властивостей. Це може допомогти їм відобразити правильну інформацію поза межами їх каркасної розмітки. Це проводять на ранній стадії створення ідеї з метою оптимізувати свій підхід до кодування. Існує в основному три частини даних, від яких залежить JSX, зокрема:

- помилка – це стандартне повідомлення, яке відображається, коли помилка є в системі або під час відтворення;
- завантаження – завантаження відбувається, коли програма отримує дані API;
- користувачі – дані, отримані з API, також є критичними.

Це допомагає створити успішну архітектуру кодування в модулі більшого інтерфейсу користувача. Поки компонент налаштовує початковий стан у конструкторі, є можливість змінити його пізніше за допомогою методу setState.

Це надає більше гнучкості в кодуванні, коли рухатись до іншого кінця життєвого циклу. DefaultProps також визначається як властивість компонента, щоб можна було ідентифікувати значення за замовчуванням для props. Існує можливість задати значення за замовчуванням, які програма може використовувати як стандартний стан. Цей стан за замовчуванням можна використовувати кілька разів у кількох програмах. Важливо правильно закодувати цю частину, оскільки вона закладає основу для наступних рівнів життєвого циклу.

Монтаж є наступним етапом життєвого циклу та критичний для запуску. Після того, як підготовлено код із основними вимогами, станами та атрибутами, монтується компонент у браузері. Це робиться за допомогою DOM браузера, і фаза дає правильні методи хуків React.js для підгонки до та після.

Рендер – це те, що монтує компонент у браузері в цьому стані. Це класичний метод, який дає той самий результат щоразу, коли надається той самий вхід. Це стандартна функція, яка широко використовується в системі кодування React.js.

ComponentWillMount – це критична функція, яка виконується безпосередньо перед монтуванням компонента охоплення. Монтування на DOM виконується після цього етапу, на якому можна ввести всі дії, які мають бути виконані програмою. Він також виконується один раз протягом життєвого циклу компонента і відбувається перед тим, як його вперше відтворено в програмі. Він також використовується для ініціалізації станів або пропсів, що робить його надійним компонентом для використання.

ComponentDidMount – це метод React.js Hook, який виконується після того, як компонент монтує DOM. Він виконується один раз у життєвому циклі та відбувається після першого відтворення. Можна отримати доступ до DOM за допомогою цього методу та ініціалізувати відповідні бібліотеки JS. За допомогою цього компонента можна ефективно отримати доступ до DOM, також можливо ініціалізувати кілька інших бібліотек, які включити в кінцевий результат. Ви можете зробити правильні виклики API за допомогою цього методу, щоб отримати дані правильним шляхом.

Оновлення – третій етап, який починається, коли компонент прийнято в браузері. Потім це може зрости, отримуючи нові оновлення від програми. Користувач може взаємодіяти з програмою, а компонент можна оновлювати відповідно. Зазвичай розробники можуть оновлювати компонент кількома основними способами. Вони можуть або надіслати нові властивості для команди, або повністю оновити стан. Залежно від складності або масштабу роботи, вони можуть вибрати будь-який спосіб і запустити програму.

Основні Методи Hook:

1. ShouldComponentUpdate. Метод повідомляє програмі про стан візуалізації під час оновлення. Якщо оновлюються нові атрибути або правила, візуалізацію можна виконати або пропустити, що важливо для правильного кодування, оскільки в програмі також є стани, які розвиваються. Оновлення методу як true/false є правильним підходом. Типовим значенням тут є true, яке можна змінити відповідно до коду.

2. ComponentWillUpdate. Метод виконується, коли попередній метод повертає відповідь true. Потім він використовується для підготовки майбутньої візуалізації, у випадку, коли необхідні попередні обчислення перед поверненням відповіді.

3. ComponentDidUpdate. Виконується, коли оновлений компонент також оновлено в DOM. Можливо ініціювати перезавантаження нових бібліотек, щоб підтримувати оновлену програму протягом усього процесу. Візуалізація може бути запущена відповідним чином відповідно до основних вимог.

Демонтування – є останнім етапом, для виконання якого не потрібен компонент і проводиться демонтування з DOM. Як кінцевий стан, він призначений для отримання результату через демонтування. Основний метод, який використовується для демонтування: componentWillUnmount. Це останній метод у життєвому циклі, оскільки він стосується демонтування ядра та видалення з DOM. Тут також виконується очищення компонента, що використовується для виходу користувачів, коли вони хочуть очистити програму зі свого браузера.

Розглянемо реалізацію клієнтської частини додатку для проведення голосування (опитування). Коли в перше ми входимо у систему, то виникає необхідність увійти в аккаунт. Вхід в платформу наведено на рис. 3.14.

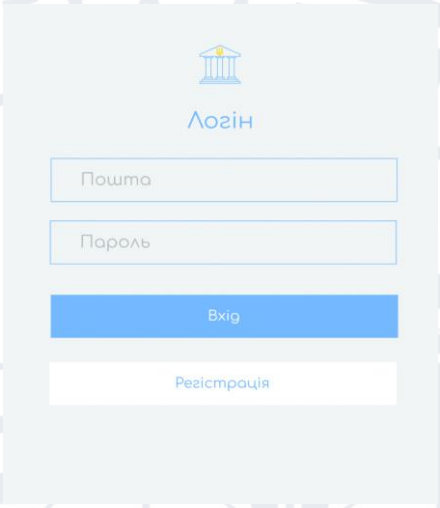


Рисунок 3.14 – Вхід у платформу

У разі відсутності аккаунту, його можна створити та перейти на сторінку реєстрації (рис. 3.15).

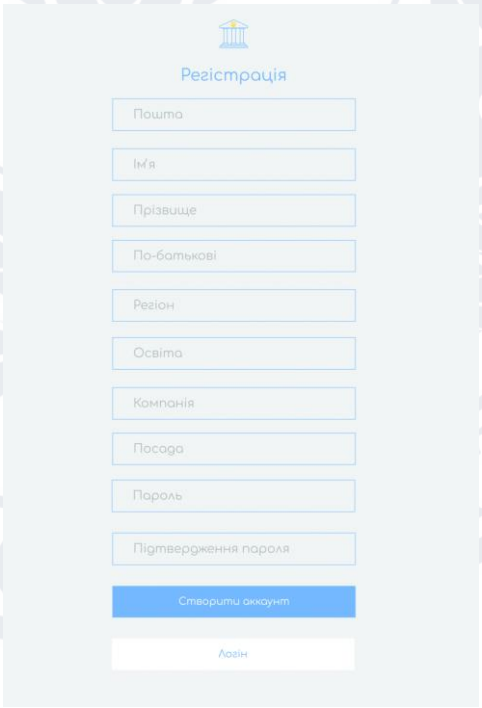


Рисунок 3.15 – Сторінка створення аккаунту

Після створення аккаунту або реєстрації у уже існуючому аккаунту, здійснюється перехід на головну сторінку (рис. 3.16).

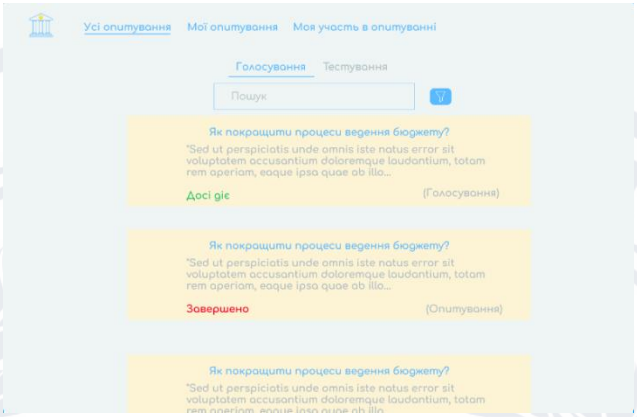


Рисунок 3.16 – Головна сторінка

На головній сторінці висвітлюються списки голосувань та опитувань, а також можливість перейти в список тестувань до голосувань. Існує також можливість зробити пошук по назві або зробити фільтр.

