ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка
до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**Розробка web-додатку для перевірки HTML форми мовою JavaScript**»

Виконав: студент 4 курсу, групи ПД–43
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

_____ Лозінський С.В.
(прізвище та ініціали)

Керівник _____ Коваленко Д.С.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Київ –2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти – «Бакалавр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Негоденко О.В.
"____" _____2022 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

Лозінський Станіслав Вадимович
(прізвище, ім'я, по батькові)

1. Тема роботи: «**Розробка web-додатку для перевірки HTML-форми мовою JavaScript**»
_____
_____
_____
Керівник роботи: Коваленко Данило Сергійович, асистент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від«18» лютого 2022 року №.
2. Строк подання студентом роботи «03» червня 2022 року
3. Вхідні дані до роботи
Методи перевірки HTML-форм; Технічна література програмного забезпечення з приводу перевірки елементів форми;
_____
_____
_____
4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).
4.1 Системи розпізнавання та вилучення текстової інформації з зображень.
4.2 Вимоги та оцінка якості системи.
4.3 Опис проектування системи.
4.4 Опис використаних технологій.
5. Перелік демонстраційного матеріалу (назва основних слайдів)
1. Актуальність проблеми
2. Огляд аналогів

3. Таблиця порівняння аналогів
4. Технічне завдання
5. Програмні засоби реалізації
6. Екранні форми
7. Висновки
8. Дата видачі завдання «11»квітня 2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04.22	Виконано
2	Аналіз та дослідження існуючих аналогів	16.04.22	Виконано
3	Дослідження програмних засобів	21.04.22	Виконано
4	Концепція та архітектура програмного забезпечення	27.04.22	Виконано
5	Розробка функціоналу додатку	05.05.22	Виконано
6	Вступ, висновки, реферат	12.05.22	Виконано
7	Розробка обов‘язкових демонстраційних матеріалів	16.05.22	Виконано
8	Попередній захист роботи	27.05.22	
9	Здача роботи	03.06.22	

Студент __ _____Лозінський С.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____Коваленко Д.С.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Ключові слова: HTML-форма, HTML, CSS, JavaScript, програмний продукт, web-додаток, елементи форм, валідація, перевірка.

Мета роботи: спрощення перевірки HTML-форми за рахунок використання web-додатку.

Об’єкт дослідження: перевірка HTML-форми на стороні клієнта.

Предмет дослідження: технології мови JavaScript для перевірки HTML-форми.

Основними методами є: аналіз та моделювання.

В ході дослідження було отримано: web-додаток для перевірки HTML-форми

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTML – HyperText Markup Language

БД – База даних

ОС – Операційна система

CSS – Cascading Style Sheets

URL – Uniform Resource Locator

IDLE – Integrated Development and Learning Environment

JSON – JavaScript Object Notation

XML – Extensible Markup Language

HTTP – Hypertext Transfer Protocol

ВСТУП

HTML-форми – це прості елементи керування HTML, які використовуються для збору інформації від відвідувачів веб-сайту. До них відносяться текстові поля для введення даних з клавіатури, списки для вибору наперед визначених даних, прапорці для установки параметрів тощо. Існує незліченна кількість способів використання HTML-форм, від реєстрації на форумі або отримання поштової скриньки до перегляду біржового курсу або покупки товару в інтернет-магазині.

Перед тим як надсилати дані на сервер потрібно переконатися, що всі обов'язкові поля форми заповнені даними у коректному форматі. Це називається валідацією на стороні клієнта і допомагає переконатися в тому що дані, введені у кожний елемент форми, відповідають вимогам.

Валідація на стороні клієнта – це первинна перевірка введених даних, яка суттєво покращує взаємодію з інтерфейсом. Виявлення некоректних даних на стороні клієнта дозволяє користувачу одразу їх виправити. Якщо перевірка буде відбуватись лише на сервері, процес заповнення буде більш трудомістким, так як потребує повторення одних і тих самих дій надсилання даних на сервер для отримання зворотної відповіді з повідомленням що саме потрібно виправити.

Що ж таке валідація форми? Зайдіть на будь-який сайт, який має реєстраційну форму. Заповнивши дані в невірному форматі, ви одразу побачите сповіщення про наявність проблеми. Подібні повідомлення мають приблизно такий вигляд:

- "Пароль повинен містити від 8 до 30 символів і мати одну велику літеру, один символ і одну цифру"
- "Будь ласка, введіть коректну email-адресу"
- "Будь ласка, введіть номер телефону у форматі +38 xxx-xxx-xx-xx"
- "Обов'язкове поле"

Це називається перевіркою HTML-форми. Браузер або сервер перевіряють дані, щоб визначити чи відповідають вони необхідному формату. Перевірка, яка

виконується у браузері, називається валідацією на стороні клієнта, а перевірка на сервері — валідацією на стороні сервера.

Коли формат введених даних відповідає вимогам, додаток дозволяє відправити їх на сервер і зберегти у БД. Якщо ж формат невірний, з'являється сповіщення з описом того, що саме потрібно виправити, дозволяючи ввести дані знову.

Об'єктом дослідження є перевірка HTML-форми на стороні клієнта. Предметом дослідження є використання технологій мови JavaScript для валідації HTML-форми.

1. ОГЛЯД ІСНУЮЧИХ АНАЛОГІВ ДЛЯ ПЕРЕВІРКИ HTML-ФОРМИ

1.1 Огляд мови програмування PHP

PHP — це мова серверних сценаріїв з відкритим вихідним кодом, яку багато розробників використовують для розробки веб-сайтів. Це також мова загального призначення, яку можна використовувати для створення багатьох проектів, включаючи графічні інтерфейси користувача (GUI).

Що означає PHP? Абревіатура PHP спочатку означала "Персональна домашня сторінка". Але зараз це рекурсивний акронім для Hypertext Preprocessor. (Рекурсивний у тому сенсі, що перше слово саме по собі є абревіатурою, тому повне значення не слідує за абревіатурою).

Першу версію PHP було випущено 26 років тому. Зараз на черзі 8-а версія, випущена у листопаді 2020 року, але 7-ма версія залишається найпоширенішою.

PHP працює на движку Zend, який є найпопулярнішою реалізацією. Є й інші реалізації, наприклад, parrot, HPVM (Hip Hop Virtual Machine) та Hip Hop, створений Facebook.

PHP переважно використовується для створення веб-серверів. Він запускається у браузері, а також може працювати у командному рядку. Тому якщо

вам не хочеться показувати виведення коду в браузері, ви можете показати його в терміналі.

PHP має ряд переваг, які зробили його таким популярним, і вже понад 15 років він є основною мовою для веб-серверів. Ось деякі з переваг PHP:

— Крос-платформність: PHP не залежить від платформи. Вам не потрібно мати певну ОС, щоб використовувати його, тому що він працює на будь-якій платформі, будь то Mac, Windows або Linux.

— Відкритий вихідний код: PHP має вихідний вихідний код. Оригінальний код доступний для всіх, хто хоче його використовувати. Це одна з причин, чому один з його фреймворків Laravel, так популярний.

— Легко вивчати: PHP неважко вивчити абсолютним новачкам. Ви можете освоїти його, навіть якщо у вас є знання в області програмування.

— PHP синхронізується з усіма базами даних: Ви можете легко підключити PHP до всіх баз даних, реляційних та нереляційних. Так, він може миттєво підключитися до MySQL, Postgress, MongoDB або будь-якої іншої бази даних.

— Підтримуюча спільнота: PHP має дуже підтримуючу онлайн-спільноту. Офіційна документація містить посібники з використання функцій, і ви можете легко вирішити проблему, якщо застрягли.

Хто використовує PHP? Багато відомих компаній і технологічних гігантів використовують PHP для роботи своїх серверів і створення безлічі неймовірних речей.

Facebook: Facebook використовує PHP для свого сайту. У свою чергу, компанія зробила свій внесок у розвиток спільноти, створивши реалізацію, відому як Hip Hop for PHP.

Wikipedia: одне з найбільших у світі джерел інформації з будь-якої теми, Wikipedia побудована на PHP.

Системи керування контентом (CMS): найпопулярніша у світі система керування контентом, WordPress, побудована на PHP. Інші системи управління контентом, такі як Drupal, Joomla та Magento, також побудовані на PHP. Shopify

також працює на PHP.

Платформи веб-хостингу: багато платформ веб-хостингу, такі як BlueHost, Site ground і Whogohost, використовують PHP у своїх хостинг-серверах.

1.1.1 Використання PHP для перевірки HTML-форми

HTML-форма містить різні поля введення, такі як текстове поле, прапорець, радіокнопки, кнопка відправки, контрольний список і т.д. Ці поля введення потребують перевірки, яка гарантує, що користувач ввів інформацію у всі необхідні поля, а також перевіряє, що інформація, надана користувачем, є достовірною та правильною.

Немає жодної гарантії, що інформація, надана користувачем, завжди є вірною. PHP перевіряє дані на стороні сервера, які надсилаються за допомогою HTML-форми. Вам потрібно перевірити кілька речей:

- Порожнє поле
- Валідація рядка
- Валідація чисел
- Валідація електронної пошти
- Валідація URL
- Довжина введення

Порожнє поле: нижче наведений код перевіряє, що поле не порожнє. Якщо користувач залишить поле порожнім, буде видано повідомлення про помилку. Помістіть ці рядки коду, щоб перевірити обов'язкове поле.

```
if (emptyempty ($_POST["name"])) {
    $errMsg = "Error! You didn't enter the Name.";
        echo $errMsg;
} else {
    $name = $_POST["name"];
}
```

Рисунок 1.1 – Перевірка на порожнє поле мовою PHP

Перевірка рядка: наведений нижче код перевіряє, що поле міститиме лише алфавіти та пробіли, наприклад - ім'я. Якщо поле name не отримає коректне введення від користувача, то буде видано повідомлення про помилку:

```
$name = $_POST ["Name"];
if (!preg_match ("/^[a-zA-z]*$/", $name) ) {
   $ErrMsg = "Only alphabets and whitespace are allowed.";
         echo $ErrMsg;
} else {
   echo $name;
}
```

Рисунок 1.2 – Перевірка рядка мовою PHP

Перевірка числа: наведений нижче код перевіряє, що поле міститиме лише числове значення. Наприклад – Mobile no. Якщо поле Mobile не отримає числові дані від користувача, код видасть повідомлення про помилку:

```
$mobileno = $_POST ["Mobile_no"];
if (!preg_match ("/^[0-9]*$/", $mobileno) ){
   $ErrMsg = "Only numeric value is allowed.";
   echo $ErrMsg;
} else {
   echo $mobileno;
}
```

Рисунок 1.3 – Перевірка числа мовою PHP

Перевірка електронної пошти: правильна адреса електронної пошти повинна містити символи @ та . PHP надає різні методи перевірки адреси електронної пошти. Тут ми будемо використовувати регулярні вирази, щоб перевірити адресу електронної пошти. Наведений нижче код перевіряє адресу електронної пошти, вказану користувачем у HTML-формі. Якщо поле не містить правильну адресу електронної пошти, код виведе повідомлення про помилку:

```php
$email = $_POST ["Email"];
$pattern = "^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,3})$^";
if (!preg_match ($pattern, $email) ){
   $ErrMsg = "Email is not valid.";
        echo $ErrMsg;
} else {
   echo "Your valid email address is: " .$email;
}
```

Рисунок 1.4 – Перевірка електронної адреси мовою PHP

Валідація довжини введення: перевірка довжини введення обмежує користувача у наданні значення між вказаним діапазоном, наприклад номер мобільного телефону. Правильний номер мобільного телефону має складатися із 10 цифр. Наведений код допоможе вам застосувати перевірку довжини значення, що вводиться користувачем:

```php
$mobileno = strlen ($_POST ["Mobile"]);
$length = strlen ($mobileno);

if ( $length < 10 && $length > 10) {
   $ErrMsg = "Mobile must have 10 digits.";
        echo $ErrMsg;
} else {
   echo "Your Mobile number is: " .$mobileno;
}
```

Рисунок 1.5 – Перевірка на довжину введення мовою PHP

Перевірка URL-адреси: наведений нижче код перевіряє URL сайту, наданого користувачем через HTML-форму. Якщо поле не містить дійсних URL-адрес, код виведе повідомлення про помилку, тобто. "URL недійсний".

```php
$websiteURL = $_POST["website"];
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "URL is not valid";
  echo $websiteErr;
} else {
   echo "Website URL is: " .$websiteURL;
}
```

Рисунок 1.6 – Перевірка URL-адреси мовою PHP

Перевірка натискання кнопки: наведений нижче код перевіряє, що користувач натиснув кнопку submit, і відправляє дані форми на сервер одним з наступних методів - get або post.

```php
if (isset ($_POST['submit']) {
   echo "Submit button is clicked.";
   if ($_SERVER["REQUEST_METHOD"] == "POST") {
      echo "Data is sent using POST method ";
   }
} else {
   echo "Data is not submitted";
}
```

Рисунок 1.7 – Перевірка натискання кнопки мовою PHP

1.2 Огляд мови програмування Python

Python є інтерпретованою, об'єктно-орієнтовною мовою програмування високого рівня, що має динамічну семантику. Містить в собі високорівневі структури даних разом із динамічною типізацією та динамічним зв'язуванням. Це робить його дуже привабливим для швидкої розробки додатків, а також для використання у якості мови сценаріїв або "клею" для з'єднання існуючих компонентів.

