Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ВЕБ-РЕСУРС ДЛЯ АВТОМАТИЗАЦІЇ АНАЛІЗУ МОЖЛИВОСТІ РЕАЛІЗАЦІЇ ПРОГРАМНИХ ПРОЕКТІВ

Текстова частина до магістерської роботи
за спеціальністю «Інженерія програмного забезпечення»

Виконав студент 2-го року навчання
Усачов Кирило Юрійович

Керівник Ковалюк Т. В.
доцент, к.н.

Магістерська робота захищена
з оцінкою «_____»

Секретар ДЕК _____
«___»_____ 2021 р.

Київ 2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
доц., канд. фіз.-мат. наук
_____ С. С. Гороховський
(підпис)
1 листопада 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на магістерську роботу

студенту Усачову К. Ю. факультету інформатики 2-го курсу МП
Тема: «Веб-ресурс для автоматизації аналізу можливості реалізації програмних проектів»

Зміст ТЧ до дипломної роботи:

Індивідуальне завдання
Вступ
1. Дослідження
2. Розробка власної системи
3. Практичне застосування
Висновки
Список літератури

Дата видачі 1 листопада 2021 р.

Керівник ___
(підпис)
Завдання отримав _____
(підпис)

Календарний план виконання роботи:

№ п/п	Назва етапу дипломної роботи	Термін виконання	Примітки
1	Отримання завдання на дипломну роботу	01.11.20	
2	Огляд теоретичного матеріалу за темою	15.01.21	
3	Опис теоретичної частини	15.02.21	
4	Продумування вимог до практичної частини	01.03.21	
5	Проектування	01.04.21	
6	Написання пояснювальної роботи	01.05.21	
7	Попередній захист	14.05.21	
8	Аналіз та корегування	20.05.21	
9	Створення слайдів для доповіді та написання доповіді	01.06.21	
10	Захист	16.06.21	

Анотація

Дипломна робота описує можливість автоматизувати шляхом створення веб-ресурсу аналіз програмного проекту з огляду на можливість його реалізації.

Перший розділ містить дослідження стосовно шляхів та критеріїв оцінки проектів, дано визначення та розібрано основні параметри, за якими можна робити аналіз.

Другий розділ описує реалізацію власного веб-ресурсу за допомогою сучасних бібліотек. У цьому розділі показано процес створення системи, описано її структуру, компоненти та використані інструменти.

Третій розділ демонструє практичне використання створеного застосунку. У даному розділі описано користувацький інтерфейс, та взаємодію з ним.

Вступ

Із появою складної комп’ютерної техніки стало можливим виконання обрахунків для тих задач, обчислення для яких були занадто складними у виконавстві людського мозку. Після розповсюдження калькулятора будь-хто може за декількасот наносекунд дістати корінь з будь-якого числа, у той час як п’ятдесят років тому для цього необхідно було просидіти деякий час над папером із розрахунками.

Утім, не до кінця довіряючи складним механізмам, ми все ще часто за інерцією надаємо перевагу своїй власній роботі, навіть у тих випадках, коли скористатися електронікою було б зручніше. Така ситуація можлива у трьох випадках: або людина не до кінця довіряє стороннім засобам, надаючи пріоритет емпіричному досвіду, або ж поставлена задача є занадто розмитою або алгоритмічно нерозв’язною, або ж програмних засобів для конкретної поставленої задачі просто не існує на даний момент. І якщо в перших двох випадках розробник фактично безсилий, то в третьому ми, як ті, хто створює програмне забезпечення, маємо на одну нову задачу більше.

Коли мова стосується аналізу нового проекту, ми частково маємо усі три випадки із зазначених вище: досліджуючи задачу на реалізованість, людина опирається на емпірику, їй складно виділити певний алгоритм для аналізу, адже задача з’ясування можливості реалізувати проект є занадто широкою і нечіткою у розумінні машини — забагато змінних, неможливість представити задачу як набір формул, — усе це унеможливлює алгоритмізацію даної задачі. Утім, ми можемо зв’язати людський досвід і машинну логіку, звужуючи набір критеріїв, які має надати алгоритму людина сама; набір, який буде формально визначений і завдяки якому можна буде провести повноцінний аналіз проекту уже з формальної, алгоритмічної точки

зору. Тут у гру вступає програмне забезпечення, яке має надати користувачу шлях повідомити усі необхідні дані у потрібній формі, що дасть можливість проаналізувати їх згідно завчасно заданим формулам і критеріям. Дослідженню цих критеріїв та реалізації відповідної системи і присвячена подальша робота.

1 Дослідження

1.1 Концепція

Метою роботи автора є дослідження шляхів аналізу можливості реалізувати програмний проект. Багато ідей намагаються знайти втілення, але з багатьох причин далеко не всім це вдається. На успіх впливає багато змінних, починаючи з людського фактору і закінчуючи стохастичними непередбачуваними подіями глобального масштабу, які впливають або на сам проект, або на його актуальність, або ж спонукають змінити шлях реалізації у той момент, коли це зробити неможливо.

Для визначеності в межах дослідження введімо певні поняття, на які будемо спиратись у подальшому. По-перше, проектом будемо вважати весь процес планування, реалізації та експлуатування певної ідеї, втіленої у чітко окресленому результаті, який ми будемо іменувати кінцевим продуктом; при цьому немає значення, є цей кінцевий продукт витвором, абстрактним концептом або ж фізичним втіленням ідеї — для нас важливим є наявність певної роботи, що йде за визначеною схемою, триває певний час, і приводить до результату. Звідки маємо сформувати деякі рамки, у межах яких на певний процес буде розповсюджуватись поняття проекту.

Очевидно, що виконуючи аналіз алгоритмічним шляхом, ми не можемо спиратися на випадкові параметри, які не можемо передбачити. З іншого боку, брати до уваги лише очевидні (визначені, невипадкові) дані під час аналізу погано через дві причини:

1) хоч на відміну від випадкових, ці знання і є завчасно визначеними, самі випадкові події можуть непередбачуваним чином внести зміни до системи;

2) надавати будь-які дані буде людина, інтерпретуючи їх суб’єктивно, що позбавляє нас гарантії дати ту оцінку, яку очікує користувач.

Наведемо приклад, щоб продемонструвати перший пункт[1]. Забудовник планує звести будинок на підготовленій ділянці. Проект поділено на етапи, для кожного визначено терміни та необхідні матеріали; у процесі реалізації n-го етапу відбувається стихійне лихо, що проявляється у зсуві поверхневих шарів ґрунту, що робить ділянку непридатною для забудови за поточним розподілом ресурсів/часу, оскільки необхідний додатковий час та витрати на усунення проблеми. До початку робіт алгоритмічний аналіз даного проекту дасть завищений позитивний результат, порівняно з тим, що ми отримали.

Приклад до другого пункту. Людина, відповідальна за прокладання доріг, хоче спланувати нову розв’язку з метою розрідити трафік, при цьому в очікуваннях людини створення нової якісної дороги призводить до збалансування завантаженості доріг і зниження витрат на їх ремонт (вважаємо, що легкий ремонт обидвох доріг — старої і нової — менш витратне, аніж глибинний ремонт однієї дороги, коли весь трафік йде нею). Після надання вхідних даних по проекту, алгоритм дасть певну оцінку[2], але в дійсності виявиться, що більшість людей їхатиме новою дорогою, що збільшить загальний простір для авто, але підвищить витрати на ремонт дороги (оскільки більшість навантаження припадає на неї, то вона потребуватиме глибинного ремонту, при цьому вартість його буде вища, ніж у випадку ремонту старої ділянки дороги, оскільки нова потребує більш якісних матеріалів). Отже оцінка, дана алгоритмом, якою б вона не була,

[1] Хоч метою поставленої задачі є розробка системи аналізу саме програмних проектів, приклади і терміни можна застосувати до будь-яких проектів, адже програмний проект — це такий же проект, як і будь-який інший, просто з передчасно визначеним обмеженням за предметною областю.

[2] Очевидно, що оцінка буває як завищеною, так і заниженою.

виявиться завищеною через некоректну інтерпретацію наявних вхідних даних людиною.

Беручи до уваги все вищезазначене, не існує алгоритмічного шляху надати універсальний аналіз усього проекту. Алгоритм можна скласти лише за допущення, що проект розробляється в певній системі, впливом зовнішніх факторів на яку можна знехтувати. Через це ми також маємо ввести певні обмеження щодо того, чи будемо ми вважати деякий процес аналізованим проектом:

а) **достатня умова**: процес має бути обмежений у часі; відомі умови, за яких буде відбуватись весь процес;

б) **необхідні умови**:

1) при розбитті процесу на етапи, для кожного етапу e_i ($e_n \in E$, де E — множина етапів, на які розбито проект) відомий набір ресурсів r_i ($r_n \in R$, де R — множина *усіх* доступних ресурсів), необхідний для виконання етапу e_i.

