ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ

ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до магістерської роботи

на ступінь вищої освіти магістр

на тему: «Розробка методики прототипування об'єктів інформаційної системи на базі технології JavaScript, Node.js»

Виконав: студент 7 курсу, групи ППЗМ–71
спеціальність
121 інженерія програмного забезпечення
(прізвище та ініціали)
Печериця В.В.
(прізвище та ініціали)
Керівник Бондарчук А.П.
(прізвище та ініціали)
Рецензент ____________________
(прізвище та ініціали)
Нормоконтроль ______________
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій
Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти -«Магістр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення

Негоденко О.В.

“____” ___________2022 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

ПЕЧЕРИЦІ ВІТАЛІЮ ВАЛИНТИНОВИЧУ

(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка методики прототипування об'єктів інформаційної системи на базі технології JavaScript, Node.js

Керівник роботи: Бондарчук Андрій петрович, д.т.н., професор кафедри ІПЗ
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «__»__2022 року №__.

2. Строк подання студентом роботи____

3. Вхідні дані до роботи

Офіційна документація Node.js;

Офіційна документація JavaScript

Науково-технічна література з питань, пов’язаних з базами даних;

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно

розробити)

4.1. Опис та аналіз загальних характеристик відомих методик прототипування

4.2. Опис опис та структура об'єктів прототипування

4.3. Порівняльний аналіз функціональних можливостей існуючих інструментів прототипування

5. Перелік демонстраційного матеріалу (назва основних слайдів)

6. Дата видачі завдання___________________________________

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів робот	Примітка
1	Підбір науково-технічної літератури	26.09-21.10	
2	Опис та аналіз загальної характеристики систем прототипування	26.10-12.11	
3	Опис структур та систем розробки прототипів	13.11-11.12	
4	Аналіз процесів роботи створеного інструменту.	18.12-27.01	
5	Вступ, висновки, реферат	02.02-16.02	
6	Розробка презентації	18.02-01.03	

Студент __________________________________

Керівник роботи __________________________

РЕФЕРАТ

Прототипування дозволяє вирішити питання постановки основних задач безпосередньо перед запуском: перевірку гіпотез, отримання певного досвіду і зменшення ризиків. На етапі прототипування не значними зусиллями для реалізації базового набору функціональностей. В процесі створення прототипу можливо формулювати більш детальну картину складових частин системи. За думкою деяких розробників, даний процес є одним із найважливіших, за яким слідують настпуні етапи розробки програмного забезпечення.

Мета – покращення одного із етапу розробки інформаціних систем, а саме прототипування на базі розробленої методики прототипування.

Об`єкт дослідження – розробка методики прототипування об'єктів інформаційних систем.

Предмет дослідження – є методики прототипування інформаційних систем.

Було проаналізовано всі основні методики, які використовують в розробці програмного забезпечення різного призначення та розміру. По кожному підходу було проведено аналіз та опис та надана інформація, що до інструментів котрі імплементують різні підходи розробки прототипу. З дослідження відомих інструментів було зроблено висновки та детальне дослідження в їх використанні.

Було розроблено окрему систему, яка створена на основі нової методики прототипування. Вона містить в собі інструменти візуалізації потоку даних, архітектури основних компонентів ПЗ яка формує ядро. Також такий інструмент дає можливість використати стоворені прототипи, як готовий шаблон на фасаді якого можна та навіть потрібно створювати об’єкти системи.

Ключові слова: *Прототипування, інформаційна система, прототип, дані, методики.*

ЗМІСТ

ВСТУП……………………………………………………………………8

1. ОГЛЯД ІСНУЮЧИХ МЕТОДИК ПРОТОТИПУВАННЯ ТА ЇХ ХАРАКТЕРИСТИКИ……………………………………………………10

1.1. Основи прототипування……………………………………………...11

1.2. Огляд інструментів прототипування…………………………………...13

1.3. Прототипування баз даних……………………………………………...26

2. ІНСТРУКМЕНТИ ДЛЯ ІМПЛЕМЕНТАЦІЇ МЕТОДИКИ ПРОТОТИПУВАННЯ…………………………………………………….. 32

2.1. Аналіз інструментів для свторення системи прототипування…………32

2.2. Правила використання архітектури MVC……………………………… 44

2.3. Основні характеристики REST API……………………………………44

2.4. Структура взаємодії REST між клієнтом і сервером…………………… 48

3. СТВОРЕННЯ ІНСТРУМЕНТУНА ОСНОВІ МЕТОДИКИ ПРОТОТИПУВАННЯ…………………………………………………….. 50

3.1. Загальна архітектура прототипу……………………………………… 50

3.2. Структура серверу……………………………………………………53

4. ЗАСТОСУВАННЯ СТОВРЕНОЇ МЕТОДИКИ ПРОТОТИПУВАННЯ В ПРОЦЕСІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ…………….. 70

4.1. Аналіз застосування створеної методики прототипування……………..70

ВИСНОВКИ…………………………………………………………………74

ПЕРЕЛІК ПОСИЛАНЬ……………………………………………………75

ВСТУП

Значна частина в сучаних успішних систем будь-якого призначення (додаток, інструмент програмування, гра) свторені на основі прототипу. З підвищенням потреб в нових програмних систем зростає потреба в збільшені швидкості та не зменшенні якості розробки тому деякі інструменти прототипування досить застаріли.

Актуальність дослідження. Однією з найважливіших частин розробки програмного забезпечення є стовврення проекту. Прототип проекту як процес створення проекту можна розділити на дві великі частини умовно: прототипування функціональності та візуальний дизайн інтерфейсу користувача. Для створення прототипу функціональності застосовуються такі інструменти, як iRise і ProtoShare, які вже стали одними із основними для розробки прототипів програмного забезпечення. У розробці графічного інтерфейсу (як елемент інформаційної системи) користувача немає встановлених стандартів, є окремі рекомендації, прийоми, особливості дизайну, традиції, умови роботи програмного забезпечення тощо. При цьому важливою, але не завжди належним чином виконаною, частиною цього процес - це прототипування, тобто створення прототипу або прототипу майбутньої системи. Прототипи можуть бути різними: паперові, презентаційні, імітаційні тощо, аж до точної відповідності майбутній програмі. Більшість сучасних інтегрованих середовищ розробки програмного забезпечення (IDE) дозволяють створювати щось подібне до прототипів, але це пов'язано зі спеціальними знаннями IDE та мови програмування. У той же час розробка інтерфейсу користувача великого програмного проекту, як правило, є завданням окремої людини, яка не обов'язково бере участь у програмуванні. Тому корисно мати інструмент для прототипування інтерфейсу користувача, адаптований для швидкого створення досить складних прототипів. В якості таких інструментів використовувалися різноманітні програмні пакети: MS Visio, Corel Draw, Adobe Photoshop.

Мета – покращення одного із єтапу розробки інформаціних систем, а саме прототипування на базі розробленої методики прототипування.

Об'єкт дослідження – процес аналізу та вивчення даних.

Предмет дослідження – програмне забезпечення яке імплементує розробленої методики прототипування.

Завдання дослідження полягають в наступному

- розглянути основні методки прототипування
- дослідити існуючі інструменти прототипування, які реалізовані на власних підходах.
- проаналізувати потреби для реалізації створення інструменту для створення прототипу.
- застосувати створений інструмент прототипування в процесі прозробки програмного забезпечення в команді.
- аналіз висновків розробки інформаційних систем.

Методи дослідження – Теоретичного дослідження: аналіз, та узагальнення, методи класифікації. А також було застосовано емпіричні методи: експеримент, та опис.

Наукова новизна полягає в розробці нового способу прототипування інформаційних систем, які будуються на основі користувацьких дій конструювання схем процесів потоку інформації. Даний спосіб, на відміну від вже розробнеих , дозволяє прискорити проце розробки прототпу та подальшого його використання та нарощування функціональності.

Проведено аналіз та дослідження методики прототипування систем. Визначені вимоги та елементи до структур взаємодій та алгоритму побудови її елементів.

На базі отриманих результатів, які були отримані в процесі досліджень та реалізації оптимального алгоритму генерації прототипу. Дані результати дають змогу створити інформаційну систему, яка буду задовольняти процес передачі , обробки та отримання даних, які надають змогу проаналізувати вимоги до

структури та її елементів для подальшого впровадження та реалізації в подальше користування користувачами чи системами .

Наукова новизна отриманих результатів.

На основі теоретичних і експериментальних досліджень було проаналізовано проблему прототипування об'єктів інформаційних систем:

– приведено класифікацію компонетів компонентів серверних додатків для підвищення швидкодії розробки програмного забезпечення з великим обсягами логіки;
– запропоновано новий метод для створення проектів та його елементів на основі прототипу .

Новизна дослідження. Зроблено структурний аналіз інформаційних систем та дослідження наукової літератури. Проведено розробку системи та аналіз з її використанян в процесі розробки програмного забезпечення. Встановлено рекомендації з використання додатку та отриманої інформації.

Структура та обсяг роботи. Відповідно до мети і завдань дослідження структура роботи складається зі вступу, трьох розділів, висновків та списку використаної літератури. За час роботи опрацьовано 52 літературних джерела. Зміст роботи викладено на 70 сторінках машинописного тексту.

Джерелами інформації для вирішення перерахованих вище завдань є збірники наукових праць, монографії, періодична література, підручники та довідники, періодичні фахові журнали.

1. ОГЛЯД ІСНУЮЧИХ МЕТОДИК ПРОТОТИПУВАННЯ ТА ЇХ ХАРАКТЕРИСТИКИ

1.1 Основи прототипуавння

Прототипування (prototyping) - це один із найважливіших та в основному найвикористовуваніших сучасних інструментів для формування потреб та вимог. Прототипи будуються для відображення системи або чи переважаючої її основної частини для користувачів з метою формування потреб.

Прототип формує в собоі демонстраційну (базову) систему - "нашвидкуруч і грубо" створену робочу модель рішення, яка демонструє взаємодію з її елементами і моделює поведінку системи при взаємодії з нею при ініціалізаці з користувачем чи іншою системою різних подій.

Складність та зростаючі вимоги, які формуються зі зростанням розвитку сучасних систем, які створюють все більший попит на прототипування та є майже невід'ємним компонентом у створені ІС або її частин. Прототипи дають змогу досить оцінити можливість її реалізаці та користь системи ще до початку її створення.

У переважаючій більшості ситуаціях , прототип - це все більш ефективний метод виявлення вимог, які складно отримати від замовника за допомогою інших інструментів. Часто такі моменти можливо побачити на системах, яким необхідно надати у користування користувачам (чи іншим системам) нову функціональність. Також дана ситуація є характерна у тих випадках котрі суперечать з вимогами і наявностю проблем у зв'язку між користувачем і розробниками.

На даний момент є декілька основних різновидів прототипування.

- *"Одноразовий" прототип ("throw-away" prototype),* який після того, як виявлення вимог сформоване, просто відкидається. Розробка "одноразового"

прототипу націлена тільки на етап встановлення вимог ЖЦ ПЗ. Як правило, цей прототип концентрується на найменш зрозумілих вимогах;

- Еволюційні прототипи є окремим випадком ітераційних прототипів, які мають на меті еволюціонувати в остаточну систему. Такі методології, як екстремальне програмування, здебільшого полягають у розробці еволюційних прототипів. Оскільки прототипи рідко бувають надійними чи завершеними, часто недоцільно, а іноді й небезпечно розвинути їх у кінцеву систему. Дизайнери повинні ретельно продумати основну архітектуру програмного забезпечення прототипу, а розробники повинні використовувати добре задокументовані шаблони проектування для їх реалізації;

Ще одним аргументов на користь у використані такого типу, як "одноразового" може являтися прагнення уникання ризиків "консервації" швидких і не чітких а саме, тому в наслідок, неефективних сформованих рішень у кінцевій сформованій струткурі. Проте міць і гнучкість сучасних інтсрументів реалізації ІС роблять не чіткий даний виновок. Якщо управління проектом виконується належним чином, то обставини, по яким не можна було б уникнути неефективних установлених для прототипу рішень, не має.

1.2 Види прототипування

Програмне прототипування уособлює в собі велику кількість різновидів. Проте, всі методики певною мірою базуються на двох основних моделях прототипування: одноразове прототипування і еволюційне прототипування.

Одноразове прототипування воно ще має назву закрите прототипування. Відкидання або швидке створення прототипів має відношення до формування моделі, котра зрештою не буде прийнята, а не стане певною частиною кінцевого програмного забезпечення, що видається. Після того, як попередній збір вимог сформовано, формується проста робоча модель системи, щоб візуально показати

користувачам, який може виглядати їх вимоги в процесі запровадження у готову систему. Також воно має назву швидке прототипування.

Швидке прототипування уособлює в собі реалізацію робочої структури різних комопнентів системи на початковій стадії, після чого достатньо не двогого дослідження. Методика яка, використовується у його побудові, у зазвичайному ході являє собою достатньо неформальнім, одним з найважливіших причин є швидкість, за якою дається модель. Після чого модель стає початковою точкою, з якою користувачі мають змогу зробити висновки своїх очікувань та зробити висновки своїм вимогам. Коли дана мета досягнута, модель-прототипу «відкидається», і отримана система формально створюється за урахуванням сформованих потреб.

Однією з більш очевидною причиною впровадження одноразових прототипів це те, що дається змога виконати швидко. Коли користувачі мають змогу по найшвидшому шляху отримати та надати відгуки щодо своїх вимог, вони можуть деталізувати їх на ранніх ітераціях розробки інформаційної системи. Інтеграція змін на ранніх етапах життєвого циклу розробки є дуже продуктивним з точки зору затрат, оскільки в даному етапі немаєє нічого, що можливо було б спробувати ще раз. Коли в проект вводяться зміни після якого значна частина обсягу праці, є невеликі зміни котрі можуть потребувати значних зусиль для її реалізації, тому, що інформаційні системи зберігають в собі велику кількість залежностей. Один з найприорітетніших пунктів при створені одноразового прототипу є швидкість, оскільки за обмеженого бюджету часу та грошей мало що може бути витрачено на прототип, який буде відкинуто

Ще одна перевага одноразового прототипу - його здатність створювати інтерфейси, які можуть тестувати користувачі. Коли користувач має можливість по бачити взаємодію з системою з якою він має можливість взаємодіяти надає більшого розуміння, як система функціонуватиме в майбутньому

Стверджується, що швидке революційне створення прототипів - більш ефективний спосіб вирішення проблем, пов'язаних з вимогами користувачів, і, отже, більше підвищення продуктивності програмного забезпечення в цілому. Вимоги можуть бути ідентифіковані, змодельовані та протестовані набагато швидше та дешевше, коли ігноруються питання, пов'язані з розвитком, ремонтопридатністю та структурою програмного забезпечення. Це, у свою чергу, призводить до точної специфікації вимог та подальшої побудови дійсної та придатної до використання системи з погляду користувача за допомогою традиційних моделей розробки програмного забезпечення.

Прототипи можуть бути класифіковані відповідно до вірності, з якою вони нагадують фактичний продукт з погляду зовнішнього вигляду, взаємодії та часу. Одним із способів створення одноразового прототипу з низькою точністю є паперове прототипування. Прототип реалізований з використанням паперу та олівця, і таким чином імітує функцію реального продукту, але зовсім не схожий на нього. Інший спосіб легко створити високоточні одноразові прототипи – використовувати GUI Builder і створити пустушку, прототип, який виглядає як система цілей, але не надає жодних функціональних можливостей.

Використання розкадровок, анімації або малюнків не зовсім те саме, що й одноразове прототипування, але, безумовно, відноситься до того ж сімейства. Це нефункціональні реалізації, але вони показують, як виглядатиме система.

Еволюційні прототипи мають різну тривалість життя: швидкі прототипи створюються для певної мети, а потім викидаються, ітераційні прототипи розвиваються або для опрацювання деяких деталей (збільшення їхньої точності) або для дослідження різних альтернатив, а еволюційні прототипи розроблені, щоб стати частиною кінцевої системи. Швидкі прототипи особливо важливі на ранніх стадіях проектування. Вони повинні бути недорогими і легкими у виготовленні, оскільки мета полягає в тому, щоб швидко вивчити широкий спектр можливих

типів взаємодії, а потім викинути їх. Зауважте, що швидкі прототипи можуть бути офлайн або онлайн.

Створення точних прототипів програмного забезпечення, навіть якщо їх потрібно повторно реалізувати в остаточній версії системи, важливо для виявлення та усунення проблем взаємодії. У розділі 4 представлені конкретні методи створення прототипів, як офлайн, так і он-лайн. Ітераційні прототипи розробляються як відображення проекту, що виконується, з явною метою розвитку через кілька ітерацій проектування. Створення прототипів, які підтримують еволюцію, іноді буває складним.

Існує напруженість між розробкою та інструментами Beaudouin Prototype та інструментами до остаточного рішення та дослідженням несподіваного напрямку дизайну, який можна прийняти або повністю викинути. Кожна ітерація повинна інформувати про деякі аспекти дизайну. Деякі ітерації досліджують різні варіації однієї і тієї ж теми. Інші можуть систематично підвищувати точність, опрацьовуючи дрібніші деталі взаємодії

Горизонтальні прототипи, метою даного прототипу є розробка всього рівня дизайну на водночас. Цей тип прототипування найчастіше зустрічається у великих командах розробників програмного забезпечення, де дизайнери з різним набором навичок звертаються до різних рівнів архітектури програмного забезпечення. Горизонтальні прототипи користувальницького інтерфейсу корисні, щоб отримати загальну картину системи з точки зору користувача та вирішити такі проблеми, як узгодженість (подібні функції доступні за допомогою подібних команд користувача), покриття (підтримуються всі необхідні функції) та надлишковість (те саме функція доступна/недоступна за допомогою різних команд користувача). Горизонтальні прототипи інтерфейсу користувача можуть починатися з швидких прототипів і переходити до робочого коду. Прототипи програмного забезпечення можна створювати за допомогою конструктора інтерфейсу, не створюючи жодної базової функціональності, що дає змогу перевірити, як користувач буде взаємодіяти з інтерфейсом користувача, не турбуючись про те, як працює решта

архітектури. Однак часто потрібен певний рівень створення, або моделювання решти програми, інакше прототип неможливо оцінити належним чином. Як наслідок, горизонтальні прототипи програмного забезпечення мають тенденцію до еволюції, тобто вони поступово трансформуються в кінцеву систему.

Метою вертикального прототипу є забезпечення того, щоб розробник міг реалізувати повну робочу систему, від рівня інтерфейсу користувача до базового системного рівня. Вертикальні прототипи часто створюються для оцінки доцільності функції, описаної в горизонтальному, орієнтованому на завдання або сценарному прототипі.

Наприклад, коли ми розробили поняття магнітних напрямних у системі CPN2000 для полегшення вирівнювання графічних об'єктів (Beaudouin-Lafon and Mackay, 2000), був реалізований вертикальний прототип, щоб перевірити не лише техніку взаємодії, але й алгоритм компонування та виконання. Було відомо, що можливо включити конкретну техніку взаємодії лише в тому випадку, якщо існує момент для реалізації достатньої швидкості відповіді.

