Міністерство освіти і науки України
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
з дисципліни
“Технологія створення програмних продуктів”

2023

Методичні вказівки до виконання лабораторних робіт з дисципліни “Технологія створення програмних продуктів” / Укл.: А. О. Олійник, Т. О. Колпакова, В. М. Льовкін, С. Д. Леощенко. – Запоріжжя : НУ «Запорізька політехніка», 2023. – 99 с.

Укладачі: А. О. Олійник, д.т.н., професор
Т. О. Колпакова, к.т.н., доцент
В. М. Льовкін, к.т.н., доцент
С. Д. Леощенко, доктор філософії, ст. викладач

Рецензент: С. О. Субботін, д.т.н., професор

Відповідальний за випуск: С. О. Субботін, д.т.н., професор

Затверджено
на засіданні кафедри
програмних засобів

Протокол № 1
від “17” серпня 2023 р.

ЗМІСТ

ВСТУП

Дане видання призначене для вивчення та практичного освоєння студентами усіх форм навчання основ технології створення програмних продуктів.

Відповідно до графіка студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою.

Для одержання заліку з кожної роботи студент здає викладачу оформлений звіт, а також демонструє на екрані комп'ютера результати виконання лабораторної роботи.

Звіт має містити:

– титульний аркуш;

– тему та мету роботи;

– завдання до роботи;

– лаконічний опис теоретичних відомостей;

– результати виконання лабораторної роботи;

– змістовний аналіз отриманих результатів та висновки.

Звіт виконують на білому папері формату А4 (210 × 297 мм) або подають в електронному вигляді.

Під час співбесіди при захисті лабораторної роботи студент повинний виявити знання про зміст роботи та методи виконання кожного етапу роботи, а також вміти продемонструвати результати роботи на конкретних прикладах. Студент повинний вміти правильно аналізувати отримані результати. Для самоперевірки при підготовці до виконання і захисту роботи студент повинен відповісти на контрольні запитання, наведені наприкінці опису відповідної роботи.

1 ЛАБОРАТОРНА РОБОТА № 1 ЗБИРАННЯ ТА АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ПРОДУКТУ

1.1 Мета роботи

Навчитися виконувати збирання, аналіз та документування вимог до програмного забезпечення.

1.2 Короткі теоретичні відомості

Одним з ключових понять проєктування інформаційних систем є життєвий цикл (ЖЦ) проєкту. В загальному випадку, життєвий цикл визначається моделлю й описується у формі методології (методу). Модель або парадигма життєвого циклу визначає загальну організацію ЖЦ і, як правило, основні його фази та принципи переходу між ними. Методологія (метод) визначає комплекс робіт, їх детальний зміст і рольову відповідальність спеціалістів на всіх етапах вибраної моделі ЖЦ; рекомендує практики, які дозволяють максимально ефективно використовувати відповідну методологію та її модель.

Життєвий цикл програмного забезпечення – період часу, що починається з моменту прийняття рішення про необхідність створення програмного продукту і закінчується в момент його повного вилучення з експлуатації. Цей цикл – процес побудови і розвитку програмного забезпечення (ПЗ).

Загалом можна виділити 4 основних етапи (стадії) в процесі створення ПЗ, які представлені на рис. 1.1.

Ініціація – відбувається висунення ідеї, а також підготовка проєктних документів. Проводяться детальне обґрунтування, а також маркетингові дослідження, які стануть підставою для реалізації наступних стадій.

Планування – визначення термінів реалізації задуму, поділ даних процесів на конкретні етапи, а також призначення виконавців та відповідальних осіб.

Виконання – починається відразу ж після того, як були затверджені плани. Мається на увазі реалізація в повному обсязі всіх намічених дій.

Завершення – аналіз отриманих даних і контроль на предмет відповідності їх запланованим. Даний обов'язок в більшості випадків покладається на керівництво.

Рисунок1.1 – Основні етапи створення ПЗ

Використання певної моделі ЖЦ дозволяє визначитися з основними моментами процесу замовлення, розроблення та супроводу ПЗ навіть недосвідченому програмісту. Також використання моделей дозволяє чітко зрозуміти, в який період переходити від версії до версії, які дії з удосконалення виконувати, на якому етапі. Знання про закономірності розвитку програмного продукту, які відбиваються в обраній моделі ЖЦ, дозволяють отримати надійні орієнтири для планування процесу розроблення та супроводу ПЗ, економно витрачати ресурси та підвищувати якість управління усіма процесами.

Також моделі життєвого циклу є основою знань технологій програмування та інструментарію, що їх підтримує. Будь-яка технологія базується на певних уявленнях про життєвий цикл та організує свої методи та інструменти навколо фаз та етапів ЖЦ.

До цього часу моделі ЖЦ розвиваються і модифікуються, уточнюючи та доповнюючи дві базові моделі – каскадну та ітеративну. Ці зміни обумовлені потребою організаційної та технологічної підтримки проєктів з розроблення ПЗ.

1.2.1 Моделі життєвого циклу

Модель життєвого циклу – це структура, що складається із процесів, робіт та задач, які включають в себе розробку, експлуатацію і супровід програмного продукту. Вона охоплює життя системи від визначення вимог до неї до припинення її використання. На сьогодні найбільшого розповсюдження набули наступні моделі:

– каскадна;

– інкрементна;

– спіральна.

Каскадна модель (одноразовий прохід, водоспадна або класична стратегія) припускає лінійну послідовність виконання стадій створення інформаційної системи (рис. 1.2). Іншими словами, перехід з однієї стадії на наступну відбувається тільки після того, як буде повністю завершена робота на поточній.

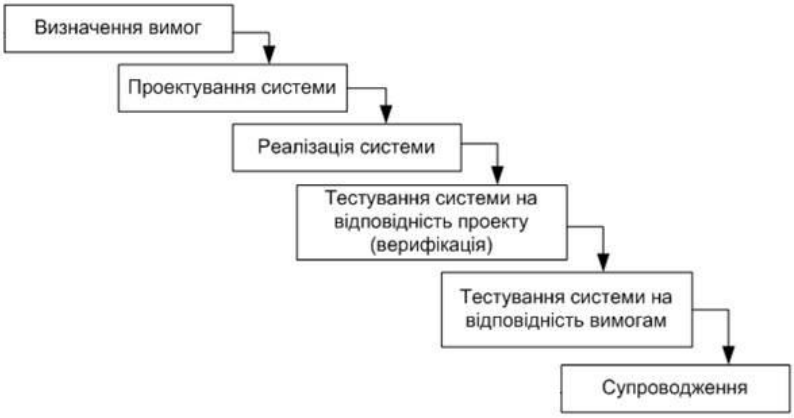

Рисунок 1.2 – Каскадна модель ЖЦ

Ця модель застосовується при розробці ПЗ, для якого на самому початку розробки можна достатньо точно і повно сформулювати всі вимоги.

Переваги моделі:

– на кожному етапі формується закінчений набір документації, програмного та апаратного забезпечення, що відповідає критеріям повноти та узгодженості;

– стадії, що виконуються в чіткій послідовності, дозволяють впевнено планувати терміни виконання робіт і відповідні ресурси (грошові, матеріальні і людські).

Недоліки моделі:

– реальний процес розробки ПЗ рідко повністю вкладається в таку жорстку схему. Особливо це відноситься до розробки нетипових та новаторських проєктів;

– заснована на точному формулюванні вихідних вимог до ПЗ. Реально на початку проєкту вимоги замовника визначені лише частково;

– основний недолік – результати розробки доступні замовнику тільки в кінці проєкту. У разі неточного викладу вимог або їх зміни протягом тривалого періоду розробки ПЗ замовник отримує систему, яка не задовольняє його потребам.

Інкрементна стратегія припускає розробку інформаційної системи з лінійною послідовністю стадій, але в кілька інкрементів (версій), тобто із запланованим поліпшенням продукту.

Рисунок 1.3 – Інкрементна модель ЖЦ

На початку роботи над проєктом визначаються всі основні вимоги до системи, після чого виконується її розробка у вигляді послідовності версій. При цьому кожна версія є закінченим і працездатним продуктом. Перша версія реалізує частину запланованих можливос-

тей, наступна версія реалізує додаткові можливості і т. д., поки не буде отримана повна система.

Дана модель ЖЦ характерна при розробці складних і комплексних систем, для яких є чітке бачення (як з боку замовника, так і з боку розробника) того, яким має бути кінцевий результат. Розробка версіями ведеться з різних причин:

– відсутність у замовника можливості відразу профінансувати весь проєкт (за умови великої ціни);

– відсутності у розробника необхідних ресурсів для реалізації складного проєкту в стислі терміни;

– вимоги поетапного впровадження та освоєння продукту кінцевими користувачами. Впровадження всієї системи відразу може викликати у її користувачів неприйняття і тільки «загальмувати» процес переходу на нові технології.

Переваги і недоліки цієї стратегії такі ж, як і у класичної. Але на відміну від класичної стратегії замовник може раніше побачити результати. Вже за результатами розробки та впровадження першої версії він може незначно змінити вимоги до розробки, відмовитися від неї або запропонувати розробку більш досконалого продукту з укладенням нового договору.

Спіральна стратегія (еволюційна або ітераційна модель) передбачає розробку у вигляді послідовності версій, але на початку проєкту визначені не всі вимоги. Вимоги уточнюються в результаті розробки версій.

Дана модель життєвого циклу характерна при розробці новаторських (нетипових) систем. На початку роботи над проєктом у замовника і розробника немає чіткого бачення кінцевого продукту (вимоги не можуть бути чітко визначені) або стовідсоткової впевненості в успішній реалізації проєкту (ризики дуже великі). У зв'язку з цим приймається рішення розробки системи по частинах з можливістю зміни вимог або відмови від її подальшого розвитку. Як видно з рис. 1.4 розвиток проєкту може бути завершено не тільки після стадії впровадження, але і після стадії аналізу ризику.

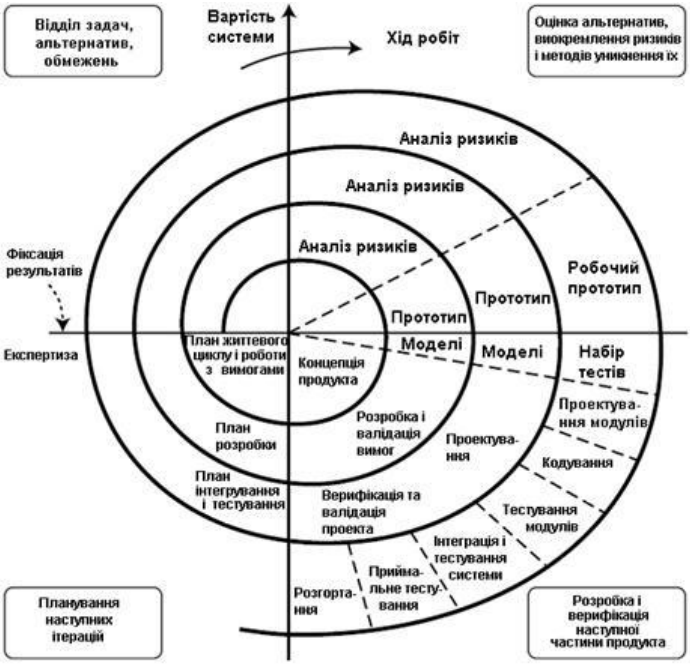

Рисунок 1.4 – Спіральна модель ЖЦ

Переваги моделі:

– дозволяє якнайшвидше показати користувачам системи працездатний продукт, тим самим активізуючи процес уточнення і доповнення вимог;

– допускає зміну вимог при розробці інформаційної системи, що характерно для більшості розробок, в тому числі й типових;

– забезпечує більшу гнучкість в управлінні проєктом;

– дозволяє отримати більш надійну і стійку систему. В процесі розвитку системи помилки і слабкі місця виявляються і виправляються на кожній ітерації;

– дозволяє вдосконалювати процес розробки – аналіз, що виконується на кожній ітерації, дозволяє проводити оцінку того, що має бути змінено в організації розробки, та поліпшити її на наступній ітерації;

– зменшуються ризики замовника. Замовник може з мінімальними для себе фінансовими втратами завершити розвиток неперспективного проєкту.

Недоліки моделі:

– збільшується невизначеність у розробника в перспективах розвитку проєкту. Цей недолік випливає з попередньої переваги моделі;

– ускладнені операції часового і ресурсного планування всього проєкту в цілому. Для вирішення цієї проблеми необхідно ввести часові обмеження на кожну зі стадій життєвого циклу. Перехід здійснюється відповідно до плану, навіть якщо не вся запланована робота виконана. План складається на основі статистичних даних, отриманих в попередніх проєктах, і особистого досвіду розробників.

Протягом значного часу саме спіральна модель ЖЦ демонструвала найбільшу придатність. Проте, навіть такий підхід через недостатню гнучкість потребував змін. Через це з часом з'явився окремий клас методологій розробки – **гнучка розробка програмного забезпечення.**

1.2.1.1 Гнучка розробка програмного забезпечення

Гнучка розробка програмного забезпечення (agile-методи) відокремилась як клас методологій розробки програмного забезпечення, що базується на ітеративній розробці, в якій вимоги та розв'язки еволюціонують через співпрацю між багатофункціональними командами, здатними до самоорганізації.

Гнучка розробка припускає, що при реалізації проєкту не потрібно спиратися тільки на заздалегідь створені докладні плани. Важливо орієнтуватися на постійно мінливі умови зовнішнього і внутрішнього середовища і враховувати зворотний зв'язок від замовників та користувачів. Це заохочує розробників та інженерів експериментувати і шукати нові рішення, не обмежуючи себе жорсткими рамками і стандартами.

До популярних agile-підходів відносяться Scrum і Kanban.

Scrum – це, так би мовити, «підхід структури». Над кожним проєктом працює універсальна команда фахівців, до якої приєднується ще дві людини: власник продукту і scrum-майстер. Перший з'єднує команду з замовником і стежить за розвитком проєкту; це не формальний керівник команди, а швидше куратор. Другий допомагає першому

організувати бізнес-процес: проводить загальні збори, вирішує побутові проблеми, мотивує команду і стежить за дотриманням scrum-підходу.

Scrum-підхід ділить робочий процес на рівні терміни (спринти) – зазвичай це періоди від тижня до місяця, залежно від проєкту і команди. Перед спринтом формулюються завдання на даний спринт, в кінці – обговорюються результати, а команда починає новий спринт. Спринти дуже зручно порівнювати між собою, що дозволяє управляти ефективністю роботи.

Kanban же – це «підхід балансу». Його завдання – збалансувати різних фахівців усередині команди та уникнути ситуації, коли частині команди треба працювати постійно, а інша не може розпочати роботу через відсутність нових завдань.

В Kanban немає ролей власника продукту і scrum-майстра – вся команда єдина. Бізнес-процес ділиться не на універсальні спринти, а на стадії виконання конкретних завдань: «Планування», «Розробка», «Тестування», «Завершення» та ін.

Головний показник ефективності в Kanban – середній час проходження завдання по дошці. Завдання пройшло швидко – команда працювала продуктивно і злагоджено. Завдання затягнулося – треба думати, на якому етапі та чому виникли затримки, чию роботу треба оптимізувати.

1.2.2 Аналіз вимог

На сьогоднішній день, процес ЖЦ, на якому фіксуються вимоги на розробку системи, є визначальним для завдання функцій, термінів та вартості робіт, а також показників якості, яких необхідно досягти в процесі розробки. Висунення вимог проводиться шляхом обговорення проєкту, аналізу предметної області та визначення підходів до проєктування проміжних продуктів на етапах ЖЦ.

Вимоги відображають потреби людей (замовників, користувачів, розробників), зацікавлених у створенні ПЗ. Замовник і розробник спільно проводять обговорення проблем проєкту, збір вимог, їх аналіз, перегляд, визначення необхідних обмежень і документування.

Аналіз вимог полягає у визначенні потреб та умов, які висуваються щодо нового, чи зміненого продукту, враховуючи можливо конфліктні вимоги різних замовників.