Має також і не складний синтаксис, що легко вивчається, підкреслює зручність читання і, отже, знижує витрати на супровід програм. Python підтримує

модулі та пакети, що сприяють модульності програми та повторному використанню коду. Python-інтерпретатор і стандартна бібліотека доступні у вихідному вигляді для всіх основних платформ і можуть вільно поширюватися.

Цінують Python через велику продуктивність, яку він забезпечує. Оскільки відсутній етап компіляції, цикл "редагування-тестування-налагодження" дуже швидкий.

Налагоджувати продукт на Python досить просто: помилка або невалідне введення ніколи не покаже помилку сегментації. Натомість, коли інтерпретатор виявляє помилку, він показує виняток. Коли програма не бачить виняток, інтерпретатор трасує стек. Налагоджувач на рівні вихідного тексту дає змогу оглядати локальні та глобальні змінні, давати оцінку довільним висловлюванням, встановлювати точки зупинки, проходити за кодом за рядком за раз і так далі. Відладчик, що написан мовою Python, говорить про силу Python. З іншого ж боку, найчастіше швидшим способом налагодження програми є додавання декількох операторів друку у вихідний текст: найшвидший цикл редагування-тестування-налагодження робить цей простий підхід дуже ефективним.

Серед основних її переваг можна назвати такі:

- Чистий синтаксис(для виділення блоків слід використовувати відступи);
- Переносність програм (що властиве більшості інтерпретованих мов);
- Стандартний дистрибутив має велику кількість корисних модулів (включно з модулем для розробки графічного інтерфейсу);
- Можливість використання Python в діалоговому режимі (дуже корисне для експериментування та розв'язання простих задач);
- стандартний дистрибутив має просте, але разом із тим досить потужне середовище розробки, яке зветься IDLE і яке написане мовою Python;
- зручний для розв'язання математичних проблем (має засоби роботи з комплексними числами, може оперувати з цілими числами довільної величини, у діалоговому режимі може використовуватися як потужний калькулятор);

- відкритий код (можливість редагувати його іншими користувачами).

Перевірка HTML-форм у цій мові відбувається за допомогою фреймворку Django. Django є високорівневим веб-фреймворком на мові Python. Він дозволяє швидко створювати безпечні веб-сайти. Був написаний досвідченими розробниками, фрейворк бере на себе велику частину роботи, пов'язану з веб-розробкою, так що ви можете зосередитися на написанні своєї програми. Він є безкоштовним і з відкритим вихідним кодом, має активну спільноту, відмінну документацію та безліч варіантів безкоштовної та платної підтримки.

Django допомагає вам писати програмне забезпечення, яке:
— **Повне**. Django наслідує філософію "батарейки в комплекті" і дає практично все, що може знадобитися розробнику. Оскільки все, що вам знадобиться, є частиною одного продукту, все це працює злагоджено та має велику та актуальну документацію.
— **Універсальне**. Django використовується для створення будь-якого типу веб-сайту - від систем керування контентом до соціальних мереж та сайтів новин. Фреймворк може працювати з будь-яким іншим фреймворком на стороні клієнта і може надавати контент у будь-якому форматі (включаючи HTML, RSS-канали, JSON, XML тощо).
— **Безпечне**. Django допомагає програмістам уникати багатьох поширених помилок безпеки, бо дає фреймворк, який спроектований щоб зробити правильні речі для автоматичного захисту веб-сайту. Як приклад, Django забезпечує безпечне керування обліковими записами юзерів та паролями, уникаючи таких поширених помилок, як розміщення інформації про сесію в файлах-кукі, де вона вразлива або пряме зберігання паролів замість хеш пароля.
Hash пароля – значення фіксованої довжини, яке було створене шляхом передачі пароля через криптографічну hash-функцію. Фреймворк має змогу перевірити правильність пароля, пропустивши його через hash-функцію та порівняти отриманий результат із збереженим хеш-значенням. Однак через свій характер функції, навіть якщо збережене хеш-значення буде скомпрометовано, хакеру буде

достатньо складно дізнатися ваш оригінальний пароль. Django за замовчуванням дає захист від багатьох вразливостей, що включає SQL-ін'єкції, міжсайтовий скриптинг, підробку міжсайтових запитів та clickjacking.
— **Масштабоване**. Фреймворк має компонентну архітектуру "shared-nothing" (кожна частина архітектури незалежна від інших, і через це може бути замінена або змінена при необхідності). Чіткий розділ між різними частинами означає, що Django ає змогу масштабуватися для збільшення трафіку через додавання обладнання будь-якого рівня: кеш-серверів, серверів баз даних або серверів додатків. Деякі з найбільш завантажених сайтів успішно масштабували Django для задоволення своїх потреб (наприклад, Instagram та Disqus, і це лише два).
— **Підтримуване**. Код фреймворку був створений з використанням принципів проектування та патернів, що сприяють створенню коду, що супроводжується і багаторазово використовується. До речі, в ньому використовується принцип "Не повторюй себе" (DRY), а це дозволяє уникнути непотрібного дублювання, скорочуючи обсяг коду. Django також дає змогу групуванню пов'язаних функціональних можливостей у багаторазово використовувані "додатки" і, на нижчому рівні, групує пов'язаний код у модулі (за зразком Model View Controller (MVC)).
— **Портативне**. Django написаний мовою Python, яка працює на багатьох платформах. Це означає, що ви не прив'язані до конкретної серверної платформи і можете запускати свої програми на багатьох версіях Linux, Windows та MacOS. Крім того, Django добре підтримується багатьма хостинг-провайдерами, які часто надають спеціальну інфраструктуру та документацію для розміщення Django-сайтів.

Django був спочатку розроблений між 2003 та 2005 роками командою веб-фахівців, які відповідали за створення та підтримку газетних сайтів. Після створення низки сайтів команда почала виявляти та повторно використовувати багато загального коду та шаблонів проектування. Цей загальний код перетворився на загальну структуру веб-розробки, яка у липні 2005 року була відкрита як проект Django.

Django продовжував розвиватися і вдосконалюватися, починаючи з першого релізу (1.0) у вересні 2008 року і закінчуючи версією 4.0 (2022), що недавно вийшла. У кожному випуску додавалися нові функціональні можливості та виправлялися помилки, починаючи від підтримки нових типів баз даних, шаблонизаторів та кешування, до додавання "загальних" функцій представлення та класів (які зменшують кількість коду, який розробникам доводиться писати для вирішення низки задач програмування).

Зараз Django - це процвітаючий спільний проект з відкритим вихідним кодом, в якому беруть участь багато тисяч користувачів та співавторів. Хоча в ньому все ще є деякі особливості, що відображають його походження, Django перетворився на універсальний фреймворк, здатний розробляти веб-сайти будь-якого типу.

Наскільки популярним є Django? Не існує легкодоступного і точного виміру популярності серверних фреймворків (хоча ви можете оцінити популярність, використовуючи такі механізми, як підрахунок кількості проектів GitHub та питань StackOverflow для кожної платформи). Краще запитати, чи Django "досить популярним", щоб уникнути проблем непопулярних платформ. Чи продовжує він розвиватися? Чи можете ви отримати допомогу, якщо вона вам знадобиться? Чи є можливість отримати оплачувану роботу, якщо ви вивчите Django?

Судячи з кількості відомих сайтів, що використовують Django, кількості людей, які роблять внесок у кодову базу, і кількості людей, які надають як безкоштовну, так і платну підтримку, так, Django - популярний фреймворк.

До відомих сайтів, що використовують Django, відносяться: Disqus, Instagram, Knight Foundation, MacArthur Foundation, Mozilla, National Geographic, Open Knowledge Foundation, Pinterest і Open Stack.

Чи є Django думкою? Веб-фреймворки часто називають "opinionated" або "unopinionated".

Фреймворки з думкою - це ті, які мають думку про "правильний спосіб" вирішення будь-якої конкретної задачі. Вони часто сприяють швидкій розробці у

певній області (вирішення проблем певного типу), оскільки правильний спосіб зробити що-небудь зазвичай добре зрозумілий і добре документований. Однак вони можуть бути менш гнучкими при вирішенні проблем за межами своєї основної області, і, як правило, пропонують менший вибір компонентів та підходів, які можна використати.

На відміну від них, неангажовані фреймворки мають менше обмежень на те, як краще склеювати компоненти для досягнення мети, і навіть на те, які компоненти слід використовувати. Вони полегшують розробникам використання найбільш підходящих інструментів для виконання конкретного завдання, хоч і ціною того, що вам доведеться шукати ці компоненти самостійно.

Django "деякою мірою орієнтований на думку", і тому надає "краще з двох світів". Він надає набір компонентів для вирішення більшості завдань веб-розробки і один (або два) варіанти їх використання. Однак, розв'язана архітектура Django означає, що ви можете вибирати з кількох варіантів або додати підтримку нових при бажанні.

Як виглядає код Django? У традиційному веб-сайті, керованому даними, веб-програма очікує HTTP-запитів від веб-браузера (або іншого клієнта). Коли запит отримано, програма визначає, що потрібно, ґрунтуючись на URL і, можливо, інформації в даних POST або GET. Залежно від того, що потрібно, воно може читати або записувати інформацію з бази даних або виконувати інші завдання, необхідні для задоволення запиту. Потім програма повертає відповідь веб-браузеру, часто динамічно створюючи HTML-сторінку для відображення браузером шляхом вставки даних у шаблони HTML.

Веб-додатки Django зазвичай групують код, що обробляє кожен із цих етапів, в окремі файли:

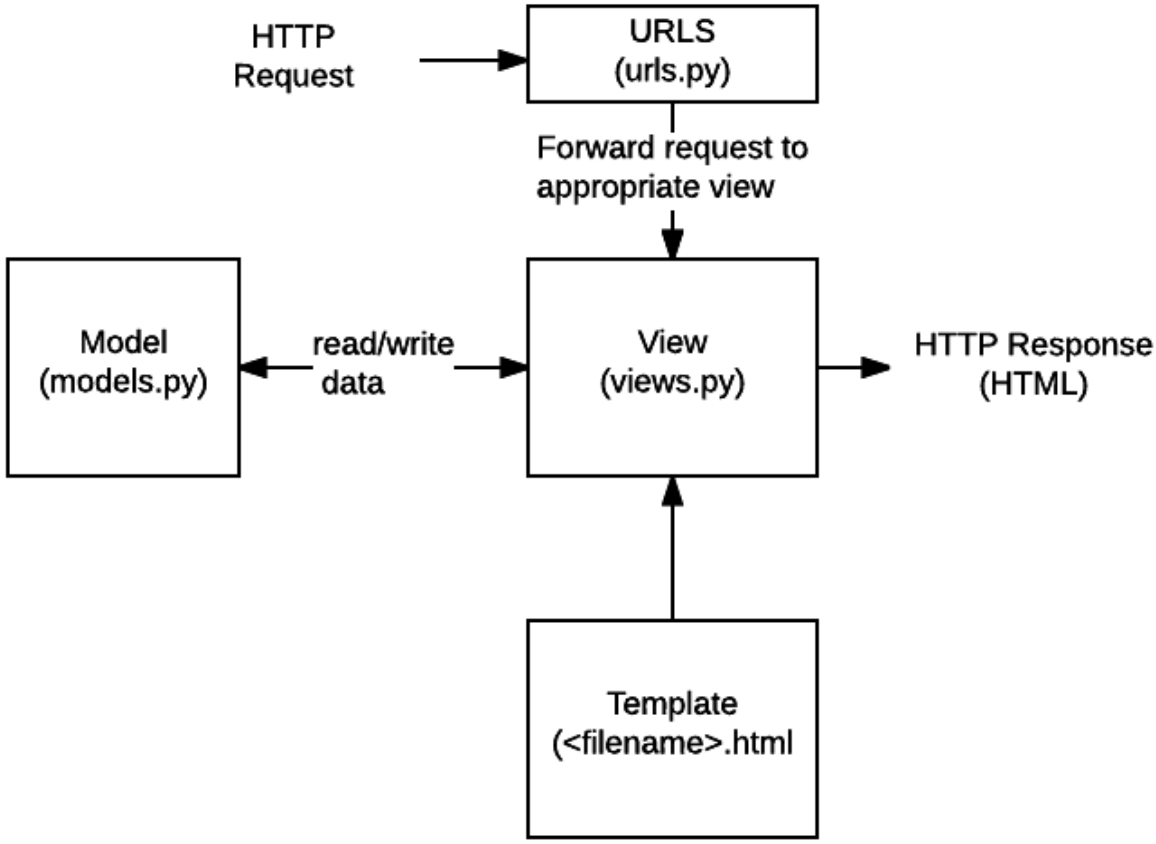


Рисунок 1.8 – Структура веб-додатку Django

URL-адреси: Хоча можна обробляти запити від кожної окремої URL-адреси через одну функцію, набагато зручніше написати окрему функцію подання для обробки кожного ресурсу. URL mapper використовується для перенаправлення запитів HTTP до відповідного представлення на основі URL запиту. URL mapper також може відповідати певним шаблонам рядків або цифр, що з'являються в URL, і передавати їх у функцію представлення даних.