Вертикальні прототипи, як правило, є високоточними програмними прототипами, оскільки їх метою є підтвердження ідеї на системному рівні. Їх часто викидають, оскільки вони зазвичай створюються на початку проекту, до того, як буде визначено загальну архітектуру, і вони зосереджені лише на одному питанні дизайну.

Наприклад, вертикальний прототип перевірки орфографії для текстового редактора не вимагає реалізації та тестування функцій редагування тексту. Однак остаточну версію потрібно буде інтегрувати в решту системи, що може передбачати значні архітектурні зміни або зміни інтерфейсу. Орієнтовані на завдання прототипи Багато дизайнерів інтерфейсу користувача починають з аналізу завдань, щоб визначити окремі завдання, які користувач повинен виконати за допомогою системи.

Кожне завдання вимагає від системи відповідного набору функцій. Прототипи на основі завдань організовані як серія завдань, що дозволяє як дизайнерам, так і користувачам тестувати кожне завдання незалежно, систематично опрацьовуючи всю систему. Орієнтовані на завдання прототипи включають лише функції, необхідні для реалізації зазначеного набору завдань. Вони поєднують широту горизонтальних прототипів, щоб охопити функції, необхідні для цих завдань, з глибиною вертикальних прототипів, що дає змогу детально аналізувати те, як завдання можуть бути підтримані. Залежно від 18мети прототипу.

Розробка прототипу та інструменти, для орієнтованих на завдання прототипів можуть використовуватися як автономні, так і он-лайнові уявлення.

Визначення архітектури програмного забезпечення традиційно виконується після написання функціональної специфікації, але до початку кодування. Дизайнери розробляють на основі структури програми та того, як функції будуть реалізовані програмним забезпеченням модулів розробки прототипів та інструментів.

Архітектура програмного забезпечення — це призначення функцій модулям. В ідеалі кожна функція повинна бути реалізована одним модулем, а модулі повинні мати мінімальні залежності між собою. Не продуктивні архітектури збільшують витрати на розробку (кодування, тестування та інтеграцію), зменшують ремонтопридатність і знижують продуктивність. Архітектура, розроблена для підтримки створення прототипів та розвитку, має вирішальне значення для того, щоб альтернативи дизайну можна було протестувати з максимальною гнучкістю та за розумною ціною. Seeheim і Arch Перша загальна архітектура для інтерактивних систем була розроблена на семінарі в Зеехаймі (Німеччина) в 1985 році і відома як модель Зеехайма (Pfaff, 1985). Він розділяє інтерактивну програму на інтерфейс користувача та функціональне ядро (тоді називалося «додаток», оскільки інтерфейс користувача розглядався як додавання «шарпа фарби» поверх існуючої програми). Інтерфейс користувача складається з трьох модулів: презентації, контролера

діалогів та інтерфейсу програми. У презентації йдеться про фіксацію введених даних користувача на низькому рівні (часто називають лексичним рівнем у порівнянні з лексичним, синтаксичним і семантичним рівнями компілятора). Презентація також відповідає за створення вихідних даних для користувача, як правило, у вигляді візуального відображення.

Контролер діалогів об'єднує введені користувачем дані в команди (він же синтаксичний рівень), забезпечує деякий негайний зворотний зв'язок для виконуваної дії, наприклад, еластичну гумову лінію, і виявляє помилки. Нарешті, інтерфейс програми інтерпретує команди у виклики до функціонального ядра (він же семантичний рівень).

Він також інтерпретує результати цих викликів і перетворює їх у вихідні дані, які будуть представлені користувачеві. Презентація Діалог Контролер Програмний інтерфейс Функціональний інтерфейс користувача Ядро. Модель Зеехайма (Пфафф, 1985) Усі моделі архітектури інтерактивних систем базуються на моделі Зеехайма. Усі вони визнають, що є частина системи, присвячена фіксації дій користувача та представленню результатів (презентація), а інша частина присвячена функціональному ядру (обчислювальна частина програми). Між ними є один або кілька модулів, які перетворюють дії користувача на функціональні виклики, а дані програми (включаючи результати функціональних викликів) у вихідні дані користувача. Сучасною версією моделі Seeheim є модель Arch (The UIMS Workshop Tool Developers, 1992). Модель Арка складається з п'яти компонентів.

Компонент набору інструментів інтерфейсу — це вже існуюча бібліотека, яка надає послуги низького рівня, такі як кнопки, меню тощо. Компонент презентації забезпечує рівень абстракції порівняно з набором інструментів інтерфейсу користувача. Як правило, він реалізує методи взаємодії та візуалізації, які ще не підтримуються набором інструментів інтерфейсу.

Він також може забезпечити незалежність від платформи, підтримуючи різні набори інструментів. Компонент функціонального ядра реалізує функціональність

системи. У деяких випадках він уже існує і не може бути змінений. Компонент адаптера домену надає додаткові послуги компоненту діалогу, які не входять до функціонального ядра. Наприклад, якщо функціональним ядром є файлова система Unix, а інтерфейс користувача є культовим інтерфейсом, подібним до Macintosh Finder, адаптер домену може надати контролеру діалогу службу сповіщень, щоб презентацію можна було оновлювати щоразу, коли файл змінюється. Нарешті, компонент діалогу є ключовим каменем арки; він обробляє переклад між світом інтерфейсу користувача та світом домену.

Розробка прототипу та інструменти Діалог Компонент Презентація Інструментарій взаємодії компонента Компонент DomainAdpater Компонент DomainSpecific Компонент Рисунок 26: Модель Arch (Інструмент розробників UIMS Workshop Developers, 1992) Модель Arch також відома як модель Slinky відносні розміри компонентів можуть відрізнятися в різних програмах, а також протягом терміну експлуатації програмного забезпечення. Наприклад, компонент презентації може бути майже порожнім, якщо набір інструментів інтерфейсу надає всі необхідні послуги, і пізніше його буде розширено для підтримки конкретних методів взаємодії чи візуалізації або кількох платформ.

Аналогічно, ранні прототипи можуть мати великий адаптер домену, що імітує функціональне ядро кінцевої системи або підключається до ранньої версії функціонального ядра; адаптер домену може скоротитися майже до нуля, коли остаточна система буде зібрана. Розділення, яке здійснюють Seeheim, Arch та більшість інших моделей архітектури між інтерфейсом користувача та

функціональним ядром, є хорошим, прагматичним підходом, але в деяких випадках це може викликати проблеми.

Типовою проблемою є зниження продуктивності, коли компоненти інтерфейсу (ліва нога) повинні запитувати компоненти домену (права нога) під час взаємодії, наприклад перетягування.

Наприклад, під час перетягування значка файлу на робочий стіл мають висвітлюватися значки папок і програм, які можуть отримати файл. Визначення піктограм для виділення є семантичною операцією, яка залежить від типів файлів та іншої інформації, і тому має виконуватися функціональним ядром або адаптером домену. Якщо перетягування реалізовано в наборі інструментів інтерфейсу користувача, це означає, що щоразу, коли курсор переходить на нову піктограму, потрібно пройти до чотирьох модулів один раз запитом і один раз у відповідь, щоб дізнатися, чи щоб виділити піктограму. Це і складно, і повільно. Рішення цієї проблеми, яке називається семантичним делегуванням, передбачає переміщення в архітектурі деяких функцій, таких як відповідність файлів для перетягування з ділянки домену в компонент діалогу або презентації. Це може вирішити проблему ефективності, але ціною додаткової складності, особливо під час підтримки або розвитку системи, оскільки це створює залежності між модулями, які в іншому випадку повинні бути незалежними.

Архітектурні моделі MVC і PAC, такі як Seeheim і Arch, є абстрактними моделями і, таким чином, є досить неточними. Вони мають справу з такими категоріями модулів, як презентація або діалог, коли в реальній архітектурі кілька модулів мають справу з презентацією, а кілька інших з діалогом. Модель Model-View-Controller або MVC модель (Krasner and Pope, 1988) є набагато більш конкретною. MVC був створений для реалізації середовища Smalltalk-80 (Goldberg & Robson, 1983) і реалізований як набір класів Smalltalk. Модель описує інтерфейс програми як набір триплетів об'єктів. Кожна трійка містить модель, вигляд і контролер. Модель представляє інформацію, яку необхідно представляти та взаємодіяти з нею. Нею керують об'єкти додатків. Подання відображає інформацію в моделі певним чином.

Контролер інтерпретує введені користувачем дані в поданні Prototype Development and Tools і перетворює його на зміни в моделі. Коли модель змінюється, вона сповіщає про свій вигляд, щоб можна було оновити дисплей. Подання та контролери тісно пов'язані між собою і іноді реалізуються як один об'єкт.

Модель є абстрактною, якщо вона не має представлення і контролера. Він неінтерактивний, якщо у нього є представлення, але немає контролера. Триплети MVC зазвичай складаються в дерево, наприклад, абстрактна модель представляє весь інтерфейс, вона має кілька компонентів, які самі є моделями, такі як рядок меню, вікна документа тощо, аж до окремих елементів інтерфейсу, таких як кнопки та смуги прокрутки. MVC досить легко підтримує кілька представлень: у них є одна модель; коли контролер змінює модель, усі представлення сповіщаються та оновлюють свою презентацію.

Модель Presentation-Abstraction-Control, або PAC (Coutaz, 1987), близька до MVC. Як і MVC, архітектура, заснована на PAC, складається з набору об'єктів, які називаються агентами PAC, організованих у вигляді дерева. Агент PAC має три аспекти: Presentation піклується про фіксацію введених даних користувача та створення вихідних даних;

Абстракція містить дані програми, як модель у MVC;

Фасет керування керує зв'язком між аспектами абстракції та представлення агента, а також із субагентами та суперагентами в дереві. Як і MVC, легко підтримується декілька переглядів. На відміну від MVC, PAC є абстрактною моделлю, тобто не має еталонної реалізації. Варіант MVC, який називається MVP (Model-View-Presenter), дуже близький до PAC і використовується в Dolphin Smalltalk від ObjectArts.

Інші моделі архітектури були створені для певних цілей, таких як програмне забезпечення груп (Dewan, 1999) або графічні додатки (Fekete and Beaudouin-Lafon, 1996).

Шаблони проектування Архітектурні моделі, такі як Arch або PAC, розглядають лише загальний дизайн інтерактивного програмного забезпечення. PAC більш дрібнозернистий, ніж Arch, а MVC більш конкретний, оскільки він заснований на реалізації. Тим не менш, розробник інтерфейсу користувача повинен вирішити багато питань, щоб перетворити архітектуру на працюючу систему. Шаблони

проектування з'явилися в останні роки як спосіб знайти ефективні рішення для періодичних проблем проектування програмного забезпечення. У своїй книзі Gamma et al. (1995) представляють 23 моделі.

Цікаво відзначити, що багато з цих шаблонів походять з інтерактивного програмного забезпечення, і більшість з них можна застосувати до проектування інтерактивних систем. Детальний опис цих моделей виходить за рамки цього розділу. Однак цікаво, що більшість шаблонів для інтерактивних систем є поведінковими шаблонами, тобто шаблонами, які описують, як реалізувати структуру управління системою. Справді, в інтерактивному програмному забезпеченні йде боротьба за контроль.

У традиційному алгоритмічному програмному забезпеченні алгоритм контролює і вирішує, коли читати вхідні дані та записувати вихідні дані.

В інтерактивному програмному забезпеченні користувальницький інтерфейс повинен контролюватись, оскільки введення користувача має керувати реакцією системи. На жаль, найчастіше функціональне ядро також потребує контролю. Це особливо часто трапляється при створенні інтерфейсів користувача для застарілих програм. У моделях Seeheim і Arch часто вважають, що управління знаходиться в контролері діалогу, хоча насправді ці архітектурні моделі не вирішують явно питання керування.

У MVC три основні класи Model, View і Controller реалізують складний протокол, щоб гарантувати, що введення користувача враховується своєчасно і що зміни моделі належним чином відображаються у поданні (або представленнях). Деякі автори насправді описують MVC як шаблон проектування, а не архітектуру. Насправді це і те, і інше: внутрішня робота трьох основних класів є шаблоном, але розкладання програми прототипів та інструментів на набір триплетів MVC є архітектурною проблемою. Зараз широко визнано, що інтерактивне програмне забезпечення керується подіями, тобто виконанням керують дії користувача, що призводить до контролю, локалізованого в компонентах інтерфейсу користувача. Шаблони проектування, такі як команда, ланцюг відповідальності, посередник і

спостерігач (Gamma et al., 1995), особливо корисні для реалізації перетворення події користувача низького рівня в команди вищого рівня, щоб з'ясувати, який об'єкт в архітектурі відповідає. до команди, а також для поширення змін, створених командою, від внутрішніх об’єктів функціонального ядра до об’єктів інтерфейсу користувача.

Використання шаблонів проектування для реалізації інтерактивних систем не тільки заощаджує час, але й робить систему більш відкритою для змін і легшою в обслуговуванні. Тому прототипи програмного забезпечення повинні впроваджуватися досвідченими розробниками, які знають мову шаблонів і розуміють потребу в гнучкості та розвитку.

Інкрементальне прототипування вона становить собою готовий продукт котрий побудувано, як роздільні частини прототипи. В кінці окремі прототипи з’єднуються у спільний дизайн. За допомогою чого поступове створення прототипу та часовий проміжок між клієнтом та розробником ІС зменшується.

Екстремальне прототипування - процес створення системи який використовується конкретно для створення веб-додатків. Тобто він розділяє веб-розробку на три етапи, кожен з яких переймає сформований на попередньому.

1. це статичний прототип, який складається з HTML-сторінок.
2. це клієнтська частина яка формується і повністю функціону при застосуванні рівня сервісів, що відтворює реальну функціональність. На етапі сервісу впроваджуються. Процес має назву екстремальне прототипування, щоб сконцентрувати увагу на другій фазі процесу, де цілком та повністю.
3. функціональний інтерфейс користувача створюється при меншій увазі до послуг, окрім крім їх безпосереднього контракту.

Дана методика дає змогу команді розробників інтегрувати функціональності чи додавати зміни котрі майже неможливо було уявити в процесі розробки вимог.

Для того щоб сформована система була якомога корисною, їй необхідно розвиватися за рахунок інтеграції у наміченому операційному середовищі. Система ніколи не розробляється; вона завжди ‘дозріває’ у ту міру змін її середовища використанн. Розробники часто намагаються визначити систему, використовуючи ту систему відліку відліку у якій вони більш компитентні– де вони наразі знаходяться.

Маємо змогу зробити припущення про те, яку взаємодію буде виконуватися в системі, і про технологічну базу, на якій віна буде реалізована. Сформований план розвитку розроблено, тому, рано чи пізно, виникає те, що нагадує зрозумілу систему.

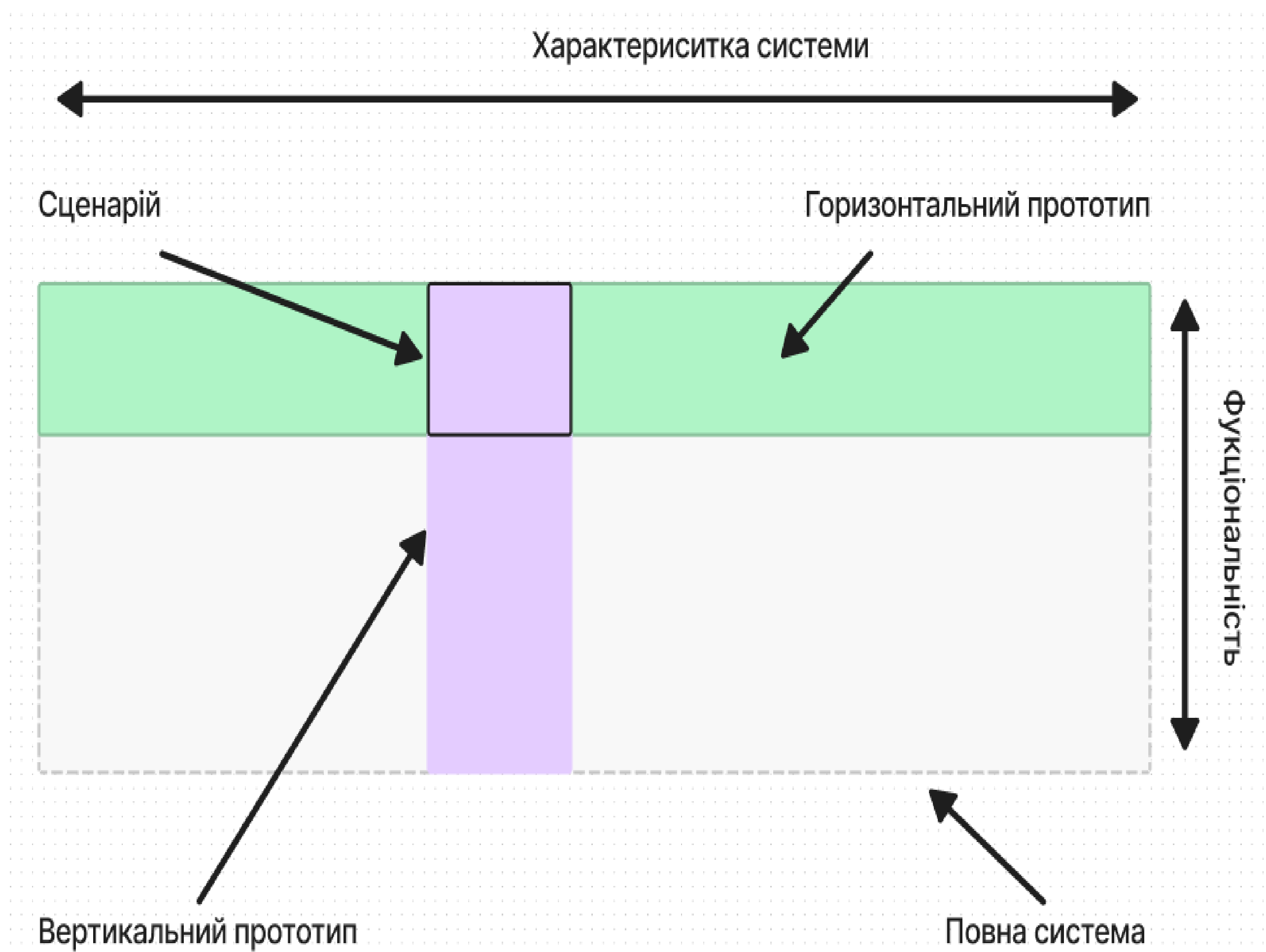


Рисунок 1.1. Співвідношення горизонтального та вертикального прототипів

Еволюційний прототип концентрує в собі перевагу над одноразовим прототипом саме тому, що є функціональними системами. Хоча вони можуть мати не всі функції, заплановані користувачами, вони можуть використовуватися на тимчасовій основі, доки не буде поставлена остаточна система.

«У сфері використання прототипів часто користувач застосовує початковий прототип для практичного використання аби в подальшому отримати версію з більшою функціональністю … Бо користувач системи може подумати, що не стабільна ІС є кращою, а ніж зовсім без неї».