Можна перейти на сторінку власних створюваних голосувань (рис. 3.17) або на сторінку, де вже прийняли участь у голосуванні (рис. 3.18).

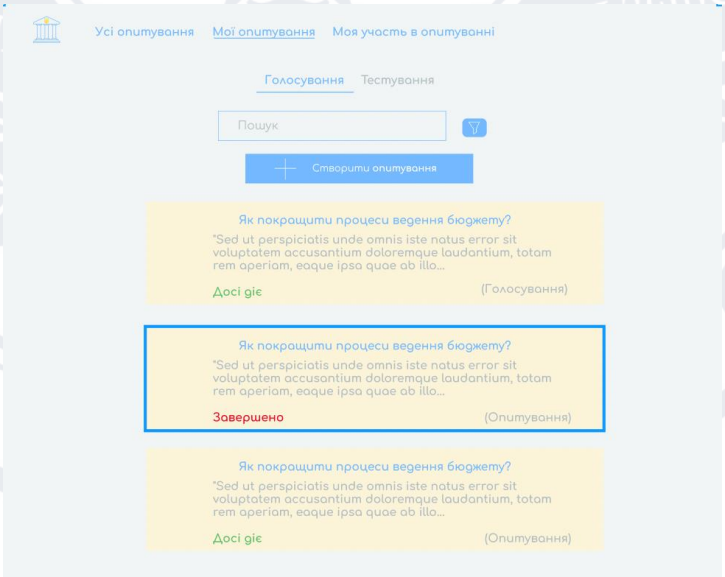


Рисунок 3.17 – Сторінка де користувач створив власні голосування

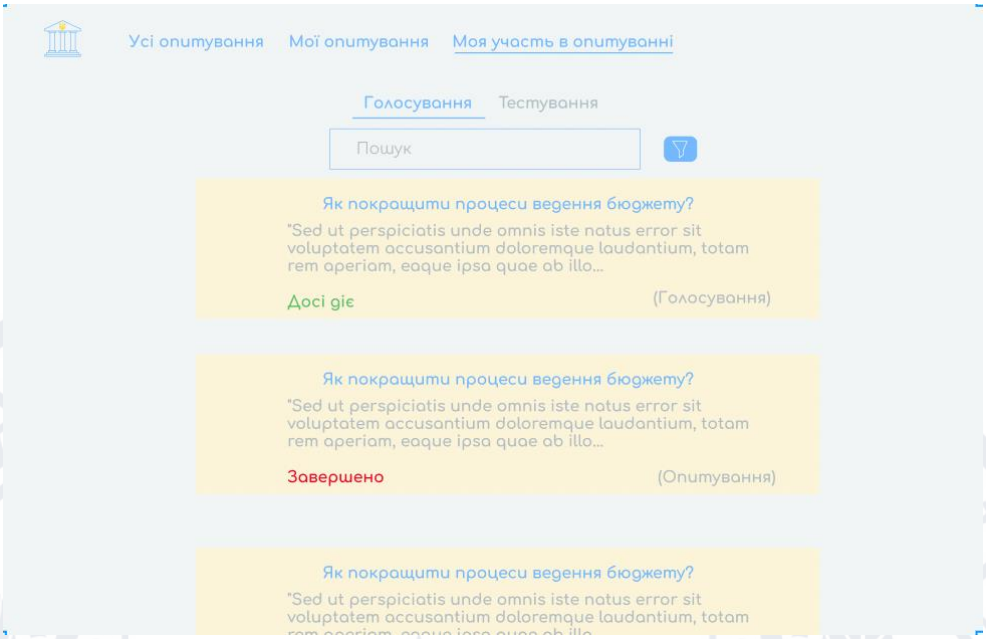


Рисунок 3.18 – Сторінка де користувач прийняв участь в голосуваннях

Створення власного голосування може супроводжуватись попереднім тестуванням та опитуванням, що дозволяє отримати більш детальну аналітику вподобань учасників (рис 3.19).

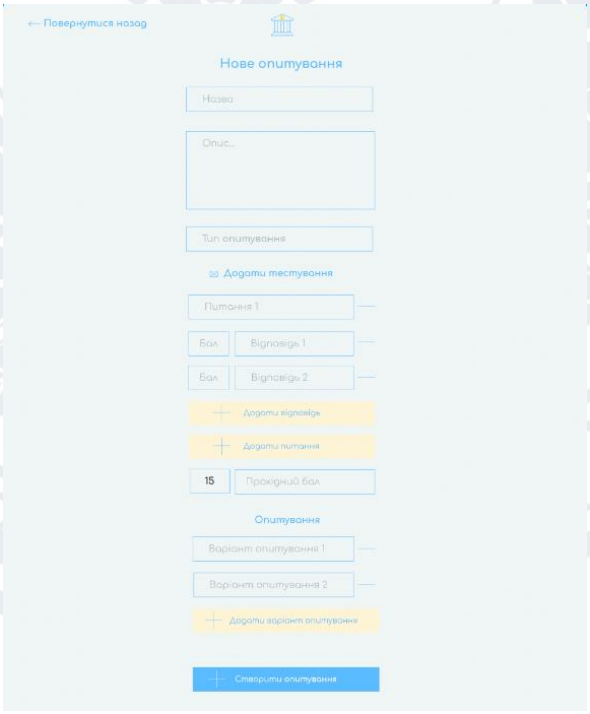


Рисунок 3.19 – Сторінка для створення голосування

Сторінка для проходження попереднього тестування зображена на рис. 3.20.

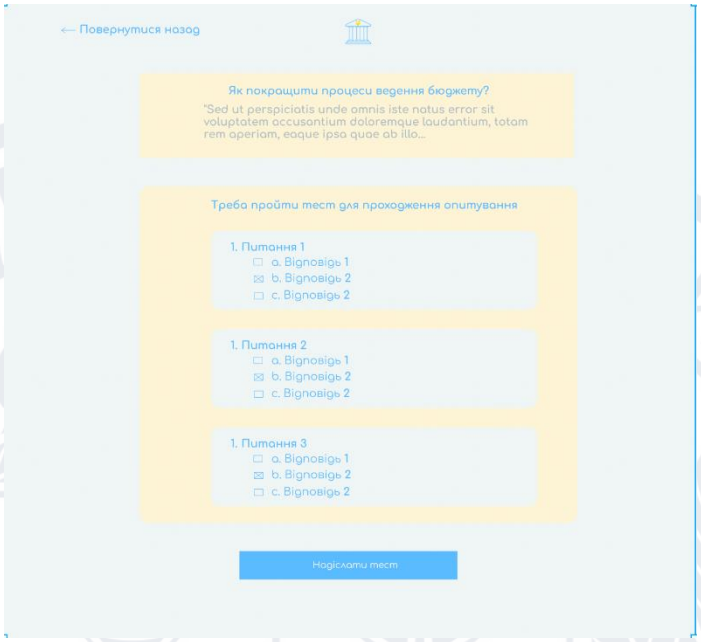


Рисунок 3.20 – Сторінка для проходження попереднього тестування

Сам процес голосування (опитування) відбувається на сторінці, вигляд якої наведено на рис. 3.21.