2) для кожного етапу e_i визначено максимальний час $t_i^{(max)}$ його виконання, або ж користувач гарантує, що на момент настання етапу e_i набір ресурсів r_i, необхідний для виконання цього етапу, буде доступний.

Зазначимо, що множина етапів, з яких складатиметься проект, може буде необмеженою, адже проект може не мати завершального терміну (приклад: браузер, який постійно оновлюється і випускає нові версії, не має наперед визначеного кінцевого терміну підтримки, завершення оновлення проекту означає припинення існування проекту).

Для кожного етапу виконання проекту (якщо розбивати його на етапи) необхідно виділяти певну кількість ресурсів. Ресурсом може бути що завгодно, включаючи незадіяну робочу силу, здатну виконувати проект, фінанси, дані тощо. Також для того, щоб ми мали право застосовувати алгоритм, користувач має гарантувати доступність ресурсів для кожного етапу, інакше алгоритм не зможе дати коректну (відносно очікувань людини) оцінку, адже за відсутності гарантії щодо стабільності і доступності ресурсів виконання проекту може піти недетермінованим шляхом, який не можна проаналізувати алгоритмом, а значить, не буде сенсу його застосовувати. Приклад некоректного застосування алгоритму: група ентузіастів планує розробку відеогри, при цьому кошти на проект збираються на добровільні пожертвування. Не маючи в розпорядженні художників та модельєрів, команда вирішує задіяти третю сторону для створення ігрових моделей. Залежно від бюджету можуть бути замовлені унікальні моделі, або ж взяті/придбані на платформах із загальнодоступними ресурсами. Намагаючись проаналізувати успішність проекту, керівники команди звертаються до алгоритму, при цьому на запит системи щодо унікальності використаних засобів неможливо буде дати точну відповідь, адже проект фінансується добровільно і бюджет не є фіксованим, а відповідно, невідомо, чи буде можливість придбати унікальні моделі або лише наявні у загальному доступі. При вказанні випадкового значення для даного пункту алгоритм надасть оцінку лише за допущення, що очікування щодо цього пункту були виконані, що може не відображати дійсність і ввести користувача в оману.

Стосовно критеріїв оцінки, ми маємо визначитись, яким саме шляхом потрібно проводити аналіз. Очевидно, що дані буде надавати людина, програмі ж потрібно буде відштовхуватись від наданої інформації. При

цьому дані завжди мають бути надані в єдиній формалізованій формі. Очевидне рішення лежить у наданні контрольних запитань користувачеві та аналізі відповідей. Форма відповідей має бути такою, щоб її можна було проаналізувати алгоритмічно, тобто закрита. Інший спосіб — емпіричний аналіз і відкрита форма відповідей, але такий спосіб потребує штучного інтелекту, який зможе аналізувати контекст та людську мову, виділяти з неї сенс. Автор прийняв рішення зупинитись на першому більш простому варіанті через дві причини:

1) реалізація штучного інтелекту потребує серйозних зусиль і допоки не існує ідеального універсального способу реалізувати подібний інтелект;

2) для поставленої задачі ускладнення механізму аналізу є надлишковим, адже за наявності умов, описаних у попередньому пункті задача зводиться до алгоритмічно розв’язної, а отже, може бути реалізована детермінованим аналізатором.

1.2 Критерії оцінки проекту

Для того, щоб оцінити потенційну успішність ідеї, необхідно визначити параметри, за якими буде відбуватися оцінка. Очевидно, що чим більше параметрів ми матимемо, тим більш точною може бути аналіз. Кожному критерію будуть відповідати питання, які необхідно підібрати таким чином, щоб людині було просто відповісти на них, а у випадку, якщо на конкретне питання буде відповісти складно, суміжні питання мають бути схожими, що дасть можливість з більшою ймовірністю покрити відповідями ті невідомі змінні, які нам бажано було б знати.

Розглянемо набір критеріїв, за якими будемо виконувати аналіз майбутнього проекту.

1.2.1 Наявність даних

Перш за все потрібно з’ясувати, чи достатньо інформації щодо бізнес-проекту ми маємо. Для того, щоб проект був успішним, нам необхідно мати повні знання про предметну область, вивчити та дослідити об’єкт нашої роботи. Неможливо зробити програму для діагностики хвороб, не знаючи медицини, або ж програму для обрахування параметрів небесних тіл, не знаючи астрономії.

У випадку замовлення зі сторони нам мають надати спеціалістів, які зможуть консультувати розробників і виступати у ролі посередників між спеціалістами чисто технічного профілю та, власне, областю, у межах якої ми ведемо роботу. Розробник також повинен мати повну уяву щодо того, що він взагалі створює, яке призначення несе створюваний ним продукт. Це необхідно не тільки для можливих зауважень та пропозицій, які може надати програміст, а й для зменшення можливих проблем, які можуть виникнути у процесі розробки та підтримки, що може потребувати написання фіксів та оновлень, що в свою чергу призводить до небажаного затягування строків реалізації проекту, фінансових втрат, часу тощо.

У разі відсутності спеціалістів предметної області або відсутності спеціалістів-консультантів необхідно докласти зусиль і провести власне дослідження, якщо ж такої можливості немає, є підстави вважати, що варто задуматись над тим, чи братися за такий проект, адже є ризик не впоратися із складнощами його реалізації і втратити час та вкладені зусилля.

Достатнім рівнем інформації слід вважати наявність базових знань для неспеціалізованих проектів (базове розуміння або досвід створення подібних проектів у керівника проекту або членів команди) і наявність внутрішнього або стороннього спеціаліста у випадку спеціалізованих проектів (тих, що потребують глибинних знать, на кшталт наукових проектів, або ж проектів областей із великим рівнем відповідальності: медицина, будівництво, військова справа тощо).

1.2.2 Конкурентоспроможність

Тут варто розглянути два випадки:

а) проект не є унікальним;

б) проект є унікальним і подібних немає.

У випадку коли проект не є унікальним, ми маємо бути впевненими, що наш продукт буде принаймні на порядок кращим за продукт конкурентів або ж що наше програмне забезпечення буде розповсюджуватись на умовах, які будуть більш вигідними для кінцевих користувачів з точки зору надання переваги саме нашому продукту. Це може бути

1) ціна;

2) бренд;

3) доступність;

4) мультиплатформенність;

5) якісна реклама;

6) відкритість коду;

7) надання можливості третім особам створювати доповнення до свого проекту (плагіни, розширення, тощо).

Якщо ж ми не можемо гарантувати виконання цих пунктів, є сенс відмовитись від виконання проекту або переглянути можливість реалізувати один або декілька з вищезазначених пунктів.

У випадку ж коли проект є унікальним і подібних немає, ми маємо визначити наскільки можливо з боку конкурентів буде створити проект, який буде кращим за наш. Ми маємо бути впевненими, що такого ризику немає, адже це швидко призведе до того, що наш проект втратить свою цінність, а також що конкуренти не будуть заохочені прийняти рішення, яке переверне конкуренцію в позитивний для них бік, навіть за умови відставання їхнього продукту в якості. Тут є два можливих виходи:

а) прийняти ризик; у такому разі ми маємо бути впевненими, що зможемо втримати позицію на ринку завдяки бренду, концепції тощо. Має бути запасний варіант для утримання ідеї на плаву. Має бути передбачено можливості випустити оновлення системи і/або перебудови (випуску нової версії);

б) неприйняття потенційного ризику; якщо є підстави вважати, що потенційний можливий продукт іншої фірми/компанії/команди буде кращим, є сенс відмовитися від проекту або внести до планів зміни, які змінять ризики або усунуть їх.

Як приклад можна згадати розробку браузеру Netscape Navigator, яка свого часу програла конкуренцію MS Internet Explorer. Продукт компанії Netscape був потенційно кращий, тим не менш, Microsoft зробила свій браузер безкоштовним і доступним за замовченням будь-кому, хто

користувався їхньою операційною системою, що зіграло свою роль у припиненні існування Navigator.

1.2.3 Новизна

Для успішності проекту ми маємо гарантувати, що користувач отримає щось нове, нові можливості, яких він не мав раніше. Людині властиво нехтувати новими продуктами, якщо досвід їхнього використання мало чим буде відрізнятись від того, із чим він уже має справу або мав справу раніше.

Новизна не завжди означає появу нових функцій в програмі, це може проявлятись в таких речах як

1) модернізація інтерфейсу на унікальний/більш сучасний;

2) інтеграція з іншими системами;

3) власне, нові функції.