В свою чергу у еволюційному прототипу розробники маєть змогу зосередитисвою увагу та зусилля в створені компонентів системи, в яких вони мають розуміння, замість того, щоб реалізовувати всю систему відразу.

Для мінімізації ризиків, розробник не розробляє не коректно сформованих функціональностей. Не повна система надсилається на сервери клієнтів. Саме тоді коли користувачі мають змогу взаємодіяти системою, коли вони вказують на можливості для нових функцій та надсилають вимоги чи поба жання на ці та інші нові можливості системи самим розробникам. Потім самі розробники отримують ці запити на покращення разом зі своїми менеджерами та застосовують перевірені методики управління конфігурацією для інтеграції зміни вимог до інформаційної системи, оновлення структури, рефакторинг такаож повторного чи автоматичного тестування.

Ці технології розраховані головним чином на забезпечення швидкої розробки прототипів, а не на такі їх системні характеристики, як продуктивність, зручність експлуатації або безвідмовність. Існує три основні методи швидкої розробки прототипів.

1. Розробка із застосуванням динамічних мов високого рівня.
2. Використання мов програмування баз даних.
3. Складання додатків з повторним використанням компонентів.

Для зручності ці методи описані в окремих розділах. Але на практиці вони часто спільно використовуються при розробці прототипів систем. Наприклад, мова програмування баз даних може застосовуватися для добування даних з їхньою наступною обробкою за допомогою повторно використовуваних компонентів. Інтерфейс користувача системи можна розробити, використовуючи візуальне програмування.

У цей час розробка прототипів зазвичай опирається на набір інструментів, що підтримують принаймні два із цих методів. Наприклад, система Smalltalk VisualWorks підтримує мову дуже високого рівня і забезпечує повторне використання компонентів. Пакет Lotus Notes включає підтримку програмування баз даних за допомогою мови високого рівня і повторне використання компонентів, які можуть забезпечити операції над базою даних.

Більшість систем прототипування сьогодні підтримують візуальне програмування, при якому деякі частини або весь прототип розробляються в інтерактивному режимі. Замість послідовного написання програм розробник прототипу віддає перевагу працювати із графічними піктограмами, що представляють функції, дані або компоненти інтерфейсу користувача сценаріями, що і відповідають, керування цими піктограмами. Програма, готова до виконання, генерується автоматично з візуального представлення системи. Це спрощує розробку програми і зменшує витрати на прототипування.

1.3 Прототипування баз даних

Еволюційне прототипування в цей час є стандартною методикою для планування маштабованих додатків малого і середнього розміру. Більшість ІС містять у собі систему керування базою даних і обробку даних, що перебувають у ній.

Для протитипування та підтримки і розробки даних додатків усі такці системи керування базами даних мають внутрішні засоби програмування. Програмування

баз даних запускається в своїй основі спеціалізованих мов, які мають інтегровану базу знань і засобів, необхідні для взаємодії з БД. Робоче середовище підтримки мови забезпечує інструментальні засоби для створення користувацьких інтерфейсів, числових обчислень і звітів. Термін ***мова четвертого покоління***застосовується як до самої мови програмування баз даних, так і до його робочого середовища.

Мови четвертого покоління досить продуктивно використовуються на практиці, тому, що більшість сучасних інформаційних систем тією чи іншою мірою виконують обробку потоку інформації, установленої в базах даних. Головні операції, виконувані такими додатками – це модифікація бази даних і створення звітів на основі, витягнутої з бази даних. Зазвичай для введення і виведення даних використовуються базові структури. Мови четвертого покоління мають інструменти для розробки інтерактивних додатків, що дозволяють користувачам редагувати, видаляти та додавати дані в базу даних. Користувацький інтерфейс будується з набору базових форм або електронних таблиць.

У базовому вигляді дане середовище мов четвертого покоління має наступний набір інструментальних засобів так функцій.

1. У якості мови створення баз даних , а саме мови запитів до бази даних використовується SQL, також є можливість виконати ті ж самі дії за допомого інтсрументів котрі взаємодіють з таблицю БД, як об'єкт, що робить процес взаємодії з даними більш гнучким.
2. Генератор інтерфейсів використовується для створення форм введення і відображення даних.
3. Електронна таблиця застосовується для аналізу даних і виконання різних дій над числовою інформацією.
4. Генератор звітів призначений для створення звітів на основі інформації, яка міститься в базі даних.

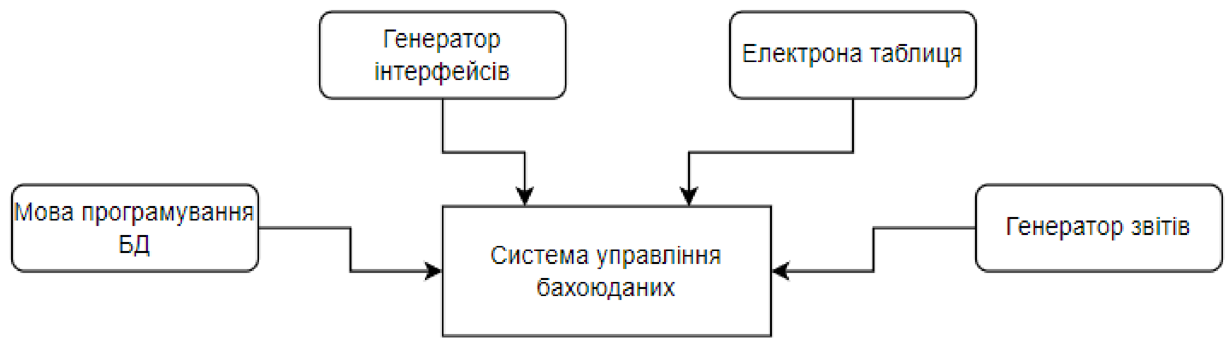


Рисунок 1.2. Компоненти мови четвертого покоління

В переважній більшості додатків передбачають формовані моделі для введення і виведення даних, тому мови четвертого покоління надають потужні інструменти для визначення форм і створення звітів. Екранні форми часто визначаються як ряд взаємозалежних форм, тому система, що генерує екрани, повинна забезпечувати наступне.

1. ***Інтерактивну структуру форм***, коли розробник встановлює поля введення та їх організацію.
2. ***Зв'язування форм,*** коли розробник встановлює дані, введення котрих встановлює відображення подальших форм:
3. ***Перевірки вхідних даних,*** коли розробник при генерацції полів форм обмежує дозволений радіус вхідних даних.

В даний момент в переважній більшості мов четвертого покоління зберігається інтеграція інтерфейсів баз даних, які в основному засновані на запуску даних інформаційних систем Web-броузерах, що робить таку систему мультиплатформену, та надає можливість для взаємодії зі смартфона, планшету … Також вони дозволяють зробити БД доступною за допомогою використання Internet. Такий підхід знижує вартість навчання ІС та дозволяє зовнішнім користувачам та системам мати доступ до бази даних. Проте обмеження протоколів Internet і повільне завантаження Web-сторінок роблять дану методику не продукктивною для систем, яким потрібна швидка та безперевна взаємодія.

Фундаментальні методики, в які сформовані на мовах четвертого покоління, дають змогу використовувати підхід еволюційного прототипування чи для генерування “одноразового” прототипу системи. Форму моделі, яку CASE-засоби накладають на створюване ПЗ та її документацію, надає якомога зручну підтримку прототипів, чим ту , що пропонують прототипи, які були запрограмовані вручну. CASE-засоби дають змогу генерувати код SQL або код мовою іншого рівня, наприклад COBOL.

Хоча мови четвертого покоління підходять для розробки прототипів, все-таки вони мають певну кількість недоліків, які проявляються при створені систем. Інформаційні системи, що створені на мовах четвертого покоління, як правило, працюють більш повільно, а ніж подібні програми, котрі написані за допомогою звичайних мовах програмування, та вимагають в рази більше пам'яті

Навіть при використані мов четвертого покоління коли знижує вартість розробки систем, загальна сума витрат на повний життєвий цикл та підтримку дані системи поки не ясна. Такі програми в загальних випадках не коректно структуровані і як правило це призводить до трудомісткого внесення змін. Не тривіальні проблеми можуть з'являтися при певних змінах конфігурації систем чи подібних її модулей. Тому в процесі мови четвертого покоління не мають типізації та установлених стандартів розробки, тому при модифікації систем, скоріше всього, доведеться робити рефакторинг програми, бо обрана мова, на якій вони написані, стає не актуальною.

З моменту появи графічних інтерфейсів користувача у вісімдесятих роках було розроблено велику кількість інструментів, які допомагають у створенні інтерактивного програмного забезпечення, переважно націленого на візуальні інтерфейси. У цьому розділі представлено набір інструментів, від низькорівневих, тобто потребують багато програмування, до високорівневих. Інструменти найнижчого рівня – це графічні бібліотеки, які забезпечують апаратну незалежність для малювання пікселів на екрані та обробки введення користувача, а також віконні системи, які надають абстракцію (вікно) для структурування екрана на кілька

«віртуальних терміналів». Набори інструментів інтерфейсу користувача структурують інтерфейс як дерево інтерактивних об'єктів, які називаються віджетами, тоді як конструктори інтерфейсу користувача надають інтерактивну програму для створення та редагування цих дерев віджетів.

Фреймворки програм базуються на наборах інструментів і конструкторах інтерфейсу користувача, щоб полегшити створення типових функцій, таких як вирізати/копіювати/вставити, скасувати, довідку та інтерфейси на основі редагування кількох документів в окремих вікнах. Інструменти на основі моделі напівавтоматично отримують інтерфейс із специфікації об'єктів і функцій предметної області, які мають підтримуватися. Нарешті, середовища розробки інтерфейсу користувача або UIDE надають інтегровану колекцію інструментів для розробки інтерактивного програмного забезпечення. Перш ніж описати кожну з цих категорій більш детально, важливо зрозуміти, як їх можна використовувати для створення прототипів.

Не завжди найкраще використовувати доступний інструмент найвищого рівня для створення прототипів. Інструменти високого рівня є найбільш цінними в довгостроковій перспективі, оскільки вони полегшують обслуговування системи, перенесення її на різні платформи або локалізацію на різні мови. Ці проблеми не мають відношення до вертикальних і одноразових прототипів, тому інструмент високого рівня може виявитися менш ефективним, ніж інструмент нижнього рівня. Основним недоліком інструментів вищого рівня є те, що вони обмежують або стереотипують типи інтерфейсів, які вони можуть реалізувати. Набори інструментів користувальницького інтерфейсу зазвичай містять обмежений набір «віджетів», і створювати нові досить дорого. Якщо дизайн повинен включати нові прийоми взаємодії, такі як біручна взаємодія (Kurtenbach et al., 1997) або інтерфейси з можливістю масштабування (Bederson & Hollan, 1994), інструментарій інтерфейсу користувача буде перешкоджати, а не допомагати розробці прототипу.

Аналогічно, фреймворки прикладних програм припускають стереотипну програму з рядком меню, кількома панелями інструментів, набором вікон, що містять документи, тощо. Така структура була б непридатною для розробки гри чи мультимедійного навчального компакт-диска, який вимагає плавного, динамічного та оригінального інтерфейс користувача.

Розробка прототипів та інструменти нарешті, розробникам необхідно по-справжньому оволодіти цими інструментами, особливо при створенні прототипів для підтримки команди дизайнерів. Успіх залежить від здатності програміста швидко змінювати деталі, а також загальну структуру прототипу. Розробник буде більш продуктивним, використовуючи знайомий інструмент, ніж якщо він змушений використовувати потужніший, але невідомий інструмент.

1. ІНСТРУМЕНТИ ДЛЯ ІМПЛЕМЕНТАЦІЇ МЕТОДИКИ ПРОТОТИПУВАННЯ

2.1 Аналіз інструментів для створення сиситеми прототипування

Всі існуючі методики протипування досить активно застосовуються при стоврені інформаційних систем є дуже корисним інструментом, який має свої недоліки, та в резулььтаті вони все одно підвищують результативність команди та підприємств.

Проаналізувавши всі загальні особливості методики, було зроблено висновки та створено новий підхід, який оснований на об’єднані декількох метод в один. Щоб підтвердити моє твердження, було створено програмне забезпечення, яке включає в собі вище описані характеристики та вимоги прототипування.

Першим кроком для створення такої системи було сформульовано наступні вимоги до інсрументів розробки.

- Встановлення, оновлення;
- Публікація / розгортання;
- Надійність;
- Доступність;
- Кросплатформеність;
- Функціональність, швидкодія;
- Безпека;

Всім цим вимогам відповідає веб-додаток.

В даний момент існує можливість створення web-додатків, які практично не можливо відрізнити від desktop чи mobile. Тому виникає великий сенс створення ІС, які запускаються з браузера.

Веб-додаток не вимагає установки, всі оновлення відбуваються на сервері, доставляються користувачам відразу - досить просто перезавантажити сторінку або вийти, а потім знову зайти в обліковий запис. Але іноді для його роботи необхідно встановити додаткові бібліотеки або використовувати захищені мережеві протоколи.

Веб-програма публікується на локальному або хмарному сервері, там же відбувається процес оновлення. У цьому сервер потрібен у разі, навіть якщо рішення дуже просте. Адже крім фронтенду, з яким користувачі працюватимуть через браузер, потрібно десь розміщувати бекенд.

Робота веб-застосування залежить не тільки від того, наскільки грамотно воно розроблено і характеристик пристрою, але також від швидкості інтернет-з'єднання, працездатності віддаленого сервера.

Веб-додаток доступний з будь-якої точки світу, з будь-якого пристрою, а файли користувача завжди будуть під рукою. Але якщо є інтернет-з'єднання або реалізована можливість роботи офлайн і завантаження-вивантаження даних.

Веб-додаток однаково добре буде працювати на будь-якому пристрої, будь то стаціонарний комп'ютер, ноутбук, планшет або смартфон - адже він практично не залежить від "заліза" або операційної системи. Головне - підходящий браузер. Як правило, для роботи більшості веб-клієнтів підходять Google Chrome, Mozilla Firefox, Safari від Apple або Windows-браузер (Microsoft Edge/Internet Explorer).

Веб-застосунок повністю залежить від браузера та технологій його роботи. Тому є ряд обмежень, наприклад, доступ до апаратного забезпечення вашого пристрою. Це й деякі інші обмеження оминути неможливо (принаймні зараз). Але цілу низку завдань можна вирішити за принципом «що не можна переписати, можна надбудовувати чи розширювати». Редактори документів, зображення, аудіо, відео, 3D графіки; системи управління проектами; сховища файлів; no-code конструктори

– успішно працюють у браузерах. Інструменти швидкої інтеграції сервісів, а також інтерфейсні бібліотеки ще більше розширюють можливості.

Веб-додаток, розроблений з використанням сучасних протоколів та засобів захисту, здатний повноцінно забезпечувати збереження даних. Однак на деякі моменти розробники не можуть вплинути: браузер, хмарний сервер, канал зв'язку можуть підвищити рівень безпеки за рахунок додаткових засобів перевірки, але також знизити його за рахунок своїх вразливостей. Безперечний плюс для користувачів: таке ПЗ простіше контролювати. Обмеження середовища знижують ймовірність, що воно приховано отримає доступ до файлів або запустить процес.

Всі вище вказані пукти доводять правильність вибору розробку саме, як web-app.

Інструменти вікористані для підтвердження методики прототипування

Node.js — це кросплатформне середовище виконання JavaScript із відкритим вихідним кодом, яке працює на движку V8 і виконує код JavaScript за межами веб-браузера. Node.js дозволяє розробникам використовувати JavaScript для написання інструментів командного рядка та для сценаріїв на стороні сервера — запуск сценаріїв на стороні сервера для створення динамічного вмісту веб-сторінки, перш ніж сторінка буде відправлена у веб-браузер користувача. Отже, Node.js являє собою парадигму «JavaScript всюди», що об’єднує розробку веб-додатків навколо однієї мови програмування, а не різних мов для сценаріїв на стороні сервера та клієнта.

Node.js має архітектуру, керовану подіями, з можливістю асинхронного введення-виведення. Ці варіанти дизайну спрямовані на оптимізацію пропускної здатності та масштабованості у веб-додатках з багатьма операціями введення/виводу, а також для веб-додатків реального часу (наприклад, комунікаційні програми в режимі реального часу та браузерні ігри).

Проект розподіленої розробки Node.js раніше керувався Node.js Foundation, а тепер об'єднався з JS Foundation, щоб утворити OpenJS Foundation, якому сприяє програма Collaborative Projects Linux Foundation.

Node.js дозволяє створювати веб-сервери та мережеві інструменти за допомогою JavaScript і набору «модулів», які обробляють різні основні функції Надаються модулі для вводу-виводу файлової системи, мереж (DNS, HTTP, TCP, TLS/SSL або UDP), двійкових даних (буферів), функцій криптографії, потоків даних та інших основних функцій.

Модулі Node.js використовують API, призначений для зменшення складності написання серверних додатків.

JavaScript є єдиною мовою, яку Node.js підтримує ізначально, але є багато мов компіляції в JS. В результаті програми Node.js можна писати на CoffeeScript, Dart, TypeScript, ClojureScript та інших.

Node.js в основному використовується для створення мережевих програм, таких як веб-сервери. Найбільш істотна відмінність між Node.js і PHP полягає в тому, що більшість функцій у PHP блокують до завершення (команди виконуються лише після завершення попередніх команд), тоді як функції Node.js не блокують (команди виконуються одночасно або навіть паралельно, і використовувати зворотні виклики, щоб повідомити про завершення або невдачу).

Node.js офіційно підтримується в Linux, macOS і Microsoft Windows 8.1 і Server 2012 (і пізніших версіях), з підтримкою рівня 2 для SmartOS і IBM AIX і експериментальною підтримкою FreeBSD. OpenBSD також працює, і версії LTS доступні для IBM і (AS/400). Наданий вихідний код також може бути побудований на операційних системах, подібних до тих, що офіційно підтримуються, або модифікований третіми сторонами для підтримки інших, таких як NonStop OSі сервери Unix.

Node.js надає веб-серверам програмування на основі подій, що дозволяє розробляти швидкі веб-сервери на JavaScript. Розробники можуть створювати масштабовані сервери без використання потоків, використовуючи спрощену модель програмування на основі подій, яка використовує зворотні виклики для сигналу про завершення завдання. Node.js поєднує простоту мови сценаріїв (JavaScript) з потужністю мережевого програмування Unix.

Node.js був побудований на основі двигуна JavaScript V8 від Google, оскільки він був з відкритим кодом під ліцензією BSD. Він володіє основами Інтернету, такими як HTTP, DNS і TCP. JavaScript також був добре відомою мовою, що робило Node.js доступним для спільноти веб-розробників.

Існують тисячі бібліотек з відкритим кодом для Node.js, більшість із них розміщені на веб-сайті npm. Існує безліч конференцій і заходів для розробників, які підтримують спільноту Node.js, включаючи NodeConf, Node Interactive і Node Summit, а також ряд регіональних заходів.

Спільнота з відкритим кодом розробила веб-фреймворки для прискорення розробки додатків. Такі фреймворки включають Connect, Express.js, Socket.IO, Feathers.js, Koa.js, Hapi.js, Sails.js, Meteor, Derby та багато інших. Також були створені різні пакети для взаємодії з іншими мовами або середовищами виконання, такими як Microsoft .NET.