Аналіз вимог є критичним для успішного розроблення проєкту. Вимоги мають бути задокументованими, вимірними, тестованими, пов'язаними з бізнес-потребами, і описаними з рівнем деталізації достатнім для конструювання системи.

1.2.3 Етапи аналізу вимог

Аналіз вимог включає наступні види діяльності:

Виявлення (збирання) вимог – задача комунікації з користувачами для визначення їх вимог.

Аналіз вимог – виявлення недоліків вимог (неточностей, неповноти, неоднозначностей чи суперечностей) і їх виправлення.

Запис вимог – документування вимог в різних формах, таких як опис звичайною мовою, прецедентами, користувацькими історіями, чи специфікаціями процесу.

1.2.4 Класифікація вимог

Вимоги споживача представляють собою вирази фактів та припущень, які описують очікування від системи в термінах цілей, середовища, обмежень та міри ефективності й придатності.

Архітектурні вимоги пояснюють, що має бути зроблено ідентифікацією необхідної системної архітектури.

Структурні вимоги пояснюють, що має бути зроблено ідентифікацією необхідної структури системи.

Поведінкові вимоги пояснюють, що має бути зроблено ідентифікацією необхідної поведінки системи.

Функціональні вимоги пояснюють, що має бути зроблено ідентифікацією необхідної задачі, дії чи діяльності, які мають виконуватись. Аналіз функціональних вимог буде використаний в функціях верхніх рівнів для функціонального аналізу.

Нефункціональні вимоги задають критерій для оцінювання операцій системи, замість її поведінки.

Вимоги продуктивності пояснюють, до якої міри місії чи функції повинні бути виконані; зазвичай вимірюється в термінах кількості, якості, охоплення, своєчасності чи готовності. Протягом аналізу вимог, вимоги продуктивності будуть інтерактивно розроблятись впродовж всіх виявлених функцій, що базуються на факторах життє-

вого циклу системи, і характеризуються в термінах ступеня визначеності в їх оцінках, ступеня критичності успіху системи і їх відношення до інших вимог.

Вимоги дизайну.

Успадковані вимоги, тобто вимоги, які обумовлені вимогами вищого рівня, чи перетворені з них.

Розподілені вимоги визначені поділом чи іншим перерозміщенням високорівневих вимог в декілька низькорівневих вимог.

1.2.5 Користувацькі історії

Користувацькі історії – в гнучкій розробці термін, що традиційній розробці називають «програмні вимоги». Вони є короткими заявами про намір або вимоги до системи.

Користувацькі історії (User Story) – спосіб опису вимог до розроблюваної системи сформульованих як одна або більше пропозицій повсякденною або діловою мовою користувача. Користувацькі історії використовуються гнучкими методологіями розробки ПЗ для специфікації вимог. Кожна користувацька історія обмежена за розміром та складністю. У методології Scrum історії пишуться або схвалюються власником продукту. Для замовників (користувачів) користувацькі історії є основним інструментом впливу на розробку програмного забезпечення.

Користувацькі історії – швидкий спосіб документувати вимоги клієнта, без необхідності розробляти великі формалізовані документи і згодом витрачати ресурси на їх підтримку. Мета користувацьких історій полягає в тому, щоб бути взмозі оперативно і без накладних витрат реагувати на мінливі вимоги реального світу.

Ось основна формула написання історії користувача: Як <роль> я хочу <функція> щоб <користь>.

Карта історії (story map) є інструментом, що допомагає в осмисленні функціональності продукту, способів його використання, а також допомагає в розстановці пріоритетів при постачанні продукту (при плануванні релізу). Цей метод декомпозиції забезпечує еволюційне розуміння продукту, починаючи з повного охоплення всіх потреб і завершуючи зануренням до детальних історій користувачів.

Карта історій служить джерелом інформації і використовується для візуалізації вимог до продукту в контексті використання та пріо-

ритетів. Карта історій часто виводиться на екран для проєктної команди під час сесій планування релізу. Аналізуючи карту історій, команда може краще ідентифікувати залежності, що утворюються в результаті передбачуваного потоку через власні історії. Карта також може бути використана для оцінки і управління ризиками, шляхом розгляду того, як історії будуть спільно працювати в контексті отримання користі для бізнесу.

Карта історій являє собою техніку візуального і фізичного представлення послідовності дій, які повинні бути реалізовані рішенням. При цьому використовується двомірна сітка, щоб показати послідовність і угруповання ключових аспектів продукту по горизонталі, а деталі і пріоритет історій по вертикалі.

Приклад побудови карти історій у випадку, коли перед командою розробників стоїть задача створити інтернет-магазин для продукції продемонстровано у Додатку Б.

1.2.6 Документування вимог

Документ, який описує вимоги, є результатом етапів виявлення та аналізу вимог. Документ опису вимог розроблюється відповідно до раніше визначеного шаблону. Шаблон визначає структуру та стиль документу.

Частина документу опису вимог, яка містить **стислий опис проєкту**, переважно орієнтує тих керівників та учасників проєкту, які відповідні за прийняття рішення, але ймовірно не стануть детально вивчати документ повністю. На початку документу необхідно визначити цілі та межі проєкту, а потім описати діловий контекст системи.

Також слід визначити учасників проєкту системи. При цьому важливо, щоб замовник виступав не як безвиразно представлений підрозділ або офіс, необхідно привести конкретні імена.

Хоча документ опису вимог може бути далеким від технічних рішень, важливо визначити ідеї, які стосуються рішення, на початкових етапах життєвого циклу розробки. Важливо також проаналізувати варіант придбання готового продукту замість його розроблення “з нуля”. Документ опису вимог повинен надавати перелік існуючих програмних компонентів та пакетів, які необхідно в подальшому вивчити в якості варіантів можливих рішень.

Основна частина документу опису вимог присвячена визначенню **системних сервісів**. Ця частина може займати до половини всього обсягу документу. Це єдина частина документу, яка може містити узагальнені моделі – моделі бізнес-вимог.

Межі системи можна моделювати за допомогою діаграм контексту. У поясненнях до діаграми контексту повинні бути чітко визначені межі системи. Без подібного визначення проєкт не може бути застрахованим від спроб "розтягнути" його межі.

Функціональні вимоги можна моделювати за допомогою діаграм бізнес-прецедентів. Однак, діаграми охоплюють перелік функціональних вимог тільки в загальному вигляді. Усі вимоги треба позначити, класифікувати та визначити.

Вимоги до даних можна моделювати за допомогою діаграми бізнес-класів. Так само як і у випадку функціональних вимог, діаграма бізнес-класів не дає повного визначення структур даних для бізнес-процесів. Кожний бізнес-клас вимагає подальших пояснень. Необхідно описати атрибут ненаповнення класів та визначити ідентифікуючі атрибути класів. У протилежному випадку неможливо правильно представити асоціації.

Системні сервіси визначають, що повинна робити система. Системні обмеження визначають, на скільки система обмежена під час виконання обслуговування. Системні обмеження пов'язані з наступними видами вимог:

- вимоги до інтерфейсу;
- вимоги до продуктивності;
- вимоги до безпеки;
- експлуатаційні вимоги;
- політичні та юридичні вимоги.

Вимоги до інтерфейсу визначають, як система взаємодіє з користувачем. У документі опису вимог визначаються тільки "відчуття" від GUI-інтерфейсу. Початкове проєктування GUI-інтерфейсу виконується під час специфікації вимог та пізніше під час системного проєктування.

У залежності від галузі застосування вимоги до продуктивності можуть грати доволі важливу роль в успіху проєкту. В обмеженому розумінні вони задають швидкість (час відгуку системи), з якою повинні виконуватися різноманітні завдання. У широкому розумінні ви-

моги до продуктивності включають інші обмеження: щодо надійності, готовності, пропускної здатності тощо.

Вимоги до безпеки описують користувацькі права доступу до інформації, що контролюються системою. Користувачам може бути наданий обмежений доступ до даних або обмежені права на виконання деяких операцій з даними.

Експлуатаційні вимоги визначають програмно-технічне середовище, якщо воно відоме на етапі проєктування, у якому повинна функціонувати система. Ці вимоги можуть впливати на інші сторони проєкту, такі як: підготовка користувачів та супроводження системи.

Важливі й інші види обмежень. Наприклад, у відношенні деяких систем можуть висуватися вимоги щодо легкості їх використання (вимоги щодо придатності їх використання) або легкості їх супроводження (вимоги щодо придатності до супроводження).

Заключна частина документу опису вимог визначає **інші проєктні питання**. Одним з важливих розділів даної частини є «Відкриті питання», у якому визначаються всі питання, які можуть вплинути на успіх проєкту і які не розглядались в інших розділах документу. До даного пункту належить очікуване збільшення значення деяких вимог, які в поточний момент виходять за межі проєкту, а також будь-які потенційні проблеми та відхилення у поведінці системи, які можуть початися у зв’язку з розгортанням системи.

У даній частині документу опису вимог необхідно представити попередній план-графік виконання основних проєктних завдань, а також попередній розподіл людських та інших ресурсів. Для вироблення стандартних планових графіків можна використовувати програмні засоби управління проєктами, наприклад, такі як система PERT (program evaluation-and-review technique – метод оцінювання та перегляду планів) або карти Гантта.

Прямим результатом складання план-графіку може бути розроблення попереднього бюджету. Вартість проєкту може бути виражена у вигляді діапазонів значень витрат, а не конкретного значення.

Додатки до документу опису вимог містять іншу корисну для розуміння вимог інформацію. Основним додатком є глосарій. **Глосарій** визначає терміни, скорочення та абревіатури, які використовуються в документі опису вимог. Значення вірно розробленого глосарію важко переоцінити. Невірне використання термінології несе велику небезпеку для проєкту.

Розділ посилань містить перелік документів, які згадуються та використовуються при підготовці документу опису вимог. До них можуть належати книги та інші опубліковані джерела інформації, а також внутрішні документи, які можливо є навіть більш важливими.

1.3 Завдання на лабораторну роботу

1.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

1.3.2 Обрати тему для подальшого проєктування системи з переліку тем у Додатку А або запропонувати власну тему. Узгодити тему з викладачем.

1.3.3 Провести документування специфікації вимог до проєкту: сформулювати цілі та межі проєкту, аналіз функціональних вимог представити у вигляді карти користувацьких історій, визначити учасників проєкту та провести розподіл учасників проєкту та ролей користувачів, розглянути існуючі рішення проблеми.

1.3.4 Визначити системні сервіси: межі системи, функціональні вимоги, вимоги до даних.

1.3.5 Сформулювати системні обмеження: вимоги до інтерфейсу, продуктивності, безпеки, експлуатаційні вимоги та вимоги до клієнтської та серверної сторін.

1.3.6 Визначити проєктні питання: представити попередній план-графік виконання основних проєктних завдань, попередній бюджет (складається із заробітної платні учасників проєкту), розглянути питання, які впливають на успіх проєкту і не були розглянуті в інших пунктах.

1.3.7 Оформити звіт з роботи.

1.3.8 Відповісти на контрольні питання.

1.4 Зміст звіту

1.4.1 Тема та мета роботи.

1.4.2 Тема, обрана для проєктування.

1.4.3 Завдання на лабораторну роботу.

1.4.4 Специфікація вимог до проєкту: цілі та межі проєкту, учасники проєкту та ролі користувачів, функціональні вимоги до проєкту,

карта користувацьких історій, існуючі аналоги, системні сервіси, системні обмеження, проєктні питання, додатки.

1.4.6 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

1.5 Контрольні запитання

1.5.1 Що таке життєвий цикл?

1.5.2 Поясніть різницю між життєвим циклом та методологією розробки.

1.5.3 Які етапи життєвого циклу розробки систем розрізняють?

1.5.4 Які існують методи виявлення вимог до програмного забезпечення?

1.5.5 Які фази включає процес аналізу вимог?

1.5.6 Кого можна віднести до осіб, зацікавлених у розробці системи?

1.5.7 Які існують способи документування вимог?

1.5.8 Поясніть принцип складання карти користувацьких історій?

1.5.9 Які типи вимог можна виокремити?

1.5.10 Які види ризиків характерні для вимог?

1.5.11 Яка типова структура документу опису вимог?

2 ЛАБОРАТОРНА РОБОТА № 2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ UML

2.1 Мета роботи

Ознайомитися з видами діаграм UML та особливостями їх застосування для проєктування програмного забезпечення, навчитися розробляти абстрактну модель програмної системи мовою UML на основі діаграм різних видів.

2.2 Короткі теоретичні відомості

UML (англ. Unified Modeling Language) – уніфікована мова моделювання, що використовується у парадигмі об'єктно-орієнтованого програмування та є невід'ємною частиною уніфікованого процесу розроблення ПЗ. UML заснована на діаграмах, що дають можливість представити систему у такому вигляді, щоб її можна було легко перевести в програмний код.

2.2.1 Діаграма класів

Діаграму класів (class diagram) використовують для подання статичної структури моделі системи в термінології класів об’єктно-орієнтованого програмування.

Класи використовуються для складання словника розроблюваної системи. Вони представляють опис сукупності об’єктів із загальними атрибутами, операціями, відношеннями та семантикою. Клас реалізує один чи декілька інтерфейсів.

Діаграми класів використовуються для моделювання статичного вигляду системи з точки зору проєктування. Сюди більшою мірою належить моделювання словника системи, кооперацій та схем. Діаграми класів складають основу діаграм компонентів.

Діаграми класів можуть містити наступні сутності:

– класи;

– інтерфейси – набори операцій, які використовуються для специфікації послуг, що надаються класом або компонентом;

– кооперації – спільнота класів, інтерфейсів та інших елементів, які працюють спільно для забезпечення кооперативної поведінки, що є більш значимою ніж сума складових;

– відношення залежності (описує існуючі між класами відношення використання), узагальнення (зв'язує узагальнені класи зі спеціалізованими) та асоціації (надає структурні відношення між об'єктами).

Залежністю називають варіант використання, відповідно до якого зміна у специфікації одного елементу може вплинути на інший елемент, що його використовує, при чому зворотне не обов'язково. Графічно залежність позначається пунктирною лінією зі стрілкою, направленою від одного елементу на той, від якого він залежить. Найчастіше залежності застосовуються під час роботи з класами, щоб відбити у сигнатурі операції той факт, що один клас використовує інший у якості аргументу.

Узагальнення – відношення між загальною сутністю (суперкласом, або батьківським класом) та її конкретним втіленням (субкласом, чи нащадком). Узагальнення іноді називають відношенням типу "є", маючи на увазі, що одна сутність є окремим вираженням іншої, більш загальної. Узагальнення означає, що об'єкти класу-нащадка можуть використовуватися всюди, де зустрічаються об'єкти батьківського класу, але не навпаки. Нащадок наслідує властивості батьківського класу, зокрема його атрибути та операції.

Проста асоціація між двома класами відбиває структурне відношення між рівноправними сутностями, коли обидва класи знаходяться на одному концептуальному рівні і не один з них не є більш важливим ніж інший. Але іноді треба змоделювати відношення типу "частина/ціле", в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин). Відношення такого типу називається агрегацією. Агрегація є окремим випадком асоціації.

Часто під час моделювання важливо зазначити, скільки об'єктів може бути пов'язано шляхом одного екземпляру асоціації. Це число називається кратністю ролі асоціації і записується або як вираз, значенням якого є діапазон значень, або у явному вигляді.

Приклад діаграми класів наведено на рис. 2.1.