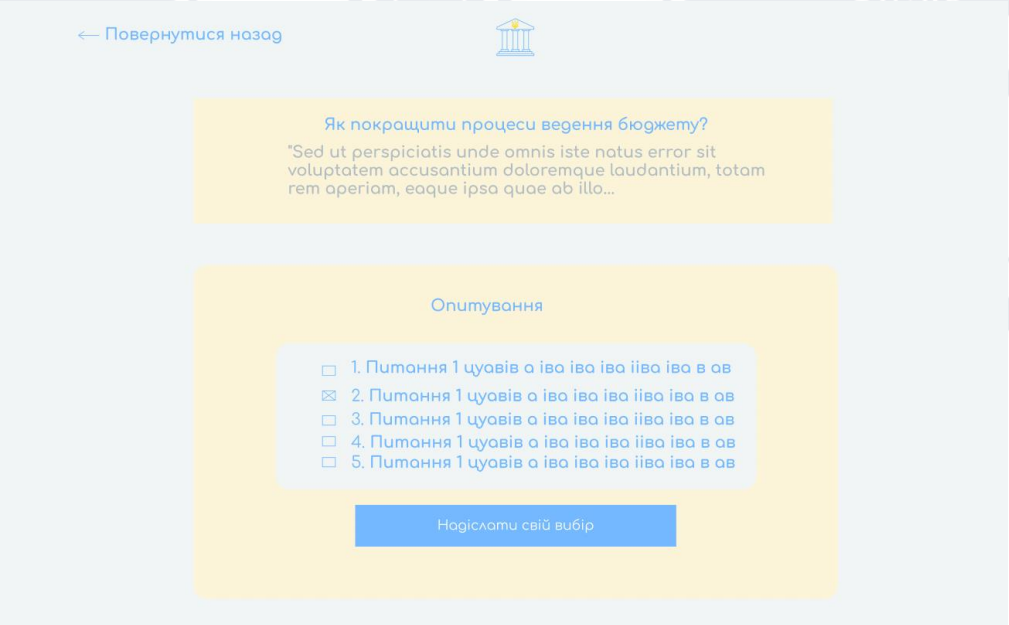


Рисунок 3.21 – Сторінка для проходження голосування або опитування

Результат голосування (опитування) наводиться у вигляді діаграми (рис. 3.22).

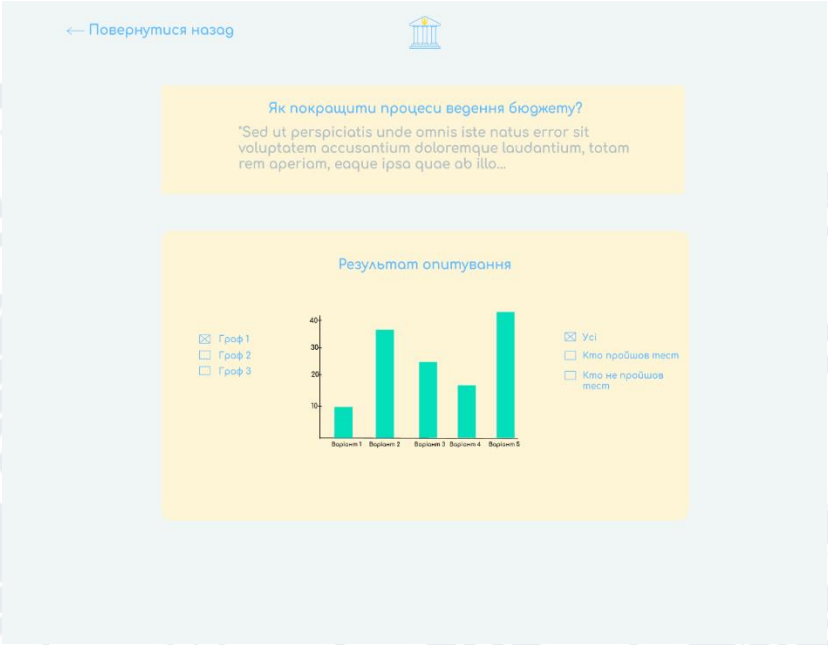


Рисунок 3.22 – Сторінка для перегляду аналітики у вигляді діаграми

Також можна переключитись на інші види графіків, наприклад, на графік у вигляді кругової діаграми (рис. 3.23).

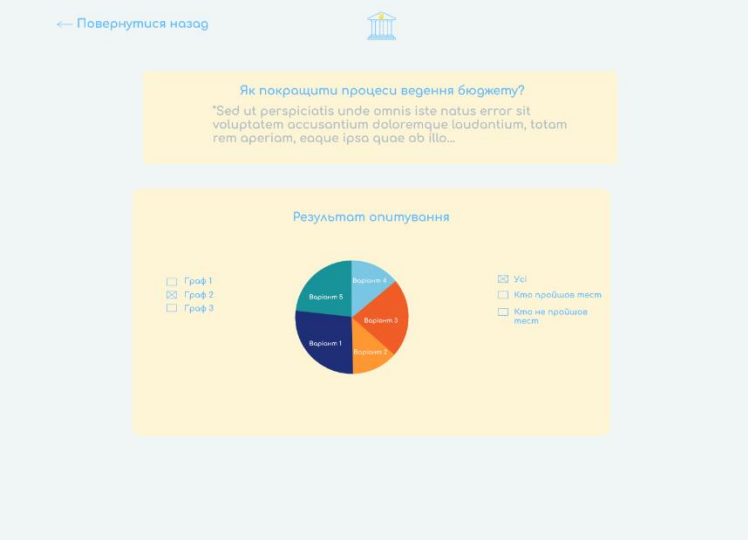


Рисунок 3.23 – Сторінка для перегляду аналітики у вигляді кругової діаграми

3.3 Розробка сервер частини платформи

Для розробки сервер частини орано суверну мову програмування, а саме NodeJS. Головна проблема полягала у виборі альтернатив програмування між бібліотекою ExpresJS та фреймворком NestJS.

Express – це бездумна, мінімалістична структура. Бездумні фреймворки довіряють розробникам зробити правильний вибір, що дозволяє розробникам вибирати будь-які інструменти чи технології, проміжне програмне забезпечення, хуки, які вони хочуть використовувати. Розробники структурують свої коди без будь-якої конкретної структури, яку нав'язує фреймворк. Express, будучи бездумним фреймворком, не має жодного конкретного механізму шаблонів, синтаксичного аналізатора тіла, обробника помилок тощо. Кожен експрес-проект з часом стає унікальним. Крім того, в рамках одного великого проекту структура проекту та інструменти різних команд мікросервісу залишаються ефективними. Через це не можна просто переходити від однієї команди мікросервісу до іншої.

Nest – це дуже впевнена структура, яка дотримується парадигми проектування «угода над конфігурацією». Це суворе керівництво розробникам використовувати певні інструменти та кодувати певним чином. За лаштунками він уже обгортає контролер блоком try-catch, уже аналізує тіло запиту, додає обробник помилок, проміжне програмне забезпечення, реєстратор тощо. Розробники мають написати контролер, сервіс, репозиторії в певних місцях певним чином. Кожен, хто знайомий з Nest, може почати проект, не витрачаючи багато часу на те, як організовано код або які інструменти використовуються. Код у проекті Nest має бути набагато зручнішим у супроводі, ніж у необачному проекті Express.

TypeScript. За допомогою вільних типізованих мов, таких як PHP, Python або JavaScript, можливо швидко написати код. Але без суворого введення завжди є ймовірність створення помилок. У міру зростання проектів потрібно завжди повертатися до кількох службових файлів або файлів компонентів, щоб перевірити, які методи доступні або які параметри вони очікують. Більшість

помилок виникає, коли очікується об’єкт користувача, але передано ідентифікатор користувача. Без підказки типу доведеться витрачати години й години на налагодження такої помилки. Проте, з Typescript від тіла запиту до відповіді – за умови використання суворої типізації можна зробити код швидше, і ймовірність створення необачних помилок зменшується. Nest – добре структурований найбільш доступний фреймворк, який повністю написаний на Typescript. Незважаючи на те, що можна використовувати Typescript у Express (дещо налаштувавши), але це не 100% Typescript.

Декоратори є запропонованою функцією JavaScript. Проте, Typescript вже має цю функцію. Усі основні серверні фреймворки Java, Python, Php активно використовують декоратори/анотації. Декоратори дозволяють програмувати метадані, механізм для модифікації класів, членів класу декларативним способом. У NEST можна прив’язати контролер до маршрутів, установити метод запиту, вставити тіло запиту або проаналізувати параметри, чудово керувати захистом авторизації/ролей за допомогою декораторів. За допомогою декораторів можна контролювати ряд подій: необхідність використання вхідного атрибуту; можливість перевіряти електронну пошту; необхідність створювати серії атрибуту у JSON. Приклад використання декораторів в NestJS наведено на рис. 3.24.

```
@Get('/me')
@ApiBearerAuth()
@UseGuards(RolesGuard)
@Roles('user', 'admin')
@UseInterceptors(ClassSerializerInterceptor)
async getMe(
    @AuthUser() user: IAuthUser,
): Promise<User> {}
```

Рисунок 3.24 – Приклад використання декораторів в NestJS

Nest надає простий спосіб декларувати власні декоратори. Отже, якщо є якесь загальне завдання, яке можна використовувати в різних частинах розробленої платформи, його записано як декоратор і використовують в багатьох місцях.