Сучасні настільні IDE надають функції редагування та налагодження спеціально для додатків Node.js. Такі IDE включають Atom, Brackets, JetBrains WebStorm, Microsoft Visual Studio (з інструментами Node.js для Visual Studio або TypeScript з визначеннями Node,) NetBeans, Nodeclipse Enide Studio (на основі Eclipse) і Visual Studio Code. Деякі онлайнові веб-інтегровані IDE також підтримують Node.js, такі як Codeanywhere, Codenvy, Cloud9 IDE, Koding та редактор візуального потоку в Node-RED.

Node.js підтримується на багатьох платформах хмарного хостингу, таких як Jelastic, Google Cloud Platform, AWS Elastic Beanstalk, Joyent та інші.

TypeScript — це мова програмування, розроблена та підтримувана Microsoft. Це суворий синтаксичний наднабір JavaScript і додає до мови необов'язковий статичний введення. Він розроблений для розробки великих додатків і транспіляції на JavaScript. Оскільки це наднабір JavaScript, існуючі програми JavaScript також є дійсними програмами TypeScript.

TypeScript можна використовувати для розробки додатків JavaScript для виконання як на стороні клієнта, так і на стороні сервера (як з Node.js або Deno). Доступно кілька варіантів транспіляції. Можна використовувати перевірку TypeScript за замовчуванням або запустити компілятор Babel для перетворення TypeScript у JavaScript.

TypeScript підтримує файли визначення, які можуть містити інформацію про типи існуючих бібліотек JavaScript, так само, як файли заголовків C++ можуть описувати структуру існуючих об'єктних файлів. Це дозволяє іншим програмам використовувати значення, визначені у файлах, так, ніби вони були статично типізованими об'єктами TypeScript. Існують сторонні заголовні файли для популярних бібліотек, таких як jQuery, MongoDB і D3.js. Також доступні заголовки TypeScript для базових модулів Node.js, що дозволяє розробляти програми Node.js у TypeScript.

Сам компілятор TypeScript написаний на TypeScript і скомпільований на JavaScript. Він ліцензований за ліцензією Apache 2.0. TypeScript включено як першокласну мову програмування в Microsoft Visual Studio 2013 Update 2 і новіших версій, поряд із C# та іншими мовами Microsoft. Офіційне розширення також дозволяє Visual Studio 2012 підтримувати TypeScript. Андерс Хейлсберг, провідний архітектор C# і творець Delphi і Turbo Pascal, працював над розробкою TypeScript

TypeScript виник із недоліків JavaScript для розробки великомасштабних додатків як у Microsoft, так і серед їхніх зовнішніх клієнтів. Проблеми, пов’язані зі складним кодом JavaScript, призвели до потреби в спеціальних інструментах для полегшення розробки компонентів у мові.

Розробники TypeScript шукали рішення, яке не порушило б сумісність зі стандартом та його міжплатформною підтримкою. Знаючи, що поточна стандартна пропозиція ECMAScript обіцяла майбутню підтримку програмування на основі класів, TypeScript був заснований на цій пропозиції. Це призвело до створення компілятора JavaScript з набором розширень синтаксичних мов, наднабору на основі пропозиції, який перетворює розширення на звичайний JavaScript. У цьому сенсі TypeScript був попереднім переглядом того, чого очікувати від ECMAScript 2015. Унікальним аспектом не в пропозиції, а доданим до TypeScript, є додаткова статична типізація, яка дозволяє статичний аналіз мови, що полегшує інструменти та підтримку IDE.

TypeScript додає підтримку таких функцій, як класи, модулі та синтаксис функції стрілки, як визначено в стандарті ECMAScript 2015.

Клієнтська частина, - це важивий елемент який відповідає за взаємодію сервера та користувача. Перевагу бело надано Vue.js.

Хоча JavaScript є повноцінним сам по собі, його екосистема означає набагато більше, ніж сама мова. Такі інструменти, як фреймворки, значно полегшують життя розробника, забезпечуючи основу для більш гладкої розробки. Якщо мова розглядається як алфавіт, фреймворк можна розглядати як розмовник, що дозволяє розробнику будувати речення та спілкуватися.

Що таке Vue.js? Історія Vue.js починається в 2013 році, коли Еван Ю працював у Google, створюючи безліч прототипів прямо в браузері. З цією метою Еван використав зручні практики з інших фреймворків, з якими він працював, і офіційно випустив Vue.js у 2014 році. Vue.js — це прогресивний фреймворк для JavaScript,

який використовується для створення веб-інтерфейсів і односторінкових програм. Vue.js також використовується не тільки для веб-інтерфейсів для розробки настільних і мобільних додатків за допомогою Electron framework.

Розширення HTML і база JS швидко зробили Vue улюбленим інструментом інтерфейсу, про що свідчить прийняття такими гігантами, як Adobe, Behance, Alibaba, Gitlab і Xiaomi. В інтерв'ю з Еваном він показує, що спочатку Vue.js був спробою використати найкраще з Angular і створити користувацький інструмент, але менший: «Для мене Angular запропонував щось круте, що пов'язує дані та керує даними спосіб поводитися з DOM, тож вам не доведеться самому торкатися DOM». Назва фреймворка – Vue – англійською збігається з фонетичною як view, і вона відповідає традиційній архітектурі Model-View-Controller (MVC). Простіше кажучи, view — це інтерфейс програми/веб-сайту, а основна бібліотека Vue.js за замовчуванням фокусує рівень перегляду. Але MVC не означає, що Vue.js не можна використовувати з іншим архітектурним підходом, як-от Component Based Architecture (CBA), що використовується в React.

Як і будь-яка популярна технологія, програмування Vue.js викликає суперечки у всьому співтоваристві. І є причини, чому Vue став другим найпопулярнішим фреймворком у 2019 році. Давайте подивимося, які це причини.

Об'єктна модель документа (DOM) — це те, з чим ви, ймовірно, зіткнетеся під час візуалізації веб-сторінок. DOM – це представлення сторінок HTML з його стилями, елементами та вмістом сторінки як об'єктами. Об'єкти, що зберігаються у вигляді деревоподібної структури, генеруються браузером під час завантаження сторінки.

Ще однією перевагою маніпуляцій DOM є двостороннє прив'язування даних, успадковане Vue від Angular. Двостороннє прив'язування даних – це зв'язок між оновленнями даних моделі та переглядом (UI). Зв'язані компоненти містять дані, які час від часу можна оновлювати. За допомогою двостороннього зв'язування даних легше оновлювати пов'язані компоненти та відстежувати оновлення даних.

У Vue зв’язані дані оновлюються реактивно, як і об’єкти DOM, що чудово підходить для будь-якої програми, яка потребує оновлення в режимі реального часу. З точки зору розробників, реактивність Vue зробить оновлення даних зрозумілішим і легшим у виконанні. Для того, щоб реактивність працювала, застосовуються деякі правила.

Кожна частина майбутньої програми/веб-сторінки у Vue є компонентом. Компоненти представляють інкапсульовані елементи вашого інтерфейсу. У Vue.js компоненти можна писати на HTML, CSS та JavaScript, не розділяючи їх на окремі файли.

Поділ коду програми насправді є архітектурним підходом, який називається Component Based Architecture (CBA), і він також використовується в Angular і React. Такий архітектурний підхід має масу переваг:

Можливість повторного використання компонентів. Інкапсульовані компоненти — це в основному фрагменти коду, які можна повторно використовувати як шаблони для подібних системних елементів.

Читабельність коду. Оскільки всі компоненти зберігаються в окремих файлах (а кожен компонент є лише одним файлом), код легше читати та розуміти, що полегшує його обслуговування та виправлення.

Добре підходить для модульного тестування. Юніт-тестування — це дія контролю якості, спрямована на перевірку того, як найменші частини програми працюють самостійно. Наявність компонентів значно спрощує це завдання.

Важливим аспектом будь-якої нової технології є можливість інтеграції з існуючими програмами. З Vue.js це так само просто, як пиріг, оскільки він покладається лише на JavaScript і не вимагає жодних інших інструментів для роботи.

Vue також дозволяє вам писати шаблони за вашим бажанням: використовуючи HTML, JS або JSX (розширення синтаксису JavaScript). Серед компонентів і легкої

природи Vue можна використовувати майже в будь-якому проекті. І ми раді повідомити, що перехід з React або Angular не буде великою проблемою, оскільки внутрішня організація Vue, в основному, є поєднанням двох.

Важливим аспектом будь-якої нової технології є можливість інтеграції з існуючими програмами. З Vue.js це так само просто, як пиріг, тому що він покладається лише на JavaScript і не вимагає жодних інших інструментів для роботи.

Vue також дозволяє вам писати шаблони за вашим бажанням: використовуючи HTML, JS або JSX (розширення синтаксису JavaScript). Серед компонентів і легкої природи Vue можна використовувати майже в будь-якому проекті.

Традиційно використовуваний для настільних графічних інтерфейсів користувача (GUI), цей шаблон став популярним для розробки веб-додатків. Популярні мови програмування мають фреймворки MVC, які полегшують реалізацію шаблону.

Модель центральний компонент візерунка. Це динамічна структура даних програми, незалежна від інтерфейсу користувача. Він безпосередньо керує даними, логікою та правилами програми. Будь-яке представлення інформації, наприклад діаграма, діаграма або таблиця. Можливі кілька переглядів однієї і тієї ж інформації, наприклад, стовпчаста діаграма для управління та табличне представлення для бухгалтерів.

Контролер приймає введення та перетворює його в команди для моделі або представлення. На додаток до поділу програми на ці компоненти, дизайн модель–представлення–контролер визначає взаємодію між ними. Модель відповідає за управління даними програми. Він отримує дані користувача від контролера. Подання відображає подання моделі в певному форматі. Контролер реагує на введення користувача та здійснює взаємодії з об’єктами моделі даних. Контролер отримує вхідні дані, за бажанням перевіряє їх, а потім передає вхідні дані моделі.

Як і інші шаблони програмного забезпечення, MVC виражає «ядро рішення» проблеми, дозволяючи його адаптувати для кожної системи. Конкретні конструкції MVC можуть значно відрізнятися від традиційного опису тут.

Незважаючи на те, що MVC спочатку був розроблений для настільних комп'ютерів, він був широко прийнятий як дизайн додатків World Wide Web на основних мовах програмування. Створено кілька веб-фреймворків, які забезпечують виконання шаблону. Ці програмні фреймворки відрізняються за своїми інтерпретаціями, головним чином у тому, як відповідальність MVC розподіляється між клієнтом і сервером.

Деякі веб-фреймворки MVC використовують підхід тонкого клієнта, який розміщує майже всю логіку моделі, представлення та контролера на сервері. У цьому підході клієнт надсилає або запити гіперпосилання, або форми до контролера, а потім отримує повну та оновлену веб-сторінку (або інший документ) із представлення; модель повністю існує на сервері.

Об'єкти моделі інкапсулюють дані, специфічні для програми, і визначають логіку, обчислення, які керують та обробляють ваші дані. Наприклад, об'єктом моделі може являти собою персонаж у грі або контакт з адресної книги. Об'єкт моделі може мати відношення "один-багатьом" з іншими об'єктами моделі, тому іноді модельний рівень додатка є одним або декількома об'єктними графами і мати ієрархічну структуру. Більшість даних, які постійно зберігаються в програмі, повинні знаходитися в об'єктах моделі після завантаження даних у програму. Оскільки об'єкти моделі є якісь знання, пов'язані з конкретним завданням, їх можна повторно використовувати в схожих завданнях. В ідеалі, об'єкт моделі не повинен мати явного зв'язку з об'єктами уявлення, які надають свої дані, і дозволяють користувачам редагувати ці дані - це не повинно стосуватися завдань інтерфейсу користувача.

Зв'язок об'єктів: Користувацькі дії на рівні представлення, які створюють або змінюють дані, повинні передаватися через об'єкт контролера і призводити до

створення або оновлення моделі. Коли об'єкт моделі змінюється (update), він повідомляє (notify) контролер, який оновлює відповідний view у додатку. Простіше кажучи: Модель повідомляє контролер про будь-які зміни даних. У свою чергу, контролер оновлює дані у виставах. Об'єкти представлення – це такі об'єкти, які користувач бачить на екрані. Ці об'єкти знають, як себе показувати і можуть реагувати на дії користувача. Основна мета об'єктів представлення - відображати дані з об'єктів моделі програми та дозволяти редагування цих даних. Незважаючи на це, об'єкти view зазвичай відокремлюються від об'єктів model у програмі MVC.

Оскільки ви зазвичай повторно використовуєте та змінюєте дані, view об'єкти забезпечують узгодженість між програмами. Обидві структури UIKit та AppKit надають колекції класів вистав, а Interface Builder пропонує десятки об'єктів виду у своїй Бібліотеці. Обидві структури UIKit та AppKit надають колекції класів вистав, а Interface Builder пропонує десятки об'єктів виду у своїй бібліотеці.

Зв'язок об'єктів: view об'єкти дізнаються про зміни даних моделі через об'єкти контролера програми і повідомляють про зміни, ініційовані користувачем, наприклад текст, введений в текстові поля дані, проходять через об'єкти контролера об'єктам моделі програми.

Подання повідомляє контролер про дії користувача. Контролер у разі потреби оновлює модель або отримує запитані дані.

Об'єкт контролера діє як посередник між одним або декількома view об'єктами програми та об'єктами model. Об'єкти контролера, таким чином, є каналом, через який об'єкти view дізнаються про зміни в об'єктах моделі і навпаки. Об'єкти контролера також можуть виконувати налаштування та координування завдань для застосування та керувати життєвими циклами інших об'єктів.

Зв'язок об'єктів: Об'єкт контролера інтерпретує дії користувача, створені у view об'єктах, і передає нові чи змінені дані модельний рівень. Коли об'єкти моделі змінюються, об'єкт контролера пов'язує ці нові моделі з об'єктами представлення, щоб вони могли його відображати.

2.2 Правила використання архітектуру MVC

Наш контролер не повинен мати взагалі ніякої логіки, пов'язаної з виконанням якихось операцій (скачування, завантаження, парсинг даних, зберігання баз даних тощо). Цим має займатися модель

Найкраще використовувати у вашому проекті 3 групи (папки) з назвою Model, View та Controller. Так ви наочно знатимете, куди відправляти і для чого потрібен конкретний файл з об'єктом.

Контролер займається виключно обробкою вашого view, відштовхуючись від логіки відповідної моделі. Не нагромаджуйте його завданнями, які можуть виконати модель.

Якщо все зроблено правильно, то у вас ніколи не буде проблем зі зміною коду в моделі (нічого з view або controller не зламається). Можна розширювати функціонал і тестувати його, т.к. кожен тип об'єкта незалежний друг від друга.

Передача стану репрезентації (REST) — це архітектурний стиль програмного забезпечення, який був створений для керівництва проектуванням і розробкою архітектури для всесвітньої павутини. REST визначає набір обмежень для того, як повинна вести себе архітектура розподіленої гіпермедійної системи масштабу Інтернет, наприклад Web. Архітектурний стиль REST підкреслює масштабованість взаємодії між компонентами, уніфіковані інтерфейси, незалежне розгортання компонентів і створення багатошарової архітектури, щоб полегшити кешування компонентів, зменшити затримку, сприйняту користувачем, забезпечити безпеку та інкапсулювати застарілі системи.

2.3 Основні характеристики REST API

REST пропонує створити об'єкт даних, запитаних клієнтом, і надіслати значення об'єкта у відповідь користувачеві. Наприклад, якщо користувач запитує

фільм у Бангалорі в певному місці та в певний час, ви можете створити об’єкт на стороні сервера.

Отже, тут у вас є об’єкт, і ви надсилаєте стан об’єкта. Ось чому REST відомий як репрезентаційна передача стану.

Якщо мені потрібно визначити REST, то, передача стану репрезентації, відома як REST, — це архітектурний стиль, а також підхід до комунікаційних цілей, який часто використовується при розробці різних веб-сервісів.

Архітектурний стиль REST допомагає використовувати меншу пропускну здатність, щоб зробити додаток більш придатним для Інтернету. Його часто вважають «мовою Інтернету» і повністю базується на ресурсах.

Щоб краще зрозуміти, давайте зануримося трохи глибше і подивимося, як саме працює REST API. В основному, REST API розбиває транзакцію, щоб створити невеликі модулі. Тепер кожен із цих модулів використовується для вирішення певної частини транзакції. Цей підхід забезпечує більшу гнучкість, але вимагає багато зусиль для створення з самого нуля.

Отже, тепер, коли ви знаєте, що таке REST API, давайте далі розберемося з обмеженнями або принципами, які повинні бути задоволені, щоб програма розглядалася як REST API. Що ж, існує шість основних принципів, закладених доктором Філдінгом, який визначив дизайн REST API у 2000 році. Нижче наведено шість керівних принципів REST. Запити, надіслані від клієнта до сервера, будуть містити всю необхідну інформацію, щоб сервер зрозумів запити, надіслані від клієнта. Це може бути частиною URL-адреси, параметрами рядка запиту, тілом або навіть заголовками. URL-адреса використовується для однозначної ідентифікації ресурсу, а тіло містить стан запитуваного ресурсу. Після того, як сервер обробить запит, відповідь відправляється клієнту через тіло, статус або заголовки.

Архітектура клієнт-сервер забезпечує єдиний інтерфейс і відокремлює клієнтів від серверів. Це покращує переносимість на кількох платформах, а також масштабованість компонентів сервера. Щоб отримати уніфікацію в усій програмі, REST має такі чотири обмеження інтерфейсу:

- Ідентифікація ресурсу;
- Маніпулювання ресурсами за допомогою уявлень;
- Самоописні повідомлення;
- Гіпермедіа як двигун стану програми;

Щоб забезпечити кращу продуктивність, програми часто кешують. Це робиться шляхом позначення відповіді від сервера як кешується або не кешується або неявно, або явно. Якщо відповідь визначена як кешована, то кеш клієнта може повторно використовувати дані відповіді для еквівалентних відповідей у майбутньому.

Багатошарова архітектура системи дозволяє додатку бути більш стабільним, обмежуючи поведінку компонентів. Цей тип архітектури допомагає підвищити безпеку програми, оскільки компоненти кожного рівня не можуть взаємодіяти за межами наступного безпосереднього рівня, на якому вони знаходяться. Крім того, він дозволяє балансувати навантаження та надає спільні кеші для підвищення масштабованості.

Це необов'язкове обмеження і використовується найменше. Це дозволяє завантажувати клієнтський код або аплети та використовувати їх у програмі. По суті, це спрощує клієнтів, створюючи розумну програму, яка не покладається на власну структуру коду. Усі ми, що працюємо з технологіями Інтернету, виконуємо операції CRUD. Коли я кажу операції CRUD, я маю на увазі, що ми створюємо ресурс, читаємо ресурс, оновлюємо ресурс і видаляємо ресурс.

REST використовується в індустрії програмного забезпечення і є широко прийнятим набором рекомендацій щодо створення надійних веб-API без стану.