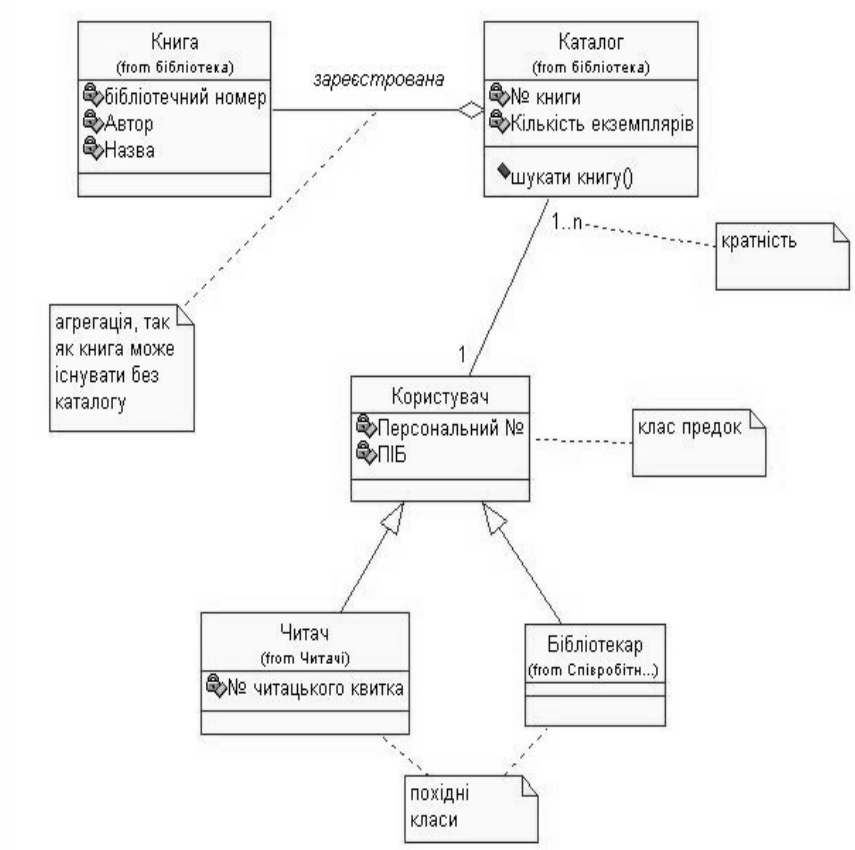

Рисунок 2.1 – Діаграма класів

2.2.2 Діаграма компонентів

Діаграма компонентів (рис. 2.2) описує особливості фізичної уяви системи. Дана діаграма застосовується для моделювання статичного вигляду системи з точки зору реалізації. За своєю сутністю діаграми компонентів – не що інше як діаграми класів, сфокусовані на системних компонентах.

Діаграма компонентів відображає залежності між компонентами ПЗ, включаючи компоненти вихідних кодів, бінарні компоненти та

компоненти, що можуть виконуватись, тобто вона дозволяє визначити архітектуру системи, що розробляється, встановивши залежності між програмними компонентами.

Діаграми компонентів зазвичай включають:

- компоненти;
- інтерфейси;
- відношення узагальнення, асоціації та реалізації.

Реалізацією називається семантичне відношення між класифікаторами, за якого один з них описує контракт, а інший гарантує його виконання. Частіше за все реалізації використовуються для визначення відношень між інтерфейсом та класом або компонентом, який надає об'явлені в інтерфейсі операції або послуги.

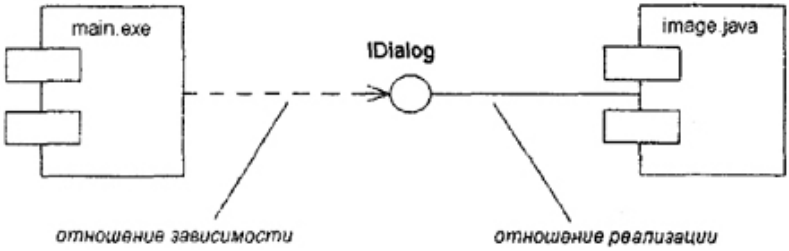

Рисунок 2.2 – Діаграма компонентів

2.2.3 Діаграма прецедентів

Діаграми прецедентів застосовуються для моделювання вигляду системи з точки зору прецедентів (або варіантів використання). Даний вид діаграм використовується для моделювання динамічних аспектів системи, так само як і діаграми станів та діяльності.

Діаграма прецедентів – діаграма, на якій зображено відношення між акторами та прецедентами в системі (рис. 2.3).

Діаграми прецедентів зазвичай включають в себе:

- прецеденти;
- акторів;
- відношення залежності, узагальнення та асоціації.

UML дозволяє моделювати контекст за допомогою прецедентів, у яких особлива увага акцентується акторів, що оточують систему. Важливо вірно визначити акторів, оскільки це дозволяє описати клас сутностей, що взаємодіють з системою.

Моделювання контексту системи складається з наступних кроків:

а) ідентифікувати акторів, що оточують систему: визначити групи, яким участь системи необхідна для виконання їхніх завдань; групи, які необхідні для виконання системою своїх функцій; групи, що взаємодіють із зовнішніми програмними та апаратними засобами, а також групи, що виконують допоміжні функції адміністрування та підтримки;

б) організувати схожих акторів за допомогою відношень узагальнення/спеціалізації;

в) ввести стереотипи для кожного актора, якщо це полегшує розуміння;

г) розташувати акторів на діаграмі прецедентів та визначити способи їх зв'язку з прецедентами системи.

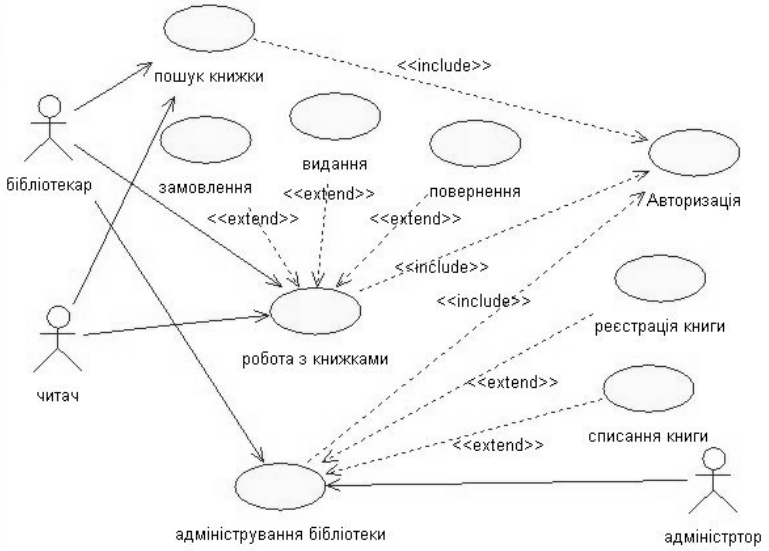

Рисунок 2.3 – Діаграма прецедентів

Проєктована система представляється у вигляді множини сутностей чи акторів, що взаємодіють із системою за допомогою варіантів використання (use case), призначених для опису сервісів, що система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій системи при діалозі з актором.

2.2.3.1 Потоки подій

Однією з вимог мови UML є самодостатність діаграм для подання інформації про моделі проєктованих систем. Однак, діаграми прецедентів описують те, що робить система, без уточнення того, як вона це робить.

Для реального опису системи потрібні специфічні дані, які відображені в потоці подій. Потоки подій уточнюють або деталізують послідовність дій, які здійснюються системою при виконанні її варіантів використання, а також описують логіку переходів через варіанти використання.

Потік подій – це певна послідовність дій, яка описує дії акторів і поведінку модельованої системи в формі звичайного тексту.

Потоки подій – це текстові описи покрокового виконання прецедентів, вони зрозумілі не тільки розробнику, але і сторонньому читачеві. Їх завдання – ще більше деталізувати опис функціональності системи до того, як розробники приступлять до написання програмного коду, і усунути можливе нерозуміння необхідної функціональності, як можна більше зблизити уявлення розробника про систему і замовника.

Потоки подій бувають трьох типів: основний, альтернативний і потік помилок.

Основний (головний) потік описує найкращий сценарій або найбільш використовуваний шлях виконання прецеденту.

Альтернативний потік специфікує відхилення від основного потоку, які не розглядаються як помилкові.

Потік помилок розглядається як відхилення від альтернативного або основного, яке породжує умови формування помилки.

Приклад опису потоку подій наведено у Додатку В.

2.2.4 Діаграма станів

Діаграми станів (рис. 2.4) зображають всі можливі стани, в яких може знаходитися конкретний об'єкт, а також зміни стану об'єкту, які відбуваються в результаті впливу деяких подій на цей об'єкт. У більшості об'єктно-орієнтованих методів діаграми станів будуються для єдиного класу, щоб показати динаміку поведінки єдиного об'єкту.

Зазвичай діаграми станів складаються з наступних елементів:

- прості та складені стани;
- переходи разом з асоційованими подіями та діями.

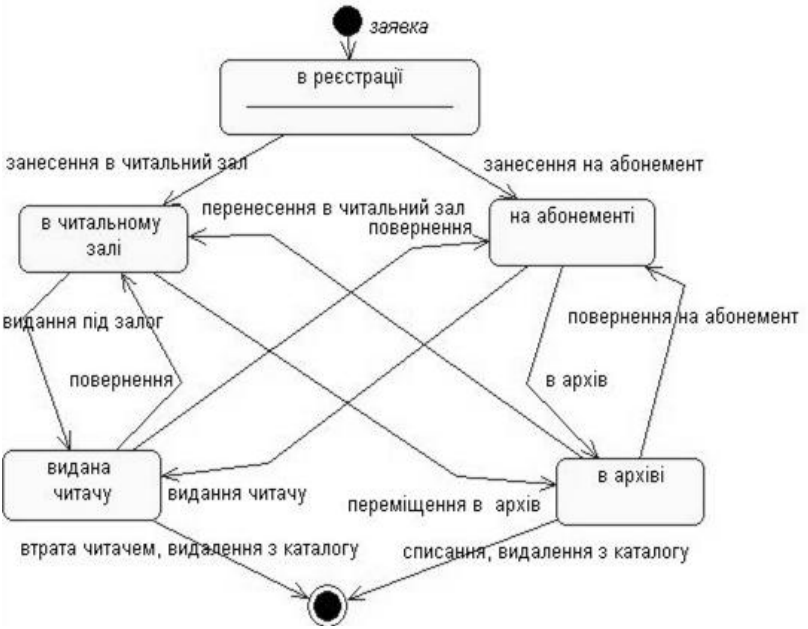

Рисунок 2.4 – Іменовані переходи між станами

2.2.5 Діаграма діяльності

Діаграма діяльності показує переходи між видами діяльності. Модель видів діяльності (activity model) може подавати в графічній формі потік подій для прецеденту (рис. 2.5).

Кожен прецедент можна моделювати за допомогою одного або декількох графів видів діяльності. Діаграма діяльності за своєю сутністю є блок-схемою, яка демонструє, як потік управління переходить від однієї діяльності до іншої.

Подія, джерелом якої служить суб'єкт-прецедент, що ініціює, це та ж сама подія, що запускає виконання графа видів діяльності. Процес виконання послідовно переходить від одного стану виду діяльності до іншого.

Діаграма діяльності у загальному випадку складається зі:

– станів діяльності та станів дії;
– переходів.

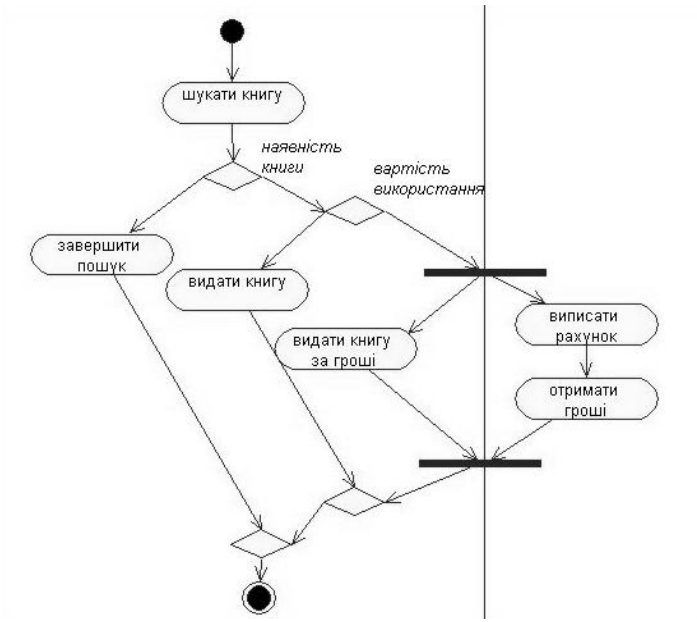

Рисунок 2.5 – Діаграма діяльності

Перехід – це відношення між двома станами, що демонструє те, що об’єкт, який знаходиться в першому стані, повинен виконати деякі дії та перейти у другий стан, як тільки відбудеться вказана подія та будуть задоволені вказані умови.

Розгалуження описує різноманітні шляхи виконання в залежності від значення деякого булівського виразу.

Послідовні переходи в діаграмах діяльності виконуються найчастіше, однак використовуються також і паралельні потоки. У UML для позначення розділення та злиття таких паралельних потоків виконання використовується синхронізаційна лінія, яка зображується у вигляді жирної вертикальної або горизонтальної лінії.

2.2.6 Діаграма кооперації

Діаграма кооперації (рис. 2.6) призначена для специфікації структурних аспектів взаємодії. Головна особливість діаграми кооперації полягає в можливості графічно уявити не тільки послідовність взаємодії, але і всі структурні відносини між об'єктами, що беруть участь в цій взаємодії.

Для створення діаграм кооперації необхідно розташувати об’єкти, що беруть участь у взаємодії, у вигляді вершин графа. Потім зв’язки, що з’єднують ці об’єкти, відображаються у вигляді дуг даного графу. Зв’язки доповнюються повідомленнями, які об’єкти приймають та посилають.

Діаграма кооперації має дві властивості, які відрізняють її від діаграми послідовності:

а) шлях: для опису зв’язку одного об’єкта з іншим до дальньої кінцевої точки цього зв’язку можна приєднати стереотип шляху (наприклад, local показує, що зазначений об’єкт є локальним по відношенню до відправника повідомлення); має сенс явним чином зображати шлях зв’язку тільки по відношенню до шляхів типу local, parameter, global та self;

б) порядковий номер повідомлення: для позначення часової послідовності перед повідомленням можна поставити номер, який повинен поступово збільшуватись для кожного нового повідомлення; для позначення вкладеності використовується десяткова нотація Дьюї.

Найчастіше моделюються нерозгалужені послідовності потоків управління. Однак можливо моделювати і більш складні потоки, що містять ітерації та розгалуження.

Рисунок 2.6 – Діаграма кооперації

2.2.7 Діаграма послідовності

Діаграма послідовності показує учасників взаємодій і послідовність повідомлень, якими вони обмінюються.

На діаграмі послідовності (рис. 2.7) зображуються виключно ті об'єкти, що безпосередньо беруть участь у взаємодії і не показуються можливі статичні асоціації з іншими об'єктами. Для діаграми послідовності ключовим моментом є саме динаміка взаємодії об'єктів у часі.

Для створення діаграми послідовності потрібно, по-перше, розташувати об’єкти, що беруть участь у взаємодії, у верхній її частині вздовж вісі X. Зазвичай об’єкт, що ініціює взаємодію, розташовують зліва, а інші – правіше (чим далі, тим більш підпорядкованим є об’єкт). Потім вздовж вісі Y розташовують повідомлення, які об’єкти надсилають та приймають, при чому пізніші розташовуються нижче. Це надає наочну картину, яка дозволяє зрозуміти розвиток потоку керування в часі.

Діаграми послідовності характеризуються двома особливостями, які відрізняють їх від діаграм кооперації.

По-перше, на них є лінія життя об’єкту – вертикальна пунктирна лінія, що відбиває існування об’єкту в часі. Більша частина об’єктів, представлених на діаграмі, існує протягом всієї взаємодії, тому їх зображують у верхній частині діаграми, а їх лінії життя промальовані згори донизу.

Об’єкти можуть створюватись і під час взаємодії. Лінія життя таких об’єктів починаються з отримання повідомлення зі стереотипом create. Об’єкти можуть також знищуватись під час взаємодії: у такому випадку їх лінії життя завершуються отриманням повідомлення зі стереотипом destroy, а в якості візуального образу використовується велика літера X, що позначає кінець життя об’єкту.