Nest надає спосіб визначення кількох модулів в одному кореневому модулі. Модулі інкапсулюють постачальників, контролер, репозиторії, перехоплювачі, проміжне програмне забезпечення тощо. Завдяки цьому логічно розподілено весь код проекту на логічні домени. Якщо постачальник/сервіс не визначено в тому самому модулі, його не можна додати до контролера чи класу сервісу. Якщо потрібно використовувати послугу з іншого модуля, цю послугу потрібно експортувати, і цей модуль потрібно включити в даний модуль. Завдяки цьому можна створювати чіткі межі для домену, писати модулі для багаторазового використання та писати слабозв'язаний і підтримуваний код.

Основною причиною зростання популярності Node.js є використання однієї мови як у бек-енді, так і в інтерфейсі. Основна перевага, яку отримано досі, це той самий синтаксис. Архітектура кодування, філософія, терміни – все було зовсім іншим. З точки зору архітектури NEST сильно натхненний Angular. Це модуль, служба, контролер, труба, декоратори – усе синтаксис, використання та філософія точно відповідають Angular. Фронтенд-розробник Angular може легко перейти до бекенд-проекту NEST – відчуваючи, ніби працюєш у тому самому проекті. З однаковою концепцією та філософією як для серверної, так і для зовнішньої частини без зайвих витрат часу на перемикання контексту.

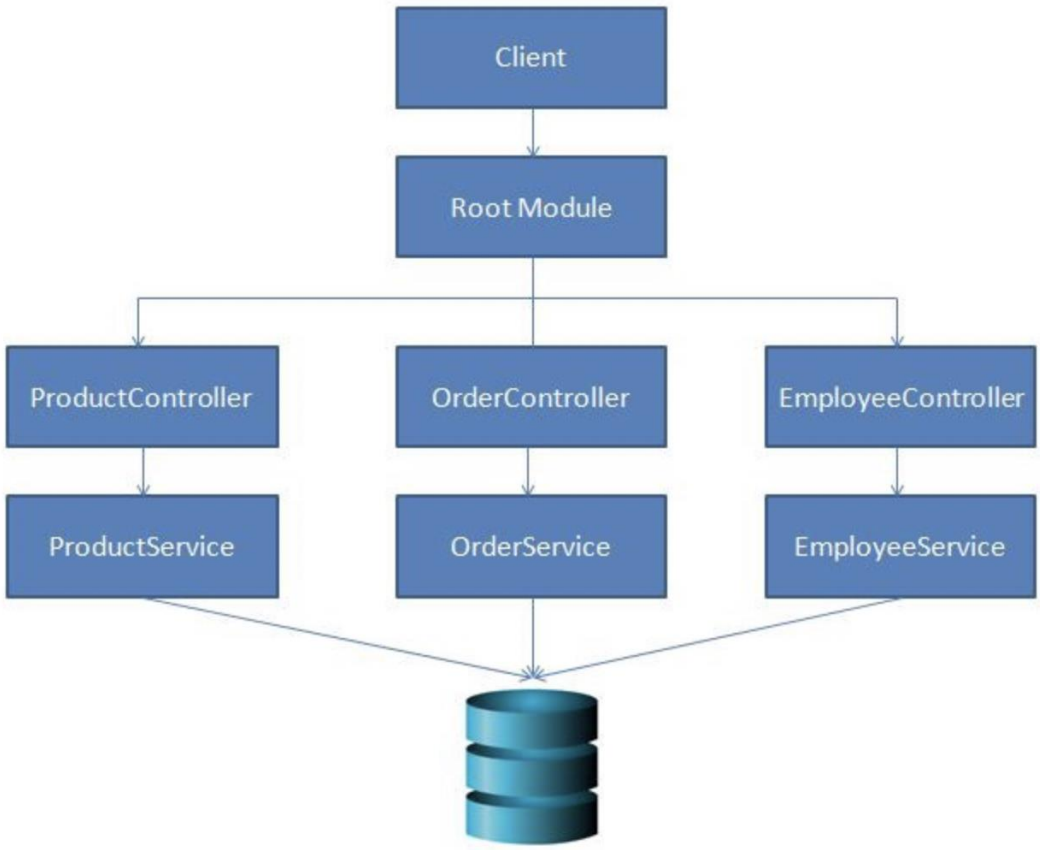
Інтерфейс командного рядка (CLI). Nest має власний інтерфейс командного рядка, щоб швидко ініціювати та створювати проект. Завдяки цьому існує можливість легко генерувати модулі, контролери, канали та проміжне програмне забезпечення.

Тестування. Запуск програми, створеної Nest CLI, постачається із середовищем тестування за замовчуванням, налаштованим за допомогою Jest framework. Щоразу, коли створюється служба, контролер, перехоплювачі тощо – CLI генерує файл «spec» разом із ним. Він надає автоматично згенерований

тестовий код, тому не потрібно писати необхідний код каркасу для модульного тестування. Отже, писати модульне тестування стає набагато простіше з NestJs.

Продуктивність. У переповненні стека є кілька потоків, де люди порівнюють тест продуктивності між Express і Nest. Оскільки сам Nest використовує Express, порівнювати їх безглуздо. Щоб отримати набагато кращу продуктивність, Nest надає альтернативний спосіб змінити реалізацію нижнього фреймворку з Express на Fastify (інший фреймворк вузла).

Архітектура. Якщо необхідно визначити архітектуру API, створеного за допомогою NestJS, це виглядає так: у нас є доступний кореневий модуль, який використовуватиметься для налаштування постачальників баз даних, визначення контролера, додавання проміжного ПЗ, додавання каналу та охоронців, а також для надання послуг. Приклад архітектури NestJS наведено на рис. 3.25



.

Рисунок 3.25 – Приклад архітектури NestJS

Існує можливість отримати модуль для кожного контролера. Як тільки модуль отримає запит, він буде знову перенаправляти до відповідного контролера (який оброблятиме запит). Послуга є необов'язковою, але її використовують у відповідності до принципу єдиної відповідальності (SRP).

Готова реалізація серверної частини додатку. Основою розробки серверної частини додатку є архітектура та структура файлів, з метою отримання оптимальної комунікації між модулями (рис 3.26).