Веб-API, що підкоряється обмеженням REST, неофіційно описується як RESTful. Веб-API RESTful зазвичай базуються на методах HTTP для доступу до ресурсів через параметри, закодовані URL-адресою, і використання JSON або XML для передачі даних.

«Веб-ресурси» вперше були визначені у всесвітній мережі як документи або файли, ідентифіковані за їх URL-адресами. Сьогодні це визначення є набагато більш загальним і абстрактним і включає кожну річ, сутність або дію, яку можна ідентифікувати, назвати, адресувати, обробити або виконати будь-яким чином в Інтернеті. У веб-сервісі RESTful запити до URI ресурсу викликають відповідь із корисним навантаженням, відформатованим у HTML, XML, JSON або іншому форматі. Наприклад, відповідь може підтвердити, що стан ресурсу було змінено. Відповідь також може включати гіпертекстові посилання на пов'язані ресурси. Найпоширенішим протоколом для цих запитів і відповідей є HTTP. Він забезпечує такі операції (методи HTTP), як GET, POST, PUT і DELETE.[2] Використовуючи протокол без стану та стандартні операції, системи RESTful націлені на швидку продуктивність, надійність і здатність розвиватися за рахунок повторного використання компонентів, якими можна керувати та оновлювати їх, не впливаючи на систему в цілому, навіть коли вона працює.

Метою REST є підвищення продуктивності, масштабованості, простоти, модифікації, видимості, портативності та надійності. Це досягається завдяки дотриманню принципів REST, таких як архітектура клієнт-сервер, відсутність стану, кешування, використання багатошарової системи, підтримка коду на вимогу та використання єдиного інтерфейсу. Цих принципів необхідно дотримуватися, щоб система була класифікована як RESTful.

Архітектурний стиль REST розроблено для мережевих програм, зокрема програм клієнт-сервер. Але більше того, він розроблений для використання в Інтернеті, тому зв'язок між користувацьким агентом (клієнтом) і вихідним сервером має бути якомога легше (вільним), щоб полегшити широкомасштабне впровадження. Це

досягається шляхом створення рівня абстракції на сервері шляхом визначення ресурсів, які інкапсулюють сутності (наприклад, файли) на сервері і таким чином приховують основні деталі реалізації (файловий сервер, база даних тощо). Але визначення є ще більш загальним: будь-яка інформація, яку можна назвати, може бути ресурсом: зображенням, запитом до бази даних, тимчасовим сервісом (наприклад, «погода сьогодні в Лондоні») або навіть колекцією інших ресурсів. Цей підхід забезпечує максимальну взаємодію між клієнтами та серверами в довготривалому середовищі Інтернету, яке перетинає організаційні (довірчі) кордони.

2.4 Структура взаємодії REST між клієнтом та сервером

Клієнти можуть отримати доступ до ресурсів лише за допомогою URI. Іншими словами, клієнт запитує ресурс за допомогою URI, а сервер відповідає представленням ресурсу. Представлення ресурсу є ще одним важливим поняттям в REST; щоб гарантувати, що відповіді можуть бути інтерпретовані якомога більшою кількістю клієнтських програм, представлення ресурсу надсилається у гіпертекстовому форматі. Таким чином, ресурсом маніпулюють за допомогою гіпертекстових уявлень, які передаються в повідомленнях між клієнтами і серверами.

Потужне відокремлення клієнта від сервера разом із текстовою передачею інформації за допомогою єдиного протоколу адресації забезпечили основу для задоволення вимог Інтернету: надійність (анархічна масштабованість), незалежне розгортання компонентів, передача великої кількості даних та низький бар’єр для читачів контенту, авторів контенту та розробників.

Архітектурні властивості

Обмеження архітектурного стилю REST впливають на такі архітектурні властивості:

- продуктивність у взаємодії компонентів, яка може бути домінуючим фактором продуктивності, сприйнятої користувачем, та ефективності мережі;
- масштабованість, що дозволяє підтримувати велику кількість компонентів і взаємодій між компонентами;
- простота єдиного інтерфейсу;
- можливість модифікації компонентів для задоволення мінливих потреб (навіть під час роботи програми);
- видимість зв'язку між компонентами сервісними агентами переносимість компонентів шляхом переміщення програмного коду з даними; надійність у стійкості до збоїв на системному рівні за наявності збоїв у компонентах, роз'ємах або даних;

2. СТВОРЕННЯ ІНСТРУМЕНТУ МЕТОДКИ ПРОТОТИПУВАННЯ

3.1. Загальна архітектура прототипу.

В створеному методу прототипування, його процес відбувається не тільки, як візуалізація потоку даних, елементів системи, які використовуються в проекті, а й програмне ПЗ, яке можливо запустити. Оскільки даний підхід не є унікальний було прийнято рішення зробити прототип не тільки з вище описаними параметрами. Даний новий параметр формує собою те, що з даної системи є можливість отримати її вихідний код. Це значить, пройшовши шлях від візуального поєднання логічних блоків, які передають, зберігають чи обробляють дані, що вона зберігає, до її запуску, для тестування та отримання досвіду відпрацювання установленої логіки.

Кінцевий крок, як було описано вище формує згенерований код, який в подальшому ми можемо модифікувати для додавання більш тонких слоїв реалізації функціональностей проекту.

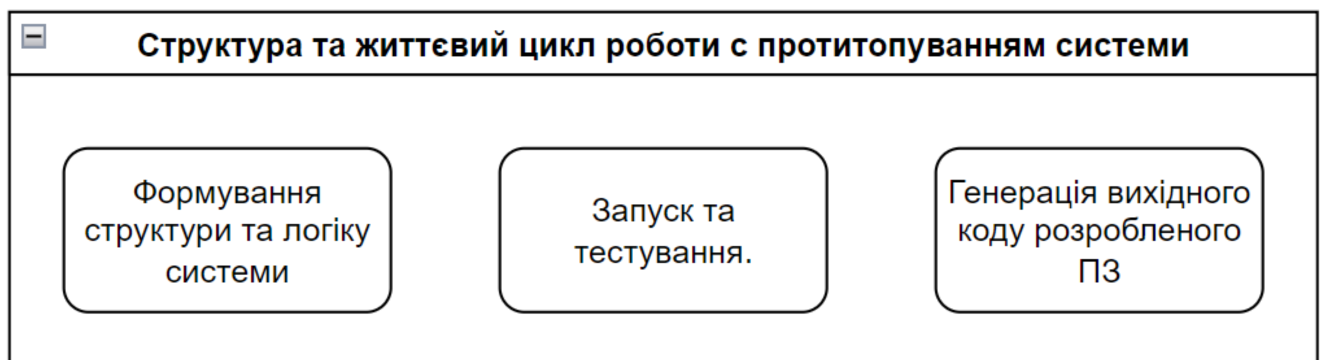


Рисунок 3.1 Цикл основних подій в системі прототипування.

Формування логіки, структури відбувається за допомогою блок схем. Вони мають свою назву (Model, Service, Controller, Router, Condition, Method), яка відображає її сутність та призначення в системі. Ці інструменти будують в сервісі логічні ланцюги , що генерують код який дає можливість для подальшого запуску на віддаленому сервері, на ньому генерується код, який і формує собою прототип.

Дана система дає можливість відпрацьовувати RESTful правила.

Вона також слідує правилам REST. А саме виконати запит до БД в сервері для отримання, видалення, оновлення, створення даних, які зберігаються в базі даних.

Також є прямий доступ до збережених даних. Ми можемо створювати її таблиці, над якими є абстракція у вигляді моделі, що за своєю структурою складається з назви таблиці, їх полів. Поле бази даних - це стовпець таблиці, що містить значення певної якості.

Рядки таблиці є записами про об'єкт; ці записи розбиті на поля стовпцями таблиці, тому кожен запис є набором значень, які у полях.

- **Name** – відповідає назві поля колонки таблиці;
- **Primary key** - поле, яке є ядентифікатором для кожного окремого запису в таблиці бази даних (далі – БД). Первинний ключ обов'язково повинен бути NOT NULL;
- **Autoincrement** - функція, що працює з числовими типами даних. Він автоматично генерує послідовні числові значення кожного разу, коли запис вставляється в таблицю поля, визначеного як автоінкремент;
- **Type** - Типи даних SQL визначають тип значення, яке може зберігатися в стовпці таблиці. Основні типи які можуть бути (Numeric, Date/Time, Character/String, Unicode Character/ String, Binary, Miscellneous);
- **Allow null** – дозволяє нульові значення в цьому стовпці;
- **Not null -** забороняє ненульові значення в цьому стовпці;

```json
{
    "source-code": {
        "model": {
            "name": "User",
            "fields": {
                "id": {
                    "name": "id",
                    "primaryKey": true,
                    "type": "INTEGER",
                    "autoIncrement": true,
                    "allowNull": false
                },
                "username": {
                    "type": "STRING",
                    "allowNull": false,
                    "unique": true
                },
                "token": {
                    "type": "STRING",
                    "allowNull": true
                },
                "password": {
                    "type": "STRING",
                    "allowNull": false
                }
            },
            "table-params": {
                "timestamps": true,
                "paranoid": true,
                "underscored": true,
                "freezeTableName": true,
                "tableName": "'users'"
            }
        }
    }
}
```

Рисунок 3.2. Структури робочої моделі

Сама ж модель складається з набору таких полів, що має структуру.

Назва моделі потрібна, що би звертатися до неї, як до об'єкту в коді. В неї вкладені всі прописані поля, вони не мають обмеження по кількості. На сонові ціїє конфігурації генерується SQL код який запускається та генерує таблицю. Даний підхід робить зручне звернення користувача, що автоматизую роботу з виконуваним кодом бази даних.

Як мінімум перших чотирьох пунктів для створення повноцінної системи, з якою можливо взаємодіяти з інформацією. Ще одним головним пунктом до цієї БД те, що для всіх інших небхідний окремий виділений сервер, а SQLlite доступний як

файл його можна копіювати, редагувати назву та зовсім видалити із операційної системи, що робить його використання досить гнучким для прототипування подібних систем.

3.2 Структура серверу

Сервер має всього одну таблицю Projects в базі даних, яка зберігає дані про прототип, а саме назву, згенерований Id, дата створення та дата оновлення поля. Сам процес творення одиниці прототипу являє собою копіювання бшаблону в директорію, назва якої, відповідає ідентифікатору проетку. Оскільки така назва буде, гарантовано унікальна, то це зберігає нас від конфлікту, якщо б його найменування було б сформоване по назві створеного прототипу.

Вони розділені за призначення. Основні їх цілі це робота з файловою системою сервера, адже там зберігається конфігурація прототипів та їх згенерований вихідний код. Для створення конфігурацій систем було розроблену базовий шаблон. В ньому існує декілька типів файлів, статичні – вони відповідають своїй назві, адже не будуть змінюватися системою прототипування, користувач не матиме доступу для їх редагування до поки не згенерую його, та сам не змінить код. Частично-статичні, в них змінюються деякі параметри, проте тільки під час створення, при використанні в майбутньому при створених параметрах, його параметри змінюватися не будуть. Динамічні, вся основна логіка програмного забезпечення буде зберігатися в ній. При потребі видалити систему прототототипу стирається відповідна папка. Вся подібна функціональність по створенняю шаблону, виділена та покрита шаром абстрації, яка віділяє прямий доступ до методів, які імплементують роботі з файлами, а тому при зміні її реалізації, зникає залежність та при потребі застосувати в середені який інший модуль, не потрібно буде змінювати код, який використовуєтья в інших модулях.

ProjectDBService

+ db: DataBase

+ getAlltables()
+ getAllDataFromTables()
+ deleteTable()
+ getDataTable()
+ getAllTableNames()
+ insertDataToTable()
+ createTable()
+ deleteDataFromTable()
+ resetTableData()
+ #rowToSqlPramsAND()
+ #getColValue()
+ #getColumn()
+ closeDB()

Рисунок 3.3. Комплекс методів для роботи з базою даних

Даний клас реалізовує повний доступ до всіх даних, які зберігає прототип, та є можливість повністю маніпулювати з нею, та надавати всю її інформацію на клієнт. Він слугує в основному для редагування на стороні клієнті та надісталати оновлену інформацію на сервер що б зафіксувати зміни. Всі фони будуть актуальні і в прототипі. Важливо за уважити, що дані створеної системи пров’язані до одного файлу, якийє зв’язковим елементом двосторонньої ланки, яка з’єднує прототип в процесі його створення так і в стані запуску, як окрему одиницю.

Для системи прототипування було принципове питання по взаємодії з базою даних. Оскільки доступ до неї необхідний, як в процесі прототипуваня так і в її запску, до того ж всі дані її маю бути ідентичні на всіх етапах. Дану задачу виконує окремий класс. Він отримує в свій конструктор шлях до файлу з БД. Далі, він за допомогою драйверу підключається до документу з розширенням .sqlite в якому всі дані по проекту. В класі контекст містить тіло базиданих та методи, як приватні які мають доступ тільки в середині модуля так і публічні, що надають доступ зовні в інших сутностях.

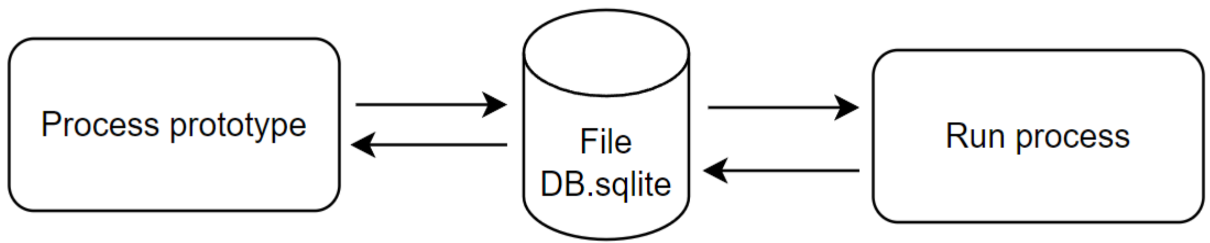


Рисунок 3.4. Зв'язність БД в процесі прототипування та запущеного прототипу.

Структура діректорії шаблону для прототипу з вкладеними файлами сутностей, що його формують.

```
Project/
├── src/
│   ├── config/
│   │   ├── config.json
│   │   └── constants.json
│   ├── controllers/
│   │   ├── nameController.json
│   │   └── name2Controller.json
│   ├── db/
│   │   ├── config.json
│   │   └── index.json
│   ├── models/
│   │   └── nameModel.json
│   ├── models/
│   │   └── nameModel.json
│   ├── routes/
│   │   └── nameRoute.json
│   ├── services/
│   │   └── nameService.json
│   ├── utils/
│   │   ├── Util.json
│   │   └── Logger.json
│   ├── db.sqlite
    └── app.json
```

Рисунок 3.5. Файлова структура конфігурації прототипу

Призначення директорії конфігурацій

- /config - змінні середовища та конфігурація;
- /controllers - містить параметри контролеру;
- /db - параметри бази даних;
- /models - моделі бази даних;
- /routes - маршруту для кінцевих;
- /services - допоміжні методи побудови логіки;
- /utils - утилітарні класи;
- db.sqlite - збрежене інформація в базі даних;
- app.json - точка входу в додаток;

Кожна директорія та її назва відповідає назві сутоності прототипу, що полегшує та покращує навігацію де заходиться необхідна структура, яку потрібно змінити чи продивитися.

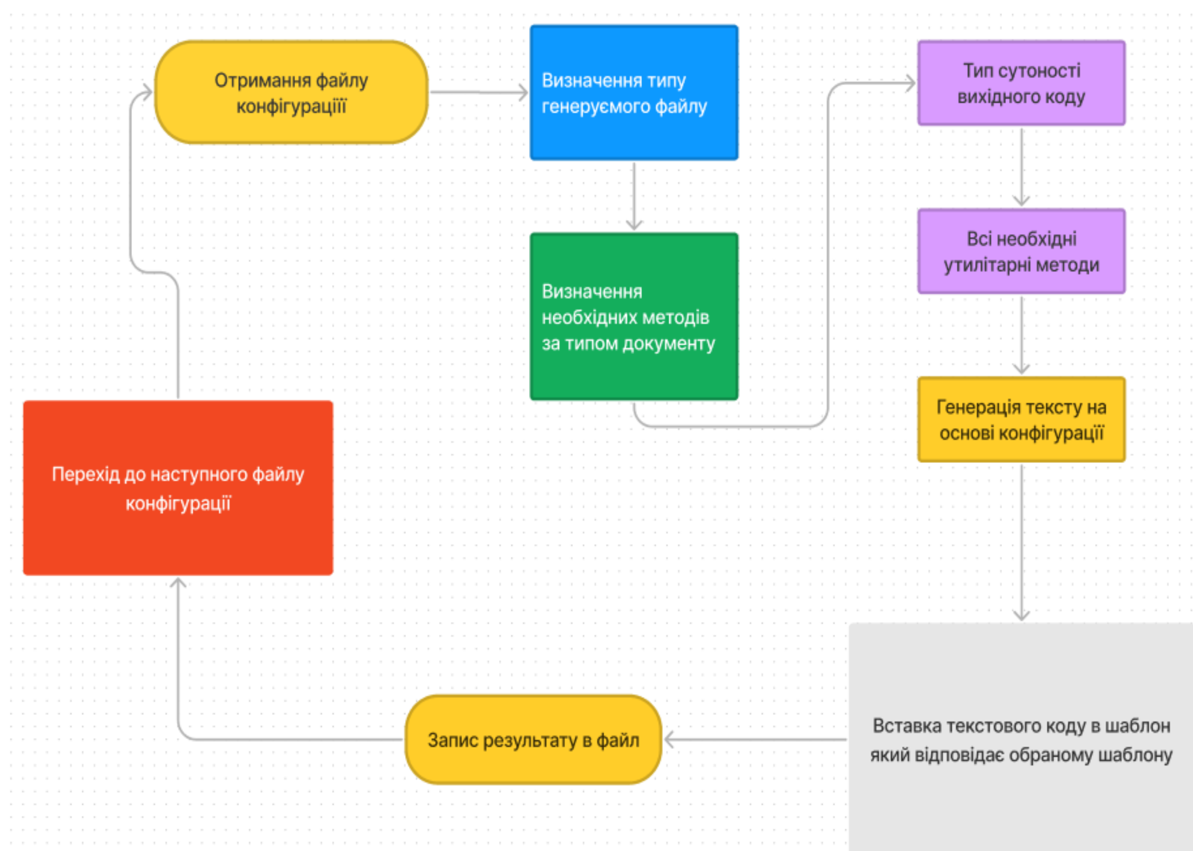


Рисунок 3.6. Алгоритм генераціїї вихідного коду

В данному прикладі зображено прохід циклу роботи скрипту, який на сонові о'бєктів, які зберігають в собі визначення сутностей, його типів та підтипів, за якими в свою чергу визначаються методи, що будуть викликатися для створення JavaScript код. Починаючи від запису умовних операторів які мають свій набір логіки.