Новизна не обов’язково означає появу можливостей, яких не було ніде раніше в принципі, це також може бути додавання нового контенту у ваш проект, який вже знаходиться в експлуатації. Якщо ж ви не бачите шляхів або ідей внести різноманіття в свій продукт, він може швидко набриднути користувачам; також не можна виключати можливість появи нових ідей у конкурентів, які зможуть реалізувати можливості, від яких ви відмовились. У такому разі є сенс задуматись над тим, чи буде ваш проект конкурентоспроможним у перспективі, навіть якщо в режимі «тут і зараз» він обіцяє бути успішним. Якщо таких перспектив не можна досягнути, є сенс переглянути реалізацію проекту. Особливо цей пункт варто підкреслити розробникам забезпечення, що спеціалізується на розважальному контенті

(game development, мультимедіа тощо, розважальні портали тощо), а також усім іншим галузям, орієнтованим на залученість людей, де новизна та унікальність є критичними.

1.2.4 Сприйняття

Плануючи новий проект, варто задуматись над тим, наскільки зрозумілим він буде людям. Дослідження показують [1], що 70 відсотків проектів, що не окупуються, не викликали сумнівів у розробників стосовно того, що продукт буде сприйнятий користувачами і буде масово використовуваний.

Дуже часто у розробників виникає необхідність у створенні інструменту, який зміг би взяти на себе або зробити зручнішим виконання певних задач, із якими стикається сам майбутній розробник, через що виникає хибне відчуття того, що даний інструмент однаково необхідний багатьом іншим. Утім, дуже часто отриманий інструмент виявляється корисним лише для специфічної категорії людей, або ж взагалі не стає вживаним і, відповідно, несе збитки.

Для уникнення подібних проблем варто проводити попередній аналіз ринку на предмет того, чи є потреба в даному продукті взагалі і чи буде він сприйнятий так, як того бажає сам розробник. Для цього варто спиратися на наступні критерії:

1) чи ті задачі, які вирішує програмний продукт, є поширеними серед аудиторії, на яку націлений замовник/автор ідеї;

2) чи є проект прийнятним для цільової аудиторії, чи не порушує він чиїсь права або не вирішує поставлені задачі шляхом, який користувачі не можуть очікувати; чи є продукт безпечним як з фізичної точки зору, так і з огляду на кібербезпеку;

3) чи планується документація проекту/створення довідки/допоміжних інструкцій, чи є продукт інтуїтивним для користувачів.

У разі відсутності даних пунктів варто переглянути план проекту або прийняти ризик не бути сприйнятим своєю аудиторією.

Продовжуючи тему прийнятності, варто не забувати про формальну частину випуску проекту. Випускаючи новий продукт, потрібно завчасно знати, чи буде необхідним для нього отримання дозволу або ліцензія на використання окремих компонентів.

Дуже часто успішний проект стикався з юридичними проблемами через конфлікти на фоні ліцензування. Приклад: після придбання проекту Android компанією Google у 2005 році було розвинуто ідею створення операційної системи для різних мобільних пристроїв. Для вирішення цієї задачі було створено власну віртуальну машину Dalvik на основі проекту Apache Harmony, який працював на мові Java. Якщо відштовхуватися від інтерпретатора мови програмування як втілення самої мови, компанія Google тим самим створила власну версію Java, не сплативши ліцензійні нарахування оригінальним авторам, що призвело до судового акту.

Окремої уваги заслуговує факт використання бібліотек та сторонніх рішень. Багато наявних рішень за свою якість пропонують оплатити ліцензію, що має бути враховано під час розробки проекту, особливо пропрієтарного характеру. Якщо керівництво проекту не планує виділення

частини бюджету на придбання допоміжного програмного забезпечення, варто задуматись, чи існує можливість власноруч реалізувати необхідні для функціонування проекту можливості. Якщо створення допоміжних бібліотек планується, і функції, які вони надають, можуть мати цінність при розробці іншого програмного забезпечення, варто розглядати можливість їхнього розповсюдження як окремого проекту, що може принести додатковий прибуток.

1.2.5 Повторне використання

Тенденції постійно рухаються вперед і розроблена система має адаптуватися під ці зміни. Будь-яке програмне забезпечення застаріває, розробник має передбачити шляхи оновлення своєї роботи, інакше з’явиться нова більш конкурентна система, що забере частину цільової аудиторії. Розвивати проект можна декількома шляхами:

1) внесенням нових функцій/можливостей;

2) оновленням супроводжувального коду (плагінів, бібліотек тощо) задля швидшої та більш стабільної роботи;

3) адаптуванням інтерфейсу під більш сучасну стилістику;

4) переходом на нові платформи (порт на нові операційні системи, тощо);

5) створенням різних версій однієї і тієї ж системи у різних середовищах (створення веб-версії програми, мобільних версій для різних платформ тощо).

Історія знає багато прикладів, коли порт на різні системи вдихав нове життя у забуті проекти. Основна мета розширення доступних платформ — залучення нової аудиторії.

Створення додаткових версій для мобільних пристроїв, а також для користувачів Web на момент проведення дослідження стає фактично обов’язковим стандартом, коли справа стосується великих систем.

Якщо план проекту не передбачає оновлення або розширення системи будь-яким іншим шляхом, варто звернути на це увагу і задуматись над можливістю покращення проекту у майбутньому, або ж прийняти ризик щодо ймовірності, що проект буде швидко забутий, що зі свого боку не дасть йому можливість розкрити всі перспективи, а розробникам отримати з цього прибуток.

Не обов’язково наробітки мають бути корисними для одного й того ж проекту, можна використати написаний код для реалізації нових ідей. Варто замислитись, чи зможе проект або компоненти, створені під час його розробки, бути корисними для інших майбутніх робіт. Більшість коду, який створює програміст, містить логіку, яку можна виділяти в окремі модулі. Ці модулі можуть бути використані повторно, що може зекономити час у подальшому. Однак не завжди розробники звертають на це увагу, починаючи нові проекти з нуля, ігноруючи можливість зекономити свої зусилля. Починаючи новий проект, слід завчасно спланувати, які функції буде необхідно реалізувати і які з них можна виділити в окремі модулі, бібліотеки або і окремі програми або цілі проекти. Компанія Valve свого часу виокремила рушій, на якому створювала свої продукти (Source) як самостійний продукт, який стали використовувати різні команди для різних

проектів, що призвело до його популяризації та розвитку, а це в свою чергу вплинуло на дохід компанії.

Якщо план проекту не передбачає розробку окремих модулів, або ж є дуже простим, варто замислитись над корисністю такого продукту. Якщо ж проект планується як немасштабний або одноразовий, тоді даний пункт не є суттєвим. Утім замислюватись над розділенням коду і його повторним використанням хоча б у межах поточного проекту варто завжди.

1.2.6 База проекту

Проект можна починати повністю з нуля, а можна створювати на основі вже наявних наробіток. Як було зазначено у попередньому пункті, значна частина коду повторюється від одного проекту до іншого і повторювану функціональність є сенс виділяти в окремі модулі, звідки слідує, що для створення нового проекту було б корисно використати вже наявний код. Що більший відсоток коду можна буде взяти зі сторонніх рішень, то менше часу доведеться витрачати на написання одних і тих же функцій, і відповідно, тим швидше можна буде перейти до експлуатації.

Наразі майже жоден серйозний проект не створюється без участі інших бібліотек або фрейморків. Поширення відкритого коду і його багаторазове використання в різних системах є основою ідеології відкритого коду (open source), проте використовуючи такий код у своїх роботах, варто звертати увагу на ліцензійні та авторські питання. Відкритість коду не завжди означає можливість його безоплатного використання, що вже було досліджено водному з попередніх пунктів, утім якщо розробник упевнений, що

інструментарій (за умови, що він існує), який він планує використати, підходить для проекту, це є перевагою, яку варто враховувати при аналізі.

Окремо стоїть питання якості сторонніх рішень, які ми беремо в свій проект, передусім важлива наявність гарної документації, відсутність якої може, навпаки, сповільнити робочий процес або ж взагалі призвести до необхідності пошуку нових сторонніх рішень, із чим на практиці розробники постійно зустрічаються.

Якщо ваш проект використовує декілька (перевірених іншими розробниками) сторонніх рішень із гарною документацією, це є бонусом для проекту, який варто враховувати під час аналізу.

1.2.7 Часові межі

Для будь-якого проекту мають бути встановлені часові терміни реалізації; у випадку довгострокового або циклічного проекту мають бути встановлені приблизні межі до введення в експлуатації першої версії.