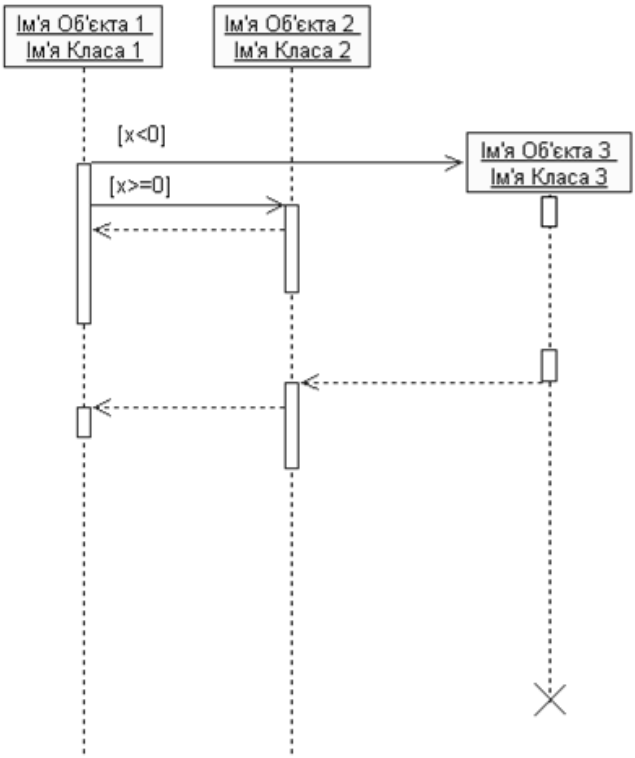

Рисунок 2.7 – Діаграма послідовностей

Друга особливість – фокус керування. Він позначається витягнутим прямокутником, що показує проміжок часу, протягом якого об’єкт виконує деяку дію безпосередньо або за допомогою підпорядкованої процедури. Верхня грань прямокутника вирівнюється за часовою віссю з моментом початку дії, нижня – з моментом його завершення. Вкладеність фокусу керування, викликану рекурсією або зворотним викликом з боку іншого об’єкту, можна продемонструвати, розташувавши інший фокус керування правіше свого батьківського об’єкту.

2.3 Завдання на лабораторну роботу

2.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

2.3.2 Вивчити правила побудови структурних діаграм: діаграм класів, компонентів, розгортання, об'єктів, профілів, кооперації.

2.3.3 Вивчити правила побудови діаграм поведінки: діаграм діяльності, станів, прецедентів.

2.3.4 Вивчити правила побудови діаграм взаємодії: діаграм кооперації, послідовності, синхронізації, огляду взаємодії.

2.3.5 Розробити абстрактну модель системи за обраною темою на основі діаграми класів.

2.3.6 Розробити абстрактну модель системи за обраною темою на основі діаграми компонентів.

2.3.7 Розробити абстрактну модель системи за обраною темою на основі діаграми прецедентів.

2.3.8 Розробити діаграми прецедентів для представлення варіантів використання системи, наведених на загальній діаграмі прецедентів.

2.3.9 До діаграм прецедентів додати словесний опис потоків подій (включаючи опис основних, альтернативних та помилкових потоків).

2.3.10 Розробити абстрактну модель системи за обраною темою на основі діаграми діяльності.

2.3.11 Розробити абстрактну модель системи за обраною темою на основі діаграми станів одного з класів (який створюється і змінюється в одному або декількох прецедентах).

2.3.12 Розробити абстрактну модель системи за обраною темою на основі діаграм кооперації основного процесу системи.

2.3.13 Розробити абстрактну модель системи за обраною темою на основі діаграм послідовності основного процесу системи.

2.3.14 Оформити звіт з роботи.

2.3.15 Відповісти на контрольні питання.

2.4 Зміст звіту

2.4.1 Тема та мета роботи.

2.4.2 Тема, обрана для проєктування.

2.4.3 Діаграма класів системи, що проєктується.

2.4.4 Діаграма компонентів системи, що проєктується.

2.4.5 Діаграма діяльності системи, що проєктується.

2.4.6 Діаграма станів одного з класів (який створюється і змінюється в одному або декількох прецедентах).

2.4.7 Діаграми прецедентів системи, що проєктується.

2.4.8 Опис потоків подій (включаючи опис основних, альтернативних та помилкових потоків).

2.4.9 Діаграма кооперації основного процесу системи.

2.4.10 Діаграма послідовності основного процесу системи.

2.4.11 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

2.5 Контрольні запитання

2.5.1 Які види структурних діаграм розрізняють в UML?

2.5.2 З якою метою використовуються діаграми класів?

2.5.3 Що таке структурні класи?

2.5.4 Які види зв'язків між сутностями на діаграмах класів представлені в UML?

2.5.5 Який вид відношень між класами позначають асоціації, і які види асоціацій розрізняють?

2.5.6 Відношення узагальнення між класами належить до часу проєктування чи до часу виконання?

2.5.7 Які типи відношення залежності між класами розрізняють і яке вони мають значення?

2.5.8 Яку інформацію несуть діаграми компонентів?

2.5.9 З яких точок зору розглядається система в діаграмах класів, об'єктів та компонентів?

2.5.10 Для чого застосовуються діаграми компонентів?

2.5.11 Що таке потік подій?

2.5.12 Які види структурних діаграм можуть бути застосовані для моделювання фізичних баз даних?

2.5.13 Які види діаграм взаємодії розрізняють в UML?

2.5.14 Які види діаграм в UML використовуються для моделювання динамічних аспектів системи?

2.5.15 За яким фактором впорядковується взаємодія об’єктів на діаграмах послідовності?

2.5.16 Чим відрізняються діаграми кооперації від діаграм послідовності?

2.5.17 Що таке сценарій в термінах діаграм послідовності?

2.5.18 Що таке лінія життя об’єкту і як вона позначається на діаграмі послідовності?

2.5.19 Що таке фокус керування і як він позначається на діаграмі послідовності?

2.5.20 Скільки фокусів керування може мати об’єкт протягом своєї лінії життя?

2.5.21 Хто може ініціювати взаємодію в системі?

2.5.22 Які існують різновиди повідомлень в UML?

2.5.23 Які види діаграм поведінки розрізняють в UML?

2.5.24 Які аспекти поведінки системи моделюються в діаграмах поведінки?

2.5.25 З чого складається діаграма діяльності?

2.5.26 Якими спільними з іншими видами діаграм властивостями володіє діаграма поведінки?

2.5.27 Для чого використовуються діаграми діяльності?

2.5.28 Для моделювання яких об’єктів використовуються діаграми стану?

2.5.29 Яке значення мають терміни “автомат”, “стан”, “подія”, “перехід”, “діяльність”, “дія” в сенсі діаграми стану?

2.5.30 Наведіть приклади застосування діаграм прецедентів.

2.5.31 Які елементи включають в себе діаграми прецедентів?

3 ЛАБОРАТОРНА РОБОТА № 3 ПРОЄКТУВАННЯ ІНТЕРФЕЙСУ ВЕБЗАСТОСУНКІВ

3.1 Мета роботи

Навчитися проєктувати дизайн та інтерфейс користувача вебзастосунків, враховуючи вимоги до інтерфейсу замовника та ергономічні показники інтерфейсу.

3.2 Короткі теоретичні відомості

Інтерфейс (від англ. Interface – поверхня розділу, перегородка) – сукупність засобів і методів взаємодії між елементами системи. Залежно від контексту, поняття застосовне як до окремого елементу (інтерфейс елементу), так і до зв'язків елементів (інтерфейс сполучення елементів).

Інтерфейс користувача – сукупність засобів, за допомогою яких користувач спілкується з різними пристроями:

– інтерфейс командного рядка – управління програмою або пристроєм здійснюються шляхом введення з клавіатури текстових рядків;

– графічний інтерфейс – управління програмними функціями реалізовано графічними елементами екрану.

Спочатку термін «інтерфейс» застосовувався до всіх видів взаємодії людини і машини. Протягом багатьох років йшов розвиток комп'ютерів і мобільних пристроїв і сьогодні цей термін найчастіше означає графічний інтерфейс користувача (GUI).

Графічний інтерфейс користувача забезпечує зручний спосіб взаємодії з електронними пристроями за допомогою значків, кнопок і інших графічних елементів (порівняно з текстовими консольними інтерфейсами, які набагато складніше використовувати).

Дизайн інтерфейсу користувача (UID) або інжиніринг інтерфейсу – це дизайн інтерфейсу користувача (UI) програм і пристроїв, таких як: комп'ютери, побутова техніка, мобільні телефони та інші електронні пристрої з акцентом на якість досвіду використання. Метою дизайну інтерфейсу користувача є створення максимально прос-

того і продуктивного взаємодії з точки зору досягнення цілей користувача (орієнтований на користувача дизайн).

Багато кількісних та евристичних методів використовуються для аналізу та вивчення інтерфейсів. Кількісні методи часто можуть зводити спірні питання до простих обчислень. Ще однією, більш важливою, перевагою даних методів є те, що вони допомагають зрозуміти найважливіші аспекти взаємодії людини з машиною.

3.2.1 Модель GOMS

Класична модель GOMS – «правила для цілей, об'єктів, методів і виділення» (the model of goals, objects, methods, and selection rules) – дозволяє передбачити, скільки часу буде потрібно досвідченому користувачеві на виконання конкретної операції при використанні даної моделі інтерфейсу.

GOMS розділяє взаємодію користувачів з комп'ютером на елементарні дії (ці дії можуть бути фізичними, пізнавальними або діями сприйняття). Інтерфейс досліджується за допомогою цих елементарних дій.

Хоча для різних користувачів час виконання того чи іншого жесту може суттєво різнитися, дослідники виявили, що для більшої частини порівняльного аналізу задач, що включають використання клавіатури та графічного пристрою вводу, замість проведення вимірів для кожного окремого користувача можна використати набір стандартних інтервалів. Так, наприклад, в оригінальній номенклатурі на натиснення клавіші відводиться 0.2 с, 1.1 с – час, необхідний користувачу для того, щоб вказати на якусь позицію на екрані монітору, 0.4 с – час, необхідний користувачу для того, щоб перемістити руку з клавіатури на графічний пристрій виводу або навпаки, 1.35 с – час, необхідний користувачу для того, щоб розумово підготовитись до наступного кроку, R – час, протягом якого користувач повинен очікувати відповідь комп'ютера. На практиці вказані значення можуть варіюватися в широких інтервалах. Для досвідченого користувача, який може друкувати зі швидкістю 135 слів/хв., час на натиснення клавіші може становити 0.08 с, для звичайного користувача, що має швидкість 55 слів/хв., – 0.2 с. Окрім того швидкість набору залежить і від того, що саме набирається.

Тим не менш, за допомогою типових значень можна виконати правильне порівняльне оцінювання між двома інтерфейсами за рівнем ефективності їх використання.

Тривалість відповіді від комп'ютера може призводити до неочікуваного ефекту на дії користувача. Якщо в процесі використання якогось керуючого елементу на екрані монітору протягом приблизно 250 мс нічого не відбувається, користувач, швидше за все, почне відчувати неспокій, вирішить зробити ще одну спробу або зробить припущення, що система несправна. Якщо затримки неминучі, важливо, щоб в інтерфейсі був передбачений зворотний зв'язок, що повідомляє про них користувачу.

Розроблення інтерфейсу починається з визначення задач або набору задач, для яких продукт призначений. Обчислення часу, необхідного для виконання тієї чи іншої дії, за допомогою моделі GOMS починається з перелічення операцій зі списку жестів моделі GOMS, які входять до даної дії.

3.2.2 Закон Фіттса

Закон Фіттса свідчить, що час досягнення мети зворотнопропорційний розміру цілі та дистанції до неї. Закон Фіттса – загальний закон, що стосується сенсорно-моторних процесів, він зв'язує час руху з точністю руху і з відстанню переміщення: чим далі чи точніше виконується рух, тим більше корекції необхідно для його виконання і, відповідно, більше часу потрібно для внесення цієї корекції.

В інтерфейсі програми довжина прямої лінії, що з'єднує початкову позицію курсора і найближчу точку цільового об'єкта, визначається у законі Фіттса як дистанція. На основі даних про розміри об'єкту і дистанції закон Фіттса дозволяє знайти середній час, за який користувач зможе перемістити курсор до кнопки.

Математично закон записується таким чином:

$$T = a + b\log_2\left(\frac{D}{W} + 1\right),$$

де T – середній час, що витрачається на вчинення дії;

a – середній час запуску / зупинки руху;

b – величина, що залежить від типової швидкості руху;

D – дистанція від точки старту до центру цілі;

W – ширина цілі, виміряна уздовж вісі руху.

Для приблизних обчислень можна використати наступні значення констант: $a = 50$, $b = 150$.

3.2.3 Закон Хіка

Закон Хіка говорить про те, що чим менше елементів меню, тим менше часу займає вибір одного з них.

Перед тим як перемістити курсор до мети або вчинити будь-яку іншу дію з набору множини варіантів, користувач повинен вибрати цей об'єкт або дію. У законі Хіка стверджується, що коли необхідно зробити вибір з *n* варіантів, час на вибір одного з них буде пропорційним логарифму за основою 2 від числа варіантів плюс 1 за умови, що всі варіанти є рівно ймовірними. У даному вигляді закон Хіка схожий на закон Фіттса:

$$T = a + b\log_2(n+1).$$

Якщо ймовірність першого варіанту дорівнює p_i, то замість логарифмічного коефіцієнту використовується:

$$H = \sum_{i=1}^{n} p_i \log_2\left(\frac{1}{p_i} + 1\right),$$

де p_i означає ймовірність *i*-го варіанту за умови теоретико-інформаційної ентропії.

Якщо варіанти для вибору представлені незрозумілим чином, значення a і b зростають. Наявність навичок і звичок у користувача при використанні системи знижує значення b.

3.2.4 Етапи проєктування та інструменти розроблення дизайну інтерфейсу користувача

Процес проєктування дизайну інтерфейсу користувача можна спробувати розділити на етапи багатьма способами. Проте найчастіше розділення відбувається за точністю представлення та характерними рисами, які відображаються. Такий розподіл дає змогу оцінити характеристики розроблюваного шаблону та чітко відокремити інструмен-

тарій розроблення шаблону інтерфейсу користувача. Так виділяють поняття: вайрфрейм, прототип та мокап.

Вайрфрейм – це низько деталізоване представлення дизайну. Вайрфрейм повинен чітко демонструвати:

– основні групи вмісту;

– інформаційну структуру;

– опис взаємодії користувача з інтерфейсом і його приблизну візуалізацію.

Вайрфрейм можна розглядати як скелет дизайну. Важливо також відзначити, що в вайрфреймі повинні бути представлені всі важливі елементи кінцевого продукту.

Тобто, вкрай важливо підтримувати правильний баланс між рівнем деталізації і швидкістю створення. Потрібно створити цілісне уявлення кінцевого дизайну та не пропустити жодного важливого елемента, але не потрібно вдаватися у деталі. На цьому етапі описується фронт робіт по проєкту для всіх задіяних осіб: розробників, дизайнерів, копірайтерів, менеджерів — всім їм потрібен добре спроєктований вайрфрейм. Вайрфрейми повинні створюватися швидко і більшу частину цього часу слід провести за обговореннями з командою і роздумами. Зовнішній вигляд повинен бути естетичним, але дуже простим. Чорно-сіро-білий — типова палітра вайрфрейма.

Якщо, наприклад, вибір піктограм або завантаження картинок займає занадто багато часу, їх можна замінити заглушками. Вважається, що вайрфрейм дає неповне уявлення про кінцевий результат. Хороший вайрфрейм лягає в основу чистового дизайну та визначає напрям роботи для всієї команди.