Один із перших файлів в директорії та важливості це conditions.js. Він потрібен для генерації логічних операцій. Коли в системі є не обхідність генерувати віконання умов за установленими параметрами. Даний модуль відпрацьовує коли генератор доходить до JOSN об'єкту ‘condition’. Він отримує обо'язковий список параметрів (без яких виконання умов не можливк) ‘params’ в ньому вкладені об'єкти

Окрім параматрів умови має бути частина коду яка виконується, якщо вона його задовільняє. Дану функцію виконує парметр “isTrue”. Та “isFalse” коли параметри не задовільняють початкові умови, дана конструкція є не критично та не великою необхідністю оскільки достаньо частини яка виконує тільки позитивний сценарій виконання.

Кожен прототип має свою глобально конфігурацію та JSON файл з установленими пакетами, які не обхідні що б npm (node package manager) розумів котрі пакети потрібно встановлювати в ту чи іншу директорію (в сновному це деректорія де знаходиться package.json).

В системі прототипізації всі залежності зберігаються в окремому файлі. Вони предаються, як параметр для створення файлу по заданому шляху. Ще один файл конфігурації - .env він містить глобальні змінні котрі доступні в будь-якій частині системи. В даному випадку там з берігається номер порту лоакльногу хосту під яким запускався прототип. Номер PORTу відповідав id.

Створення бази даних реалізовується в декілька кроків, береться абсолютний шлях та копіюється по ньому файл db.sqlite.

Ще одним важливим елементом генерації коду э запис в файл імплементаці методів котрі містять в собі процедури коду.

В прототипу виикористовуються, як самописні елемнти коду, а й зовнішні підключені модулі які містять власні інкапсульовані фаргменти логіки. Але вони не будуть глобально доступні, що б мати можливість їх використорвути треба імпортувати. Глобальні пакети складаються з назви змінної котра по суті формує посилання на нього та назву модулю, а локаьний окрім назви, локальний шлях до файлу.

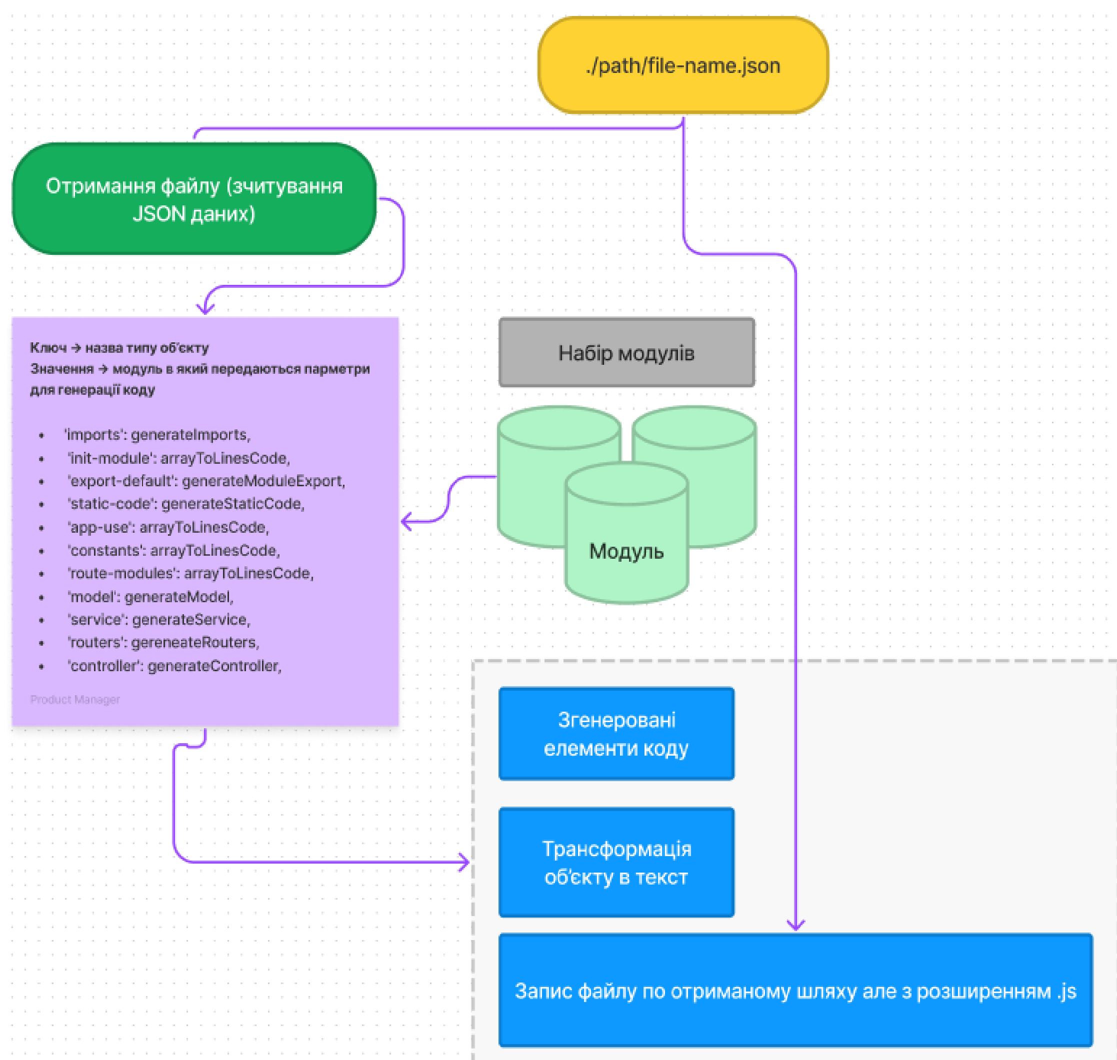


Рисунок 3.7 Структура з підключенням модулю в залежності від типу структури коду

Всі вище описані модулі описані для файлів крнкретного призначення, та для гнучкості ситеми, дані сутності треба розділяти. Для досягнення даного результату було застосовано прийом The Single Responsibility Principle (SRP).

Принцип єдиної відповідальності (SRP) стверджує, що клас чи модуль повинен мати лише одну причину для змін. У цій формі його вперше процитував Роберт К. Мартін у статті, яка пізніше стала розділом у його книзі «Принципи, шаблони та практики гнучкої розробки програмного забезпечення».

Принцип єдиної відповідальності тісно пов'язаний з концепціями зв'язку та згуртованості. Зв’язок означає, наскільки нерозривно пов’язані різні аспекти програми, тоді як згуртованість стосується того, наскільки тісно пов’язаний вміст конкретного класу чи пакета. Весь вміст одного класу тісно пов’язаний один з одним, оскільки сам клас є єдиною одиницею, яку потрібно використовувати повністю або не використовувати взагалі (за винятком статичних методів і даних на даний момент). Коли інші класи використовують певний клас і цей клас змінюється, залежні класи повинні бути перевірені, щоб переконатися, що вони продовжують правильно функціонувати з новою поведінкою класу. Якщо клас має погану згуртованість, деяка його частина може змінитися, що використовують лише певні залежні класи, тоді як інша частина може залишитися незмінною. Тим не менш, усі класи, які залежать від класу, мають бути повторно перевірені в результаті зміни, збільшуючи загальну площу поверхні програми, на яку вплинула зміна. Якби замість цього клас був розбитий на кілька, дуже згуртованих класів, кожен з них використовувався б меншою кількістю інших елементів системи, і тому зміна будь-якого з них мала б менший вплив на всю систему. Якщо э можливість придумати більше ніж одну мотивацію для зміни класу, він, ймовірно, має більше ніж одну відповідальність.

Кожен модуль по генераці коду не залежить один від одного, тому при зміні якогось елементу, інші частни коду будуть працювати, як і до змін, що підвищує модульність структури проекту. Єдина залежності котра пристня – файл

конфігупаціїі, яикй модуль отримує та його утилитарні функції проте вони являють собою компонети багаторазового використання.

Всі операції по створенню файлів прототипу для запуску, як окремий проект, генерються на сервері. Для виклику серверних подій з браузера є розроблений клієнт, який робить запит на певні дії з переданими параметрами та отримує результат по закінченню виконання єндпоінту.

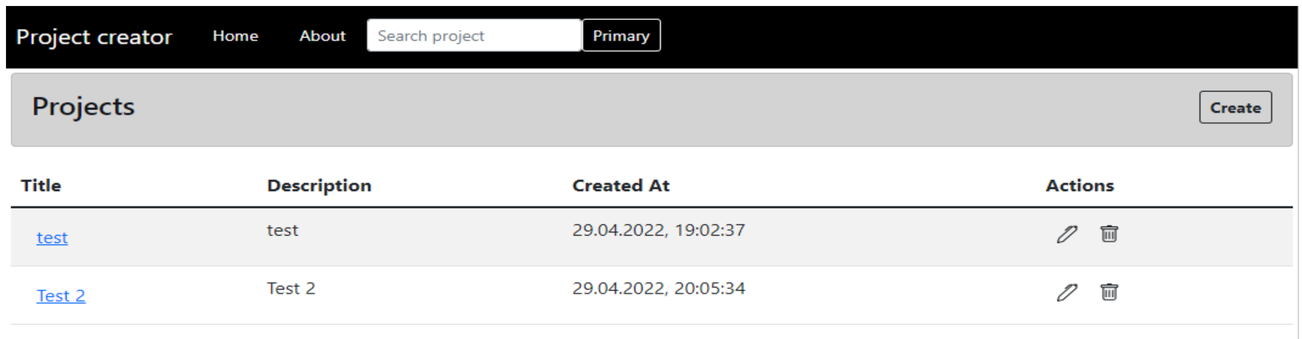


Рисунок 3.8. Початкова сторінка додатку зі списком проектів.

Є форма для стводтення прототипу в якому всього 2 поля: назва, опис, всі інші додаткові дані автоматично додаються разом з інформацією полей в базу даних

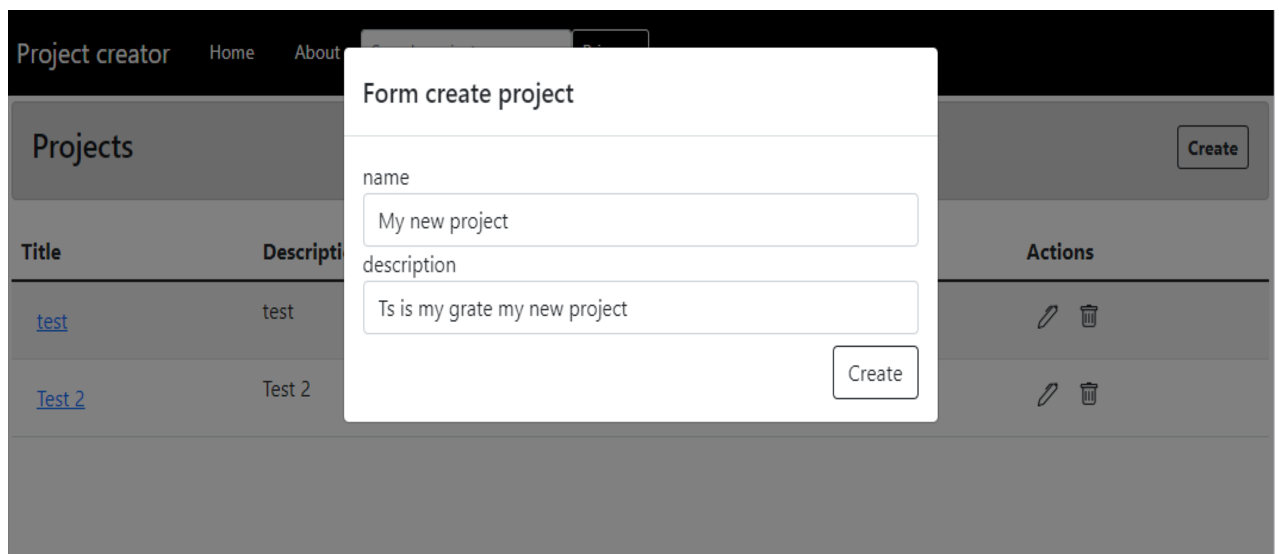


Рисунок 3.9. Форма створення проекту (прототипу)

Project creator Home About Search project Primary

Projects Create

Title	Description	Created At	Actions
My new project	Ts is my grate my new project	20.05.2022, 18:36:54	
test	test	29.04.2022, 19:02:37	
Test 2	Test 2	29.04.2022, 20:05:34	

Рисунок 3.10 Список проектів після створення прототипу.

В списку заявок, назва проекту ще і посилання на його детальну сторінку.

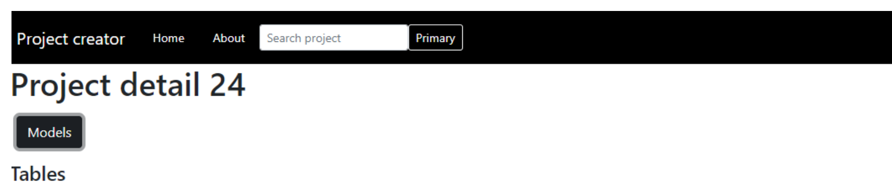


Рисунок 3.11. Детальна сторінка прототипу

Детальна сторінка проекту Після творення, вона має такий вигляд, можна побачити, що ніякої інформації не має. Проте є кнопка для створення моделі.

За допомого неї можливо створити стрктуру моделі, яка після створення форматує дані в SQL код, та створить таблицю з відповідною структурою.

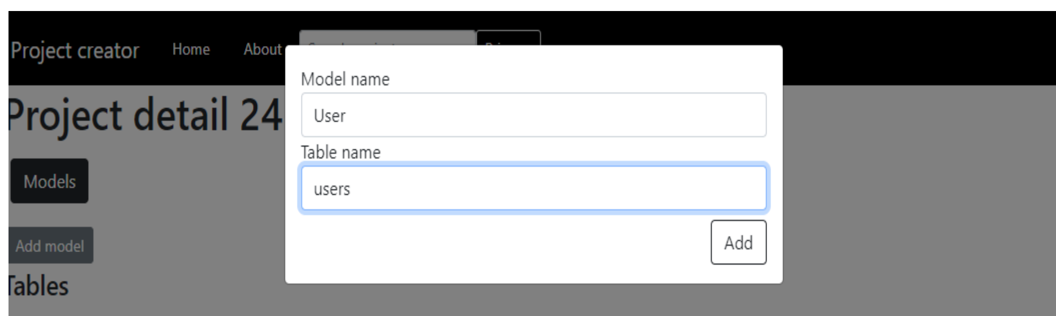


Рисунок 3.12. Форма створення моделі

ModelUser

table users

Name	Primary Key	Type	Auto Increment	Allow Null
name	☐	STRING		☐
second_name	☐	STRING		☑
id	☑	INTEGER	☐	☐

Рисунок 3.13. таблиця моделі

Дана таблиця створення, що б заповнювати параметри моделі, які користувач хоче додати. В ній є поля з назвою, яка приймає в себе тільки текст, primary key, type, який можна обрати зі списку, обрати параметр auto increment та allow null.

Таблиця БД вже створено, проте потрібно ці дані додати, що б мати можливість з ними працювати та отримувати на основі неї результати. Дані додаються відповідно до вказаних типів даних.

Оскільки модель сформована, таблиця готова для її заповнення. Кристувач може додати інформацю, яку може редагувати та видаляти. Все це необіхдно, що б прототип мав можливість обробляти потоки данах за параметрами які в нього надсилаютсья.

Вся інформація, надсилаєтся на сервер, таким чином вона зберігаєтсья в базіданих обраної таблиці, це дозволяє в падльшому(навіть після оновлення веб-додатку на стороні браузеру їх не втратити). Наступним кроком прототипування є візуальна складова, яка за допомогою побудови блоків в фонофому процесі формує на кожну дію користувача json об'єкт, який згенерує повноцінний вихідний код, який можна буде запустити в коремому проекту та модернізувати та добаляти більш “тонку” структуру.

Кожний блок відповідає за свою логіку та його призначенняв системі. Якщо їх зібрати в один логічний ланцюг то вони повернуть результат обробки інформації яка береться з БД.

Візуалізацю блоків та їх потік даних, починаючи з моделі та закінчуючи формуванням результату на запит за вказаними параметрами та логічними ланцюгами.

По закінченню процесу користувач (чи інша система) поверне отримані дані за вказанми параметрами в фоорматі JSON. Які в подальшому можуть змінюватись та передаватись знову в подібний потік для обробки інформації яле він може мати вже іншу структуру та приймаючі аргументи.

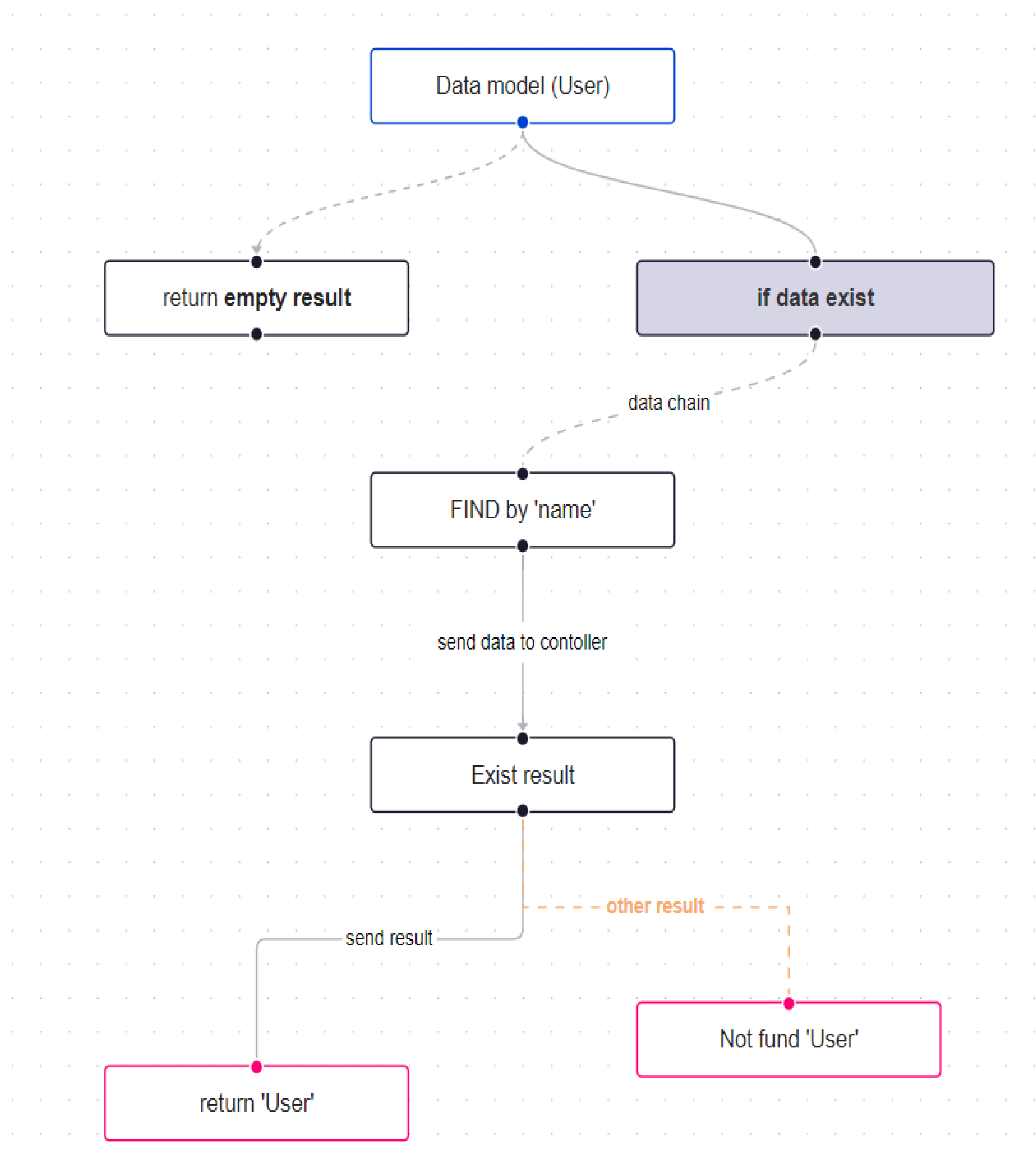


Рисунок 3.14. Побудовані блоки системи

Кожен елемент лянцюга прирймає та передає наступному компоненту сформований набір параметрів який буде записуватись для кожної сутності структури в окремий файл. Відразу після збереження всі параметри конфігураці запишуть в сервер.