Часовий показник є одним із найважливішим при аналізі проекту. Темп сучасної розробки форсує програмістів якомога швидше виконувати свою роботу, адже несвоєчасний вихід продукту на ринок може означати появу нових рішень, які будуть більш конкурентоспроможними. Інша ситуація, але зі схожим результатом — приурочення програмного проекту до певної події, після проходження якої сама програма буде неактуальною. Як приклад можна навести систему моніторингу захворювань COVID; подібна система буде корисна під час пандемії, проте стане неактуальною після її завершення, звідки слідує, що на обмеження по проекту накладається ще й часові рамки, недотримання яких знівелює будь-яку цінність розробки, не кажучи про те,

що за цей час встигнуть з’явитися аналоги, які випереджатимуть розроблювану систему за можливостями. Середній допустимий час на розробку, очевидно, залежить від самого проекту, але можна виділити середні показники в період менше року як короткий термін, від року до двох — як середній, і занадто довгий термін як той, що перевищує два роки [2].

Очевидно, що чим менше розробка займає часу тим більше шансів на успіх, проте цей показник не впливає на швидкість, з якою можна буде окупити витрати. Будь-який серйозний проект потребує певних інвестицій, як грошових, так і інших. Якщо проект створено не на добровільних засадах, він має окупити себе, щоб його можна було вважати успішним, інакше зусилля розробників було витрачено дарма. І якщо передбачити, чи окупить себе проект, завчасно неможливо, оцінити очікуваний термін окупності проекту є обов’язковою задачею під час планування. Що довший термін окупності маємо для проекту, то ризикованіше братися за його розробку. Приклад: невелика команда займається створенням інтернет-магазину, капіталовкладення на рекламу мінімальні. Оскільки дана задача є досить шаблонною, її реалізація не займає багато часу, налаштовується готовий макет, завершується написання серверної частини. Після введення в експлуатацію проходить певний час, доки клієнти почнуть заходити на сайт і здійснювати покупки. Через відсутність реклами активність покупців є єдиним засобом приурочення нових клієнтів, що робить процес популяризації сайту повільним. Тим часом розробники потребують виплати заробітної плати, а також виплат потребує утримання серверу для сайту. Якщо активність покупців буде зростати довго, це призведе до провалу проекту і появи боргів.

Щоб упевнитись в прийнятності термінів окупності, треба звернути увагу на наступні моменти:

1) від чого залежить прибуток продукту (активність користувачів, придбання продукту на платній основі тощо);

2) чи орієнтується проект на тимчасову подію (програми, приурочені до періодичних або разових подій, обмежених у існуванні).

Що коротший термін, за який робота має принести винагороду, то більше шансів на успіх має проект.

1.2.8 Прибутковість

Як правило, серйозна розробка націлена на отримання прибутку. Передчасно не можна сказати, яку кількість людей вдасться залучити, утім, плануючи свою роботу, розробник завжди повинен мати уявлення про те, яка у нього цільова аудиторія. Якщо цільова аудиторія не є чітко визначеною, є ризик не досягти успіху, адже невідомо, для якої кількості людей проект взагалі буде прийнятний. Як було зазначено раніше, дуже часто розробник створює проект, спираючись на свої власні потреби, хибно очікуючи, що інші люди мають точно такі ж самі потреби й проблеми, що наприкінці обертається неприйняттям кінцевого результату роботи і втраченим часом [3]. Щоб уникнути цього, варто дотримуватись наступних правил:

1) потрібно створювати програмне забезпечення, спираючись на актуальні потреби, а не навпаки, створювати на сподіваннях, що люди раптово приймуть нову ідею, для втілення якої не було передумов;

2) для аналізу потреб інших потрібно проводити опитування різних категорій людей, щоб виявити серед них потенційних користувачів.

Стосовно опитування, вибірка опитуваних має бути максимально широкою та випадковою, інакше можлива ситуація, за якої буде отримана хибна оцінка щодо груп людей (та їхньої кількості), яким може підійти розробка.

Окремо варто виділити охоплення ринку; не завжди нас може влаштовувати та аудиторія, яку ми можемо отримати виходячи з географічного положення, соціальних обмежень, розподілу кількості людей за віком тощо. У такому випадку варто шукати шляхи розширення впливу на ринок для залучення нових людей. Робити це можна декількома шляхами:

1) передбачити мультинаціональність; це може включати

а) багатомовність програми/системи;

б) вкладання домовленостей для офіційного розповсюдження програмного забезпечення на території різних країн;

3) зробити рекламу проекту:

2) передбачити варіативність; це може включати

а) брендову варіативність (зміна логотипу продукту, слогану для країн, де поточний логотип або слоган може бути неприйнятним);

б) варіативність контенту (приклад: окрема версія сайту для країн із оберненим напрямком написання);

в) вікову варіативність (приклад: режим батьківського контролю у програмах та іграх з елементами, які можуть бути неприйнятними для дітей);

Охоплення ринку є важливим показником, хоча його вплив і залежить від масштабу проекту і початкових планів щодо його розповсюдження. Але є очевидним, що звуження аудиторії і можливостей розповсюджувати свій проект суттєво знижує його шанси на успіх, особливо якщо це серйозний проект від відомого бренду, бюджет яких є, як правило, відчутно великим, а отже і ризики стосовно розглянутого питання ігнорувати не можна.

1.2.9 Досвід

Швидкість виконання роботи безпосередньо залежить від організованості команди, вона у свою чергу залежить від досвіду керівників проекту у організації подібних проектів. На практиці неможливо передбачити усі можливі проблеми, які потенційно можуть виникнути під час розробки, завжди буде присутній фактор випадковості; якщо проблема є новою, для неї потрібно заново шукати рішення, що псує плани і сповільнює реалізацію. І хоча багаторазове вирішення однієї і тієї ж задачі не є стовідсотковою гарантією захисту від неї у подальшому, це все ж пришвидшує роботу, адже команда знає, де і як шукати рішення.

Не обов’язково досвід повинен бути особистим. Чужий досвід також може допомогти, якщо ми стикаємось із поширеною проблемою. Плануючи той чи інший проект, та обираючи технології для його реалізації, необхідно звертати увагу на те, наскільки популярною є дана технологія, адже що ширша спільнота розробників, що постійно мають із нею справу, то вища

ймовірність, що ваша проблема уже виникала неодноразово у інших людей і була подолана.

Досвід не обов’язково пов’язаний з технічними аспектами розробки. Організація є не менш важливим моментом. Вибір правильної методології управління проектом, підхід до спілкування з командою безпосередньо впливають на працездатність та продуктивність усіх її членів. Що більший досвід керівництво проекту має у організації робочого процесу, то вищою буде кооперація з членами команди, і то вищою буде ймовірність завершити проект вчасно.

Звичайно ж, наявність досвідчених спеціалістів також є ключовим моментом, за яким можна оцінити успішність усієї ідеї, адже якою б не була задумка, без уміння втілити її у життя нічого не вийде.

Обираючи технології для проекту, варто спиратись на те, якими навичками вже володіють члени команди (якщо така є). Іноді корисніше зробити вибір на користь більш застарілого інструменту, але перевіреного часом і такого, з яким розробники вже працювали. Не слід забувати, що новіші технології несуть із собою і нові проблеми, які потрібно вміти вирішувати, і дуже часто вигода від використання більш застарілого інструментарію для команди, яка має з ним досвід, більша, аніж від використання найновішого фреймворку, який потрібно вивчати, досліджувати заново, тестувати тощо.

У разі відсутності досвідчених розробників створити якісний продукт майже неможливо, а підтримка стає дуже складною. Досвідченість спеціалістів, звісно ж, не обмежується технічними знаннями. Сюди також відноситься

а) уміння аналізувати документацію;

б) уміння шукати потрібну інформацію в Інтернеті;

в) уміння тестувати свій продукт;

г) уміння налагоджувати код (debugging);

ґ) уміння писати якісні інструкції та документацію;

д) уміння комунікувати з іншими членами команди та доносити свою думку.

Без наявності команди людей з цими якостями починати будь-який проект ризиковано.

1.2.10 Фінанси

Особливо варто відмітити фінансовий показник. Будь-яка робота повинна мати фінансове винагородження, це стосується як пропрієтарних продуктів, так і некомерційних. Плануючи проект, потрібно обов'язково звертати увагу на цей пункт. Об'єм роботи, її складність та швидкість виконання напряму пов'язані з ціною. Якісно виконану роботу, як правило, роблять повільно, а нашвидкуруч зроблений проект якісним навряд чи буде. Утім, за наявності досвідчених спеціалістів можна поєднати ці два пункти, але грошовий еквівалент винагороди за створення даного продукту буде значно вищим.

Якщо коштів на реалізацію проекту недостатньо, існують шляхи залучення капіталу через інвестиції та пожертвування, але ви маєте

гарантувати, що проект буде дієздатним, інакше це може обернутися значними витратами для інвесторів і, у першу чергу, для виконавчої сторони.