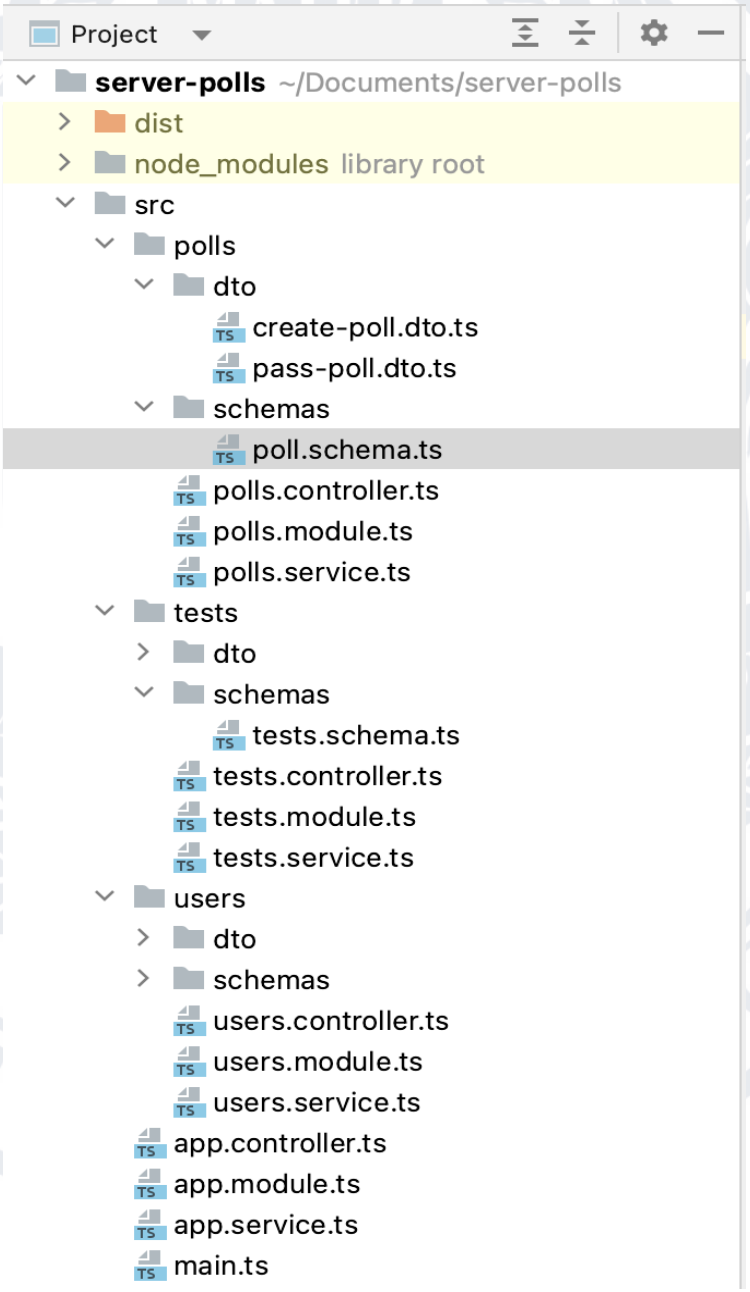


Рисунок 3.26 – Структура файлів

Проєкт складається з трьох модулів та одного головного, що зв’язує всі модулі, у кожному модулі є контролер, де описано всі запити по даній сутності та є сервіс, де описано функції, які будуть робитися при певному запиті та робота з базою даних. Dto, де описано усі інтерфейси об’єктів, з якими працюють, та є schema, де описано колекцію бази даних. Головний модуль збирає разом усі модулі та передає головному модулю, який в свою чергу підключає до комунікації з іншими модулями.

Модель зв’язків у базі даних для проведення голосування та опитування наведена на рис. 3.27.

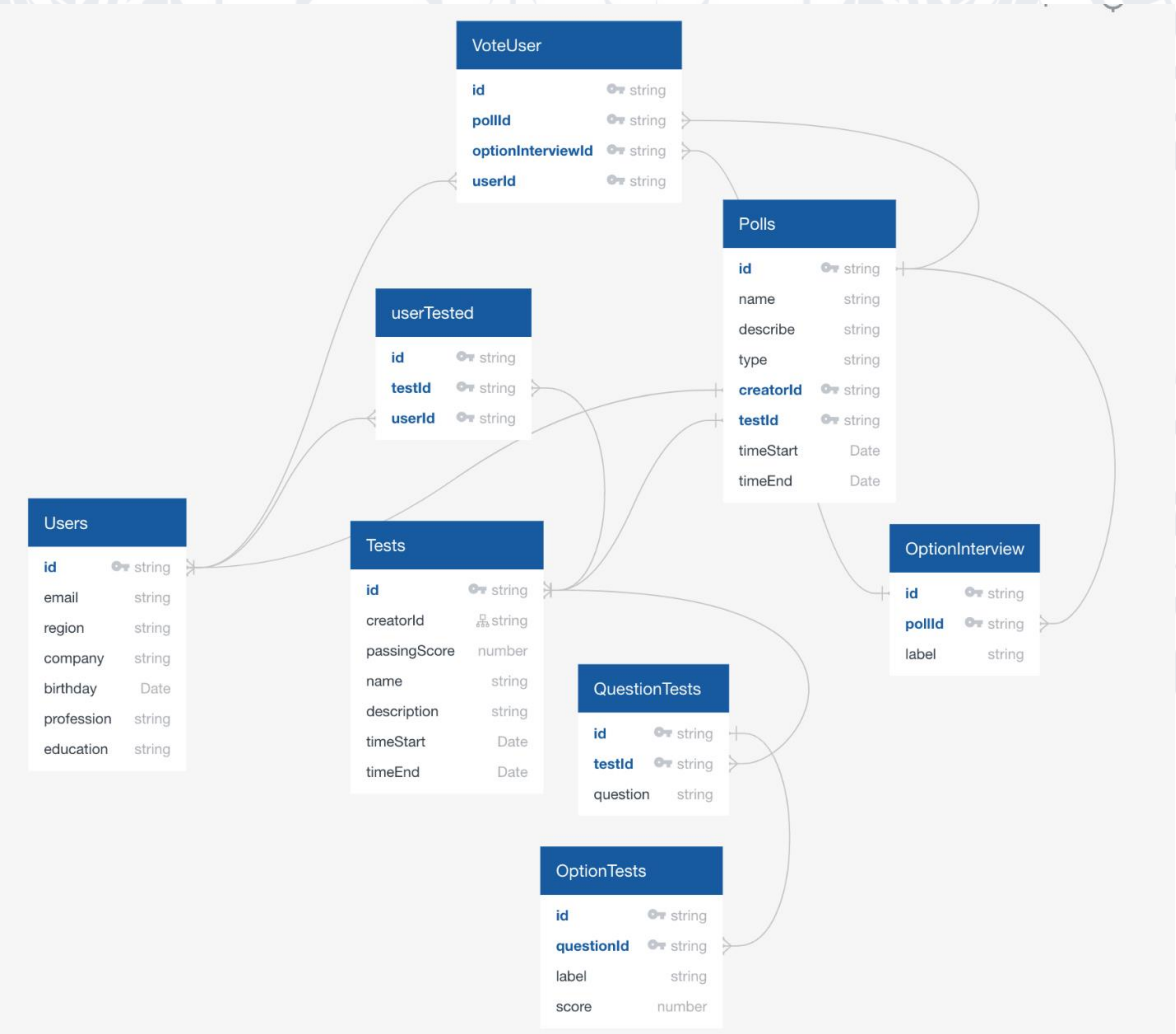
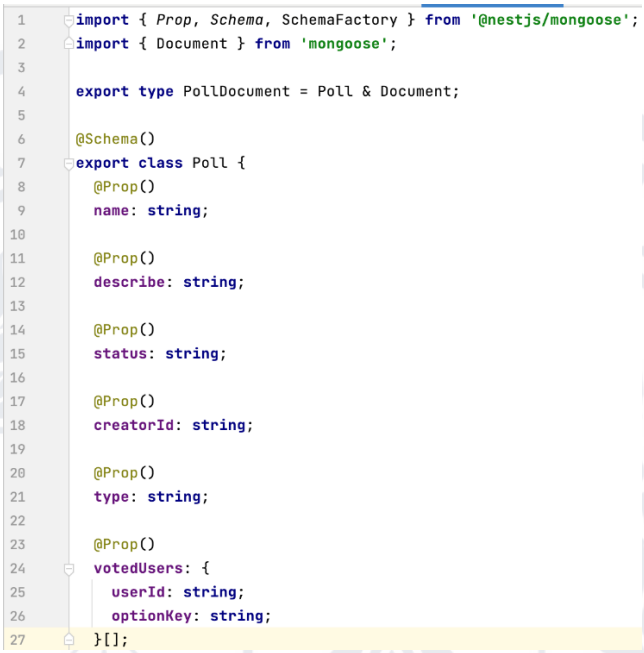


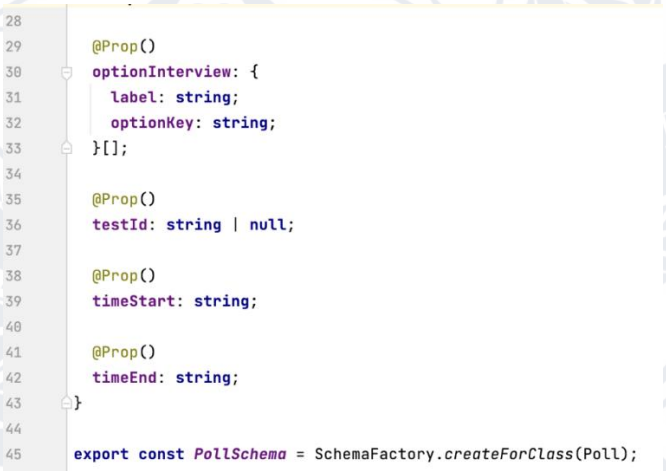
Рисунок 3.27 – Модель зв’язків бази даних для проведення голосувань та опитувань