Блок який відповідає за модель містить в собі прописані зарання методи. Вони відповідають всіс правилам REST. Проте ми можемо прописати за якими параметрами здійснювати подію ящко методи моделі їх потребують.

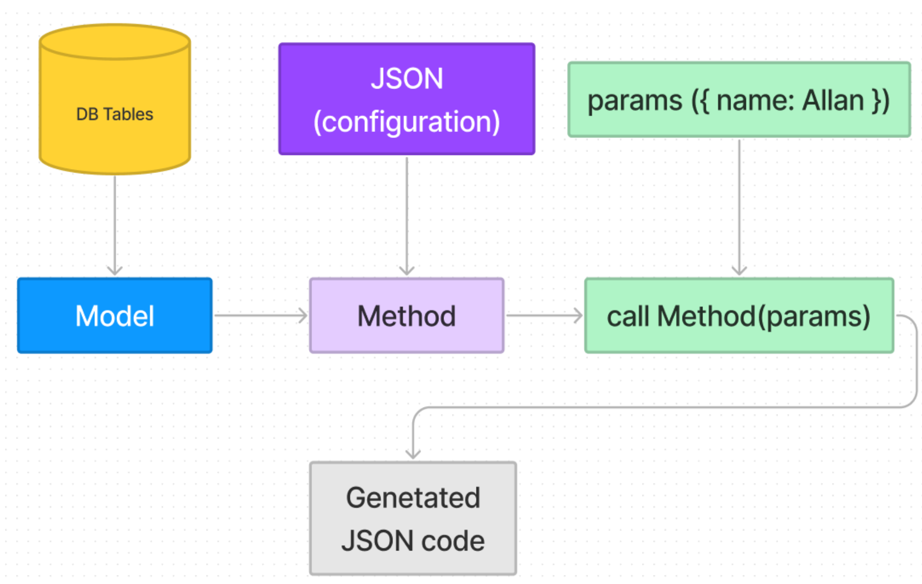


Рисунок 3.15. Структура генерації моделі в побудові потоку даних

Наприклад ми можемо отримати всі записи з таблиці обрашви з об'єкту метод “findAll” чи обрати більш конкретний запит “findAll({ where: { name } })” де name – параметр який ми передаємо при запиті в систему. Таки чином ця обгортка буде за допомогою методів генерувати щапит до базиданих. Вони допомагають уникнути руками прописувати SQL скрипти, яким б складними чи довгими вони не були б.

На кожному етапі виборки параметрів чи вибору методі з об'єкту моделі виконується зміна конфігурації кожно елементу структури також вона відразу надсилає запит на сервер для збереження ланцюгу подій в файл. Наступним кроком виконується процес обробки отриманих даних в сутності Controller.

Структура Contoller відповідає за подальшу обробку чи отримання даних з моделі. Він може викристовувати за джерело запрошених даних, декілька моделей, знову обробити їх та відправити результат в фінальну ланку ланцюга. Також, саме він відповідає за отримані дані з тіла запиту які надсилається за прописаним шляхом.

Таким чином він ініціалізує об'єкт який отримава, передає в модель, як вже відомо вона видає результат за обраним методом(аналогічно запиту в БД). Чи взагалі можна обробити в контролері параметри, виконати зі ними певну дію та відразу (без додавання додаткових комопнетів) повернути дані на запит. Кожен етап формування сутності Contoller супроводжується динамічній зміні його тіла (конфігурації структури). Всі елементи логіки компонету також прописані завчасно на стороні клієнта, проте дані елементи можна кобінувати між собою, що б мати біль гнучкі можливості для взаємодії з системою. Це дозволяє поєднувати різні елементи додаттку та інформацію яку ми в неї передаємо.

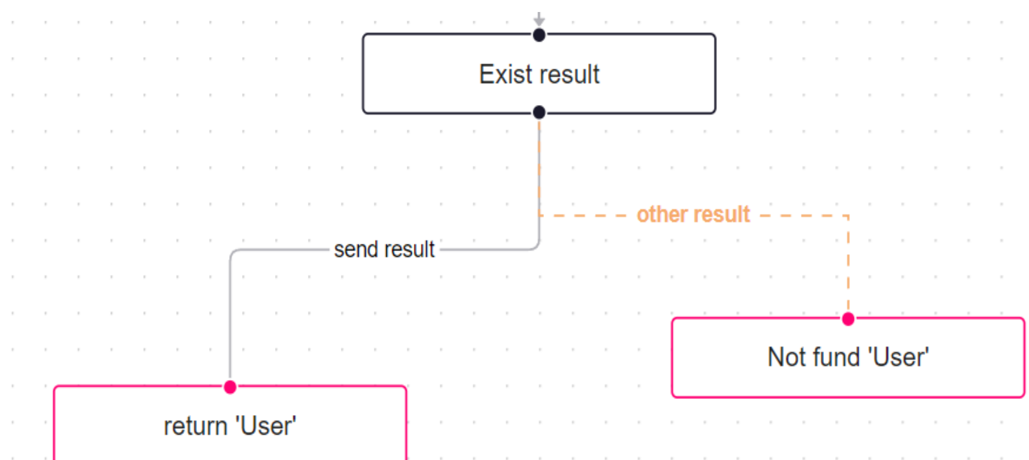


Рисунок 3.16. Приклад блоків розгалудження потоку даних.

Наприклад виконати перевірку, підключивши модуль умови на виконання події за певної умови таким чином, відбудеться розгалудження потоку даних, як візуально так і в вихідному коді який буде згенеровано після формування потоку даних прототипу.

Даний приклад візуалізує аналіз отриманих параметрів та видасть ту чи іншу дію в залежності ві виконання вказанлї умови. Таким чином мийожемо обирати, що ми хочемо знайти чи отримати або взагалі “відфільтрувати” результат отриманий з об'єкту Model, що б мати можливість розширити умови використання інформації в системі.

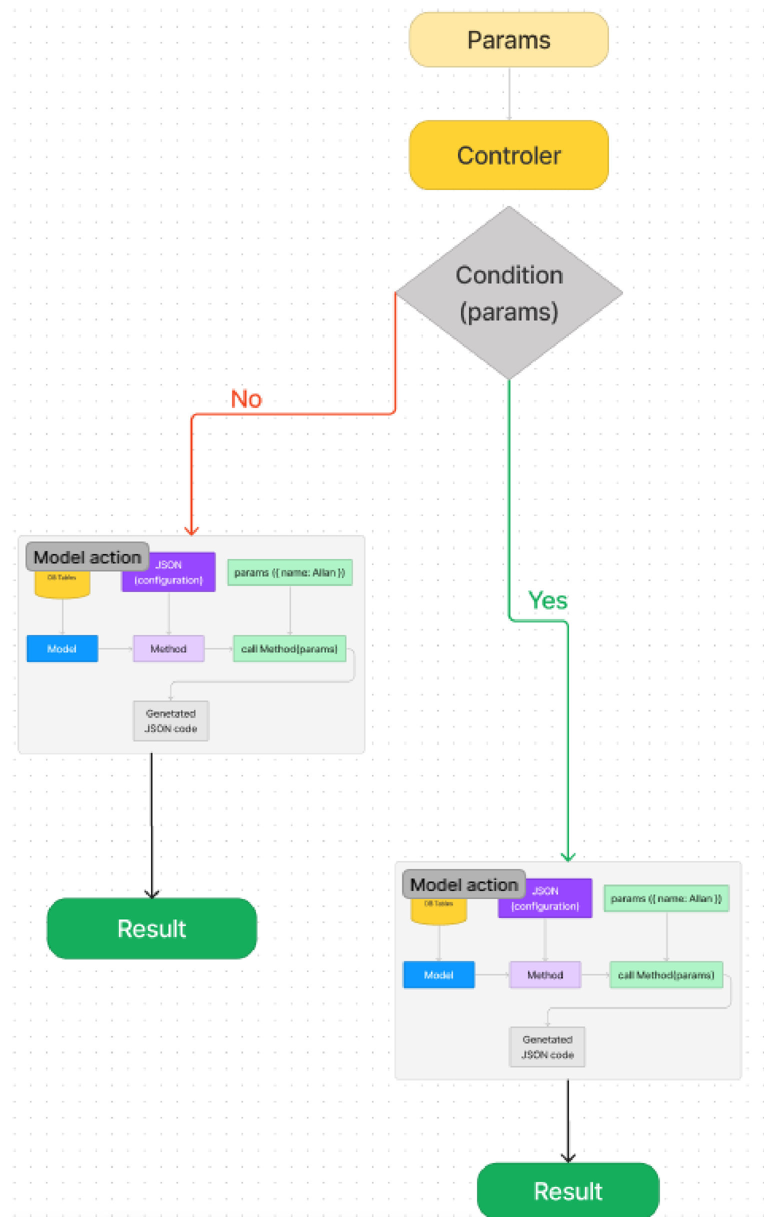


Рисунок 3.17. Блок схема взаємодії з Controller

Над контроллером є обгортка роут – адрес по якому звертаються до системи. Він має бути інікальним, для того, що б не було конфліктів. Перейшовши по прописаному шляху в який входить тип методу (GET, POST, PATCH, DELTE) назва

в яку можуть передаватися аргументи в контроллер. В свою чергу він генерує сформовані дані та подбльшої обробки та парметризаці. Роут – це початкова точка прототипу, який викликається при звернені до нього за адресом, та з нього надсилає результат заапиту. В самі системі прототипування він являє собой JSON об'єкт.

Він зберґає в собі назву (тобто метод за яким виконується звернення), адрес зверненння, контроллер в якому зберігається набори методів логіки роботи частини системи, там назва методу який виконує операцію.

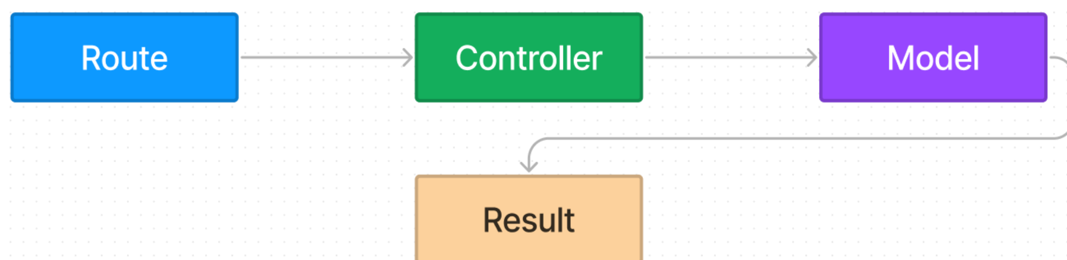


Рисунок 3.18. Циклу глобальних компонентів прототипу

Даний цикл відображає поток та його напрямок в передачі інформації в системі. Суттєвою характеристикою схеми системи прототипування є те, що вона передає відображення багатовимірного розділення співідношення, які сформовані на унікальних опціях артефактів, наприкалад таких як шар абстракції, також формування позицій всередині архітектури, функціональні чи експлуатаційні параметри.

При формувані опису інформаційної системи за допомогою багатьох артефактів інформація, зафіксована одним артефактом, та може бути пов'язана з записаної інформації чи зафіксованою іншим . Наприклад сутність модель яка описує схему таблицю базт даних, яка може бути уточненням до неї, що описує логіку даних. Так само інформація, яка сформована унікальними компонентами, має можливість повторюватися. Наприклад структури об'єктів логіки системи може посилатися використовувати посилання на модулі, що й модель, яка описує правила безпеки

для всіх сутностей прототипу у компексі. Звідси виникає потреба формування відношення між модулями для підтримки узгодженості при оновленн конфігурації.

Відношення між сутностями системи визначають через відношення між точками зору, що описують ці сутності.

Обчислення між відображеннями мають назву трансформація, вони можуть бути абсолютно створені в ручними методами, абсолютно автоматизовані чи напівавтоматичні в залежності від об'єму необідних оновлень. Коли потрібна нескінченна оновленість, трансформація може бути взагалі ручною. Якщо зміни не потрібні, трансформація може відбуватись автоматичними. Базовим прикладом для даної часткової автоматизованої трансформації слугує ручне прототипування інформаційної системи за вимогами, щоб кожен потік інформації, відображається властивість об'єкту, поверненого логікою, і що кожен метод та полі властивостей має відображатисб блоком. В інших сценаріях є можливість зробити перевірку результатів змінити початкові сутності або його параметри трансформації, щоб змінити результати. Деякі оновлення мають змогу бути застосовані безпосередньо над результатами потоку інформації для оптимізування чи налагодження. Деяка частина з них можуть бути інкапсульовані до результатів. Зміни також застосовуються для синхронізації пов'язаних сутностей системи.

Сформована методика прототипізація ІС можу розглядатися, як сценарій для розробки цілого сімейства інформаційних систем. Проте для побудови більш точного ПЗ його слід параметризувати.

Розробка системи формується та обмежується процесом, який визначений в процесі розробки фабрики програмного забеспечення та зміненим за урахуванням характеристик конкретної системи при конфігуруванні схеми фабрики.

Процес може виконуватися згори донизу це залежить від вимоги до виконуваної системної логіки, при цьому зберігаючи зв'язані сутності синхронізованими, наприклад, послідовність «модель, контроллер, оптимізована схема бази даних». Слід зауважити, що необхідно враховувати певні обмеження, сформовані у горизонтальних відображеннях між моделями. Наприклад, інформацію про

середовище збору (доступні протоколи і служби) їх ми можемо застосувати для обмеження дизайну відношення служб

Застосування нової методики прототипування інформаційної системи на практиці.

Дуже часто, розпочинаючи розробку над новим проектом, у дизайнера / менеджера / розробника виникають непорозуміння. Відбувається це через досить розповсюджені нюанси. По перше — помилкове формування часу та бюджету проекту, який не зафіксований у ТЗ. Ще одна поширена проблема: ціна системи, яка може змінюватися при умові імплементації тієї чи іншої функціональності. Щоб уникнути даних проблем, менеджер зазвичай може пропонувати запровадити розробку в етап прототипування. Звичайно, у багатьох людей які керують процесом розробки часто виникають питання на приклад: «Прототипування? А для чого це взагалі потрібно?», «Чому хтось повинен витрачати на це свій час ”?», «Навіщо розтягуватим час який надаэться для написання коду !».

Прототипом можна назвати план системи (сторінки / додатка / інтерфейсу і т.д.), основну увагу в якому формуэться на функціональності, та структурі потоку даних.

Прототипування — це інструмент знаходження необхідних вимог для розробки інформаційної системи.

Основне, що потрібно розуміти — це те, що система може мати не так багато спільного з остаточною системою. Прототип може слугувати просто, як план, та в подальшому на основі нього розробляти остаточний варіант.

За складністю по взаємодії прототипи можна розділити на:

- статичні — вони в своїй суті, відображають звичайне зображення, яке доступне в процесі перегляду та коментування. Це досить не поганий варіант

для не складних систем, на які не має потреб витрачати дуже велкий об'єм часу та розбиратися в функціональності;

- інтерактивні — це клікабельні прототипи, що показують взаємозв'язок між моделями даних та можуть відображати потік даних в анімованому режимі. Така методика застосовується в більш складних системах, що вимагають більш детальних переговорів та дискусій.
- Функціональні – це невелика система в яку включено базові елементи методів, які реалізовують основні вимоги, що вимагають максимально складне ПЗ, яке розраховане на довгострокову розробку та підтримку командою розробників.

4. ЗАСТОСУВАННЯ НОВЇ МЕТОДИК ПРГОТОТИПУВАННЯ В ПРОЦЕСІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз застосування створенної методики прототипування

У підсумку, можна розглянути реальний досвід з яким мав справу: завдання було розробити систему контролю роботи працівників компанії. Наявний сайт не влаштовував керівництво оскільки мав застарілий дизайн системи, тому що конверсія була занадто низька.

В ході обговорення, менеджер пояснив, що паралельно проводиться процес взаємодії з маркетологами та він ще не зміг визначитися остаточно, які важелі підвищення продуктивноті можуть застосуватися. Враховуючи дані чинник роботу потрібно було починати, щоб не було зривів закладених строків виконання. Було запропоновано застосувати нову методику прототипування об'єктів інформаційної системи і поетапно просуватися з таким таймінгом:

- Проектування: 50 годин;
- Дизайн: 80 годин;
- Фронт-енд розробка: 100 годин;
- Бек-енд розробка: 140 годин;

ВСЬОГО: 370 годин

Під час процесу прототипування, був проаналізований весь список функціоналальностей, команда створила та підготувала прототип, надала поислання на ознайомлення всім співробітникам, залученим в проект, і представникам менеджера. Це дало змогу обговорити в реальному часі всю функціональність та витрати на його реалізацію. Узгодження проміжних і кінцевих версій прототипу- зробило більшим витрати часу на прототипи, проте значно зменшило затратина дизайн і подальші правки. Але головним, досягненням було

розуміння всієї команди, що до потреб, обсягу роботи, і рішення приймались більш швидше і продуктивно.

Остаточний час розробки проекту занав певних змін, проте не критичних:

- Проектування: 68 годин;
- Дизайн: 63 години;
- Фронт-енд розробка: 96 годин;
- Бек-енд розробка: 152 години;

ВСЬОГО: 379 годин

Як можна побпчити, різниця в часі для проектування та дизайну склала всього 9 годин. Було зведено до мінімуму всі можливі ризики затрати часу для наступних ітерацій розробки.

Взявши для порівняння подібний проект в якому етап проектування був повністю упущений та робота стартувала відразу з дизайну, тоді можна побачити метрики затрат часу на проекту був такий:

- Дизайн: 100 годин;
- Фронт-енд розробка: 100 годин;
- Бек-енд розробка: 140 годин;

ВСЬОГО: 340 годин + ПРАВКИ

Різниця становить всього 39 годин, проте потрібно урахувати, що необхідно було внести правки — 2 ітерації по 10 годин кожна, тоді до витрат часу необхідно додати ще

- Ітерація 1: Дизайн - 10 годин;
- Ітерація 1: Фронт-енд розробка - 10 годин;
- Ітерація 1: Бек-енд розробка - 10 годин;
- Ітерація 2: Дизайн - 10 годин;
- Ітерація 2: Фронт-енд розробка - 10 годин;

- Ітерація 2: Бек-енд розробка - 10 годин;

ВСЬОГО: 60 годин це вже перевищує витрати часу, та шансів впоратися за декілька правок.