Обов’язково має бути шлях окупити проект після введення в експлуатацію; керівник проекту зобов’язаний прорахувати усі ризики та можливості втратити заробіток. Якщо коштів на проект не вистачає і немає засобів залучити інвесторів, братися за проект немає сенсу, якщо тільки ви не плануєте його цілком на добровільних засадах, — у такому разі під час аналізу слід для себе вважати, що коштів умовно достатньо, щоб алгоритм врахував це і не занижував оцінку успішності проекту. У всіх інших випадках нестаток фінансування є підставою вважати, що проект не окупиться.

Інше питання — ціна, встановлена на проект. Залежно від того, наскільки серйозним є проект, користувачам може бути запропоновано сплатити певні кошти за право користуватися продуктом. Ціна може варіюватися залежно від:

1) масштабу проекту;

2) брендовості;

3) плану (професійний, стандартний тощо);

4) специфічності версії (різні версії програми для різних потреб);

5) користувача (безкоштовно для студентів та працівників вузів);

6) підтримки (довгострокова, повна версія без підтримки тощо);

Не обов’язково проект має бути платним, ліцензія, за якою поширюється забезпечення, може передбачати безплатне використання, але наявність платних компонентів, або ж бути повністю відкритою і

безоплатною, але це можливо лише за умов, що розробка добровільна. У такому разі для аналізу слід вважати будь-який можливий прибуток прийнятним.

Ціна на продукт має бути конкурентною. Якщо існує аналог з еквівалентними можливостями і нижчою ціною, клієнти будуть схильні перейти на нього. Оскільки ціна залежить також від затрат на розробку, існує мінімальна вартість, яку можна встановити на продукт. Відштовхуючись від неї, можна оцінити, чи буде така ціна взагалі прийнятною з огляду на ціни конкурентів.

1.2.11 Інші засоби реалізації

Не одні фінанси є необхідними для успішної роботи. Як правило, проект має стосунок до певної предметної області. Якщо це є область науки або медицини, наявність кваліфікованих спеціалістів необхідна, утім, цього часто буває недостатньо. Хоч програміст і має справу з нематеріальними речами, чим є програмне забезпечення, часто саме ПО пов’язане зі специфічною технікою, яку необхідно придбати для роботи. Так, створюючи програмне забезпечення для електронних термостатів, потрібно мати екземпляр термостатів для можливості тестування, а створюючи адаптивну веб-сторінку, необхідно мати різні пристрої і дисплеї з різними пропорціями і діагоналлю екрану для перевірки коректного адаптування.

Оскільки розробники є живими людьми, є необхідність забезпечити їх усім необхідним для виконання своєї роботи, це можуть бути як матеріальні речі (приміщеннями для роботи, відпочинку тощо), так і нематеріальні (підтримка, виділення лікарняних/вихідних/відпусток тощо).

Підсумовуючи, маємо наступні негрошові засоби, у яких може виникнути необхідність:

1) цільові пристрої, для яких призначена платформа, над якою ведеться розробка;

2) приміщення для роботи (офіс);

3) стороннє ПО для організації робочого процесу (системи менеджменту, контролю версій тощо);

4) нематеріальна підтримка робітників різноманітного характеру.

Плануючи проект, слід звернути увагу на наявність та доступність усього необхідного, адже це безпосередньо впливає на продуктивність і можливість реалізувати проект.

1.2.12 Ризики

Хоч плануючи проект, ми розраховуємо на успіх, неможливо гарантувати його на сто відсотків. Провал проекту обов’язково стане причиною втрат, і плануючи проект, ми маємо обрахувати потенційні втрати у випадку, якщо проект виявиться нежиттєздатним. Не обов’язково недосягнення всіх елементів плану щодо проекту є фатальним, але будь-яке порушення плану призводить до того, що ми вже отримуємо не той продукт, який хотіли з самого початку, а значить, сприйняття його потенційними користувачами вже буде відрізнятись, і можливо це завадить планам реалізувати певні очікування, що в свою чергу стане причиною падіння продаж, непокриття інвестицій і часткових або серйозних збитків.

Плануючи проект, ми завжди маємо усвідомлювати, чим обернеться порушення планів і провал проекту; у разі потенційних великих збитків є сенс переглянути проект і внести зміни до його плану, або взагалі відмовитись від його реалізації.

1.2.13 Зовнішні обставини

На робочий процес впливають не тільки навички розробників і їхніх керівників, завжди буде присутній фактор впливу обставин ззовні. Залежно від тематики проекту цей вплив може бути суттєвим, а іноді ним можна знехтувати. Задача керівників проекту при плануванні звести сторонній вплив до мінімуму, поставивши результат роботи в максимальну залежність від навичок розробників.

Вплив ззовні може проявлятись як:

1) деактуалізація проекту (будь-який момент можуть відбутися події, що зменшать або зведуть до нуля значимість усієї роботи);

2) зміна умов фінансування (стекхолдер може відмовитись підтримувати проект за власним бажанням або через недотримання певних умов);

3) людський фактор.

Людський фактор є найвпливовішим порушником формалізації будь-яких планів, адже неможливо передбачити, що може статися з робітниками і як це вплине на робочий процес: людина може захворіти, звільнитися тощо, а це порушить початкові умови, відносно яких і відбувався аналіз.

Плануючи проект, керівництво має (наскільки це можливо) передбачити вплив зовнішніх обставин на результат роботи. Варто зазначити, що будь-який алгоритм за визначенням може проводити лише аналіз детермінованих подій, тобто тих, які можна передбачити, вплив інших буде гарантовано псувати оцінку.

1.3 Проблема точності оцінки

Оскільки неможливо повністю нівелювати вплив зовнішніх факторів на проект, неможливо і дати стовідсоткову оцінку успішності проекту. Необхідно формалізувати систему оцінювання так, щоб для одних і тих же вхідних даних завжди була одна і та ж оцінка, тобто мова йтиме про функцію оцінки (далі функція *M*).

Визначимо, відносно чого будемо робити оцінку. Для цього введімо поняття «ідеальних умов» виконання проекту. Уявімо, що ми живемо в імпровізованому ідеальному світі, де всі події, пов’язані з виконанням проекту, є детермінованими. Такі умови ми і будемо називати «ідеальними». Оцінка проекту буде надана так, ніби умови, за яких триває робочий процес, є ідеальними. Це гарантує нам детермінованість результатів, тобто для одних і тих же вхідних даних функція оцінки буде давати один і той же результат, що дає нам право визначити функцію *M* як відображення із множини вхідних даних у множину оцінок:

$$M : I \to V$$

де I — множина вхідних даних, а *V* — множина усіх допустимих оцінок.

Оцінка буде дана на основі аналізу відповідей на ряд питань. Відповіді будуть мати закриту форму, тобто на кожне контрольне питання користувач буде обирати один з завчасно заданих варіантів відповідей. Вектор обраних варіантів $i = (i_1,...,i_n)$ формуватиме вхідні дані алгоритму ($i \in I$, $n \in \mathbb{N}$). Питання будуть стосуватись критеріїв оцінки, які ми розглядали у попередніх пунктах.

Тепер залишається відкритим лише питання обов'язковості всіх відповідей. Очевидно, що дати відповідь на всі питання може бути складно або неможливо. Форсувати користувача це робити немає сенсу з очевидних причин: не маючи можливість не надати дані, користувач намагатиметься дати приблизну відповідь, суб'єктивно оцінюючи, якою вона має бути, виходячи з тих даних, що ж йому відомі, або ж узагалі дасть відповідь навмання. Замість цього дамо можливість залишити питання без відповіді. Таким чином, будь-яка відповідь i_x із вектору вхідних даних може містити порожню відповідь, яку ми будемо далі називати невизначеністю. Очевидно також, що зі зростанням кількості невизначеностей оцінка, дана алгоритмом, буде неточною, адже з появою однієї невизначеності ми примусово позбудемось одного параметру, за яким ми цю оцінку робимо.

Оскільки ми вирішили не форсувати користувача дати відповіді на всі питання, немає шляху уникнути падіння точності оцінки при зростанні невизначеностей, утім, ми маємо знайти спосіб повідомити користувачу про зниження точності. Для цього введемо поняття коефіцієнту довіри $k \in [0, 1]$. Даний параметр прямо пов'язаний із точністю оцінки, що більше значення має довірчий коефіцієнт, то ближча отримана завдяки алгоритму оцінка до тієї, що гарантовано передбачає майбутній рівень успішності проекту так,

якби він протікав у ідеальних умовах (визначення яким ми дали у попередньому пункті).

У якості відповіді користувач буде бачити два значення: оцінку потенційної успішності, виражену у відсотках від абсолютно успішного проекту, який мав би місце в ідеальному світі, де вплив недетермінованих обставин був би відсутнім (або ним можна було б знехтувати), та оцінку точності отриманого результату. Для звичайної людини найбільш інтуїтивно зрозумілим виводом буде відображення результату у відсотках, адже така форма запису зустрічається дуже часто в різних тестах та підрахунках.