Зазвичай вайрфрейми використовуються як документація по проєкту. Так як вони показують взаємодію користувача з інтерфейсом в окремих статичних моментах, їх потрібно супроводжувати текстовими коментарями: як короткими поясненнями, так і комплексної технічною документацією. Приклад розроблення вайрфрейму для головної сторінки інтернет-магазину наведено на рис. 3.1.

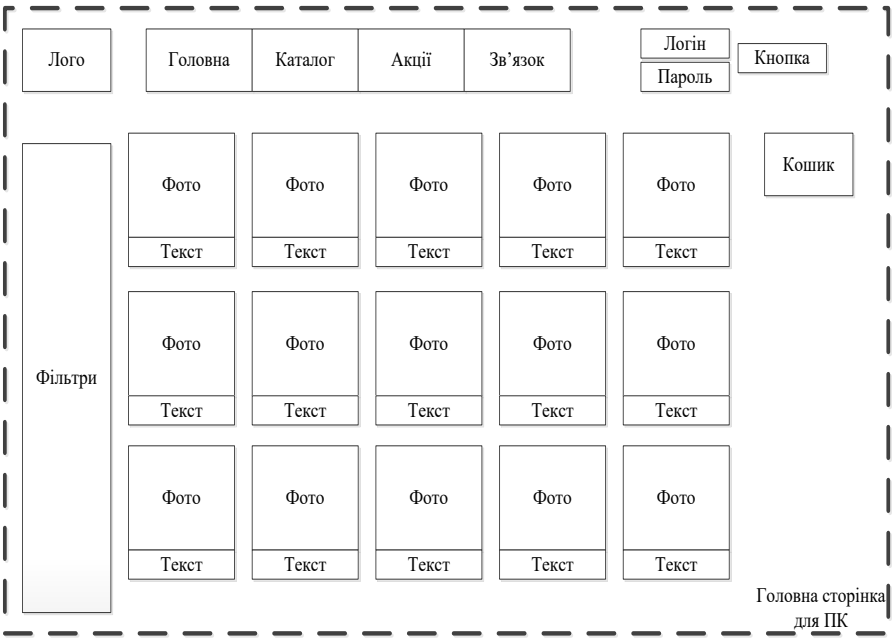

Рисунок 3.1 – Приклад розроблення вайрфрему

Прототип – це середньо або високо деталізоване уявлення кінцевого продукту, яке імітує взаємодію користувача з інтерфейсом. Він повинен дозволяти користувачеві:

– оцінити зміст і інтерфейс;

– протестувати основні способи взаємодії, як якщо б це був готовий продукт.

Прототип – це імітація взаємодії користувача з інтерфейсом кінцевого продукту. Він може не виглядати точно як кінцевий продукт, але безумовно не повинен бути контуром у відтінках сірого. Взаємодії повинні бути акуратно змодельовані та бути максимально схожими на те, що буде представлено в кінцевому продукті.

Прототипи зазвичай не дуже підходять для документації, так як зрозуміти роботу інтерфейсу можна тільки в процесі взаємодії з прототипом. З іншого боку, прототип – це найбільш приваблива форма документування дизайну, так як інтерфейс представляється «як є». Приклад розроблення прототипу для головної сторінки інтернет-магазину наведено на рис. 3.2.

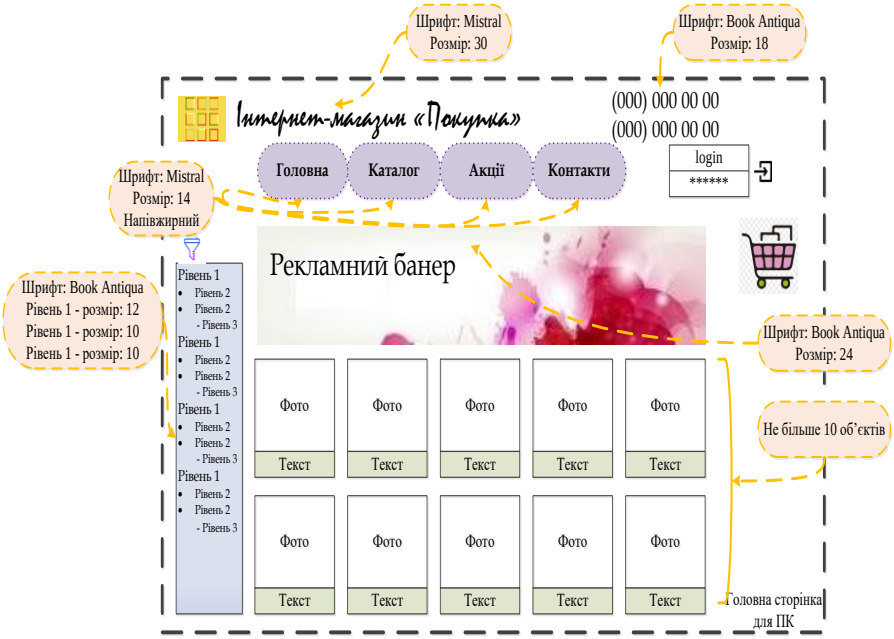

Рисунок 3.2 – Приклад розроблення прототипу

Мокап – це середньо або високо деталізоване статичне уявлення дизайну. Дуже часто мокап – це чернетка дизайну або навіть фактичний дизайн-макет. Хороший мокап повинен:

– чітко представляти інформаційну структуру, візуалізувати контент і демонструвати базову функціональність в статиці;

– заохочувати оцінку візуальної сторони проєкту.

Мокапи підходять для отримання схвалення від зацікавлених осіб, не залучених в розробку, оскільки завдяки візуальній природі мокап сприймається легше, ніж низько деталізовані артефакти, і при цьому створюється швидше за прототип. Вони хороші для збору відгуків, і їх можна використовувати в документації. Приклад розроблення прототипу для головної сторінки інтернет-магазину наведено на рис. 3.3.

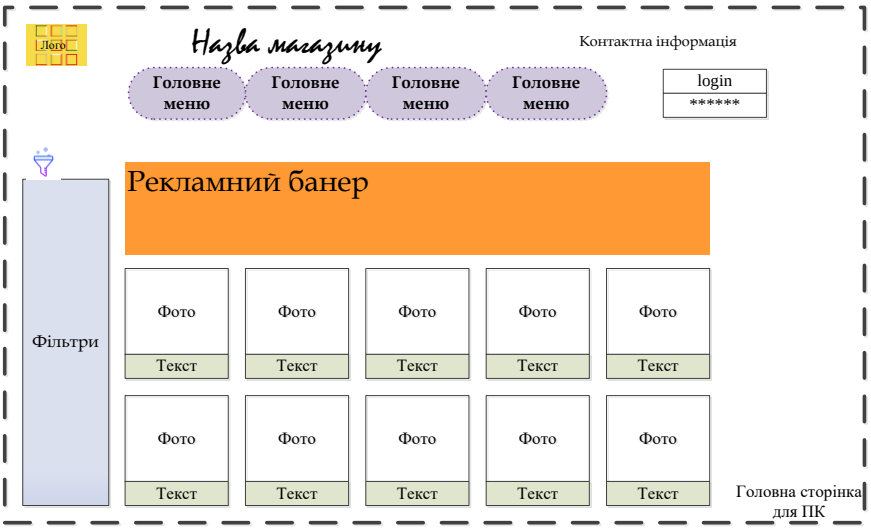

Рисунок 3.3 – Приклад розроблення мокапу

Порівняння методів проєктування дизайну представлені у вигляді таблиці 3.1.

Таблиця 3.1 – Порівняння етапів та інструментарію розроблення дизайну інтерфейсу користувача

	Точність представлення	Основні риси	Застосування
Вайрфрейм	низька	монохромний ескіз	програмна документація
Прототип	висока	інтерактивність, найповніше відображення	каркас для розробки, проходження тестування дизайну
Мокап	середня	статична візуалізація	затвердження у зацікавлених сторін

3.3 Завдання на лабораторну роботу

3.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

3.3.2 Проаналізувати вимоги до інтерфейсу ПЗ на основі специфікації вимог до ПЗ.

3.3.3 Визначити особливості роботи кожної групи користувачів з кожним елементом взаємодії користувача з вебзастосунком.

3.3.4 Розробити повний прототип головної сторінки системи (включаючи кольорові схеми, шрифти та їх розміри, оформлення для елементів керування), використовуючи Photoshop або аналогічні графічні редактори.

3.3.5 Розробити вайрфрейми інших сторінок, що реалізують роботу системи.

3.3.6 Обґрунтувати вибір запропонованих елементів інтерфейсу, ґрунтуючись на вимогах до інтерфейсу та ергономічних показниках інтерфейсу.

3.3.7 Оформити звіт з роботи.

3.3.8 Відповісти на контрольні питання.

3.4 Зміст звіту

3.4.1 Тема та мета роботи.

3.4.2 Тема, обрана для проєктування.

3.4.3 Загальна схема роботи системи.

3.4.4 Інтерфейс вебзастосунку за обраною темою.

3.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

3.5 Контрольні запитання

3.5.1 Які ергономічні показники інтерфейсу можна виділити?

3.5.2 Які існують методи кількісного аналізу інтерфейсу?

3.5.3 Що дозволяє передбачити модель GOMS?

3.5.4 Які жести розрізняє модель GOMS?

3.5.5 Що визначає закон Фіттса?

3.5.6 Що визначає закон Хіка?

3.5.7 Поясніть різницю між вайрфрейм, прототип та мокап?

3.5.8 Яке програмне забезпечення використовують для розробки вайрфреймів, прототипів та мокапів?

4 ЛАБОРАТОРНА РОБОТА № 4 АНАЛІЗ ТА ПРОЄКТУВАННЯ АРХІТЕКТУРИ ВЕБЗАСТОСУНКІВ НА ОСНОВІ ПРЕДМЕТНОЇ ОБЛАСТІ

4.1 Мета роботи

Навчитися розробляти та проводити аналіз архітектури вебзастосунків на основі предметної області.

Навчитися розробляти архітектуру вебзастосунків з використанням фреймворків та CMS, провести порівняльний аналіз архітектур вебзастосунків, розроблених на основі предметної області та за використання фреймворків або CMS

4.2 Короткі теоретичні відомості

Архітектур програмного забезпечення існує досить багато. Кожна з них має як плюси так і недоліки. Більш того, серед різноманіття є схожі представники. На сьогоднішній день найчастіше використовуються: Model-view-controller (MVC), Model-view-presenter (MVP), Model-View-View Model (MVVM).

4.2.1 Схема проєктування MVC – Model-View-Controller

MVC – це фундаментальний патерн, який знайшов застосування в багатьох технологіях.

Спільна риса для усіх патернів – відокремлення інтерфейсу користувача (UI) від логіки програмування, що дозволяє дизайнерам робити свою роботу, не замислюючись про код програми. Так само надається можливість виділення моделі даних, що дає розробникам можливість створення модульних тестів.

MVC складається з трьох компонент: View (подання, представлення, інтерфейс), Model (модель, бізнес логіка) і Controller (контролер, містить логіку на зміну моделі при певних діях користувача). Основна ідея цього патерну в тому, що і контролер і представлення залежать від моделі, але модель ніяк не залежить від цих двох компо-

нент. Це як раз і дозволяє розробляти і тестувати модель, нічого не знаючи про подання і контролери.

В ідеалі контролер так само нічого не повинен знати про представлення (хоча на практиці це не завжди так), і в ідеалі для одного представлення можна перемикати контролери, а так само один і той же контролер можна використовувати для різних представлень (так, наприклад, контролер може залежати від користувача, який увійшов в систему). Користувач бачить представлення, на ньому ж робить якісь дії, ці дії представлення перенаправляє контролеру та «підписується» на зміну даних моделі.

Контролер у свою чергу проводить певні операції над моделлю даних, представлення отримує останній стан моделі і відображає її користувачеві. Архітектура MVC представлена рис. 4.1.

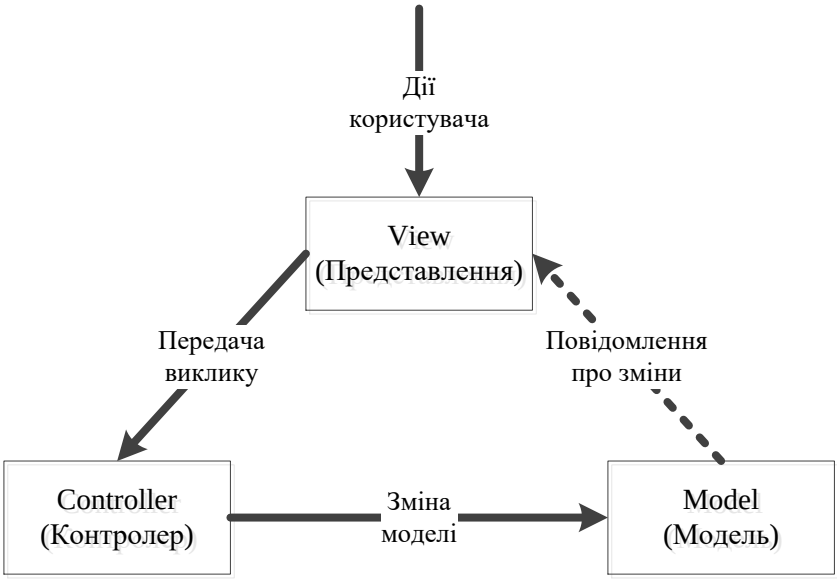

Рисунок 4.1 – Архітектура MVC

4.2.2 Архітектура MVP – Model-View-Presenter

Даний патерн також складається з трьох компонент. Тільки подивившись на наведену схему його архітектури (рис. 4.2) стає ясно,

що представленню немає потреби підписуватися на зміни моделі, тепер контролер, перейменований в Presenter (представник) дає знати представленням про зміни. Даний підхід дозволяє створювати абстракцію представлення. Реалізувати даний патерн можна за допомогою винесення інтерфейсів представлення. У кожного представлення будуть інтерфейси з певними наборами методів і властивостей, що необхідні представнику.

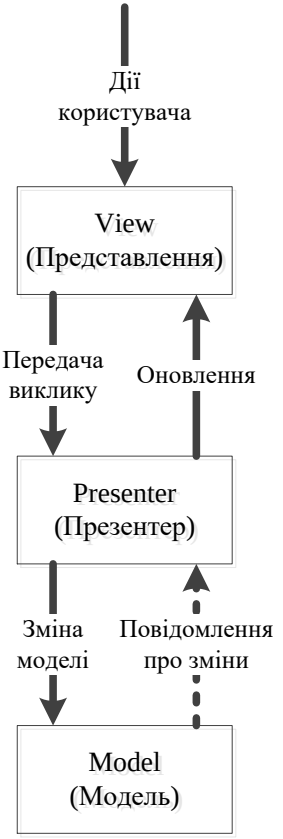

Рисунок 4.2 – Архітектура MVP

Представник в свою чергу ініціалізується з даним інтерфейсом, підписується на події подання та за необхідності відсилає дані. Даний підхід дозволяє розробляти додатки з використанням методології TDD (Test-driven development).

4.2.3 Архітектура MVVM – Model-View-ViewModel

У цій архітектурі знову присутні три компоненти: модель, представлення та третій компонент – додаткова модель під назвою ViewModel. Даний патерн підходить до таких технологій, де присутня двостороння синхронізація (біндінг) елементів управління на модель. ViewModel – це деякий супер-конвертер, який перетворює дані моделі подання, в ньому описуються основні властивості подання, а також логіка взаємодії з моделлю. Архітектура MVVM представлена на рис. 4.3.