Подання: подання - це функція обробника запитів, яка отримує HTTP-запити та повертає HTTP-відповіді. Уявлення отримують доступ до даних, необхідних задоволення запитів через моделі, і делегують форматування відповіді шаблонів.

Моделі: моделі - це об'єкти Python, які визначають структуру даних програми та надають механізми керування (додавання, зміни, видалення) та запиту записів у базі даних.

Шаблони: шаблон - це текстовий файл, який визначає структуру або макет файлу (наприклад, HTML-сторінки) із заповнювачами, які використовуються для представлення фактичного вмісту. Подання може динамічно створювати HTML-сторінку, використовуючи HTML-шаблон, заповнюючи даними з моделі. Шаблон можна використовувати визначення структури будь-якого типу файла; це не обов'язково має бути HTML.

Валідація форми - дуже важливе завдання для будь-якого веб-програми, щоб ввести до бази даних достовірні дані. Користувачі програми не можуть вставити недійсні дані, якщо дані форми будуть перевірені перед надсиланням. Django називається MVT (Model View Template) фреймворком, у якому завдання контролера виконується самим фреймворком. Завдання, пов'язані з базою даних, виконуються моделлю, а дані надаються шаблоном за допомогою представлення.

Поля форми генеруються на основі конкретної моделі, яка вставлятиме дані в базу даних Django після валідації. Один тип валідації виконується браузером на основі типу поля, визначеного моделі. Використання функції is_valid() - це інший спосіб перевірити дані форми, чи є вони дійсними чи ні після надсилання форми. Цей підручник покаже вам, як дані можуть бути вставлені до бази даних Django після перевірки форми.

1.2.1 Використання Python для перевірки HTML-форми

Перш ніж почати, необхідно виконати такі дії:

- Встановити Django версії 3+ на Ubuntu 20+
- Створити проект Django
- Запустити сервер Django, щоб перевірити, чи сервер працює правильно чи ні.

Встановлення програми Django:

1. Створення програми Django з ім'ям validationapp.

```
$ python3 manage.py startapp validationapp
```

2. Створюємо користувача, який використовуватиметься для доступу до бази даних Django.

```
$ python3 manage.py createsuperuser
```

3. Додаємо ім'я програми до частини INSTALLED_APP файлу py.

```
INSTALLED_APPS = [
    ....
    'validationapp'
]
```

4. Створюємо папку з іменем templates всередині папки validationapp і задаємо розташування шаблону програми в частині TEMPLATES файлу py.

```
TEMPLATES = [
    {
....
            'DIRS': ['/home/fahmida/django_pro/validationapp/templates'],
             ....
      },
]
```

Для валідації форм у Django: треба написати код для наступних файлів, щоб перевірити правила перевірки форми Django:

- models.py
- admin.py
- views.py
- forms.py
- customer.html

Створюємо модель. Відкриваємо файл models.py з папки app і додаємо наступний сценарій для створення структури таблиці customers. Таблиця міститиме чотири поля. Це повне_ім'я, email, contact_no та customer_type. Тут customer_type створюється на кшталт вибору. Це означає, що користувач повинен вибрати будь-яке значення зі списку.

models.py

```python
# Import models
from django.db import models

# Define class for customer entry
class Customer(models.Model):
    # Define values for the customer type
    type = (
        ('paid', 'Paid'),
        ('free', 'Free')
    )
    # Define the fields for the customer table
    full_name = models.CharField(max_length=50)
    email = models.EmailField()
    contact_no = models.CharField(max_length=20)
    customer_type = models.CharField(max_length=32, choices=type, default='free')
```

Рисунок 1.9 – Створення моделі мовою Python

Реєструємо модель. Додаємо наступний скрипт до файлу admin.py для реєстрації моделі Customer у базі даних Django, щоб отримати доступ до таблиці customers з адміністративної панелі Django.

admin.py

```python
# Import admin module
from django.contrib import admin
# Import customer model
from .models import Customer
# Register customer model
admin.site.register(Customer)
```

Рисунок 1.10 – Реєстрація моделі мовою Python

Створюємо файл forms.py у папці app та додаємо наступний сценарій, щоб визначити клас для створення форми на основі вказаних полів моделі Customer.

forms.py

```python
# Import forms module
from django import forms
# Import Customer model
from validationapp.models import Customer

# Define the class for the customer form
class CustomerForm(forms.ModelForm):
    class Meta:
        model = Customer
        fields = ('full_name', 'email', 'contact_no', 'customer_type')
```

Рисунок 1.11 – Визначення класу для створення форми мовою Python

Створюємо шаблон. Створюємо HTML-файл з ім'ям customer.html із наступним сценарієм для відображення раніше розробленої форми з кнопкою Save.

customer.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>User Registration Form</title>
</head>
<body>

    <h2>Customer Entry Form</h2&gt

   <form method="POST" class="post-form" >
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Save</button>
    </form>

</body>
</html>
```

Рисунок 1.12 – Створення HTML файлу мовою Python

Вставка даних після валідації форми. Змінюємо вміст views.py за допомогою наступного сценарію. Функція AddCustomer() використовується в сценарії для вставлення нового запису в таблицю customers після валідації форми. Якщо форму не відправлено, то форму буде завантажено за допомогою файлу customer.html. Якщо форму надіслано з правильними даними, то новий запис буде вставлено за допомогою функції save(), а в браузер буде виведено повідомлення про успіх з можливістю додати ще один запис.

views.py

```
# Import HttpResponse module
from django.http.response import HttpResponse
# Import render module
from django.shortcuts import render
# Import CustomerForm
from validationapp.forms import CustomerForm
# Define function to add customer entry
def AddCustomer(request):
    if request.method == "POST":
        form = CustomerForm(request.POST)
        # If the form data are valid or not
        if form.is_valid():
            try:
                # Save the form data into the database
                form.save()
                # Define the message for the user
                data = ['<h3>The Customer data inserted properly.</h3><br /><a
href="http://localhost:8000/customer">Add another</a>']
                # Return the response
                return HttpResponse(data)
            except:
                pass
    else:
        # Define the form object
        form = CustomerForm()
        # Show the customer entry form
        return render(request, 'customer.html', {'form': form})
```

Рисунок 1.13 – Реалізація функції AddCustomer()

Змінюємо файл urls.py проекту Django за допомогою наступного сценарію. У скрипті визначено два шляхи. Шлях 'customer/' використовуватиметься для відображення форми додавання нових записів клієнтів. Шлях 'admin/' використовуватиметься для відображення панелі адміністрування Django.

urls.py

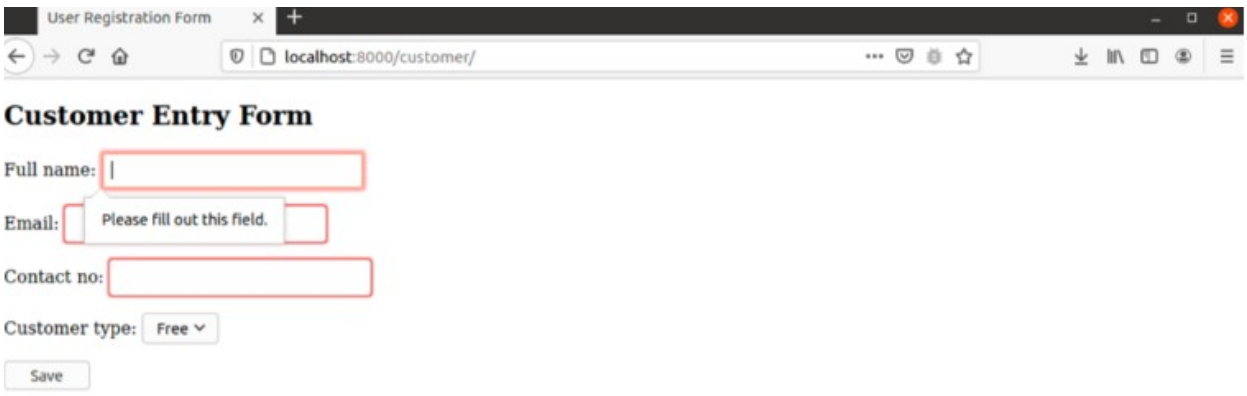
```python
# Import admin module
from django.contrib import admin
# Import path module
from django.urls import path
# Import view
from validationapp import views