Колекція голосувань для бази даних, що містить структуру уведеної інформації при голосуванні наведена на рис. 3.28 та рис. 3.29.



```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Document } from 'mongoose';

export type PollDocument = Poll & Document;

@Schema()
export class Poll {
  @Prop()
  name: string;

  @Prop()
  describe: string;

  @Prop()
  status: string;

  @Prop()
  creatorId: string;

  @Prop()
  type: string;

  @Prop()
  votedUsers: {
    userId: string;
    optionKey: string;
  }[];
```

Рисунок 3.28 – Колекція голосувань для бази даних_частина 1



```
  @Prop()
  optionInterview: {
    label: string;
    optionKey: string;
  }[];

  @Prop()
  testId: string | null;

  @Prop()
  timeStart: string;

  @Prop()
  timeEnd: string;
}

export const PollSchema = SchemaFactory.createForClass(Poll);
```

Рисунок 3.29 – Колекція голосувань для бази даних_частина 2

Робота над голосуванням завершується складанням звітних форм, які розміщують в собі кількість тих, хто прийняв участь у голосуванні; розподіл голосувань по запропонованих варіантах і т.д.

ВИСНОВКИ З РОЗДІЛУ 3

Основними складовими розробки платформи електронного голосування в умовах прямої демократії є: дизайн, клієнтська частина, сервер частина. Клієтька частина – це більш організований формат кодування складних інтерфейсів користувача, засобами якої користувачі можуть взаємодіяти з програмою та отримувати відповідну кількість відгуків та інформації у відповідь.

Знання компонентів життєвого циклу є необхідними розробникам створювати феноменальні веб-програми на основі React! Розробка серверної частини базується на модулях, кожен із яких має свої контролери. Контролери працюють з окремими сервісами, що в свою чергу пов’язані з окремою колекцією бази даних, та проводять перевірку даних, які присилає користувач платформи. Якщо типи даних неправильні або деяких даних не вистачає, повідомлення про помилку надходить користувачу та його запит не опрацьовується. Якщо всі дані правильні, то опрацьовується процес та повертається результат роботи.

Для оптимізації веб додатку введено режим формування запитів готових даних, щоб виключити можливість їх фільтрації у серверній або клієнтській частині.

ВИСНОВКИ

В результаті проведення досліджень алгоритмів електронного голосування в умовах прямої демократії було отримано наступні висновки:

1. В результаті дослідження теоретичних засад функціонування виборчих процесів та голосування було підтверджено їх ключову історичну роль в історії суспільного розвитку як основного елементу волевиявлення народу при реалізації своєї участі у прийнятті важливих рішень.

2. Процеси голосування розповсюджуються і функціонують на різних рівнях ієрархії управління суспільством: держави, регіону, підприємства. Особлива зацікавленість в організації таких процесів спостерігається на підприємствах, де подібна можливість створює підґрунття для прийняття колективних управлінських рішень за участю усіх працівників.

3. За результатами аналізу оцінок виборчих процесів визначено, що найбільш проблемними є: відкритість та поінформованість; встановлення несправедливих правил при організації; втручання в передвиборчий процес та механізм проведення; фальсифікації. Одним із шляхів подолання даних проблемних ситуацій є запровадження електронного голосування із використанням веб-платформ, які дозволять одночасно організувати, провести та оцінити механізми голосування, опитування та попереднє тестування учасників.

4. Проведена формалізація механізму процедури голосування дозволила розробити алгоритм організаційний механізм електронного голосування через веб-платформу шляхом створення особистого кабінету учасника з подальшою реалізацією процедури голосування на основі бази даних питань, що ініціюють вибір кандидату (альтернативні рішення). Включено блок реалізація створювання опитувань, що дозволяє відтворити можливість передбачення вподобань відносно поставленого вибору. Аналіз усвідомлення вибору реалізовано шляхом побудови тестування.

5. З метою отримання всебічної аналітики в спроектованій платформі

передбачено фільтрацію даних користувачів в розрізі критеріїв належності до регіону, рівня освіти, займаної посади, компанії. Це дозволяє отримати аналітику у вигляді графіків результатів голосувань (опитувань, тестувань) для моніторингу груп (кластери) за ознаками подібності виборчих вподобань та запропонованих альтернатив.

6. Розроблено та програмно реалізовано дизайн онлайн платформи засобами Figma (графічного онлайн-сервісу для створення прототипів сайтів та інтерфейсів), що відповідає поставленим вимогам розмірів для адаптивності під різні екрани пристроїв: мобільна версія працює за розміром нижче 550 пікселів широти; планшетна версія працює за розміром вище 550 пікселів та нижча 1200 пікселів; комп'ютерна версія працює за розміром вище 200 пікселів.

7. Для реалізації технічної частини був проведений аналіз варіантів різновидів архітектурних підходів, за результатами якого обрано архітектуру MVC (Model-View-Controller). Розробка клієнтської частини побудована засобами компонентів бібліотеки ReactJS, особливістю якої є наявність віртуального DOM дерева для швидкого оновлення компонентів.

8. Розробка серверної частину онлайн платформи для організації та проведення електронного голосування здійснена на базі використання NodeJS (серверний JavaScript) та фреймворку NestJS (розробка сервера модульним видом за окремими абстракціями).

9. База даних для інформації про процеси голосування (опитування, тестування) містить 8 таблиць та реалізована через документо-орієнтовану систему управління базами даних MongoDB, найбільш придатної до налаштування, масштабування та розширення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Бекешкіна І., Горбань Ю., Філіпчук І. Опитування громадської думки. Київ: Фонд «Демократичні ініціативи» імені Ілька Кучеріва, 2020. 110 с.

2. Бойко, О. (2020). Електронне голосування: перспективи для України. *Публічне урядування*, *23*(3), 24-34. URL: https://doi.org/10.32689/2617-2224-2020-3(23)-24-34

3. Головатий, М. Ф. Демократія: історія, теорія, практика: навч. посіб. для студ. вищ. навч. закл. та аспірантів-політологів. К.: ДП «Вид. дім «Персонал», 2011. 230 с.

4. Гомза І.А. Суспільно-політичні рухи: навчальний посібник. Київ: НаУКМА, 2018. 176 с.

5. Іщенко В. Сучасні дослідження суспільних рухів: головні теоретико-методологічні підходи. Соціальні виміри суспільства. Київ: Інститут соціології НАНУ, 2006. С. 183–184.

6. Ковальчук В.Б., Забокрицький І.І. Представницька демократія: поняття, ознаки, критерії. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2017/may/2222/ vnulpurn201582426.pdf.

7. Константинівська А.А. як у них? Електронне голосування: за матеріалами, підготовленими в рамках Програми USAID, що виконується Фондом Східна Європа та партнерами РАДА: підзвітність, відповідальність, демократичне парламентське представництво: URL: https://rada.oporaua.org/analityka/a-iak-u-nykh/23280-a-iak-u-nykhelektronne-holosuvannia.

8. Конституція України. URL: https://ips.ligazakon.net/document/Z960254K

9. Михальченко М., Самчук З. Порівняльний аналіз європейських виборчих систем. URL: https://ipiend.gov.ua/wp-content/uploads/2018/07/mychalchenko_porivnialnyi.pdf.

10. Олефір В., Валерій Б. Розрахунок обсягу вибірки як наріжний камінь планування наукового дослідження. Вісник Львівського університету. Серія

психологічні науки. 2021. Випуск 9. С. 186–195.

11. Палінчак М.М., Лешанич М.М. Виборча система України. Регіональні студії. № 24. 2021. С. 67-76.