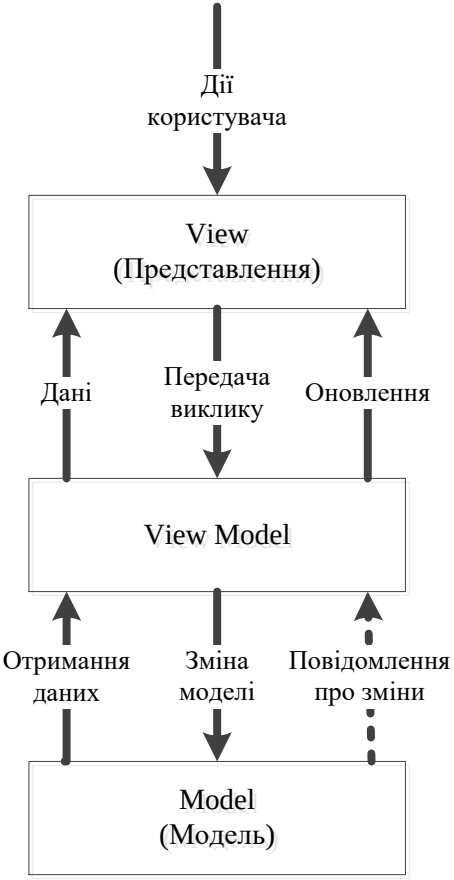

Рисунок 4.3 – Архітектура MVVM

Загалом MVC, MVP, MVVM – це схеми використання декількох шаблонів проєктування, за допомогою яких модель даних програми, користувацький інтерфейс і взаємодія з користувачем розділені на три окремих компоненти.

Логіка відображення (front end, frontend) – відповідає за збір даних від користувачів в різних формах та обробку їх у відповідності зі специфікацією.

Логіка управління даними (back-end, backend) – обробляє дані, отримані від користувача для передачі на наступний рівень.

Бізнес-логіка – сукупність правил, принципів, залежностей поведінки об'єктів предметної області. Це реалізація предметної області в інформаційній системі.

Особливості розробки архітектури вебзастосунку на основі предметної області:

– можливість більш докладного опису логіки взаємодії об’єктів предметної області;

– можливість гнучкої зміни логіки роботи додатку;

– відсутність надлишкового коду в кінцевій реалізації;

– поділ логіки відображення, логіки управління даними і бізнес-логіки.

Недоліки розробки архітектури вебзастосунку на основі предметної області:

– велика трудомісткість при створенні проєкту;

– більш жорсткі вимоги до кваліфікації розробника.

4.2.4 Фреймворки

Фреймворк (англ. framework) – у веб-програмуванні це спеціальна програмна платформа або комплекс компонентів і моделей, що полегшують процес веб-розробки. Хоча фреймворки відомі і в інших областях програмування, саме в веб-розробці в останні роки вони набули інтенсивного розвитку.

Зараз існують найрізноманітніші фреймворки, створені за допомогою таких мов програмування, як Java, PHP, Ruby і деяких інших. Особливо популярні проєкти для Java і PHP. Для найбільш затребуваних фреймворків створюються спільноти, розробляються підручники та документація.

4.2.4.1 Фреймворк Laravel

Laravel – безкоштовний веб-фреймворк з відкритим кодом, призначений для розробки з використанням архітектурної моделі MVC. Laravel випущений під ліцензією MIT. Сирцевий код проєкту розміщується на GitHub.

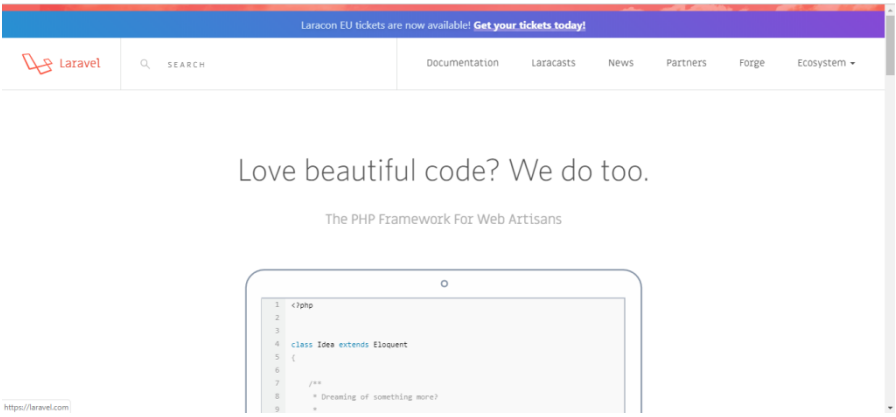

Рисунок 4.4 – Головна сторінка проєкту Laravel

Ключові особливості, що лежать в основі архітектури Laravel:

– пакети (англ. packages) – дозволяють створювати і підключати модулі у форматі Composer до додатка на Laravel. Багато додаткових можливостей вже доступні у вигляді таких модулів;

– Eloquent ORM – реалізація шаблону проєктування ActiveRecord на PHP. Дозволяє строго визначити відносини між об'єктами бази даних;

– логіка застосування – частина застосування, що розробляється, оголошена або за допомогою контролерів, або маршрутів (функцій-замикань);

– зворотна маршрутизація пов'язує між собою згенеровані застосуванням посилання і маршрути, дозволяючи змінювати останні з автоматичним оновленням пов'язаних посилань. При створенні посилань за допомогою іменованих маршрутів Laravel автоматично генерує кінцеві URL;

– REST-контролери – додатковий шар для поділу логіки обробки GET- та POST-запитів HTTP.

– автозавантаження класів – механізм автоматичного завантаження класів PHP без необхідності підключати файли їх визначень в *include*. Завантаження на вимогу запобігає завантаженню непотрібних компонентів; завантажуються тільки ті з них, які дійсно використовуються;

– укладачі представлень (англ. view composers) — блоки коду, які виконуються при генерації представлення;

– інверсія управління – дозволяє отримувати екземпляри об'єктів за принципом зворотного управління. Також може використовуватися для створення та отримання об'єктів-одинаків (англ. singleton);

– міграції – система управління версіями для баз даних (БД). Дозволяє зв'язувати зміни в коді програми зі змінами, які потрібно внести в структуру БД, що спрощує розгортання і оновлення програми;

– модульне тестування (юніт-тести) – відіграє дуже велику роль в Laravel, який сам по собі містить велику кількість тестів для запобігання регресій (помилок внаслідок оновлення коду або виправлення інших помилок);

– посторінкове виведення (англ. pagination) – спрощує генерацію сторінок, замінюючи різні способи вирішення цього завдання єдиним механізмом, вбудованим в Laravel.

Структура застосунку, що розробляється з використанням Laravel продемонстрована на рис. 4.5.

Структура Laravel проста і зрозуміла. Коренева директорія містить різні папки і файли:

– app – ця папка містить основний код програми;

– bootstrap – скрипт початкового завантаження програми;

– config – ця папка містить файли конфігурації програми;

– database – ця папка містить перенесену базу даних;

– public – це коренева папка документа програми. З неї запускається додаток Laravel. Вона також містить ресурси програми, такі як JavaScript, CSS, зображення, тощо;

– resources – ця папка містить вихідні ресурси, такі як файли LESS та Sass, файли локалізації та мов і шаблони, які відображаються як HTML;

– storage – ця папка містить сховище застосунку, наприклад, завантажені файли, тощо. Сховище для платформи (кеш) і журнали, створені за стосунком;

– test - ця папка містить різні тестові приклади;

– vendor – ця папка містить залежності Composer.

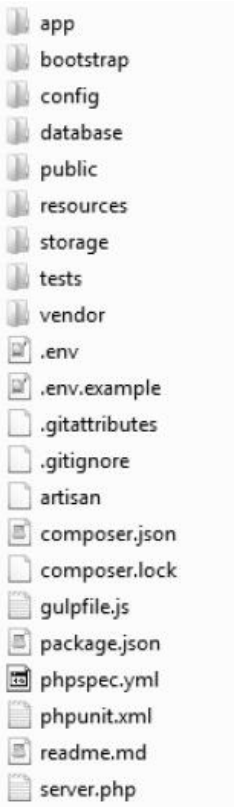

Рисунок 4.5 – Структура застосунку, що розробляється з використанням Laravel

App Directory – це каталог програми. Він містить інші додаткові папки, а саме:

– Console – всі команди майстра зберігаються в цій папці;

– Events – у цій папці зберігаються події, які підтримує застосунок. Події можуть використовуватися для оповіщення інших частин

програми про те, що відбулася певна дія, що забезпечує розробникам велику гнучкість;

– Exceptions – ця папка містить обробник винятків програми, і в ній також можна зберігати будь-які винятки, створювані додатком;

– Http – ця папка містить контролери, фільтри та запити;

– Jobs – ця папка містить завдання для застосунку;

– Listeners – ця папка містить класи обробників подій. Обробники приймають подію і виконують логіку відповідно до цього. Наприклад, подія UserRegistered може оброблятися обробником SendWelcomeEmail;

– Policies – ця папка містить різні політики, які можна встановити для програми;

– Providers – ця папка містить провайдери служб.

4.2.4.2 Фреймворк Yii2

YesItIs (Yii) – це високопродуктивний компонентний PHP фреймворк, призначений для швидкої розробки вебзастосунків. Yii – це універсальний фреймворк і може бути задіяний у всіх типах вебзастосунків. Завдяки його компонентній структурі та відмінній підтримці кешування, фреймворк особливо підходить для розробки таких великих проєктів, як портали, форуми, CMS, магазини або RESTful-додатки.

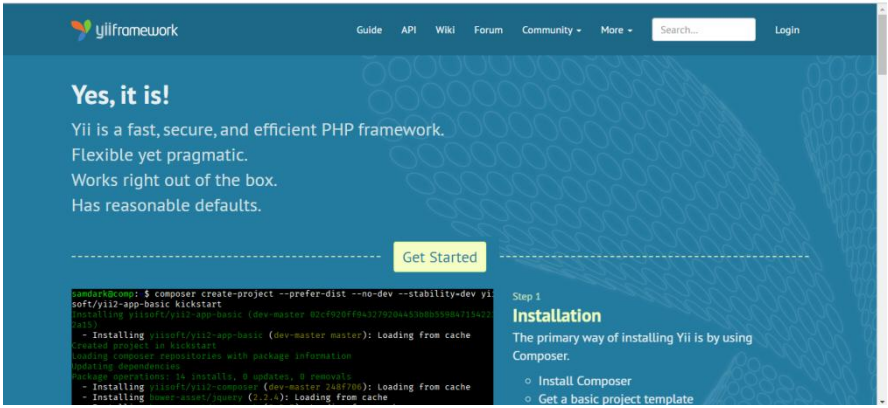

Рисунок 4.6 – Головна сторінка проєкту Yii Framework

Якщо порівняти Yii з іншими популярними фреймворками, можна відмітити:

– як і багато інших PHP фреймворків, для організації коду Yii використовує архітектурний шаблон MVC;

– Yii дотримується філософії простого і елегантного коду, не намагаючись ускладнювати дизайн тільки заради проходження будь-яким шаблонами проєктування;

– Yii є full-stack фреймворком і включає в себе перевірені і добре зарекомендували себе можливості, такі як ActiveRecord для реляційних і NoSQL баз даних, підтримку REST API, багаторівневе кешування та інші;

– Yii відмінно розширюється. Можна налаштувати або замінити практично будь-яку частину основного коду.

Yii підтримується і розвивається сильною командою і великою спільнотою розробників. Автори фреймворку стежать за тенденціями веб-розробки і розвитком інших проєктів. Найбільш підходящі можливості і кращі практики регулярно впроваджуються в фреймворк у вигляді простих і елегантних інтерфейсів.

Структура застосунку, що розробляється за допомогою Yii продемонстровано на рис. 4.7.

Рисунок 4.7 – Структурна схема Yii-застосунку

Структура найпростішого додатку на Yii буде наступною:

– assets – файли ресурсів;

– css – каскадні таблиці стилів;

– framework – коди фреймворку Yii;

– image – зображення;

– protected – функціональна частина сайту;

– themes – теми оформлення Yii;

– index-test.php – тестовий вхідний файл, для опробування різноманітних функцій;

– index.php – вхідний скрипт будь-якого проєкту на Yii, саме цей файл запускає роботу фреймворку.

У protected знаходяться 4 найважливіші складові:

– config – через цей файл можна налаштувати роботу сайту за потребами людини, тобто підключити якісь модулі, базу даних, GII, тощо;

– controllers – класи, які управляють логікою сайту;

– models – моделі, які визначають привила роботи із базою (тобто, майже із усією інформацією на сайті);

– views – файли відображення даних кінцевому користувачеві.

Основна схема роботи Yii така: при вході користувача на сайт (заповненні форми, перехід на іншу сторінку і т. д.) починає працювати контролер (який контролер починає працювати, залежить від дій користувача), який в свою чергу починає використовувати моделі (якщо потрібно отримати якісь дані з бази даних або навпаки записати туди) і представлення (якщо користувачеві необхідно надати результати роботи контролера).

Контролери складаються з подій (action), які можуть виконувати різні дії. Всі ці події починаються зі слова action і закінчуються приставкою, наприклад Index. У найпростішому випадку контролер складається з таких подій (action):

– actionIndex – подія запускається за замовчуванням при першому зверненні до контролера;

– actionError – подія описує роботу сайту при появі помилки;

– actionContact – подія описує роботу з контактною формою;

– actionLogin – подія авторизації користувача на сайті;

– actionLogout – подія виходу користувача з особистого кабінету.

Представлення зберігаються в папці views, і кожне з них відповідає своєму контролеру.

4.2.5 Системи управління вмістом

CMS (Content Management System – система управління вмістом) – програмний комплекс, що надає функції створення, редагування, контролю та організації структури вебзастосунку (сайту), а також текстової та графічної інформації (вмісту).

Переваги використання CMS при розробці:

– протестований і перевірений багатьма розробниками код;

– універсальність;

– простота використання.

Недоліки використання CMS при розробці:

– обмеження предметної області тими об'єктами, які передбачені в CMS;

– визначення основної логіки роботи програми розробниками CMS. Внесення змін в логіку проблематично;

– зайва універсальність і надлишковий функціонал.

CMS поділяються на ECMS (Enterprise Content Management System – система управління вмістом масштабу підприємства) та WCMS (Web Content Management System – система управління вебвмістом).

WCMS складається з клієнтської та серверної частин.

Клієнтська частина – веб-сайт, доступний для відвідувачів та зареєстрованих користувачів.

Серверна частина містить шар адміністрування веб-сайту, з яким взаємодіє адміністратор, де виконується конфігурація, обслуговування, очищення, генерація статистики та створення нового контенту.

Wordpress, Joomla та Drupal є прикладами безкоштовних CMS.

WordPress дозволяє створювати сайти різних типів, інформаційні, новинні тощо, але в першу чергу використовується для створення блогів або нескладних сайтів інформаційного типу.

Недоліки WordPress: не досить швидка робота сайту, можливість збоїв у випадку високої відвідуваності.

Joomla має більш широку галузь застосування порівняно з Wordpress і загалом є універсальною. Не позбавлена проблем зі швидкістю роботи за високої відвідуваності.

Drupal може використовуватись для створення форумів, блогів, онлайн-енциклопедій, сайтів спільнот. Проте є менш універсальною порівняно з Joomla. До недоліків відносять слабке використання об'єктних можливостей PHP та відсутність зворотної сумісності API.

Magento – система управління інтернет-магазинами. Magento – це досить молодий движок. Движок побудований на ZendFramework, що відразу визначає його обсяг: Він великий і важкий, але й досить потужний. На відміну від Joomla, де прямо в адміністративній частині можна призначити місце відображення і параметри конкретного модуля на сторінці, Magento в основному націлена на редагування екранних блоків вітрини не через адміністративну частину, а через файли логічної розмітки і файли шаблонів.

OpenCart — система керування вмістом з відкритим кодом, призначена для створення інтернет-магазинів. Розповсюджується за ліцензією GNU General Public License. OpenCart може бути встановлено на будь який веб-сервер Apache з підтримкою PHP5 і MySQL. Навколо OpenCart сформувалася велика спільнота, завдяки якій створено понад 8500 безкоштовних розширень у вигляді додаткових модулів. Найвагомішими перевагами OpenCart над системами Magento, Virtuemart і osCommerce є сучасна MVC-архітектура, підвищена швидкість роботи, vQmod, багатофункціональна адміністративна панель управління контентом та менше споживання серверних ресурсів. OpenCart добре зарекомендував себе в комерційному секторі як надійна і недорога в обслуговуванні система електронної торгівлі, з підтримкою розрахунку всіма найвідомішими системами електронної оплати.