# Define path for customer and admin
urlpatterns = [
    path('customer/', views.AddCustomer),
    path('admin/', admin.site.urls)
]
```

Рисунок 1.14 – Фрагмент коду для заміни файлу мовою Python

Запускаємо сервер Django і виконуємо наступну URL-адресу з браузера, щоб відобразити форму введення даних клієнта. Якщо натиснути кнопку Save, не додавши до форми жодних даних, з'явиться наступна сторінка з помилками, пов'язаними з незаповненістю обов'язкових полів.

Результат:



1.3 Огляд мови програмування Java

Java - це широко використовується об'єктно-орієнтована мова програмування та програмна платформа, яка працює на мільярдах пристроїв, включаючи ноутбуки, мобільні пристрої, ігрові консолі, медичні прилади та

багато інших. Правила та синтаксис Java засновані на мовах C та C++.

Однією з основних переваг розробки програмного забезпечення Java є його переносимість. Написавши код для програми на Java на ноутбуці, його дуже легко перенести на мобільний пристрій. Коли мова була винайдена в 1991 році Джеймсом Гослінг з компанії Sun Microsystems (згодом придбаної Oracle), основною метою була можливість "написати один раз, запустити в будь-якому місці".

Важливо також розуміти, що Java істотно відрізняється від JavaScript. Javascript не потребує компіляції, в той час як код Java потребує компіляції. Крім того, Javascript працює тільки у веб-браузерах, в той час як Java може бути запущена будь-де.

Нові та вдосконалені інструменти розробки програмного забезпечення з'являються на ринку з разючою швидкістю, витісняючи існуючі продукти, які колись вважалися незамінними. У світлі цієї безперервної зміни, довговічність Java вражає: через два десятиліття після створення Java залишається найпопулярнішою мовою для розробки прикладного програмного забезпечення - розробники продовжують вибирати його замість таких мов, як Python, Ruby, PHP, Swift, C++ та інших. В результаті Java залишається важливою вимогою для конкуренції на ринку праці.

Перш ніж досліджувати причини незмінної популярності Java, давайте детальніше розглянемо, що таке Java і яке її значення для розробки корпоративних додатків.

Java - це технологія, що складається з мови програмування та програмної платформи. Щоб створити програму з використанням Java, необхідно завантажити Java Development Kit (JDK), доступний для Windows, macOS і Linux. Ви пишете програму мовою програмування Java, потім компілятор перетворює її на байткод Java - набір інструкцій для віртуальної машини Java (JVM), яка є частиною середовища виконання Java (JRE). Java байткод запускається без змін на будь-якій системі, що підтримує JVM, що дозволяє запускати ваш Java-код будь-де.

Програмна платформа Java складається з JVM, Java API та повного середовища розробки. JVM аналізує та виконує (інтерпретує) байткод Java. Java API складається з великого набору бібліотек, включаючи базові об'єкти, функції мережі та функції безпеки; генерацію розширюваної мови розмітки (XML); та веб-сервіси. У сукупності мова Java та програмна платформа Java створюють потужну, перевірену технологію для розробки корпоративного програмного забезпечення.

Якщо ви є розробником корпоративних програм, ви вже знаєте, що таке Java, і цілком імовірно, що у вашій організації вже є тисячі і навіть мільйони рядків виробничого коду, написаного на Java. Швидше за все, вам знадобиться певний рівень знань Java, щоб ви могли усувати неполадки, підтримувати та оновлювати існуючу кодову базу.

Проте було б помилкою розглядати Java лише з погляду успадкованих програм. Мова Java лежить в основі операційної системи Android, на якій працює більша частина смартфонів у світі. Java також є однією з найпопулярніших мов для програм машинного навчання та науки про дані. Надійність, простота використання, кросплатформеність та безпека роблять Java мовою вибору для інтернет-рішень у багатьох корпоративних магазинах.

Зокрема, технологія Java є ідеальною основою для розробки веб-додатків – фундаменту цифрового бізнесу у будь-якій галузі. Сервери програм Java - це веб-контейнери для компонентів Java, XML та веб-сервісів, які взаємодіють з базами даних та надають динамічний веб-контент. Сервери програм Java формують стабільне середовище розгортання корпоративних програм з такими можливостями, як управління транзакціями, безпека, кластеризація, продуктивність, доступність, підключення та масштабованість.

1.3.1 Використання Java для перевірки HTML-форми

Для валідації даних в Java існує Bean Validation. Перша версія цього набору API була специфікована в JSR-303 і опублікована як частина Java EE 6. Поточна версія - 2.0 є частиною Java EE 8 і описана в JSR-380. Еталонною реалізацією

Bean Validation є Hibernate Validator. Bean Validation може використовуватися не тільки в класичних програмах на основі Java EE, але і в програмах на основі Spring, і навіть в програмах, що не мають відношення до Java EE.

Залежності. Для використання API Bean Validation буде потрібна залежність validation-api з javax.validation:

pom.xml

```xml
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>2.0.1.Final</version>
</dependency>
```

Якщо ви розробляєте проект на чистому Java EE, то будь-яких інших залежностей вам не знадобиться, навіть можна використовувати відповідну версію javaee-api, тому що API Bean Validation включені в неї.

Інакше вам знадобляться ще дві залежності: реалізація Bean Validation та EL (Expression Language, мова виразів). В якості реалізації Bean Validation логічніше використовувати еталонну реалізацію - Hibernate Validator, а в якості реалізації EL можна використовувати реалізацію від Glassfish:

pom.xml

```xml
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.10.Final</version>
</dependency>
<dependency>
    <groupId>org.glassfish</groupId>
    <artifactId>javax.el</artifactId>
    <version>3.0.1-b10</version>
</dependency>
```

Отримання валідатора. Найпростіший спосіб отримання валідатора – створити фабрику валідаторів за замовчуванням та запросити у неї валідатор.

ValidationFactoryExample.java

```java
public class ValidationFactoryExample {

    public static void main(String[] args) {
        ValidatorFactory validatorFactory = Validation.buildDefaultValidatorFactory();
        Validator validator = validatorFactory.getValidator();
        // ...
    }
}
```

Надалі отриманий об'єкт типу Validator можна використовувати для валі
дації об'єктів, аргументів, що передаються методи і значень, повертаних методами.

Обмеження. Правила валідації в Bean Validation задаються за допомогою обмежень (constraints), анотацій, розміщених у пакеті javax.validation.constraints. Обмеження можуть застосовуватися до властивостей класів, аргументів методів і конструкторів, їх значенням, що повертаються, а так само до типів узагальнень.

Стандартний набір обмежень включає найчастіше використовувані і універсальні. Крім того, Bean Validation дозволяє розробникам додавати власні обмеження.

Числові обмеження:

— DecimalMax. Застосовується до змінних типів BigDecimal, BigInteger, CharSequence, byte, short, int, long та їх класів-оберток. Значення має бути меншим, або дорівнює зазначеному значенню, або бути null для непримітивів.

— DecimalMin. Аналогічно DecimalMax, значення змінної має бути числом і бути більшим, або дорівнює зазначеній значенню, або бути null для непримітивів.

— Digits. Кількість символів ліворуч від коми має бути меншою, або рівною integer, а праворуч — меншою, або рівною fraction, null є валідним значенням. Застосовується BigDecimal, BigInteger, CharSequence, byte, short, int, long та його класам-обертам.

— Max. Значення має бути меншим, або дорівнює вказаному значенню, або бути null. Застосовується до змінних типів BigDecimal, BigInteger, byte, short, int, long та його класів-оберток.

— Min. Значення має бути більшим, або дорівнює зазначеному значенню, або бути null. Застосовується до змінних типів BigDecimal, BigInteger, byte, short, int, long та його класів-оберток.

— Negative. Значення має бути негативним або бути null. Застосовується до змінних типів BigDecimal, BigInteger, byte, short, int, long та його класів-оберток.

— NegativeOrZero. Значення має бути негативним, дорівнювати 0, або бути null. Застосовується до змінних типів BigDecimal, BigInteger, byte, short, int, long та його класів-оберток.

— Positive. Значення має бути позитивним або бути null. Застосовується до змінних типів BigDecimal, BigInteger, byte, short, int, long та його класів-оберток.

— PositiveOrZero. Значення має бути негативним, дорівнювати 0, або бути null. Застосовується до змінних типів BigDecimal, BigInteger, byte, short, int, long та його класів-оберток.

Обмеження дати та часу:

— Future. Значення змінної має бути майбутнім часом. Застосовується до Date, Calendar та багатьох типів з пакету java.time.

— FutureOrPresent. Значення змінної має бути майбутнім чи сьогоденням. Застосовується до Date, Calendar та багатьох типів з пакету java.time.

— Past. Значення змінної має бути минулим часом. Застосовується до Date, Calendar та багатьох типів з пакету java.time.

— PastOrPresent. Значення змінної має бути минулим чи сьогоденням. Застосовується до Date, Calendar та багатьох типів з пакету java.time.

Строкові обмеження:

— Email. Значення має бути адресою електронної пошти; застосовна до CharSequence. Поведінка залежить від конкретної реалізації.

— NotBlank. Значення типу CharSequence не повинно бути null, порожнім або складатися лише з пробілових символів.

— Pattern. Значення типу CharSequence має відповідати вказаному регулярному виразу.

Булеві обмеження:

— AssertFalse. Інструкція застосовна до змінних типів boolean і Boolean, значення яких має бути false, чи null.

— AssertTrue. Протилежність @AssertFalse, значення має бути true або null.

Універсальні:

— NotEmpty. Значення типів CharSequence, Collection, Map чи масив має бути null і має містити хоча б 1 елемент.

— NotNull. Значення повинно бути null.

— Null. Значення має бути null.

— Size. Розмір значення типів CharSequence, Collection, Map або масиву має бути у вказаному діапазоні між min і max.

Валідація властивостей. Для валідації властивостей класів достатньо одну або кілька з перерахованих вище анотацій застосувати до потрібної властивості, наприклад:

Event.java Java

```java
public class Event {

    @NotNull
    @PastOrPresent
    private Date dateCreated;
}
```

При валідації об'єкта типу Event буде проведена перевірка, що властивість dateCreated містить дату, значення якої - час або поточний час.

Приклад валідації:

ValidationFactoryExample.java Default

```java
public class ValidationFactoryExample {

    public static void main(String[] args) {
        ValidatorFactory validatorFactory = Validation.buildDefaultValidatorFactory();
        Validator validator = validatorFactory.getValidator();

        Set<ConstraintViolation<Event>> violations = validator.validate(new Event());
    }
}
```

Метод validate валідатора поверне непустий список порушень (ConstraintViolation), якщо властивості об'єкта порушують правила валідації.

Валідація аргументів і значень методів, що повертаються. Для валідації аргументів і значень методів, що повертаються (у тому числі і конструкторів) можна використовувати ExecutableValidator, введений в Bean Validation 1.1.

Допустимо, у нас є клас EventHandler:

EventHandler.java

```java
public class EventHandler {

    public EventHandler(@NotNull EventRepository repository) {
        // ..
    }

    @NotNull
    public Optional<@NotBlank String> handle(@NotNull @Valid Event event) {
        // ..
    }
}
```

Валідація аргументів конструктора та методу handle виконується таким чином:

```java
// constructor args validation
Constructor<EventHandler> constructor = EventHandler.class.getConstructor(String.class)
Set<ConstraintViolation<EventHandler>> constructorViolations = executableValidator
        .validateConstructorParameters(constructor, new Object[]{"name"});

EventHandler eventHandler = new EventHandler(repository);
Event event = new Event();

// method args validation
Method method = eventHandler.getClass()
        .getMethod("handle", Event.class);

Set<ConstraintViolation<EventHandler>> argsViolations = executableValidator
        .validateParameters(eventHandler, method, new Object[]{event});

// returned value validation
String id = eventHandler.handle(event);

Set<ConstraintViolation<EventHandler>> returnViolations = executableValidator
        .validateReturnValue(eventHandler, method, id);
```

1.4 Огляд мови програмування Ruby

Ruby - це інтерпретована, високорівнева мова програмування загального призначення, яка підтримує кілька парадигм програмування. Він був розроблений з акцентом на продуктивність та простоту програмування. Ruby все є об'єктом, включаючи примітивні типи даних. Він був розроблений в середині 1990-х років Юкіхіро "Мац" Мацумото в Японії.

Ruby динамічно типізується і використовує складання сміття та компіляцію "точно в термін". Він підтримує безліч парадигм програмування, включаючи

процедурне, об'єктно-орієнтоване та функціональне програмування. За словами автора, на Ruby вплинули Perl, Smalltalk, Eiffel, Ada, BASIC і Lisp.

Мацумото говорив, що Ruby був задуманий у 1993 році. У повідомленні 1999 року у списку розсилки ruby-talk він описує деякі зі своїх ранніх ідей про мову:

«Я розмовляв зі своїм колегою щодо можливості створення об'єктно-орієнтованої мови сценаріїв. Я знав Perl (Perl4, не Perl5), але він мені не дуже подобався, тому що від нього пахло іграшковою мовою (він і зараз такий). Об'єктно-орієнтована мова здавалася дуже перспективною. Тоді я знав Python. Але він мені не сподобався, тому що я не вважав його справжньою об'єктно-орієнтованою мовою – ОО-функції здавалися доповненням до мови. Як мовний маніяк та фанат ГО протягом 15 років, я дуже хотів справжній об'єктно-орієнтований, простий у використанні мову сценаріїв. Я шукав, але не міг знайти такої мови. Тож я вирішив створити його».

Мацумото описує дизайн Ruby як те, що у своїй основі він схожий на просту мову Lisp, з об'єктною системою, як у Smalltalk, блоками, натхненими функціями вищого ладу, та практичною корисністю, як у Perl.

Назва "Ruby" виникла під час онлайн-чату між Мацумото та Кейджу Ішицука 24 лютого 1993 року, до того, як було написано будь-який код для мови. Спочатку було запропоновано дві назви: "Coral" та "Ruby". Мацумото вибрав останній у пізнішому електронному листі Ішицуку. Пізніше Мацумото наголосив на факторі, що вплинув на вибір назви "Рубін" - це був камінь народження одного з його колег.

Перший публічний випуск Ruby 0.95 був анонсований у японських внутрішніх групах новин 21 грудня 1995 року. Згодом за два дні випустили ще три версії Ruby. Випуск співпав із запуском япономовного списку розсилки ruby-list, який став першим списком розсилки для нової мови.

Вже на цьому етапі розробки були присутні багато функцій, знайомих за пізнішими версіями Ruby, включаючи об'єктно-орієнтований дизайн, класи з успадкуванням, міксини, ітератори, закриття, обробку виключень та складання

сміття.

Після випуску Ruby 0.95 в 1995 році, в наступні роки було випущено кілька стабільних версій:

- Ruby 1.0: 25 грудня 1996 року
- Ruby 1.2: грудень 1998 року
- Ruby 1.4: Серпень 1999
- Ruby 1.6: вересень 2000 року

Синтаксис Ruby загалом схожий на синтаксис Perl та Python. Визначення класів та методів позначаються ключовими словами, тоді як блоки коду можуть бути визначені або ключовими словами, або дужками. На відміну від Perl, змінні немає обов'язкового префікса як сигила. При використанні сигіла змінюється семантика зони видимості змінної. Для практичних цілей немає різниці між висловлюваннями і твердженнями. Переклад рядка є значним і сприймається як кінець висловлювання; як еквівалент може використовуватися точка з комою. На відміну від Python, відступи не значущі.

Однією з відмінностей від Python і Perl є те, що Ruby зберігає всі змінні екземпляри повністю приватними для класу та розкриває їх лише через методи-доступники (attr_writer, attr_reader тощо). На відміну від методів "getter" та "setter" інших мов, таких як C++ або Java, методи-аксесори Ruby можуть бути створені одним рядком коду за допомогою метапрограмування; однак методи-аксесори можуть бути створені традиційним способом C++ і Java. Оскільки виклик цих методів не вимагає використання круглих дужок, можна тривіально перетворити змінну екземпляра на повноцінну функцію, не змінюючи жодного рядка коду, що викликає, і не проводячи жодного рефакторингу, домагаючись функціональності, аналогічної членам властивостей в C# і VB.NET.

Дескриптори властивостей Python схожі, але в процесі розробки доводиться йти на компроміс. Якщо почати в Python з використанням загальнодоступної змінної екземпляра, а потім змінити реалізацію, щоб використовувати приватну змінну екземпляра, відкриту через дескриптор властивості, то код всередині класу

може вимагати коригування для використання приватної змінної, а не загальнодоступної властивості.

Дизайн Ruby змушує всі змінні екземпляри бути приватними, але при цьому надає простий спосіб оголошення методів set та get.

Це відповідає ідеї, що Ruby ніколи не відбувається прямого доступу до внутрішніх членів класу ззовні; швидше, передається повідомлення класу і виходить відповідь.

1.4.1 Використання Ruby для перевірки HTML-форми

Крок 1: Початкова установка

1. rails new rails-real-time-form-validation --webpack=stimulus
2. rails g scaffold Post title body:text
3. rails db:migrate

Крок 2: Додавання валідацій у Post Model