Оскільки сенс оцінки може не бути до кінця зрозумілий за одним лише своїм значенням, оцінку необхідно супроводжувати додатковою інформацією, яка буде пояснювати отриманий результат.

1.4 Висновки до частини 1

Автором було досліджено можливості дати оцінку можливості реалізувати проект. Було розглянуто формальні підходи до визначення поняття проекту та оцінки. Також було визначено спосіб, яким буде зручно отримувати та обробляти інформацію по проекту, надану від користувачів системи.

Наступним кроком було визначено критерії оцінки. Було детально розглянуто причини, чому був обраний той чи інший критерій, обмірковано силу їх впливу на проект тощо. Для тесту було обрано наступні параметри:

а) повнота інформації по проекту;

б) строки створення проекту;

в) строк окупності вкладень;

г) можливість використання наробіток у подальшому;

ґ) можливість розширення;

д) можливість оновлення;

є) конкурентоспроможність;

ж) зрозумілість для широкої публіки;

з) новизна;

і) досвід;

к) інші другорядні параметри.

У кінці було досліджено проблеми при оцінюванні, та запропоновано шляхи їх усунення. Було визначено функцію оцінки та запропоновано шляхи вказання точності отриманих результатів. Запропоновано найоптимальніший з точки зору автора шлях надання результатів користувачу.

2. Реалізація

2.1 Розробка веб-ресурсу

2.1.1 Основний інструмент

Сервіс являтиме собою вебсайт, нам необхідно підібрати стек веб-технологій, які ми будемо використовувати. Автором було обрано бібліотеку React (рисунок 2.1.1.1) — сучасний гнучкий інструмент для конструювання односторінкових веб-застосунків.

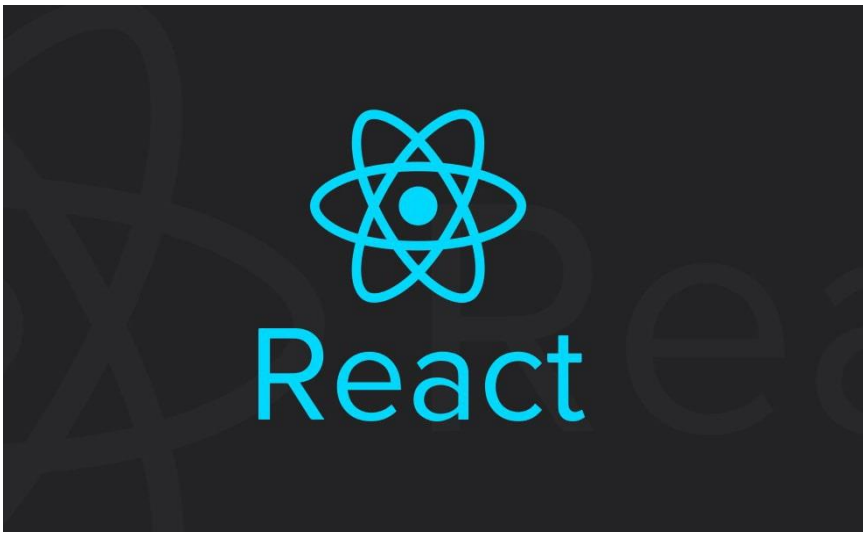

Рисунок 2.1.1.1 — Логотип React

React було розроблено компанією Facebook у 2013 році як альтернатива Angular, що розроблений Google. Метою було створити інструмент, який буде мати інтуїтивну конструкцію та синтаксис, відносно низький поріг входу, можливість легко розділяти логіку від представлення та розширювати власний код за допомогою сторонніх компонентів [4].

Весь код проекту на React ділиться на окремі частини — компоненти. У такий спосіб реалізовано компоненто-орієнтовану парадигму

програмування. Компоненти можна вкладати один в інший, тим самим формуючи ієрархію компонентів (структура яких показана на рисунку 2.1.1.2), задавати для них різну поведінку, стилі, анімацію, надавати їм аргументи тощо. Кожен компонент може бути чисто логічним (не мати жодного візуального представлення), або бути репрезентацією деякого об’єкту у об’єктній моделі документу сторінки (DOM). Для останніх можна задавати стилі, налаштовуючи їх зовнішній вигляд і відображення під вільний простір, розмір екрану, пристрій тощо.

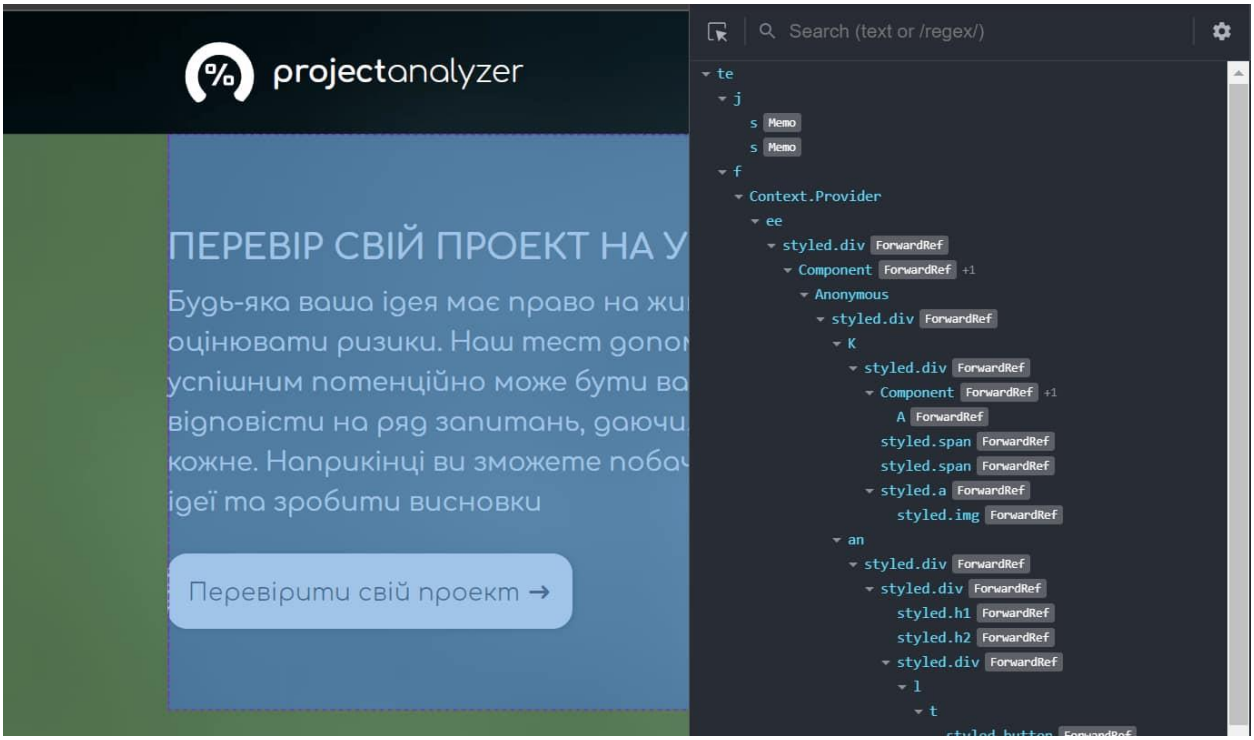

Рисунок 2.1.1.2 — Ієрархія компонентів

Застосунок, створений за допомогою React є односторінковим (SPA, single-page application): при завантаженні сторінки клієнту надходить порожній HTML-документ із підключеними файлами JavaScript, після чого браузер звертається до свого JavaScript-рушія, який запускає отримані скрипти (файли з кодом). Код скриптів містить повний цикл відображення

всіх елементів на сторінці. Усі елементи — кнопки, блоки, таблиці, списки, — усі створюються, завантажуються в пам’ять та відображаються динамічно за допомогою коду. Перевагами даного підходу є малий розмір файлу сайту і, відповідно, економне використання трафіку. По-друге, оскільки сторінка повністю генерується клієнтом, код сторінки не мусить містити жодну серверну логіку — у випадку наявності бекенду (серверної частини) код розділяється на два окремих проекти для серверу та для клієнту, над якими можуть працювати окремі команди людей незалежно одна від одної.

React будує сторінку, створюючи її віртуальне дерево. Інформація про структуру сторінки завантажується в оперативну пам’ять і у разі оновлення синхронізується з реально структурою. Такий підхід називається віртуальним DOM. Остання на момент написання роботи версія React надає також можливість оновлювати елементи сторінки інкрементально (частина за частиною, а на все разом). Це дає можливість не чекати повного оновлення компоненту, а одразу відобразити те, що можна, доки не буде підвантажено всі зміни.