PrestaShop – CMS для інтернет-магазинів з відкритим кодом. Система написана на PHP, для написання шаблонів використовується Smarty, для створення баз даних використовується MySQL. Система призначена для малого та середнього бізнесу і має більше 310 стандартних функцій для швидкого створення функціонального магазину, підтримує величезну кількість модулів і плагінів без додаткових налаштувань.

Для роботи сайту та його просування фреймворки, можливо, будуть навіть ефективніше, ніж системи управління сайтом. На відміну від останніх, фреймворки забезпечують сайтам швидкодію і не ви-

магають багато ресурсів. Втім, CMS, по суті, теж є фреймворком, а точніше, більш модернізованим фреймворком, в той час як, наприклад, Laravel або Yii вважаються чистими фреймворками. Рішення що використовувати для розробки сайту: фреймворк або CMS – приймається під час проєктування та залежить від: мети, конкретного завдання, складності проєкту, ресурсів та кінцевого користувача.

4.3 Завдання на лабораторну роботу

4.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

4.3.2 Проаналізувати предметну область.

4.3.3 Розробити архітектуру вебзастосунку на основі предметної області.

4.3.4 Обрати фреймворк для розроблення вебзастосунку.

4.3.5 Ознайомитись з архітектурою обраного фреймворку.

4.3.6 Розробити архітектуру вебзастосунку на основі використання обраного фреймворку.

4.3.7 Порівняти архітектуру вебзастосунку, розроблену з використанням фреймворку, з архітектурою, розробленою на основі предметної області. Визначити, який варіант розроблення архітектури є більш ефективним для обраної теми проєктування.

4.3.8 Оформити звіт з роботи.

4.3.9 Відповісти на контрольні питання.

4.4 Зміст звіту

4.4.1 Тема та мета роботи.

4.4.2 Тема, обрана для проєктування.

4.4.4 Схематичне зображення архітектури вебзастосунку на основі предметної області.

4.4.5 Схематичне зображення архітектури вебзастосунку (на основі фреймворку або CMS).

4.4.6 Обґрунтування вибору архітектури вебзастосунку.

4.4.7 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

4.5 Контрольні запитання

4.5.1 Наведіть особливості архітектури MVC.

4.5.2 Наведіть особливості архітектури MVP.

4.5.3 Наведіть особливості архітектури MVVM.

4.5.4 Які недоліки несе в собі проєктування вебзастосунків на основі предметної області?

4.5.5 Наведіть узагальнену схему вебзастосунку на основі предметної області.

4.5.6 Що таке frontend-частина вебзастосунку?

4.5.7 Що таке backend-частина вебзастосунку?

4.5.8 Що таке бізнес-логіка?

4.5.9 Наведіть узагальнену схему веб- застосунку на основі Laravel.

4.5.10 Наведіть узагальнену схему вебзастосунку на основі Yii.

4.5.11 Як відрізняється структура застосунку в Yii та Laravel?

4.5.12 Які переваги та недоліки має проєктування вебзастосунків на основі CMS порівняно з проєктуванням на основі предметної області?

4.5.13 Наведіть узагальнену схему вебзастосунку на основі CMS.

4.5.14 За якого варіанту побудови архітектури зв’язність між компонентами вебзастосунку є вищою?

4.5.15 Які функціональні можливості надають системи керування вмістом?

5 ЛАБОРАТОРНА РОБОТА №5 ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗРОБЛЕНОЇ АРХІТЕКТУРИ ВЕБЗАСТОСУНКУ

5.1 Мета роботи

Навчитися використовувати середовища розробки для роботи з мовою програмування PHP.

Навчитися використовувати пакетний менеджер Composer для встановлення та роботи із сучасними фреймворками.

5.2 Короткі теоретичні відомості

5.2.1 Менеджер пакетів Composer

При створенні вебзастосунку з використанням PHP використовуються поняття **пакету** та **репозиторію**.

Пакет – власне, та сама бібліотека, яка створюється або використовується в проєкті, як залежність.

Репозиторій (Registry) – сховище пакетів PHP, яке називається Packagist. Кожен бажаючий може опублікувати пакет в Packagist, витративши буквально хвилину часу, а інші зможуть його використовувати. Пакети в PHP ніяк не пов'язані з Git і GitHub.

Для вирішення проблем з повторним використанням та управлінням пакетами використовуються менеджери пакетів.

Composer – менеджер пакетів прикладного рівня для мови програмування PHP, що забезпечує стандартний формат для управління залежностями у програмному забезпеченні та необхідними бібліотеками.

Composer працює з командного рядка і встановлює залежності (наприклад, бібліотек) для застосунку. Він також дозволяє користувачам встановлювати PHP пакети, доступні на ресурсі «Packagist», який є його основним сховищем, що містить доступні пакети. Він також реалізує автозавантажувач класів, для встановлених бібліотек і це полегшує використання коду від сторонніх розробників.

Composer використовується як складова частина декількох популярних PHP проєктів з відкритим вихідним кодом, серед яких є бі-

льшість популярних фреймворків: Symfony, CodeIgniter, CakePHP, FuelPHP та, звісно ж, Laravel й Yii. Серед CMS Composer використовує Drupal починаючи з 8-ої версії.

5.2.2 Середовища розробки

При роботі з PHP у якості середовища розробки можна використовувати навіть звичайний «Блокнот» від компанії Microsoft, проте це буде вкрай не зручно та мало результативно.

Серед середовищ розробки при роботі із PHP можна виділити декілька вдалих рішень, що поширюються із вільною ліцензією.

NetBeans IDE – вільне інтегроване середовище розробки. Середовище підтримує велику кількість мов програмування, серед яких: Java, JavaFX, C/C++, PHP, JavaScript, HTML5, Python, Groovy. Середовище може бути встановлене як для підтримки окремих мов, так у повній конфігурації.

NetBeans IDE доступна для платформ Microsoft Windows, GNU/Linux, FreeBSD, і Solaris (як SPARC, так x86). За якістю і можливостям останні версії NetBeans IDE змагаються з найкращими інтегрованими середовищами розробки, підтримуючи рефакторинг, профілювання, виділення синтаксичних конструкцій кольором, автодоповнення мовних конструкцій на льоту, шаблони коду та інше.

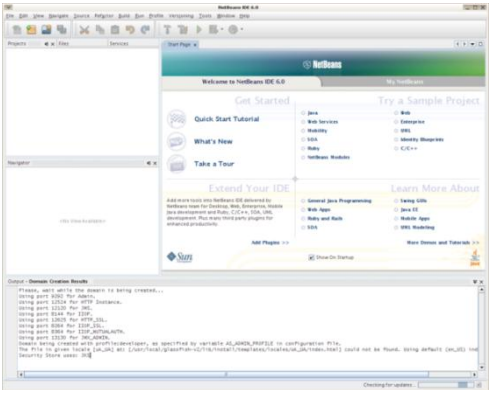

Рисунок 5.1 – Середовище розробки NetBeans

Eclipse PHP Development Tools – інтегроване середовище розробки застосунків на мові програмування PHP, розроблене на основі Eclipse, поширюване на умовах ліцензії Eclipse Public License. Серед особливостей можна виділити:

– згортання коду;

– рефакторинг;

– генерація коду (методи доступу до членів класу, майстер класів та інтерфейсів)

– аналіз та виправлення коду;

– підтримка PHP 4 і PHP 5 (включаючи замикання і простору імен);

– ієрархічне представлення класів і методів

– налагодження PHP-скриптів (як локально, так і за допомогою інтеграції з Zend Server і XDebug);

– підтримка HTML, CSS, JavaScript.

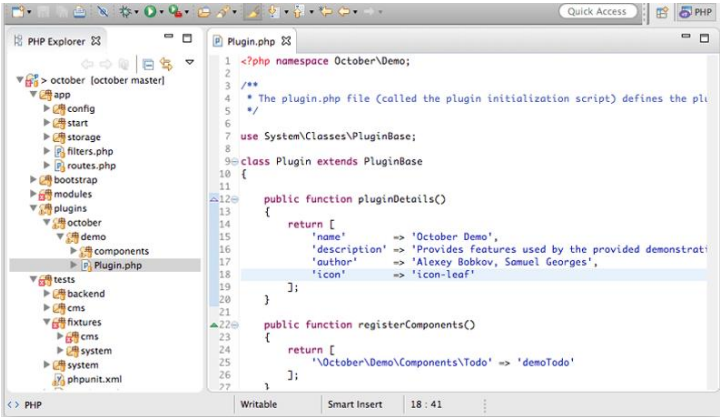

Рисунок 5.2 – Середовище розробки Eclipse PHP Development Tools

Atom – розроблений компанією «GitHub» вільний текстовий редактор і редактор коду, який може використовуватися як самодостатнє рішення, так і у ролі технологічного стека для побудови різних спеціалізованих рішень. Atom надає засоби крос-платформового редагування коду, включає вбудований пакетний менеджер і інтерфейс навігації файловою системою, надає засоби для одночасної спільної роботи з кодом, має інтелектуальну систему автодоповнення вводу, надає

режими сумісності з Vim і Emacs, підтримує API для розробки розширень. Кілька файлів можуть бути відкриті в різних вкладках і одночасно відображені з використанням вертикального або горизонтального розбиття панелей. Інтерфейс може налаштовуватися через теми оформлення, підтримуються вкладки, закладки, розумний контекстний пошук коду, схлопування блоків коду, одночасне використання декількох курсорів і областей виділення, наочна позначка змін, автодоповнення та перевірка коду для різних мов (Ruby, Python, SQL, PHP, Perl, Objective-C, C/C++, JavaScript, Java, Go тощо).

Для безпроблемної роботи з PHP рекомендується підключити пакети: Script, Language php, Linter php та Php Twig.

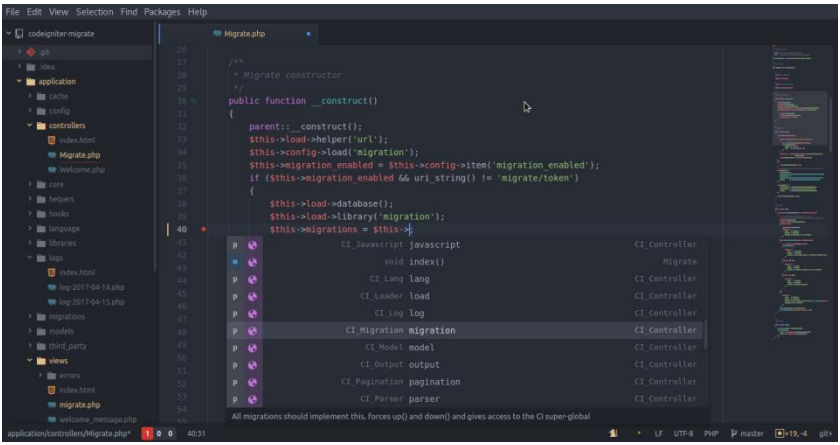

Рисунок 5.3 – Середовище розробки Atom

5.3 Завдання на лабораторну роботу

5.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

5.3.2 Встановити менеджер пакетів Composer.

5.3.3 Використовуючи Composer встановити обраний фреймворк.

5.3.4 Розробити структуру вебзастосунку на основі використання обраного фреймворку (або на основі предметної області, якщо обрано такий варіант побудови архітектури).

5.3.5 Розробити та підключити БД до вебзастосунку.

5.3.6 Розробити основні (або статичні) сторінки вебзастосунку.
5.3.7 Оформити звіт з роботи.
5.3.8 Відповісти на контрольні питання.

5.4 Зміст звіту

5.4.1 Тема та мета роботи.
5.4.2 Тема, обрана для проєктування.
5.4.3 Структура вебзастосунку, що реалізує систему.
5.4.4 Модель розробленої БД для вебзастосунку.
5.4.5 Скріншоти та сирцевий код розроблених веб-сторінок.
5.4.6 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

5.5 Контрольні запитання

5.5.1 Поясніть поняття пакету при розробці застосунку з використанням PHP.
5.5.2 Поясніть поняття репозиторію при розробці застосунку з використанням PHP.
5.5.3 Що таке менеджер пакетів?
5.5.4 У чому полягає ідея використання менеджеру пакетів?
5.5.5 Які середовища розробки зазвичай використовуються при розробці з використанням PHP?

6 ЛАБОРАТОРНА РОБОТА № 6 РОБОТА ІЗ СИСТЕМАМИ КЕРУВАННЯ ВЕРСІЯМИ. РОЗГОРТАННЯ РЕПОЗИТОРІЮ

6.1 Мета роботи

Навчитися використовувати системи керування версіями та розгортати репозиторій для роботи над вебзастосунком групи розробників.

6.2 Короткі теоретичні відомості

Система керування версіями (англ. Version Control System або Revision Control System) – це ПЗ для полегшення роботи зі змінною інформацією. Система управління версіями дозволяє зберігати кілька версій одного і того ж документа, при необхідності повертатися до більш ранніх версій, визначати, хто і коли вносив правки та зміни, та багато іншого.

Такі системи найбільш широко використовуються при розробці ПЗ для зберігання вихідних кодів розроблюваної програми. Однак вони можуть з успіхом застосовуватися і в інших областях, в яких ведеться робота з великою кількістю електронних документів, які безперервно змінюються. Зокрема, системи управління версіями застосовуються в системах автоматизованого проєктування, зазвичай в складі систем управління даними про виріб. Управління версіями використовується в інструментах конфігураційного управління.

Кожна система управління версіями має свої специфічні особливості в наборі команд, порядку роботи користувачів та адмініструванні. Тим не менш, загальний порядок роботи для більшості систем керування версіями абсолютно стереотипний. Тут передбачається, що проєкт, яким би він не був, вже існує, а на сервері розміщений його репозиторій, до якого розробник отримує доступ.

Тож загальній порядок роботи можна поділити на наступні етапи:

1) початок роботи з проєктом;
2) щоденний цикл роботи;
    а) оновлення робочої копії;

б) модифікація проєкту;
в) фіксація змін.

Git – розподілена система керування версіями файлів та спільної роботи. Проєкт було створено для управління розробкою ядра Linux. Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і комітів.

Система спроєктована як набір програм, спеціально розроблених з врахуванням їхнього використання у скриптах. Це дозволяє зручно створювати спеціалізовані системи управління версіями на базі Git або користувацькі інтерфейси. Система має ряд користувацьких інтерфейсів: наприклад, gitk та git-gui розповсюджуються з самим Git.

Серед основних переваг розгортання репозиторію з використанням Git можна виділити наступні:

– при завантаженні нових версій з віддаленої копії автоматично оновлюється live-копія сайту;

– виправлення файлів на сервері не будуть руйнувати історію оновлення версій;

– простота, не потрібні особливі правила виконання оновлення версій;

– можна застосувати до вже запущеного сайту, без повторного розгортання або переміщення файлів.

Слід відзначити, що механізм розгортання Git репозиторію на локальному комп’ютері (за умови використання його у якості локального серверу) та на сервері значно різняться. У випадку використання локального серверу вам не потрібен веб-сервер для Git. Ви можете створити сховище Git на локальній машині та клонувати його на тому ж комп'ютері.

Основні кроки для розгортання репозиторію на локальній машині мають наступний вигляд.

```
mkdir /c/GIT
cd /c/GIT/
git init --bare myproject.git

cd /c/UserWebServer/www/myproject/
```

```
git init
git add .
git commit -m "first commit"
git remote add origin /c/GIT/myproject.git
git push -u origin master
```

Тепер можна почати роботу над проєктом в папці `C:\UserWebServer\www\myproject\`.

Щоб створити і переключитися на використання нової гілки, `git checkout -b new_branch`.

6.2.1 Онлайн-репозиторії

Для більш зручної командної роботи над проєктами найчастіше використовують дистанційні або онлайн репозиторії. Найбільш популярні серед таких: GitHub, GitLab, Bitbucket. В цьому випадку можна вважати, що Git – це технологія – репозиторій коду, а онлайн-репозиторій – сервер який надає цю технологію.