```ruby
# app/models/post.rb
class Post < ApplicationRecord
  validates :body, length: { minimum: 10 }
  validates :title, presence: true
end
```

Крок 3: Створення кінцевої точки валідації форми

1. rails g controller form_validations/posts
2. Оновлення контролера для успадкування від PostsController

```ruby
# app/controllers/form_validations/posts_controller.rb
class FormValidations::PostsController < PostsController
  def update
    @post.assign_attributes(post_params)
    @post.valid?
    respond_to do |format|
      format.text { render partial: 'posts/form', locals: { post: @post }, formats:
    end
  end

  def create
    @post = Post.new(post_params)
    @post.validate
    respond_to do |format|
      format.text { render partial: 'posts/form', locals: { post: @post }, formats:
    end
  end
end
```

Крок 4: Створення контролера стимулів

1. touch app/javascript/controllers/form_validation_controller.js

```javascript
// app/javascript/controllers/form_validation_controller.js
import Rails from "@rails/ujs"
import { Controller } from "stimulus"

export default class extends Controller {
  static targets  = [ "form", "output"]
  static values   = { url: String }

  handleChange(event) {
    Rails.ajax({
      url: this.urlValue,
      type: "POST",
      data: new FormData(this.formTarget),
      success: (data) => {
        this.outputTarget.innerHTML = data;
      },
    })
  }

}
```

2. Оновлення розмітки.

```
<%# app/views/posts/_form.html.erb %>
<%= form_with(model: post, data: { form_validation_target: "form" }) do |form| %>
  <% if post.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(post.errors.count, "error") %> prohibited this post from being

      <ul>
        <% post.errors.each do |error| %>
          <li><%= error.full_message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= form.label :title %>
    <%= form.text_field :title, data: { action: "form-validation#handleChange" } %>
  </div>

  <div class="field">
    <%= form.label :body %>
    <%= form.text_area :body, data: { action: "form-validation#handleChange" } %>
  </div>

  <div class="actions">
    <%= form.submit disabled: post.errors.any?  %>
  </div>
<% end %>
```

и переглянути відповідь сервера та побачити роботу у процесі.

Крок 5: Запити на відмову

1. yarn add lodash.debounce

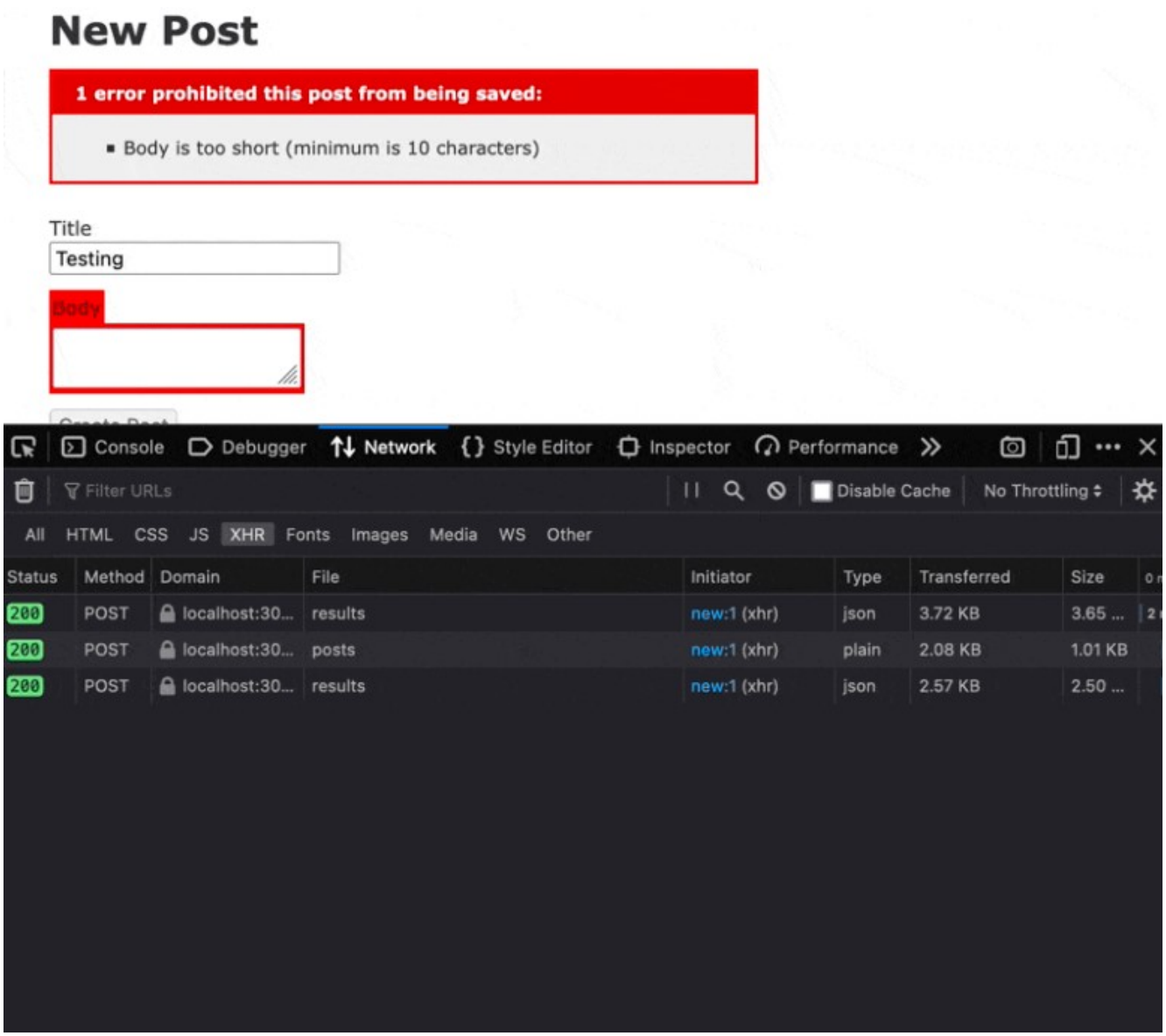


Рисунок 1.16 – Ілюстрація запиту на відмову

Крок 6: Фокусування введення

```
import Rails from "@rails/ujs"
import { Controller } from "stimulus"
const debounce = require('lodash.debounce');

export default class extends Controller {
  static targets  = [ "form", "output"]
  static values   = { url: String }

  initialize() {
    this.handleChange = debounce(this.handleChange, 500).bind(this)
  }