12. Рибачук М., Шкурат І. Історичні аспекти становлення виборчої системи України. Політичний менеджмент. №1. 2004. С. 83-98.

13. Цифрові технології у виборах: питання, висновки та перспективи. [Електронний ресурс] – Режим доступу до ресурсу: https://rm.coe.int/publication digital-technologies-regulations-ukr/16809e8040.

14. Ancient Greek Democracy. [Електронний ресурс] – Режим доступу до ресурсу: https://www.history.com/topics/ancient-greece/ancient-greece-democracy.

15. Bernard Manin. The Principles of Representative Government.

16. Biplab Dasgupta. European Trade and Colonial Conquest.

17. Brian Solis. X: The Experience When Business Meets Design.

18. Burma: 20 Years After 1990 Elections, Democracy Still Denied. [Електронний ресурс] – Режим доступу до ресурсу: https://www.hrw.org/news/2010/05/26/burma-20-years-after-1990-elections-democracy-still-denied.

19. Democracy Facing Global Challenges [Електронний ресурс] – Режим доступу до ресурсу: https://web.archive.org/web/20190605230333/https://www.vdem.net/media/filer_public/99/de/99dedd73-f8bc-484c-8b91-44ba601b6e6b/vdem_democracy_report_2019.pdf.

20. Encyclopædia Britannica. Solon - Greek statesman and poet. [Електронний ресурс] – Режим доступу до ресурсу: https://www.britannica.com/biography/Solon.

21. Encyclopædia Britannica. Spartan magistrate [Електронний ресурс] – Режим доступу до ресурсу: https://www.britannica.com/topic/ephor.

22. Encyclopedia Britannica. Election. [Електронний ресурс] – Режим доступу до ресурсу: https://www.britannica.com/topic/election-political-science .

23. Eric W. Robinson. The First Democracies: Early Popular Government Outside Athens.

24. Failure to Vote. [Електронний ресурс] – Режим доступу до ресурсу: https://www.elections.wa.gov.au/vote/failure-vote.

25. Garner, J. L.; Walkling, R. A. (2009). "Electing Directors". Journal of Finance. [Електронний ресурс] – Режим доступу до ресурсу: https://onlinelibrary.wiley.com/ doi/10.1111/j.1540-6261.2009.01504.x

26. Glazer, Amihai; Glazer, Debra G.; Grofman, Bernard (1984). "Cumulative Voting in Corporate Elections: Introducing Strategy into the Equation". South Carolina Law Review. 35 (2). 295-311.

27. Google Forms. [Електронний ресурс] – Режим доступу до ресурсу: https://www.google.com/intl/ua-ua/forms/about/.

28. H. Levin. When the Great Power Gets a Vote: The Effects of Great Power Electoral Interventions on Election Results.

29. Herodotus [Електронний ресурс] – Режим доступу до ресурсу: https://www.gutenberg.org/ files/2707/2707-h/2707-h.htm.

30. N.C. board declares a new election in contested House race after the GOP candidate admitted he was mistaken in his testimony. [Електронний ресурс] – Режим доступу до ресурсу: https://www.washingtonpost.com/politics/candidate-says-new-congressional-election-warranted-in-north-carolina/2019/02/21/acae4482-35e0-11e9-854a-7a14d7fec96a_story.html.

31. Nitish K. Sengupta. Land of Two Rivers: A History of Bengal from the Mahabharata to Mujib.

32. Rachel Feig Vishnia. Roman Elections in the Age of Cicero: Society, Government, and Voting (Routledge Studies in Ancient History).

33. Reuven Hazan, 'Candidate Selection', in Lawrence LeDuc, Richard Niemi and Pippa Norris (eds), Comparing Democracies 2, Sage Publications, London, 2002.

34. Sandri, Giulia; Seddone, Antonella (11 September 2015). *Party Primaries in Comparative Perspective*. Routledge. p. 1.

35. Sham Election Law and Legal Definition. [Електронний ресурс] – Режим доступу до ресурсу: https://definitions.uslegal.com/s/sham-election/.

36. Solon the Lawgiver. [Електронний ресурс] – Режим доступу до ресурсу: http://agathe.gr/ democracy/solon_the_lawgiver.html.

37. The Athenian constitution. [Електронний ресурс] – Режим доступу до ресурсу: https://www.gutenberg.org/files/26095/26095-h/26095-h.htm#part08.

38. The crisis of election security. [Електронний ресурс] – Режим доступу до ресурсу: https://www.nytimes.com/2018/09/26/magazine/election-security-crisis-midterms.html.

39. V.K. Agninotri. Indian History.

40. Voting right before 1832. [Електронний ресурс] – Режим доступу до ресурсу: https://www.nationalarchives.gov.uk/education/resources/citizenship/.

41. What if blockchain technology revolutionised voting? European Parliamentary Research Service. Retrieved from: https://www.europarl.europa.eu/RegData/etudes/ATAG/2016/581918/EPRS_ATA(2016)581918_EN.pdf

42. База даних MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/.

43. Бібліотека Nest js. [Електронний ресурс] – Режим доступу до ресурсу: https://nestjs.com/.

44. Серверний JavaScript Node js. [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk/.

45. Трирівнева архітектура. Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/cloud/learn/three-tier-architecture

46. John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, Bill Wihl. Globally Distributed Content Delivery. 2017

47. Сервіс для генерації програми Jamstack. [Електронний ресурс] – Режим доступу до ресурсу: https://jamstack.org/generators/

48. Прогресивний веб-додаток. [Електронний ресурс] – Режим доступу до ресурсу: https://mobidev.biz/blog/web-application-architecture-types

49. Архітектура PWA. [Електронний ресурс] – Режим доступу до ресурсу: https://web.dev/learn/pwa/architecture/

50. Nikolas C. Zakas. Node.js and the new web front-end. [Електронний ресурс] – Режим доступу до ресурсу: https://humanwhocodes.com/blog/2013/10

/07/node-js-and-the-new-web-front-end/

51. Liz Parody. Choosing the right Node.js Framework: Next, Nuxt Nest?. [Електронний ресурс] – Режим доступу до ресурсу: https://nodesource.com/blog/next-nuxt-nest/

52. Стаття про MVC. [Електронний ресурс] – Режим доступу до ресурсу: https://levelup.gitconnected.com/mvc-vs-mvp-vs-mvvm-35e0d4b933b4

53. Бібліотека React js. [Електронний ресурс] – Режим доступу до ресурсу: https://reactjs.org/.

54. Бібліотека ReduxJS. [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org/

55. Психологія свідомості: теорія і практика. [Електронний ресурс] – Режим доступу до ресурсу: http://kpppo.nau.edu.ua/files/Konfer22019.pdf