Будь-який застосунок містить у собі дані, які постійно змінюються. Від цих даних залежить стан окремих або усіх компонентів. Для доступу та управління станом усіх даних створено бібліотеку Redux (рисунок 2.1.1.3) та розширення react-redux, яке дозволяє синхронізувати систему контролю за станом із компонентами.

Рисунок 2.1.1.3 — Redux

У такий спосіб встановлюється залежність між елементами сторінки та даними. Зміна даних викликає оновлення елементів, і навпаки. Окремі елементи можуть змінювати стан системи у відповідь на дії користувача або самостійно з часом [5].

2.1.2 Розміщення сервісу

Оскільки веб-ресурс має бути доступним усім охочим в Інтернеті, необхідно подбати про його розміщення. Автором було обрано сервіс GitHub Pages, який дозволяє достатньо легко на безплатній основі завантажувати сторінки без серверної частини.

При розміщенні сайту на GitHub Pages людям також буде доступний код сторінки, це дозволить розвивати проект іншим охочим, якщо у когось виникне така зацікавленість у подальшому.

При розміщенні сторінки її вихідні коди зберігаються в одному репозиторії, а код, готовий для експлуатації — в іншому. У такий спосіб розділяється представлення і робоче місце, що дозволяє вносити зміни до проекту, оновлюючи сам сайт лише за необхідності (наприклад, коли будуть виконані всі задачі, які було заплановано для поточного оновлення сайту).

2.2 База проекту

Проект створено за допомогою бібліотеки create-react-app. Це рішення від Facebook, завдяки якому можна згенерувати сконфігурований проект. Для контролю типів використовується Typescript версії 4.1.2.

Усі компоненти проекту розділено на дві категорії. Перша категорія являє собою елементи сторінки, окремі великі блоки, які з'єднуються в головну сторінку. До другої категорії відносяться самостійні компоненти, які можна використовувати багато разів у різних місцях (reusable).

Елементи сторінок інкапсульовано у функцію App модуля App (рисунок 2.2.1).

```
import { Provider } from 'react-redux';

import { store as appStore } from 'store';

import GlobalStyle from 'components/shared/GlobalStyle';

import MainPage from 'components/pages/Main';

export default () => (
  <>
    <GlobalStyle />
    <Provider store={appStore}>
      <MainPage />
    </Provider>
  </>
);
```

Рисунок 2.2.1 — Головна функція

Функція завантажує компонент головної сторінки та декілька допоміжних компонентів:

GlobalStyle — компонент глобальних стилів, використовується для скасування стилів браузера, встановлених за промовченням.

Provider — компонент, що відповідає за делегування викликів функцій, що відповідають за зміну стану застосунку; інкапсулює об’єкт store, який є Redux-сховищем.

2.3 Стилізація

Для встановлення стилів компонент використано бібліотеку styled-components. На момент написання роботи ця бібліотека є відносно новою розробкою, яка, утім, зарекомендувала себе як одну з найбільш перспективних для своїх задач.

Ідея styled-components полягає в розділенні компонент, що відповідають за логіку, від суто стильових. Кожна директорія проекту, що містить компонент, також (за необхідності) містить файл styled.tsx, який зберігає всі стилі, зібрані у окремі обгортки. Приклад обгортки дано на рисунку 2.3.1.

```
export const Wrapper = styled.div`
  height: ${cssVars.heightNavbar};
  background-color: rgba(0, 0, 0, .1);

  display: flex;
  align-items: center;
  box-sizing: border-box;
  padding-left: 10vw;
`;
```

Рисунок 2.3.1 — Стильовий компонент

2.4 Стан

Як було зазначено, для контролю над станом використовується Redux-сховище. Redux дозволяє створити об’єкти, які міститимуть необхідні дані для всього застосунку, і змінювати компоненти при зміні цих даних. Структуру модулів, що відповідають за контроль над даними, зображено на рисунку 2.4.1.

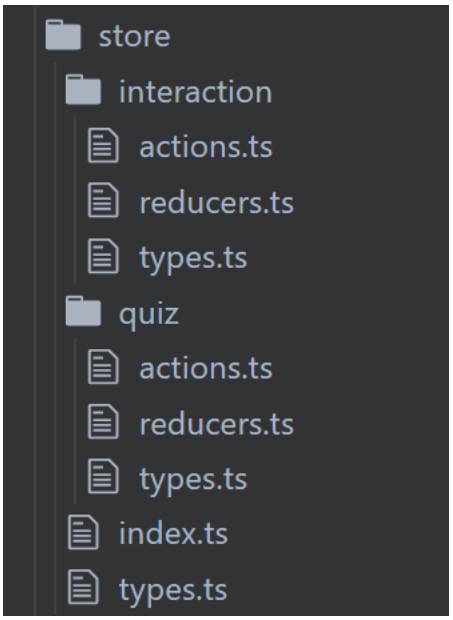

Рисунок 2.4.1 — Структура сховища

Пакет interaction відповідає за дані, пов’язані із взаємодією між користувачем та сайтом. Модулі пакету:

types — типи даних Typescript, необхідні для контролю помилок;

actions — об’єкти, що є репрезентаціями певних подій, що виникають в системі (користувач почав опитування, була натиснута певна кнопка тощо);

reducers — функції, що змінюють стан системи у відповідь на надходження до них у якості аргументу об’єкту з модуля actions.

Пакет quiz відповідає за дані, що стосуються безпосередньо опитування. Структура та призначення модулів аналогічні пакету interaction.

2.5 Основний алгоритм

Опитування та оцінка проекту здійснюється модулями QuizSection та ResultsSection відповідно. Компонент QuizSection зберігає дані про відповіді користувача. Кожен варіант відповіді має певну вагу. Коли усю інформацію буде зібрано, модуль ResultsSection обраховує сумарну вагу та знаходить відсоток успішності відносно максимально можливого значення. Після чого береться кількість ненаданих відповідей і обраховується коефіцієнт довіри. Що більше ненаданих відповідей, то нижчим буде цей коефіцієнт (враховується загальна кількість питань, та кількість питань, на які не було надано відповідь). Після обрахунків результат буде показано користувачеві у зручному вигляді. До відповіді також надаються текстові пояснення, які є спільними для конкретних діапазонів кожного з двох отриманих значень.

2.6 Висновки до частини 2

Було розглянуто реалізацію веб-сервісу із використанням декількох сучасних бібліотек, таких як React, Redux, styled-components та інших. Надано пояснення щодо структури проекту загалом та окремих його частин. Отриманий проект запущено в мережу Інтернет. Проведено тести, результати яких показують справну роботу сторінки.

3. Практичне використання

3.1 Інтерфейс та взаємодія

Після завантаження сайту користувача зустрічає головна сторінка (рисунки 3.1.1-3.1.2).

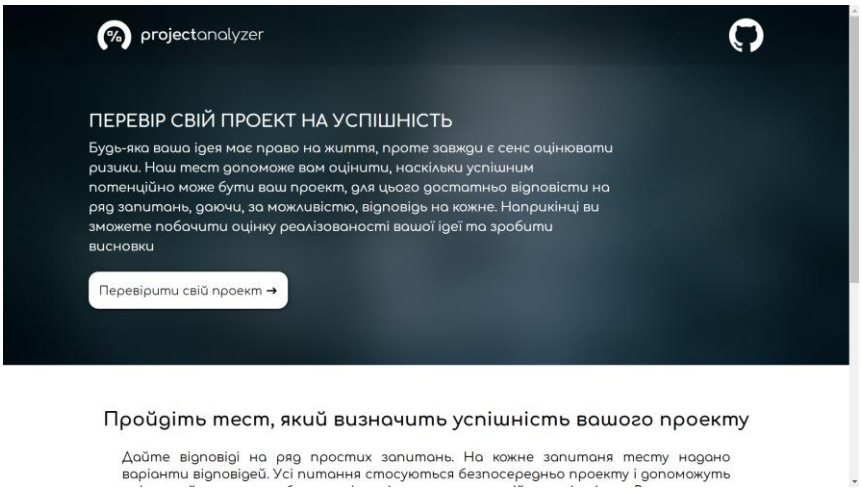

Рисунок 3.1.1 — Головна сторінка

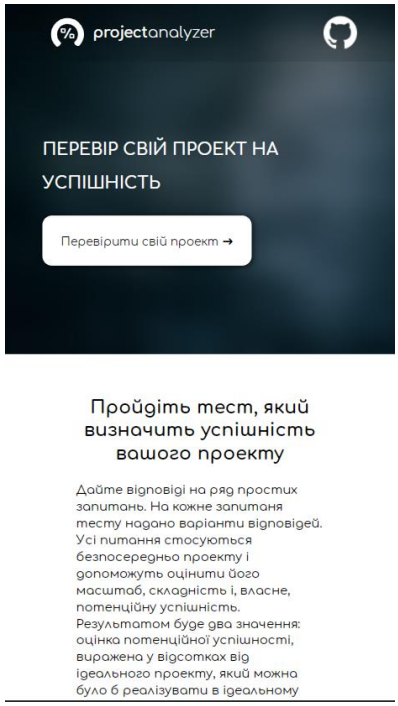

Рисунок 3.1.2 — Мобільна версія головної сторінки

Користувач має можливість переглянути репозиторій із вихідним кодом, перейшовши за посиланням на GitHub.