  handleChange(event) {
    let input = event.target
    Rails.ajax({
      url: this.urlValue,
      type: "POST",
      data: new FormData(this.formTarget),
      success: (data) => {
        this.outputTarget.innerHTML = data;
        input = document.getElementById(input.id);
        this.moveCursorToEnd(input);
      },
    })
  }
```

2. МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Модель предметної галузі

Моделю предметної галузі називають систему, що дає уявлення про структуру або функціонування предметної області.

Завчасна побудова моделі предметної області дозволяє отримати ефективний та якісний програмний продукт. Така модель зменшує шанс допущення помилок, тим самим запобігає фінансовим втратам.

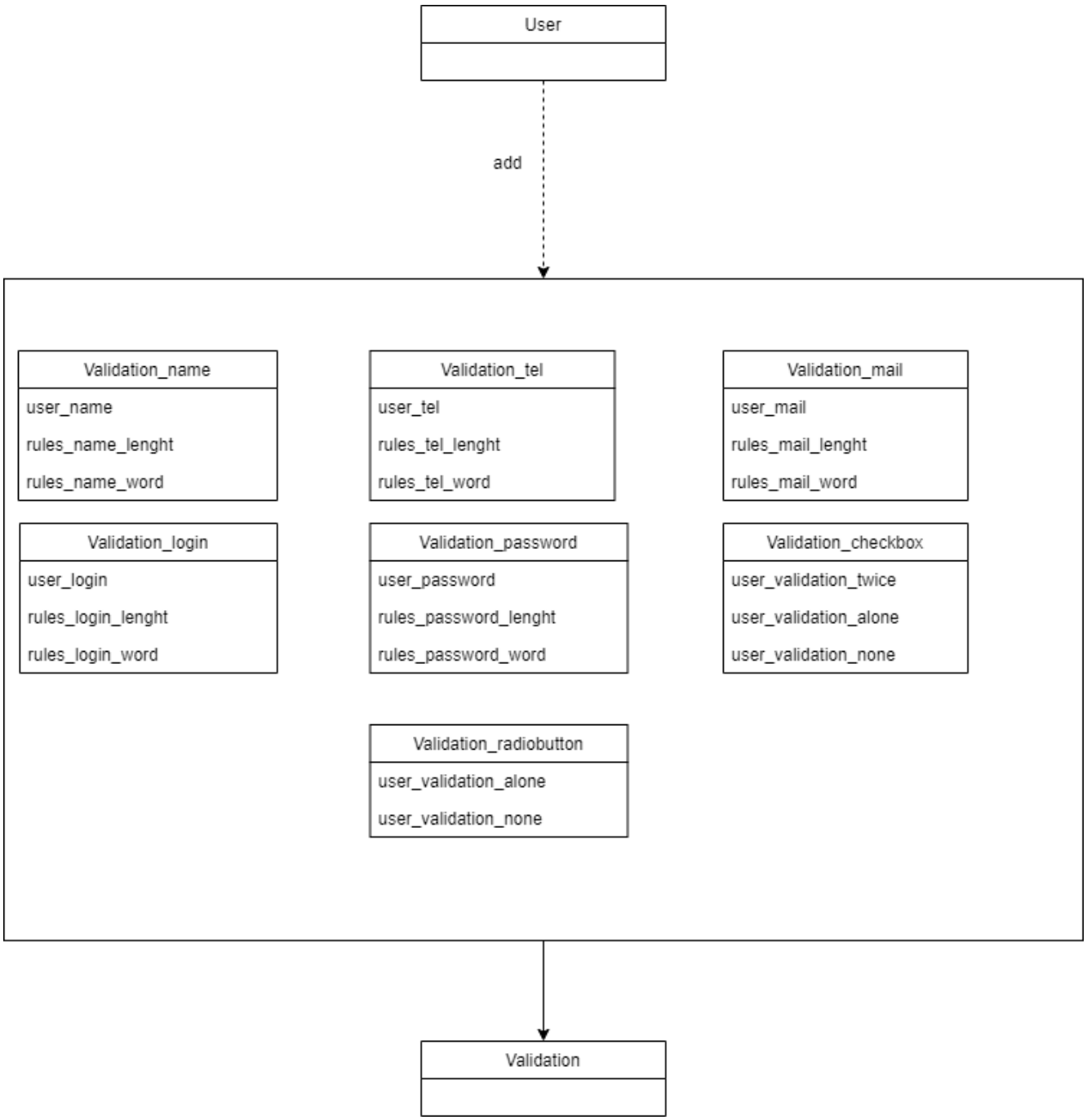


Рисунок 2.1 – Складові компоненти програмного продукту

2.2 Модель прецедентів

Діаграма прецедентів в UML – це діаграма, яка відображає відношення між акторами і прецедентами. Така діаграма дозволяє описати систему на концептуальному рівні.

Прецедент — частина функціональності системи, завдяки якій користувач може отримати конкретний і необхідний йому результат. Прецедент відповідає окремому сервісу системи, визначає один із варіантів її використання і описує спосіб взаємодії між користувачем і системою.

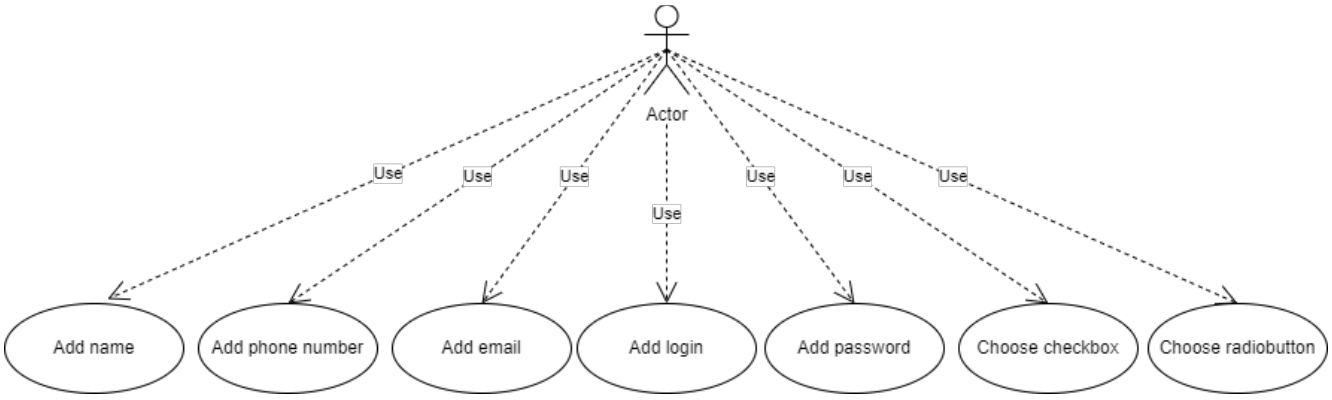


Рисунок 2.2 – Діаграма прецедентів UML

2.3 Діаграма діяльності

На діаграмі діяльності UML показані дії, стан яких відображає діаграма станів. Ця діаграма використовується при модулюванні бізнес-процесів, технологічних процесів, послідовних і паралельних обчисленнях.

Діаграма діяльності складається з обмеженої кількості фігур, об’єднаних стрілками. Основні фігури:

1. Прямокутники із закругленнями – дії (операція). Вузол управління (control node) - це абстрактний вузол дії, що координує потоки дій.
2. Ромби – рішення. Вузол рішення призначений для визначення правила розгалуження та різних варіантів подальшого розвитку сценарію. У точку розгалуження входить рівно один перехід, а виходить два або більше.
3. Широкі смуги - початок (розгалуження) та закінчення (сходження) розгалуження дій. Вузол об'єднання має два і більше вхідні вузли та один вихідний.
4. Чорне коло – початок процесу (початковий вузол). Початковий вузол діяльності (або початковий стан діяльності) (activity initial node) є вузлом

управління, у якому починається потік (або потоки) при виклик даної діяльності ззовні.

5. Чорне коло з обведенням - закінчення процесу (фінальний вузол). Кінцевий вузол діяльності (або кінцевий стан діяльності) є вузлом управління, який зупиняє всі потоки даної діаграми діяльності. На діаграмі може бути більше ніж один кінцевий вузол.

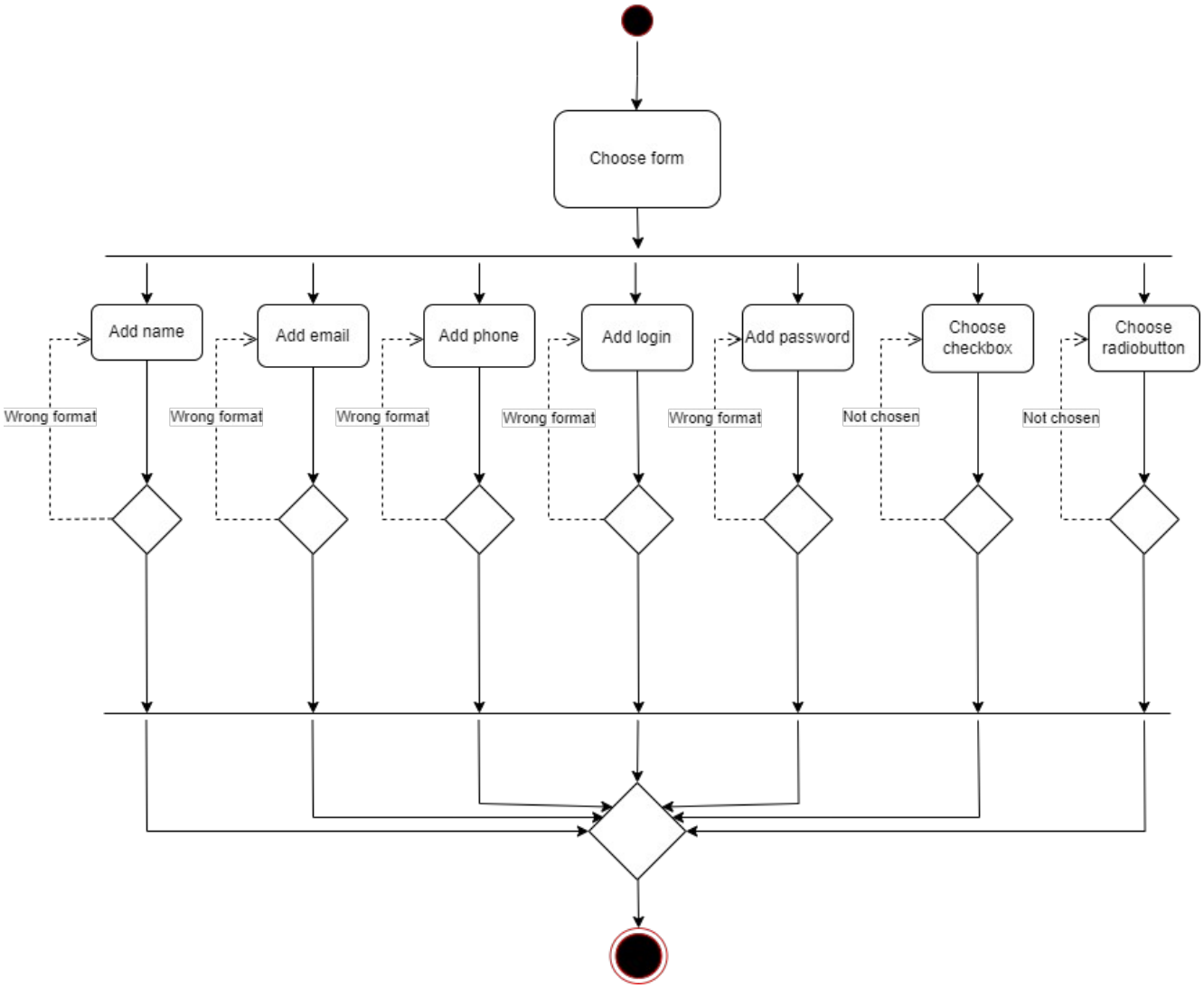


Рисунок 2.3 – Діаграма діяльності UML

2.4 Нефункціональні вимоги

Нефункціональні вимоги - це глобальні обмеження на програмну систему, наприклад вартість розробки, експлуатаційні витрати, продуктивність, надійність, ремонтопридатність, переносимість, міцність і т.д.

Вимога, яка стосується не функціональності, а таких атрибутів, як надійність, ефективність, зручність використання, ремонтопридатність і переносимість.

У системної інженерії та розробці вимог дисфункція - це вимога, що визначає критерії, які можуть бути використані для оцінки роботи системи, а не конкретні характеристики поведінки.

Програмне забезпечення має нефункціональні вимоги, необхідні для створення позитивного досвіду для користувача. І якщо ви не виконаєте ці вимоги, у вас можуть виникнути затримки у розкладі, бюджетні проблеми та незадоволені клієнти.

У проекті, в якому відсутні нефункціональні вимоги, також може бракувати ясності, що може призвести до розпливчастих рамок проекту та розриву між дизайнером та очікуваннями клієнта.

Нефункціональні вимоги усувають розрив між тим, що, на думку розробників, потрібно клієнтам і тим, що клієнти насправді хочуть.

Потрібно створити зручний web-додаток для перевірки HTML-форми, де користувач зможе не тільки перевірити валідацію, а й побачити як вона реалізована програмно.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Набір інструментів використаних для розробки

Для розробки було використано HTML, CSS, JavaScript.

3.1.1 Огляд HTML

HTML (від англійської HyperText Markup Language) - це мова гіпертекстової розмітки сторінки. Використовується для того, щоб дати браузеру зрозуміти, як потрібно відображати завантажений сайт.

Мова складається з тегів - це своєрідні команди, які перетворюються на

візуальні об'єкти в браузері користувача. Наприклад, тег <img> використовується для розміщення зображень на сторінці. Він має обов'язковий атрибут src, в якому вказується посилання на файл.

HTML – це мова розмітки для кодування веб-сторінок. Він був розроблений британським вченим сером Тімом Бернерсом-Лі у лабораторії ядерної фізики CERN у Швейцарії у 1980-х роках. Теги розмітки HTML визначають елементи документа, такі як заголовки, абзаци та таблиці. Вони розмічають документ для відображення комп'ютерної програми, відомої як веб-браузер. Браузер інтерпретує теги, відображаючи заголовки, абзаци та таблиці в макеті, адаптованому до розміру екрана та доступним шрифтам.

3.1.2 Огляд CSS

Каскадні таблиці стилів, ласкаво звані CSS, - це проста мова дизайну, покликана спростити процес створення презентабельних веб-сторінок.

CSS відповідає за зовнішній вигляд та оформлення веб-сторінки. За допомогою CSS можна керувати кольором тексту, стилем шрифтів, інтервалом між абзацами, розміром та розташуванням колонок, фоновими зображеннями та кольорами, дизайном макета, варіаціями відображення для різних пристроїв та розмірів екрану, а також безліччю інших ефектів.

CSS легко вивчити і зрозуміти, але забезпечує потужний контроль над поданням HTML-документа. Найчастіше CSS використовується у поєднанні з мовами розмітки HTML або XHTML.

Переваги CSS: CSS заощаджує час - ви можете написати CSS один раз і потім повторно використовувати той самий аркуш на декількох HTML-сторінках. Ви можете визначити стиль для кожного елемента HTML та застосувати його до будь-якої кількості веб-сторінок.

Сторінки завантажуються швидше. Якщо ви використовуєте CSS, вам не потрібно щоразу написати атрибути тегів HTML. Достатньо написати одне правило CSS для тега і застосувати його до всіх вхід цього тега. Таким чином, менше коду означає швидший час завантаження.

Простота обслуговування. Для внесення глобальних змін достатньо змінити

стиль і всі елементи на всіх веб-сторінках будуть оновлені автоматично.

Перевага стилів над HTML: CSS має набагато ширший набір атрибутів, ніж HTML, тому ви можете надати HTML-сторінці набагато кращий вид у порівнянні з HTML-атрибутами.

Сумісність із кількома пристроями: таблиці стилів дозволяють оптимізувати вміст для кількох типів пристроїв. Використовуючи один і той же документ HTML, можна надати різні версії веб-сайту для портативних пристроїв, таких як КПК і мобільні телефони, або для друку.

Глобальні веб-стандарти. Зараз атрибути HTML старіють і рекомендується використовувати CSS. Тому гарною ідеєю почне використовувати CSS на всіх HTML-сторінках, щоб зробити їх сумісними з майбутніми браузерами.

Хто створює та підтримує CSS? CSS створюється та підтримується групою людей у рамках W3C під назвою Робоча група CSS. Робоча група CSS створює документи, які називаються специфікаціями. Коли специфікація обговорюється та офіційно ратифікується членами W3C, вона стає рекомендацією.

Ці ратифіковані специфікації називаються рекомендаціями, тому що W3C не контролює фактичної реалізації мови. Незалежні компанії та організації створюють це програмне забезпечення.

Консорціум Всесвітньої павутини, або W3C, - це група, яка дає рекомендації про те, як працює Інтернет та як він має розвиватися.

3.1.3 Огляд JavaScript

JavaScript – це динамічна мова комп'ютерного програмування. Він легкий і найчастіше використовується у складі веб-сторінок, реалізації яких дозволяють сценарієм на стороні клієнта взаємодіяти з користувачем та робити динамічні сторінки. Це мова програмування, що інтерпретується, з об'єктно-орієнтованими можливостями.

JavaScript спочатку був відомий як LiveScript, але компанія Netscape змінила його назву на JavaScript, можливо через ажіотаж, викликаний Java. JavaScript вперше з'явився у Netscape 2.0 у 1995 році під назвою LiveScript. Ядро мови

загального призначення було вбудоване в Netscape, Internet Explorer та інші браузери.

3.2 Розробка web-додатку

Для початку створимо 3 файли: index.html, script.js, style.css та під’єднаємо до першого файлу інші два.