ДОДАТКИ

Програмний код сервер частини веб-платформи для голосування

```
// app.module.ts

import { Module } from '@nestjs/common';
import { MongooseModule } from '@nestjs/mongoose';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { UsersModule } from './users/users.module';
import { PollsModule } from './polls/polls.module';
import { TestsModule } from './tests/tests.module';

@Module({
 imports: [
   UsersModule,
   PollsModule,
   TestsModule,
   MongooseModule.forRoot('mongodb://localhost/polls'),
 ],
 controllers: [AppController],
 providers: [AppService],
})
export class AppModule {}

// tests.module.ts

import { Module } from '@nestjs/common';
import { MongooseModule } from '@nestjs/mongoose';
import { UsersService } from './users.service';
import { UsersController } from './users.controller';
import { User, UserSchema } from './schemas/user.schema';

@Module({
 providers: [UsersService],
 controllers: [UsersController],
 imports: [
   MongooseModule.forFeature([{ name: User.name, schema: UserSchema }]),
 ],
 exports: [UsersService],
})
export class UsersModule {}
```

```
// tests.controller.ts

import { Controller, Post, Body, Get, Param } from '@nestjs/common';
import { TestsService } from './tests.service';
import { Test } from './schemas/tests.schema';
import { CreateTestDto } from './dto/create-test.dto';
import { PassTestDto } from './dto/pass-test.dto';

@Controller('tests')
export class TestsController {
 constructor(private readonly testsService: TestsService) {}

 @Post()
 Create(@Body() createTestDto: CreateTestDto): Promise<Test> {
  return this.testsService.create(createTestDto);
 }

 @Post('/pass')
 PassTest(@Body() passTestDto: PassTestDto): Promise<Test> {
  return this.testsService.passTest(passTestDto);
 }

 @Get('/:testId')
 GetTestById(@Param('testId') testId: string): Promise<Test> {
  return this.testsService.getTestById(testId);
 }

 @Get('/all/:creatorId')
 GetAllTestsById(@Param('creatorId') creatorId: string): Promise<Test[]> {
  return this.testsService.getAllTestsById(creatorId);
 }

 @Get()
 GetAll(): Promise<Test[]> {
  return this.testsService.getAllTests();
 }
}
```

```
// tests.services.ts

import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Model, Types } from 'mongoose';
import { Test, TestDocument } from './schemas/tests.schema';
import { CreateTestDto } from './dto/create-test.dto';
import { PassTestDto } from './dto/pass-test.dto';
import { UsersService } from '../users/users.service';

@Injectable()
export class TestsService {
 constructor(
  @InjectModel(Test.name) private testModel: Model<TestDocument>,
  private userService: UsersService,
 ) {}

 async create(testDto: CreateTestDto): Promise<Test> {
  const newTest = new this.testModel(testDto);

  return newTest.save();
 }

 async getTestById(testId: string): Promise<Test> {
  return this.testModel.findById(testId, {
   passingScore: 0,
   'tests.options.score': 0,
   usersTested: 0,
  });
 }

 async getTest(testId: string): Promise<Test> {
  return this.testModel.findById(testId);
 }

 async getAllTests(): Promise<Test[]> {
  return this.testModel.find(
   {},
   { passingScore: 0, 'tests.options.score': 0, usersTested: 0 },
  );
 }

 async getAllTestsById(creatorId: string): Promise<Test[]> {
  return this.testModel.find(
```

```
      { creatorId },
      { passingScore: 0, 'tests.options.score': 0, usersTested: 0 },
    );
  }

  async passTest(passTestDto: PassTestDto): Promise<Test> {
    const test = await this.testModel.findById(passTestDto.testId);
    let totalScore = 0;

    const userIsTested = test.usersTested.map((el) =>
el.userId).indexOf(passTestDto.userId) !== -1;

    if (userIsTested) {
      return test;
    }

    test.tests.forEach((el) => {
      for (let i = 0; i < el.options.length; i++) {
        if (el.options[i].optionId === passTestDto.result[el.questionId]) {
          totalScore += el.options[i].score;
          break;
        }
      }
    });

    test.usersTested.push({
      userId: passTestDto.userId,
      totalScore,
    });

    return test.save();
  }
}

// polls.modules.ts

import { Module } from '@nestjs/common';
import { MongooseModule } from '@nestjs/mongoose';
import { PollsService } from './polls.service';
import { PollsController } from './polls.controller';
import { Poll, PollSchema } from './schemas/poll.schema';

import { TestsModule } from '../tests/tests.module';
import { UsersModule } from '../users/users.module';
```

```
@Module({
  providers: [PollsService],
  controllers: [PollsController],
  imports: [
    TestsModule,
    UsersModule,
    MongooseModule.forFeature([{ name: Poll.name, schema: PollSchema }]),
  ],
})
export class PollsModule {}

// polls.controllers.ts

import { Controller, Post, Body, Get, Param } from '@nestjs/common';
import { PollsService } from './polls.service';
import { Poll } from './schemas/poll.schema';
import { CreatePollDto } from './dto/create-poll.dto';
import { PassPollDto } from './dto/pass-poll.dto';

@Controller('polls')
export class PollsController {
  constructor(private readonly pollService: PollsService) {}

  @Post()
  Create(@Body() createPullDto: CreatePollDto): Promise<Poll> {
    return this.pollService.create(createPullDto);
  }

  @Get()
  GetAllPolls(): Promise<Poll[]> {
    return this.pollService.getAllPolls();
  }

  @Get('/all/:creatorId')
  GetAllPollsById(@Param('creatorId') creatorId: string): Promise<Poll[]> {
    return this.pollService.getAllPollsById(creatorId);
  }

  @Get('/:pollId')
  GetPollById(@Param('pollId') pollId: string): Promise<Poll> {
    return this.pollService.getPollById(pollId);
  }

  @Post('/pass')
  PassPollWithTest(@Body() passPollDto: PassPollDto): Promise<Poll> {
```

```
    return this.pollService.passPull(passPollDto);
  }
}

// polls.services.ts

import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Model } from 'mongoose';
import { CreatePollDto } from './dto/create-poll.dto';
import { PassPollDto } from './dto/pass-poll.dto';
import { Poll, PollDocument } from './schemas/poll.schema';
import { Test } from '../tests/schemas/tests.schema';
import { TestsService } from '../tests/tests.service';
import { UsersService } from '../users/users.service';

interface PullAndTest {
  poll: Poll;
  test: Test;
}

@Injectable()
export class PollsService {
  constructor(
    @InjectModel(Poll.name) private pollModel: Model<PollDocument>,
    private testService: TestsService,
    private userService: UsersService,
  ) {}

  async create(pollDto: CreatePollDto): Promise<Poll> {
    const newPoll = new this.pollModel(pollDto);
    return newPoll.save();
  }

  async getAllPolls(): Promise<Poll[]> {
    return this.pollModel.find({}, { votedUsers: 0 });
  }

  async getAllPollsById(creatorId: string): Promise<Poll[]> {
    return this.pollModel.find({ creatorId }, { votedUsers: 0 });
  }

  async getPollById(pollId: string): Promise<Poll> {
    return this.pollModel.findById(pollId, { votedUsers: 0 });
  }
```

```
  async passPull(passPollDto: PassPollDto): Promise<Poll> {
    const poll = await this.pollModel.findById(passPollDto.pollId);

    if (new Date().getTime() > new Date(poll.timeEnd).getTime()) {
      // todo: time for vote ended
      return poll;
    }

    if (poll.testId) {
      // todo: if we have test for this poll then we will do this
      const dataTest = await this.testService.getTest(poll.testId);
      const userIsTested = dataTest.usersTested.find(
        (el) => el.userId === passPollDto.userId,
      );

      if (!userIsTested) {
        // todo: not tested
        return poll;
      }

      if (userIsTested?.totalScore < dataTest.passingScore) {
        // todo: less score than need for passing poll
        return poll;
      }
    }

    const indexIfUserVoted = poll.votedUsers
      ?.map((el) => el.userId)
      .indexOf(passPollDto.userId);

    if (indexIfUserVoted) {
      // todo: if we voted we can change us vote
      poll.votedUsers.splice(indexIfUserVoted, 1);
    }

    poll.votedUsers.push({
      userId: passPollDto.userId,
      optionKey: passPollDto.optionKey,
    });

    return poll.save();
  }
}
```

Онуфрієв Олексій Сергійович
Прізвище, ім’я по батькові

Інформаційних і прикладних технологій
Факультет

122 «Комп’ютерні науки»
Шифр і назва спеціальності

«Комп’ютерні технології обробки даних (Data Science)»
Освітня програма

ДЕКЛАРАЦІЯ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ

Усвідомлюючи свою відповідальність за надання неправдивої інформації, стверджую, що подана кваліфікаційна (магістерська) робота на тему: «Дослідження алгоритмів електронного голосування в умовах прямої демократії» є написаною мною особисто.

Одночасно заявляю, що ця робота:
- не передавалась іншим особам і подається до захисту вперше;
- не порушує авторських та суміжних прав, закріплених статтями 21-25 Закону України «Про авторське право та суміжні права»;
- не отримувалась іншими особами, а також дані та інформація не отримувалась у недозволений спосіб.

Я усвідомлюю, що у разі порушення цього порядку моя кваліфікаційна робота буде відхилена без права її захисту, або під час захисту за неї буде поставлена оцінка «незадовільно».

05.12.2022
(дата)


(підпис здобувача)