Як показує ввище представлена метрика та досвід, нова методика прототипування об’єктів інформаційної системи є досить зручним та продуктивним інструментом розробки. Він допомагає сформувати межі роботи над подальшими проектами, економить час і бюджет (що важливо під час розробки, наприклад, корпоративних чи державних проектів) зменшує людські ресурси для прийняття рішень, робота стає набагато більш гнучкою і швидкою, а ККД при командній роботі — на порядок вище.

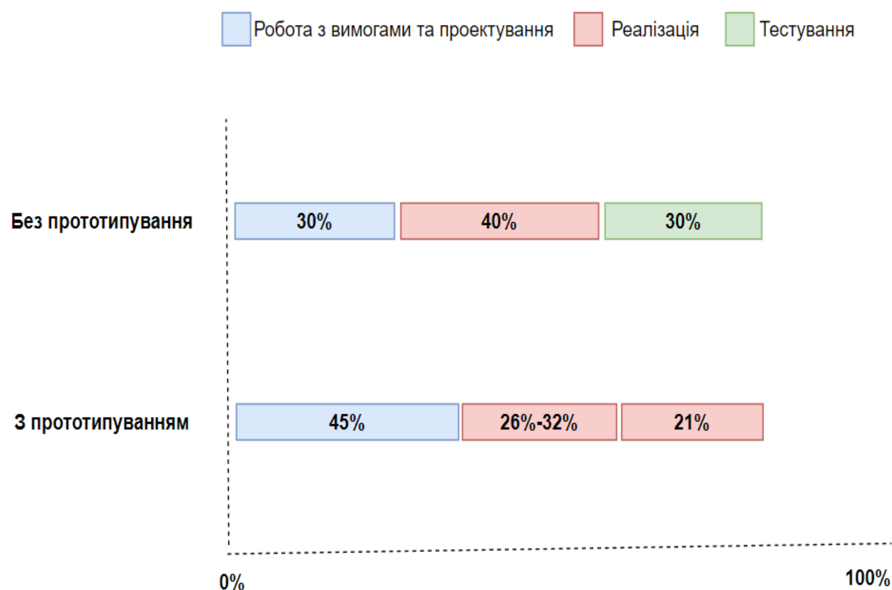


Рисунок 4.1 Підрахунок продуктивності використання прототипування

Результат аналізу створення інформаційних систем з прототопуванням та без нього на більшості проектів.

ВИСНОВКИ

Метою данної дипломної роботи була розробка веб додатку для імплементації на основі методики прототипування об'єктів інформаційних системи. На базі технології JavaScript, NodeJS, TypeScript. В процесі створення роботи було проведено аналаіз та розгляд доступних програмних продуктів, який показує, переважна більшість подібних додатків має значний недолік у вигляді високої вартості використання застосунків, що в свою чергу робить їх користування обмеженим для застосування значного кола користувачів.

Навіть якщо виключити недолік вартості значним, то ще виділити це відстуність дуже вагомих та потрібних наборів функціональностей, які будуть в зкомпоновані в одному застосунку. Також було зібрано дослідницьк дані, які були добуті з досвіду інтеграції системи прототипування, в результаті якого було сформовано метрики результативності застосування даного інструменту розробки інформаційних систем.

Перед початком стоврення даного додатку були розглянуті найбільш відомі та дослджені технології та підходи розробки. Рішення яке відповідало всім вище заданим вимогам були обрані основні технології: JavaScript, NodeJS, TypeScript, Nest. Після вибору стеку технології було розроблено ядро системи, яке генерувало вихыдний код прототипу.. Важливими особливостями інтеграції технологій в створені додатку є оптимізація коду та гнучка розробка адже будь-яка створена програмна система вимагатиме обов`язкового оновлення чи заміни вже існуючих модулей коду які інтегровані в застосунок. Також надано рекомендації щодо подальшого вдосконалення та розширення розробленого застосунку, який можна адаптувати під потреби користувачів для більшого комфорту в роботі з більш не тривіальними задачами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Axure Prototyping Blueprints. John Henry Krahenbuhl, 2017. 58 с.
2. Прототипування в процесі розробки. https://studfile.net/preview/4452595/page:13/.
3. Особливості прототипування об’єктно-орієнтованих програмних систем. http://ekmair.ukma.edu.ua/bitstream/handle/123456789/1918/Domina_Osoblyvosti_prototypuvannia.pdf?sequence=1&isAllowed=y.
4. Автоматизація Прототипування Інформаційних Систем. https://posibniki.com.ua/post-avtomatizaciya-prototipuvannya--informaciynih-sistem.
5. Rapid Prototyping Technology Applications and Bene. https://ru.scribd.com/document/442026453/Rapid-prototyping-technology-Applications-and-bene
6. Effective Prototyping for Software Makers. Jonathan Arnowitz, Michael Arent, Nevin Berger. 2006. 532 с.
7. Reenskaug, Trygve; Coplien, James O. (20 March 2009). "The DCI Architecture: A New Vision of Object-Oriented Programming".
8. Todd Grimm: The Human Condition: A Justification for Rapid Prototyping. Time Compression Technologies, vol. 3 no. 3. Accelerated Technologies, Inc. May 1998 . Page 1..
9. Dewar, Robert B. K.; Fisher Jr., Gerald A.; Schonberg, Edmond; Froelich, Robert; Bryant, Stephen; Goss, Clinton F.; Burke, Michael (November 1980). "The NYU Ada Translator and Interpreter". ACM SIGPLAN Notices – Proceedings of the ACM-SIGPLAN Symposium on the Ada Programming Languag.
10. С. Меліса Макклендон, Ларрі Регот, Геррі Акерс: Аналіз та прототипування ефективних графічних інтерфейсів користувачів. Жовтень 1996 р.
11. Доктор Рамон Акоста, Карла Бернс, Вільям Жепка та Джеймс Сідоран. Застосування методів швидкого прототипування в технічному середовищі вимог. IEEE, 1994.

12. Сміт М.Ф. Прототипування програмного забезпечення: прийняття, практика та управління. McGraw-Hill, Лондон (1991).
13. Роберт Б.К .; Фішер-молодший, Джеральд А .; Шонберг, Едмонд; Фреліх, Роберт; Брайант, Стівен; Госс, Клінтон Ф .; Берк, Майкл (листопад 1980). "Перекладач і перекладач Ада Нью-Йоркського університету". Повідомлення ACM SIGPLAN - Матеріали симпозіуму ACM-SIGPLAN з мови програмування Ada. 15 (11): 194–201.
14. SofTech Inc., Waltham, MA (1983-04-11). "Підсумковий звіт про перевірку компілятора Ada: NYU Ada / ED, версія 19.7 V-001".
15. Лукі; Berzins, Yeh (жовтень 1988). "Мова прототипу для програмного забезпечення в режимі реального часу" (PDF). Транзакції IEEE з програмної інженерії. 1409–1423.
16.Лукі; Кетабчі (березень 1988). "Комп’ютерна система прототипування". Програмне забезпечення IEEE. 5 (2): 66–72
17.Лукі (травень 1989). "Еволюція програмного забезпечення завдяки швидкому прототипуванню".
18.GE Research and Development. https://web.archive.org/web/20061013220422/http://www.crd.ge.com/esl/cgsp/fact_sheet/objorien/index.html.
19.С. П. Овермайер: Революційне проти еволюційного швидкого прототипування: збалансування продуктивності програмного забезпечення та проблем з дизайном HCI. Центр передового управління, управління, зв’язку та розвідки (C3I), Університет Джорджа Мейсона, 4400 University Drive, Ферфакс, Вірджинія.

ДОДАТКИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

«Розробка методики прототипування об'єктів інформаційної системи на базі технології JavaScript, Node.js»

Виконав: студент 7 курсу, групи ППЗМ–71
Печериця Віталій Валентинович
Керівник роботи: Бондарчук А.П.

Київ – 2022

Об`єкт, Предмет та Мета Дослідження

Об`єкт дослідження – процес прототипування об'єктів інформаційних систем.

Предмет дослідження – інстремунти прототипування інформаційних систем.

Мета – розробка методики прототипування інформаційних систем на базі автогенерації коду для зменшення часу розробки.

Порівняльна характеристика аналогів

Переваги\ Недоліки	Axure RP Pro	iRise	Sketch
Переваги	• Стандартна структура системи, яка подібна до Microsoft Office, Microsoft Visio, що зменшує затрати на ознайомлення з програмним забезпеченням. • Інтерактивність створеного прототипу(що поліпшує користувацький досвід).	• Широкий набір функціональностей • Можливість інтеграції з різними системами (jama, Jira, Azure DevOps).	• Повний автономний процес • Наявність інструментів експорту графіку • Інтеграція з нативними пристроями
Недоліки	• Застосування обмежене тільки створенням візуального інтерфейсу. • У системі не передбачено роботу кількох користувачів з одним проектом на сервері. • Не має автогенерації коду	• Система по якій мало інформаційних публікацій чи матеріалів • Дуже дорога система. • Не має автогенерації коду	Можна установити тільки на macOS Виникають проблеми з файлами великих розмірів

Схема вибору способу прототипування

Архітектурний прототип – використовується для демонстрації можливостей створюємої системи.

Прототипу вимог - це часткова реалізація програмного забезпечення, створена з метою допомогти розробникам, користувачам краще зрозуміти вимоги до системи.

Одноразовий - передбачається, що створюється макет, який на якомусь етапі залишиться («викинутий») і не стане частиною готової системи.

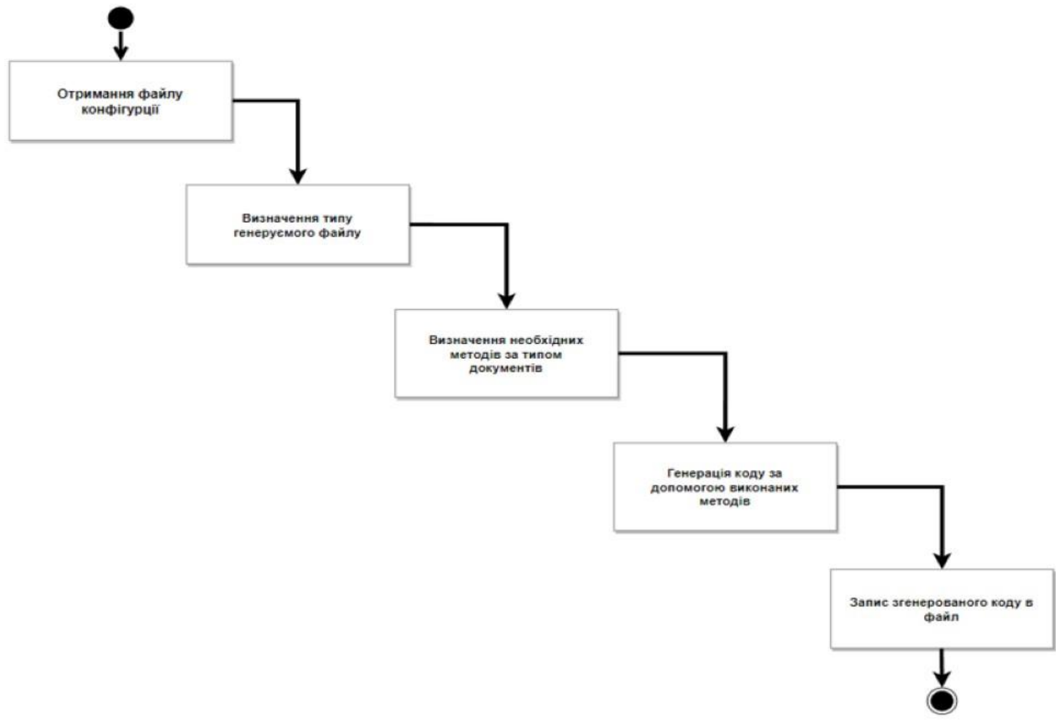
Еволюційне прототипування - ставить за мету послідовно створювати макети системи, які будуть все ближче і ближче до реального ПЗ.

Вертикальні прототипи - втілює зріз функціональності застосування (додатку) від інтерфейсу користувача до сервісних функцій.

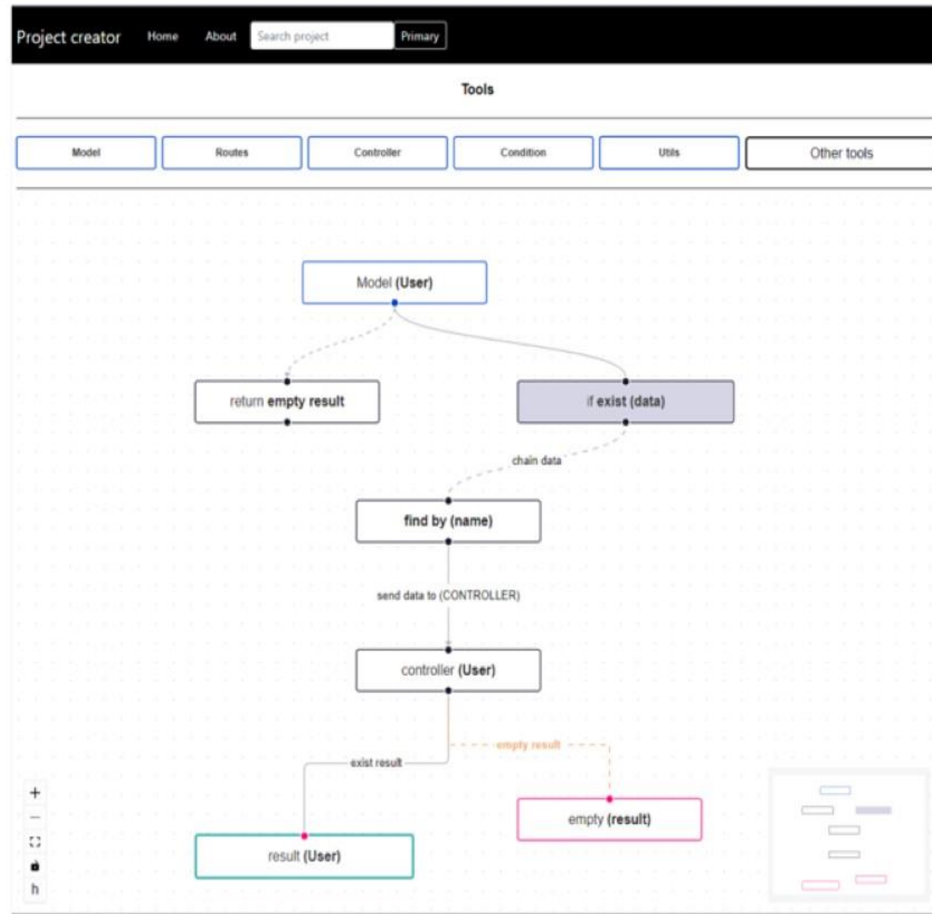
Горизонтальний - прототип в якому не реалізуються всі шари архітектури і нюанси системи, але втілюються деякі особливості інтерфейсу користувача.

Схема імплементації методики прототипування в життєвий цикл розробки системи
Робота з даними (запис, видалення, оновлення)
Розробка алгоритму роботи з даними
Запуск
Робота з прототипом як з фасадом ПЗ
Формування структури БД на основі моделей
Візуальна постановка блоків
Генерація вихідного коду. Запуск на стороні серверу.
Завантаження вихідного коду та робота з ним.

Алгоритм генерації вихідного коду



Приклад побудови логіки за допомогою блок схем



Досвід застосування методики в компанії ТОВ "РЕГІОНАЛЬНА ГАЗОВА КОМПАНІЯ"

Система управління працівниками

Система управління та моніторингу інвентаря

Було створено два проекти, вони мали такі спільні характеристики

Сервер створений за архітектурним стилем REST, для створення системи використовувалась мова JavaScript

Клієнтська частина написана на Vue.js

Члени команди розробки систем

- Manager
- Back-end розробник
- Front-end розробник
- Дизайнер
- Тестувальник

Проте відрізнялись

- Призначенням
- Внутрішньою логікою систем

Досвід застосування мемтодки в компанії ТОВ "РЕГІОНАЛЬНА ГАЗОВА КОМПАНІЯ"

Характеристика проектів		
Однакові\Різні(характеристики)	Система управління працівниками	Система управління та моніторингу інвентаря
Однакові	Характеристики Сервер створений за архітектурним стилем REST, для створення системи використовувалась мова JavaScript Клієнтська частина написана на Vue.js Члени команди розробки систем • Manager • Back-end розробник • Front-end розробник • Дизайнер • Тестувальник	
Різні	Основна функціональність • Створення моніторинг та створення задач для користувачів. • Формування звітів роботи працівників.	Основна функціональність • Відслідковування наявних інструментів. • Закупка інструментів та інвентарю

Досвід застосування мемтодки в компанії ТОВ "РЕГІОНАЛЬНА ГАЗОВА КОМПАНІЯ"

Витрачений час на зрозробку	
Система управління працівниками	Система управління та моніторингу інвентаря (з **застосуванням створеної методики прототипування**)
Закладений час на розробку: 320год. Кількість завдань: 52 • Обговорення: 20 годин • Дизайн: 40 годин + 4 годин (правки) • Front-end розробка: - 140 годин + 24 (правки) • Back-end розробка: 120 годин + 32(правки) Всього: 400 годин розробки	Закладений час на розробку: 325 год. Кількість завдань: 54 • Обговорення: 15 годин • Прототипування: 13 годин • Дизайн: 38 годин + 3 годин (правки) • Front-end розробка: 120 годин + 12 (правки) • Back-end розробка: 135 годин + 16(правки) Всього: 352 годин розробки

Метрики застосування методики на практиці в компанії ТОВ "РЕГІОНАЛЬНА ГАЗОВА КОМПАНІЯ"

В процесі розробки систем

Система управління працівниками та Система управління та моніторингу інвентаря (з **застосуванням створеної методики прототипування**)

Було підраховано затрачений час та порівняно з запланованим часом на розробку.

Та зроблено висновки

Для системи яку розробляли з застосуванням створеної методки прототипування було затрачено на 13,6% меншу часу чим на систему створену без прототипування.

l - Кількість завдань t - Час затрачений на виконання завдань

$$V = \frac{l}{t}$$

Швидкість розробки без застосування прототипування становить 0.13 завдань на годину

Швидкість розробки з застосуванням прототипування становить 0.15 завдань на годину

Висновки

Було проаналізовано існуючі методи та інструментів прототипуваня.

Розроблено власний інструмент і методику для створення прототипів, який в порівнянні з іншими аналогами, безкоштовний, та має систему автоматичної генерації коду.

За допомогою даного інструменту було скорочено час створення програмного забезпечення на 13.6%, що зменшує затрати на створення систем.

ДЯКУЮ ЗА УВАГУ