Основна кнопка направляє користувача на блок із коротким описом системи, після чого користувачу буде запропоновано пройти тест (рисунок 3.1.3).

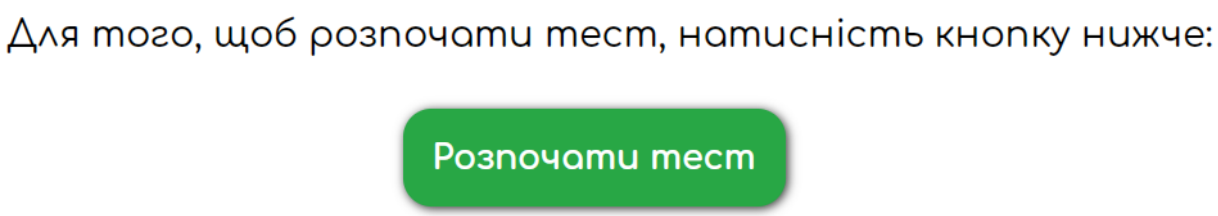

Рисунок 3.1.3 — “Call to action”

Після її натискання буде запущено тест. Користувачу буде дано ряд запитань для встановлення інформації (рисунок 3.1.4), необхідної для аналізу. Необхідні питання було розглянуто в попередньому розділі.

Рисунок 3.1.4 — Тест

Після проходження тесту, система зробить підрахунки та виведе на екран свою оцінку (рисунок 3.1.5).

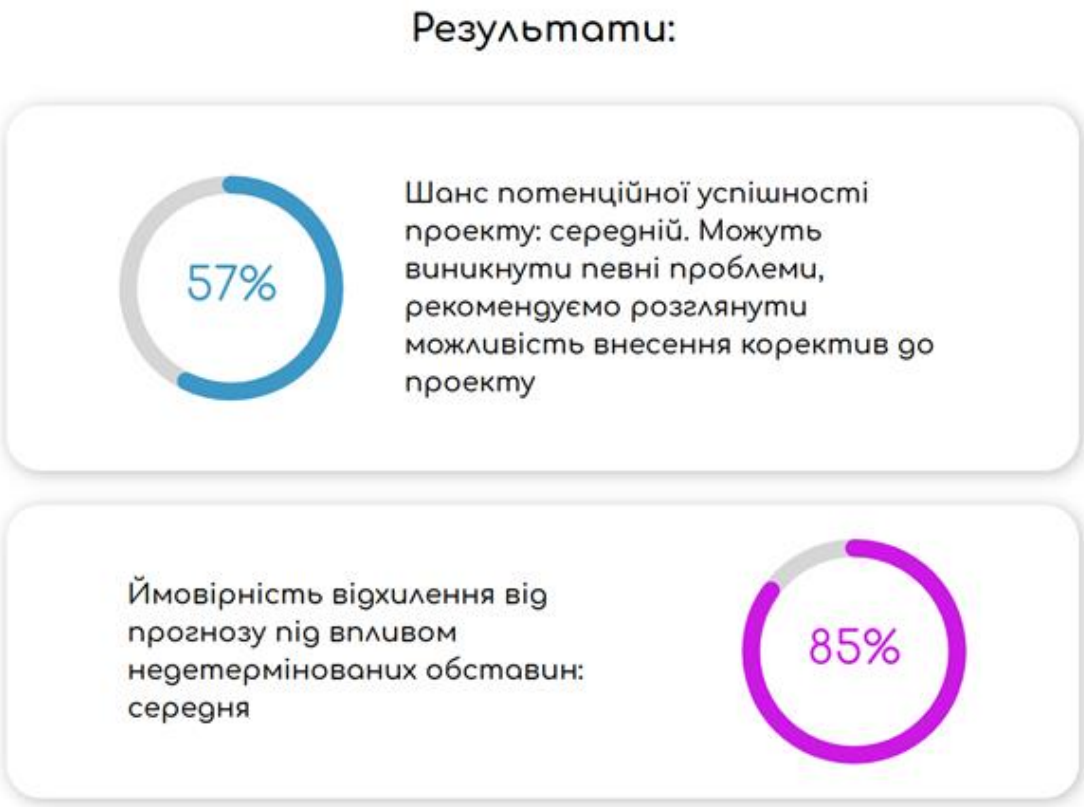

Рисунок 3.1.5 — Результати

Результат подається у вигляді двох значень (їх форма виводу та обґрунтування було розглянуто у попередньому розділі).

Першим значенням йде оцінка потенційної успішності: що вище це значення, то вищим є оцінювана успішність проекту за умови високої точності оцінки.

За точність, власне, відповідає довірчий коефіцієнт — друге число, яке залежить від кількості наданих (не пропущених) відповідей. Тобто що більше питань не було пропущено і що більше інформації було надано системі для аналізу, то, відповідно, вищим є довірчий коефіцієнт, і вища релевантність отриманої оцінки відносно тієї, яку б хотів отримати користувач в ідеальних умовах.

Веб-сервіс побудовано за принципом Mobile-First, тобто відображається лише необхідне, при цьому вся інформація, що бачить користувач на сторінці, повністю співпадає і у настільний версії і в мобільній та версії для планшетів (рисунок 3.1.6).

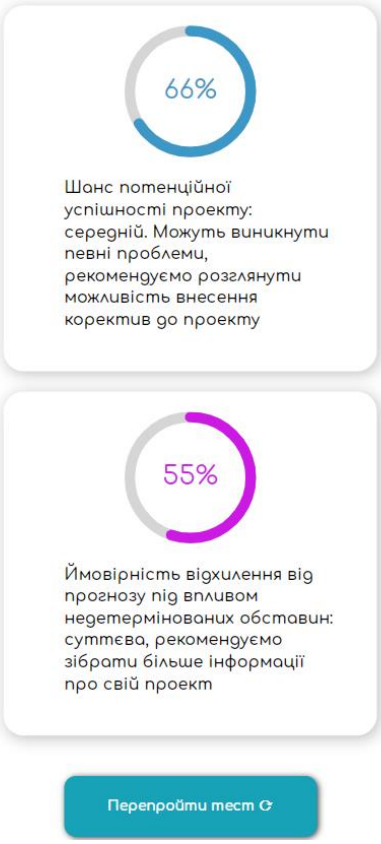

Рисунок 3.1.6 — Вивід результатів для мобільної версії

3.2 Висновки до частини 3

Було розглянуто інтерфейс сервісу та його використання. Застосунок побудовано з огляду на можливість його використання різними пристроями на різних платформах. Заплановані можливості було реалізовано, тестування не виявило технічних недоліків. Сервіс справно працює, дає користувачам можливість пройти тест, надавши інформацію щодо свого проекту, та дає оцінку у зручному та зрозумілому вигляді, супроводжуючи результат поясненнями та рекомендаціями.

Висновки

У теоретичній частині роботи було розглянуто критерії оцінювання для автоматизації аналізу реалізованості проекту та описано їх значимість. Також розглянуто основні проблеми, пов’язані з алгоритмом аналізу. Запропоновано рішення.

Практична частина описує архітектуру системи, використані бібліотеки та інструменти. Наведено приклади коду та надано опис основних модулів проекту.

Розроблене рішення може бути корисним для планування свого проекту. Завдяки створеному веб-ресурсу можна проаналізувати для себе потенційні проблеми проекту та визначити можливість його реалізації. У подальшому планується розширення системи завдяки ускладненню алгоритмів, використанню нейронних мереж, аналітики статистичних даних тощо.

Список використаних джерел

1. Project management. How much is enough? [Електронний ресурс] — Режим доступу до ресурсу: https://www.pmi.org/learning/library/project-management-much-enough-appropriate-5072

2. General Terms of the Project Implementation Agreement — Revised Version [Електронний ресурс] — Режим доступу до ресурсу: https://www.kkl-jnf.org/files/about_kkl/Tenders-RFI/Projects/General-Terms_Calls-for-Proposals.pdf

3. Методика оцінювання проектів [Електронний ресурс] — Режим доступу до ресурсу: https://ucf.in.ua/storage/docs/10122018/Методика%20оцінювання%20проектів%20-%202019.pdf

4. A JavaScript library for building user interfaces [Електронний ресурс] — Режим доступу до ресурсу: https://reactjs.org/

5. A Predictable State Container for JS Apps [Електронний ресурс] — Режим доступу до ресурсу: https://redux.js.org/