```
8     <link href="style.css" rel="stylesheet" type="text/css">
```

Рисунок 3.1 – Підключення файлу css до html

```
66    <script src="script.js"></script>
```

Рисунок 3.2 – Підключення файлу js до html

Після цього у файлі index.html створимо скелет нашої HTML-форми. Використовуємо найпоширеніші елементи форм:

1. Чекбокси
2. Перемикачі
3. Поле вводу електронної адреси
4. Поле вводу номеру телефона
5. Поле вводу Ім’я
6. Поле вводу логіну
7. Поле вводу пароля

До кожного елементу форми будемо використовувати подію onchange, що з’являється при зміні значення елементу форми, типу текстового поля або списку. Також подія спрацьовує, коли поле втрачає фокус.

```html
<div id="form1"; style="width:max-content ;">
    <form class="form">
    <div class="mainer">
    <input onchange="Validate_radiobutton()" type="radio" class="maininputs radiob" id="radiobs" name="radiob"/>
    <input onchange="Validate_radiobutton()" type="radio" class="maininputs radiob" id="radiobs1" name="radiob"/>
    <p id="validate_radio">Перемикач не може бути не вибраний</p>
    </div>
    <div class="mainer">
    <input onchange="Validate_checkbox()" type="checkbox" id="checkers" class="maininputs checkers"/>
    <input onchange="Validate_checkbox()" type="checkbox" id="checkers1" class="maininputs checkers"/>
    <p id="validate_checks">Чекбокси не можуть бути не вибрані</p>
    </div>
    <div class="inputsdiv">
        <div class="flexinput">
    <input onchange="Validation_mail()" placeholder="example@mail.com" type="email" name="mail" id="mail" class="maininputs inputs_box mail">
    <p id="validate_mail">Електронна адреса не заповнена</p>
        </div>
        <div class="flexinput">
    <input onchange="Validation_tel()"  placeholder="380000000000" type="tel" name="tel" id="tel" class="maininputs inputs_box phone">
    <p id="validate_tel">Номер не заповнений</p>
</div>
<div class="flexinput">
    <input onchange="Validation_name()" placeholder="Григорій" type="text" name="name" id="name" class="maininputs inputs_box name">
    <p id="validate_name">Ім'я не заповнене</p>
</div>
<div class="flexinput">
    <input onchange="Validation_password()"  placeholder="********" type="password" name="password" id="password" class="maininputs inputs_box password">
    <p id="validate_password">Пароль не заповнений</p>
</div>
<div class="flexinput">
    <input onchange="Validation_login()"  placeholder="student_007" type="text" name="login" id="login" class="maininputs inputs_box login">
    <p id="validate_login">Логін не заповнений</p>
</div>
```

У файлі script.js створимо власні словники для перевірки елементів HTML-форми.

```
JS script.js > ...
1   var symbols = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz@.';
2   var symbRu = 'АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеёжзийклинопрстуфчцчшщьыъэюя';
3   var symbUa = 'АБВГДЕЄЖЗИІЙЇКЛМНОПРСТУФХЦЧШЩЬЮЯабвгдеєжзиійїклмнопрстуфхцчшщьюя';
4   var emailsymb = '@';
5   var numbers = '0123456789';
```

Далі створимо функцію валідації для поля електронної адреси:

```
function Validation_mail(){
    let text = document.getElementById("validate_mail");
    let mail = document.getElementById("mail");
    if(mail.value.length==0){
        mail.style.backgroundColor = "#ff000069"
        text.textContent = "Електронна адреса не заповнена"
        return false;
    }
    if(mail.value.length<4)
    {
        mail.style.backgroundColor = "#ff000069"
        text.textContent = "Електронна адреса не може мати менше 4 символів";
        return false;
    }
    else
    {
        let valid = massiveEqualmail(mail.value);
        if(valid)
        {
            mail.style.backgroundColor = "#7eff7e52"
            text.textContent = "Електронна адреса валідна!";
        }
        else{
            mail.style.backgroundColor = "#ff000069"
            text.textContent = "Ваша електронна адреса не може мати кирилицю або не має символів @ .";
        }
    }
}
```

Створюємо функцію для перевірки поля номеру телефону:

```
function Validation_tel(){
    let text = document.getElementById("validate_tel");
    let phone = document.getElementById("tel");
    let phonecounter = 0;
    let phonetrue = false;
    if(tel.value.length==0){
        phone.style.backgroundColor = "#ff000069"
        text.textContent = "Номер не заповнений"
        return false;
    }
    if(phone.value.length>12 || phone.value.length<12)
    {
        phone.style.backgroundColor = "#ff000069"
        text.textContent = "Номер повинен мати 12 цифр";
        return false;
    }
    else
    {
        for(let q = 0; q<phone.value.length; q++)
        {
            for(let i=0;i<numbers.length;i++)
            {
                if(phone.value[q]==numbers[i]){
                    phonecounter++;
                }
            }
        }
        if(phonecounter==12){
            phonetrue = true;
        }
        else{
            phonetrue = false;
        }
        if(phonetrue)
        {
            phone.style.backgroundColor = "#7eff7e52"
            text.textContent = "Номер валідний!";
        }
        else
```

Створюємо функцію для перевірки поля вводу паролю:

```
function Validation_password(){
    let text = document.getElementById("validate_password");
    let password = document.getElementById("password");
    let pass_value = password.value;
    if(password.value.length==0){
        password.style.backgroundColor = "#ff000069"
        text.textContent = "Пароль не заповнений"
        return false;
    }
    if(password.value.length<6){
        password.style.backgroundColor = "#ff000069"
        text.textContent = "Пароль не може бути корочший ніж 6 символів"
        return false;
    }
    if(pass_value.indexOf(' ') != -1){
        password.style.backgroundColor = "#ff000069"
        text.textContent = "Ваш пароль має символ відступу"
        return false;
    }
    password.style.backgroundColor = "#7eff7e52"
    text.textContent = "Пароль валідний"
}
```

Створюємо функцію для перевірки поля вводу логіну:

```
function Validation_login(){
    let text = document.getElementById("validate_login");
    let login = document.getElementById("login");
    for(let i=0;i<login.value.length;i++)
    {
        for(let q = 0; q<symbRu.length;q++)
        {
            if(login.value[i]==symbRu[q]){
                login.style.backgroundColor = "#ff000069"
                text.textContent = "Логін не повинен мати символів кирилиці"
                return false;
            }
        }
        for(let q = 0; q<symbUa.length;q++)
        {
            if(login.value[i]==symbRu[q]){
                login.style.backgroundColor = "#ff000069"
                text.textContent = "Логін не повинен мати символів кирилиці"
                return false;
            }
        }
    }
    if(login.value.indexOf(' ') != -1){
        login.style.backgroundColor = "#ff000069"
        text.textContent = "Логін не повинен мати символ відступу"
        return false;
    }
    if(login.value.length==0){
        login.style.backgroundColor = "#ff000069"
        text.textContent = "Логін не заповнений"
        return false;
    }
    login.style.backgroundColor = "#7eff7e52"
    text.textContent = "Логін валідний"
}
```

Створюємо функцію для перевірки поля вводу імені:

```
function Validation_name(){
    let text = document.getElementById("validate_name");
    let name = document.getElementById("name");
    let errorcounter = 0;
    let symbcounter = 0;
    if(name.value.length==0){
        name.style.backgroundColor = "#ff000069"
        text.textContent = "Ім'я не заповнене"
        return false;
    }
    if(name.value.length<1){
        name.style.backgroundColor = "#ff000069"
        text.textContent = "Ім'я не може бути короче ніж 1 символ"
        return false;
    }
    for(let q = 0; q<name.value.length; q++)
    {
        for(let i=0;i<symbols.length;i++)
        {
            if(name.value[q]==symbols[i]){
                symbcounter++;
            }
        }
    }
    if(symbcounter!=name.value.length){
        symbcounter = 0;
        errorcounter++;
    }
    else{
        name.style.backgroundColor = "#7eff7e52"
        text.textContent = "Ім'я валідне"
        return true;
    }
```

Наступною буде функція перевірки поля вводу електронної адреси користувача:

```
function massiveEqualmail(text){
    let counter_s = 0;
    let counter_d = 0;
    for(let i=0;i<text.length;i++)
        {
            for(let q = 0; q<symbRu.length;q++)
            {
                if(text[i]==symbRu[q]){
                    return false;
                }

            }
            for(let q = 0; q<symbUa.length;q++)
            {
                if(text[i]==symbUa[q]){
                    return false;
                }

            }
            if(text[i]==emailsymb){
                counter_s++;
            }
            else if(text[i]=='.')
            {
                counter_d++;
            }
        }
        if(counter_d>0 && counter_s>0){
            return true;
        }
        else{
            return false;
        }
}
```

Функція перевірки чекбоксів та перемикачей:

```
function Validate_checkbox(){
let text = document.getElementById("validate_checks");
let checkbox = document.getElementById("checkers");
let checkbox1 =  document.getElementById("checkers1");
    if(checkbox.checked || checkbox1.checked){
        text.textContent = "Чекбокси валідні"
        return true;
    }
    else{
        text.textContent = "Чекбокси не можуть бути не вибрані"
        return false;
    }
}

function Validate_radiobutton(){
    let text = document.getElementById("validate_radio");
    let radiobs = document.getElementById("radiobs");
    let radiobs1 =  document.getElementById("radiobs1");
    if(radiobs.checked || radiobs1.checked){
        text.textContent = "Перемикач валідний"
        return true;
    }
    else{
        text.textContent = "Перемикач не може бути не вибраний"
        return false;
    }
}
```

Написавши основний функціонал, додатку залишається лише додати стилі та опис для кожного типового елементу HTML-форми.

Додаток призначений для користувачів, які тільки починаються своє знайомство з Frontend розробкою(HTML, CSS, JS).

Зайшовши на веб-сторінку, користувачі зможуть одразу побачити типові елементи форм, реалізацію їх перевірки і стислий довідник до кожного елементу з описом їх аргументів та типових умов валідації.

ВИСНОВКИ

Дана робота була спрямована на проектування та реалізацію web-додатку для перевірки HTML-форми.

1. Було висвітлено актуальність обраної теми. Був проведен аналіз аналогів.

2. Проведено проектування та розробка додатку.

3. Описано інструменти та програмні засоби, що були застосовані для розробки додатку.

Результати дослідження бакалаврської роботи апробовані на IV Міжнародній студентській науковій конференції «Наука сьогодення: від досліджень до стратегічних рішень» (м. Івано-Франківськ, Україна).

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Dave Thomas. Programming Ruby. The Pragmatic Programmers' Guide. Second Edition./ 2004 – с. 107-144.

2. Офіційний веб-сайт RUBY. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.ruby-lang.org/

3. Офіційний веб-сайт Python. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.python.org/

3. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.djangoproject.com/

4. Elisabeth Robson. Head First JavaScript Programming: A Brain-Friendly Guide./ 2016 – с. 37-118

5. Офіційний веб-сайт Java. [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.java.com/

6. Alan Forbes. The Joy of PHP Programming: A Beginner's Guide to Programming Interactive Web Applications with PHP and MySQL./ 2009 – с. 23-89.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-додатку для перевірки HTML-форми мовою JavaScript

Виконав студент 4 курсу
групи ПД-43
Лозінський Станіслав Вадимович
Керівник роботи
Коваленко Данило Сергійович
Асистент

Київ – 2022

АНАЛОГИ

Мова програмування RUBY

Мова програмування PHP

Мова програмування Python

Мова програмування Java

ПОРІВНЯННЯ АНАЛОГІВ

Мова програмування	RUBY	Python	Java	PHP	• JavaScript
Інтерпретатор командної строки	+	+	-	+	+
Статична типізація	-	-	+	-	-
Динамічна типізація	+	+	-	+	+
Явна типізація	-	+	+	+	-
Неявна типізація	+	+	-	+	+
Об'єктно-орієнтовна парадигма	+	+	+	+	+
Наявність бібліотек для роботи з графікою та мультимедіа	-	+	+	+/-	+

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи**: спрощення реалізації та перевірки HTML-форми за рахунок використання web-додатку
- **Об'єкт дослідження:** перевірка HTML-форми на стороні клієнта
- **Предмет дослідження:** технології мови JavaScript для перевірки HTML-форми

ТЕХНІЧНЕ ЗАВДАННЯ

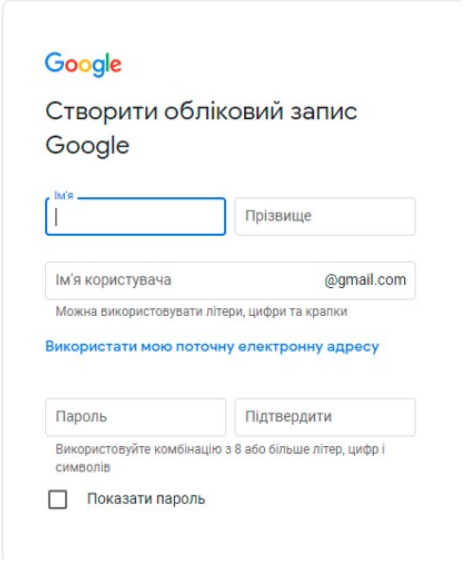
1. Розробити web-додаток з типовими елементами HTML-форми
2. Зробити перевірку HTML-форми мовою JavaScript
3. Створити у web-додатку блок-пояснення до кожного з типового елементу HTML-форми

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

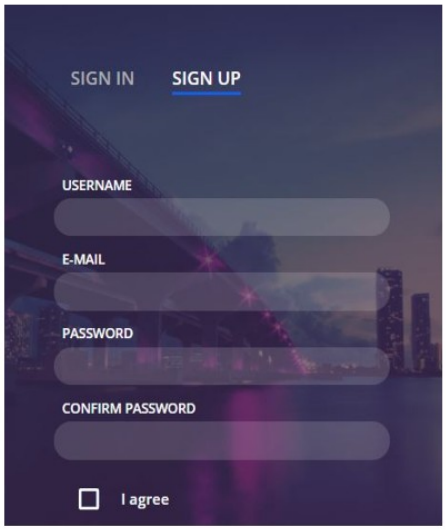
Типові елементи HTML-форми та умови їх валідації

Елементи HTML-форми	Умови валідації
Radiobutton	Обрано один із варінтів
Checkbox	Обрано один чи декілька варіантів
Name	Не містить цифр; написано кирилицею/латиницею; не містить спеціальних символів
Email	Не має кирилиці; написано у форматі «xxx@xx.xx»
Password	Містить велику літеру; містить цифри; не має кирилиці; не містить спеціальних символів; довжина від 8 до 16 символів
Login	Не має кирилиці; не містить спеціальних символів
Phone number	Містить лише цифри; довжина 10 символів

Екранні форми типових HTML-форм

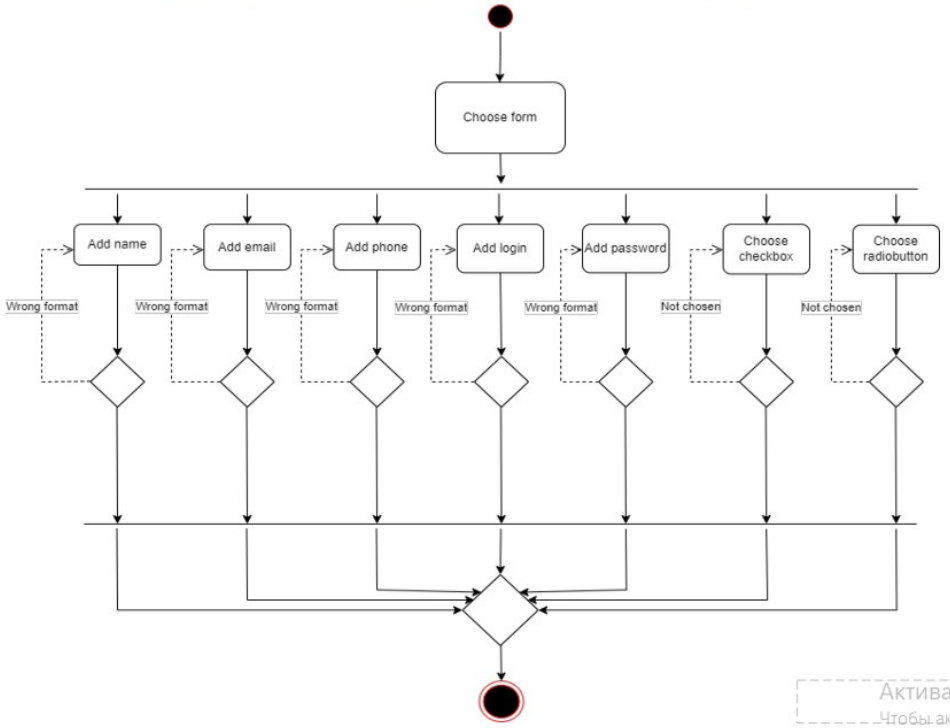


Приклад 1

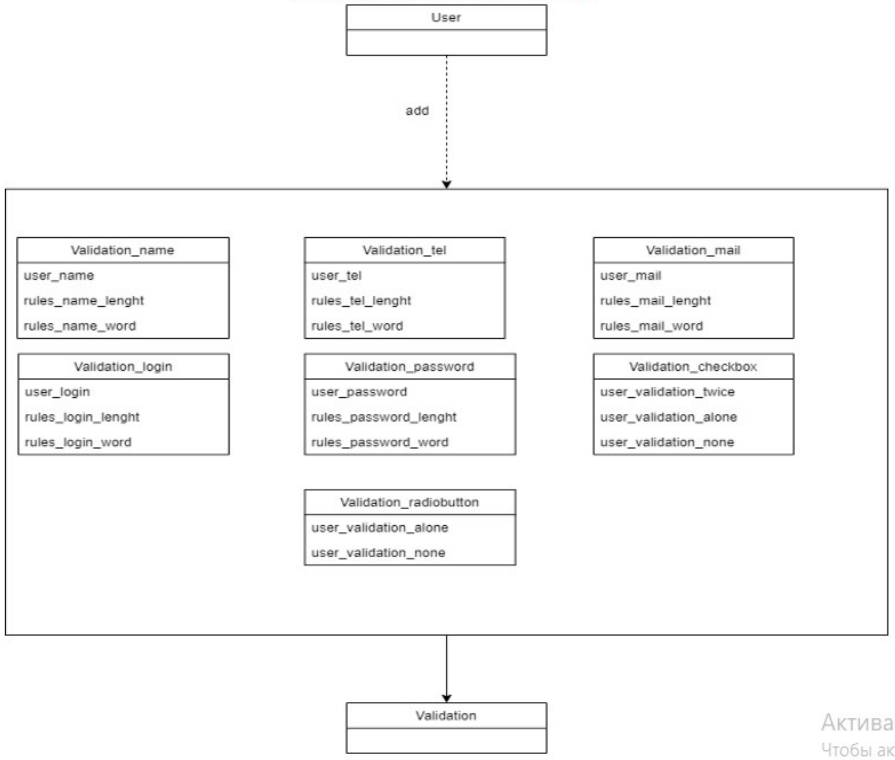


Приклад 2

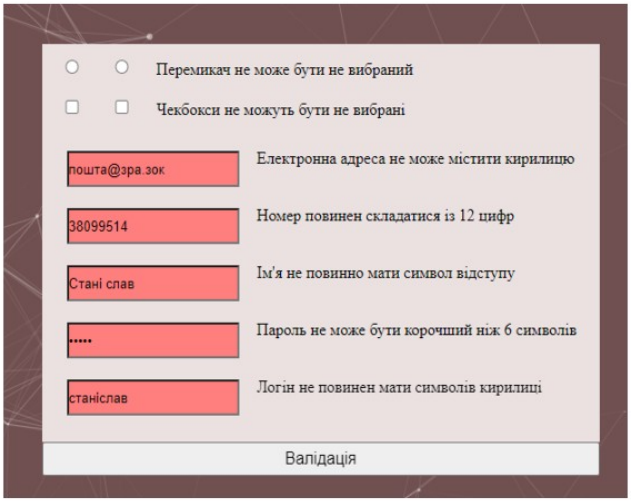
Діаграма перевірки HTML-форми



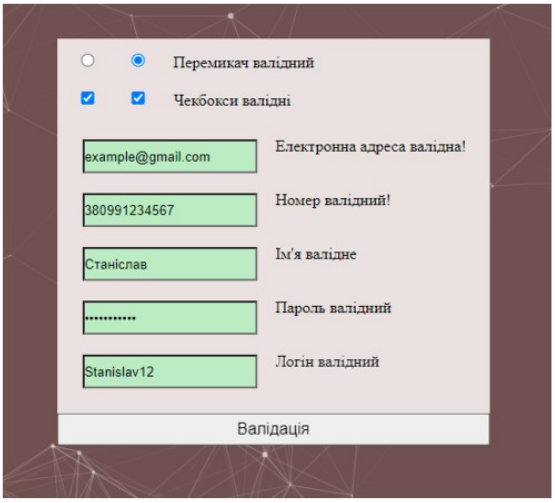
Діаграма компонентів



Екранні форми валідації



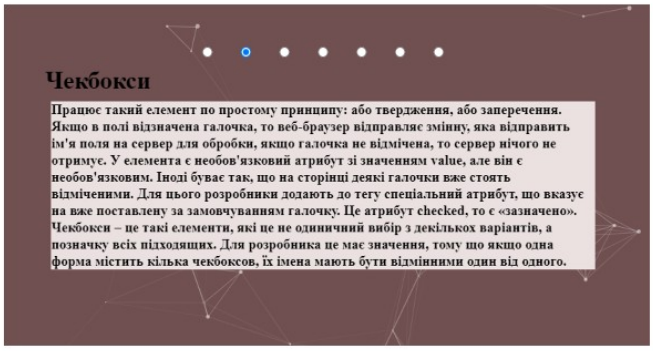
1. Заповнення невалідними даними



2. Заповнення валідними даними

Екранні форми довідника до елементів HTML-форми

Довідник Radiobutton



Довідник Checkbox

Апробація результатів дослідження

С.В. Лозінський використання мови програмування JavaScript для перевірки HTML-форми / Лозінський С.В. // IV Міжнародна студентська конференція «Наука сьогодення: від досліджень до стратегічних рішень». Подано до друку // м. Івано-Франківськ

ВИСНОВКИ

1. Проведено аналіз найпоширеніших мов програмування, які використовують для валідації HTML-форм. Розглянуто аналоги
2. Створений web-додаток з типовою HTML-формою
3. Реалізовано перевірку елементів HTML-форми за допомогою мови JavaScript
4. Створений опис елементів HTML-форми

ДЯКУЮ ЗА УВАГУ!