Github – потужна, безпечна і найпопулярніша онлайн платформа для розміщення проєктів з контролем версій з використанням Git. Хоч Github більше відомий як платформа для розробки open source проєктів, ресурс також підтримує можливість використання приватних репозиторіїв. Окрім розміщення коду учасники можуть спілкуватися та коментувати правки один одного. За допомогою широких можливостей Git програмісти можуть об'єднувати свої репозиторії – GitHub пропонує зручний інтерфейс для цього і може відображати внесок кожного учасника у вигляді дерева. Прямо на сайті можна переглянути файли проєктів з підсвічуванням синтаксису для більшості мов програмування. Можна створювати приватні сховища, які будуть доступні лише розробнику та обраним вами людям. Є можливість прямого додавання нових файлів в свій репозиторій через веб-інтерфейс сервісу.

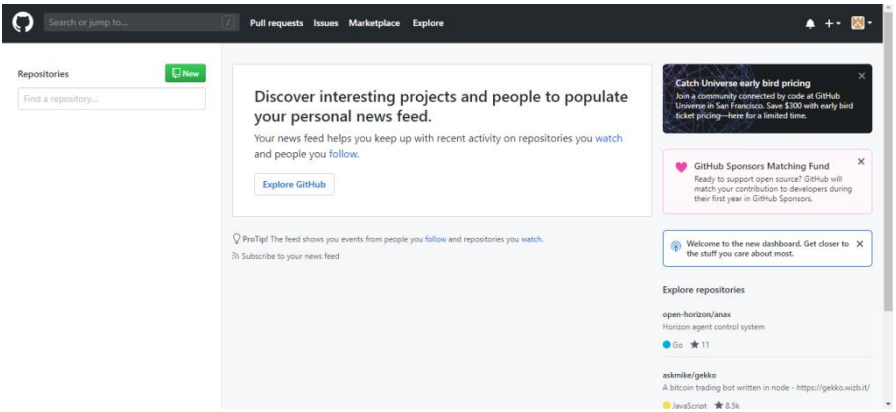

Рисунок 6.1 – Сервіс GitHub

GitLab – відкритий, потужний, безпечний, ефективний, надійний, наповнений різним функціоналом сервіс для розробки ПЗ на різних етапах життєвого циклу. Мабуть, це краща альтернатива Github. Платформа підтримує облік робочого часу, надає потужні інструменти для розгалуження, можливість захистити гілки і теги, функції блокування файлів, об'єднання запитів, персоналізації повідомлень, створення дорожніх карт проєктів, виставлення пріоритетів тікетам, створення конфіденційних і пов'язаних тікетів, графіка виконання робіт за проєктами. Крім цього, розробник може створювати тікети з листів і переглядати свої зміни за допомогою програм перевірки. GitLab також надає можливість використовувати Web IDE і дає доступ до численних шаблонів проєктів для того, щоб легше було почати роботу над проєктом, а також багато іншого. Використовуючи модуль імпорту, можна імпортувати репозиторії з GitHub в GitLab або на власний сервер GitLab.

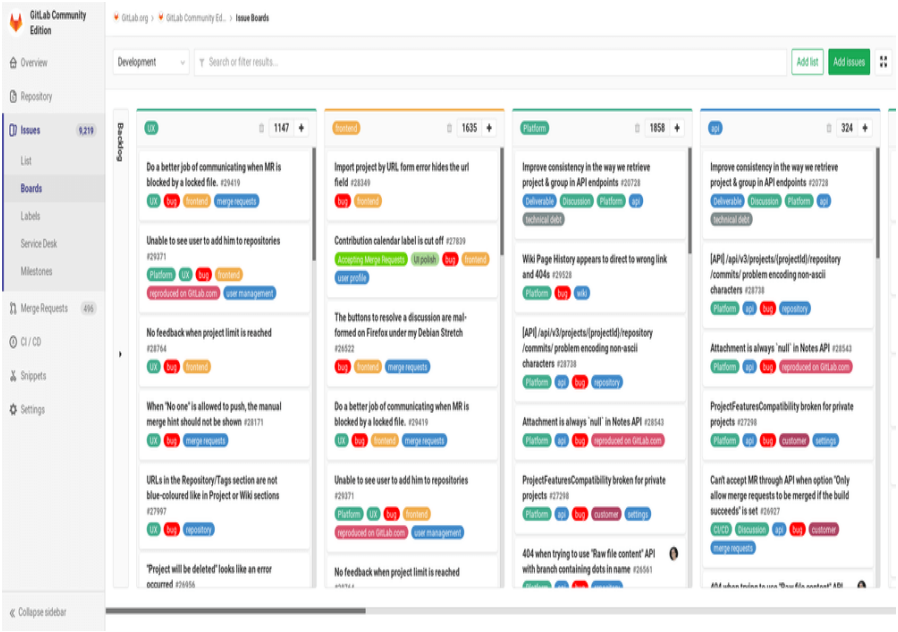

Рисунок 6.2 – Сервіс GitLab

Bitbucket – це потужна, повністю масштабована і високопродуктивна платформа для розробників, призначена спеціально для професійних команд. Початківці та команди, які працюють над проєктами з відкритим вихідним кодом безкоштовно отримують доступ до аккаунту Bitbucket і до багатьох його функцій. Bitbucket дозволяє легко імпортувати репозиторії з Github всього лише за 6 простих кроків, а також підтримує сторонні інтеграції. Від конкурентів платформу відрізняє наступний функціонал: Bitbucket pipelines, пошук коду, запити на включення коду, гнучкі моделі розгортання, порівняння, смарт-дублювання, відстеження тікетів і списку дозволених IP-адрес, а також можливості розгалуження для забезпечення безпеки робочого процесу. Bitbucket пропонує функцію підтримки Git сховища великих файлів (LFS) для розробки ігор, що дозволяє створювати необмежену кількість приватних репозиторіїв, і легко інтегрувати нову систему у вже існуючий робочий процес, а також має вбудований модуль для безперервних поставок.

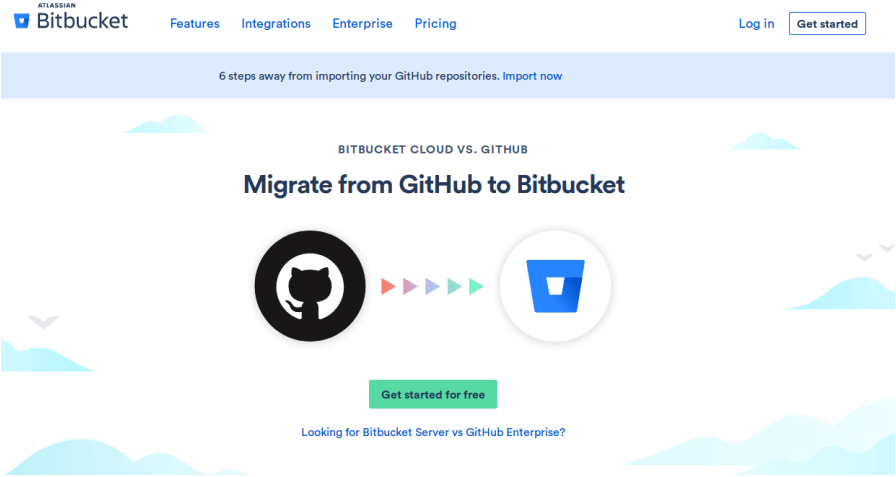

Рисунок 6.3 – Сервіс Bitbucket

6.3 Завдання на лабораторну роботу

6.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

6.3.2 Відправити (push) першу версію свого проєкту до репозиторію на GitLab, додавши його як віддалений і відправивши гілку.

6.3.3 Реалізувати основні функціональні вимоги до системи.

6.3.4 Відправити (push) до репозиторію зміни (реалізовані функціональні вимоги).

6.3.5 Оформити звіт з роботи.

6.3.6 Відповісти на контрольні питання.

6.4 Зміст звіту

6.4.1 Тема та мета роботи.

6.4.2 Тема, обрана для проєктування.

6.4.3 Скріншоти та сирцевий код розроблених веб-сторінок.

6.4.4 Скріншоти роботи із Git репозиторієм.

6.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

6.5 Контрольні запитання

6.5.1 Що таке система керування версіями?

6.5.2 Які задачі вирішує система керування версіями?

6.5.3 Назвіть системи керування версіями, окрім Git.

6.5.4 Поясніть різницю між локальним та дистанційним репозиторієм.

7 ЛАБОРАТОРНА РОБОТА №7 РОЗШИРЕННЯ ФУНКЦІОНАЛЬНОСТІ ВЕБЗАСТОСУНКІВ

7.1 Мета роботи

Навчитися використовувати модулі розширення функціональності для розроблення вебзастосунку.

7.2 Короткі теоретичні відомості

Більшість фреймворків та CMS зроблені за принципом модульності: є базова основа (ядро), до якої можливе підключення модулів, що надають додаткові функції, яких немає в ядрі фреймворка або CMS. Модулі можуть вже входити до складу комплексу, можуть бути встановлені окремо, а також багато систем допускають самостійну розробку та включення додаткових модулів. Таким чином, загальний функціонал сайту залежить від функцій включених в нього модулів.

Модулі – це закінчені програмні блоки, що складаються з моделей, представлень, контролерів та інших допоміжних компонентів. При встановленні модулів в застосунок, кінцевий користувач отримує доступ до їх контролерів. З цієї причини модулі часто розглядаються як мініатюрні застосунки.

7.2.1 Приклади розширення функціональності Yii

Віджети – це багаторазові будівельні блоки, використовувані в уявленнях для створення складних і параметрів елементів користувальницького інтерфейсу в рамках об'єктно-орієнтованого підходу. Наприклад, віджет вибору дати (datepicker) дозволяє створювати інтерактивний інтерфейс для вибору дат, надаючи користувачам програми зручний спосіб для введення даних такого типу.

В комплект Yii входить велика кількість віджетів, наприклад: active form, menu, віджети jQuery UI, віджети Twitter Bootstrap. Для отримання відомостей щодо використання конкретного віджета, слід звернутися до документації відповідного класу.

Для встановлення віджету, слід перейти до Composer та ввести наступну команду:

```
composer require адреса_віджету
```

Приклад встановлення віджету наведено на рис. 7.1.

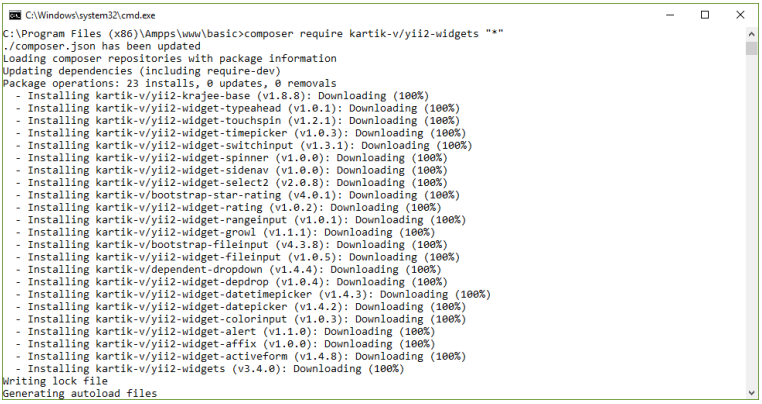

Рисунок 7.1 – Встановлення віджету

Далі потрібно підключити потрібний віджет:

```
use yii\widgets\ActiveForm;
```

Прикладом використання такого віджету може стати форма для заповнення на сторінці зворотного зв’язку (рис. 7.2) або на сторінці авторизації (рис. 7.3).

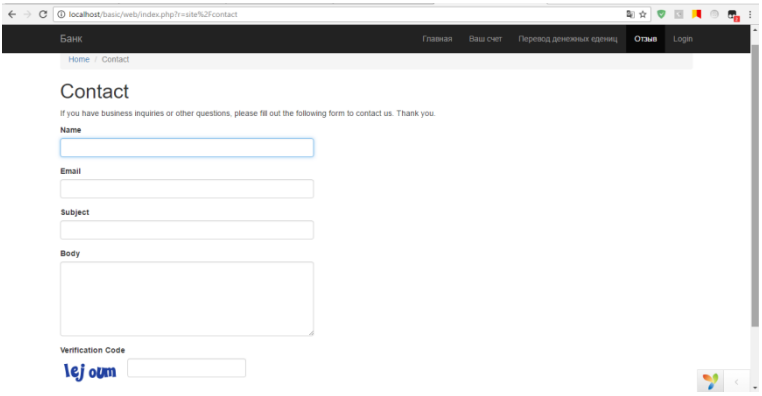

Рисунок 7.2 – Сторінка зворотного зв’язку

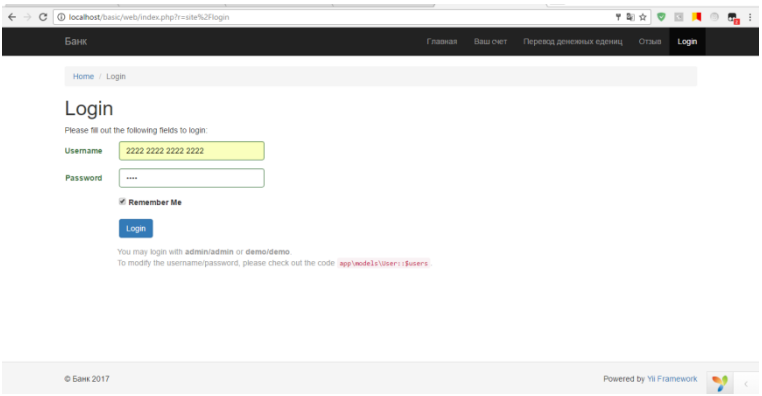

Рисунок 7.3 – Сторінка авторизації

7.2.2 Приклади розширення функціональності Laravel

Пакети (packages) – основний спосіб додавання нового функціоналу в Laravel. Пакети можуть бути всім, чим завгодно – від класів для зручної роботи з датами зразок Carbon, до цілих бібліотек BDD-тестування зразок Behat.

Звичайно, є різні типи пакетів. Деякі пакети автономні, що дозволяє їм працювати в складі будь-якого PHP-фреймворка, не тільки Laravel. Прикладами таких окремих пакетів є Carbon і Behat. Будь-який з них може бути використаний в Laravel за допомогою простого додавання їх до файлу composer.json.

З іншого боку, деякі пакети розроблені спеціально для використання в Laravel. Вони можуть містити маршрути, контролери, подання та налаштування, спеціально розраховані для поліпшення програми на Laravel.

Всі пакети Laravel поширюються через Packagist та Composer.

Composer використовує пакети з репозиторію Packagist. Ви не можете просто використовувати GitHub-пакети безпосередньо з GitHub.

У розробників є 2 варіанти:

– можна використовувати пакет, доступний на Packagist для інтеграції з API-інтерфейсом shopify. Щоб додати такий пакет до проєкту Laravel, у папці, яка містить файл composer.json, слід використати наступну команду:

```
composer require rocket-code/shopify
```

– можна створити свій власний пакет packagist, використовуючи сховище Git. В цьому випадку спочатку клонується репозиторій, а потім створюється свій власний обліковий запис у пакеті.

7.2.3 Приклади розширення функціональності CMS

Нижче наведено декілька прикладів модулів, які можна використати для розширення функціональності CMS Drupal, завантаживши з репозиторію на drupal.org:

– Captcha – захисний механізм картинок «CAPTCHA», що використовується при реєстрації;

– Ecommerce, Ubercart – системи електронної комерції;

– FCKeditor, CKEditor, Tinymce – візуальні редактори;

– Gallery – інтеграція з галереєю зображень Gallery2;

– Project – ведення проєктів, включає багтрекер і інтеграцію з CVS і Subversion;

– SPAM – блокування спаму;

– WebForm – гнучкий модуль для швидкого проєктування інтерактивних форм (опитування, зворотній зв'язок).

Для інсталяції модуля в Drupal необхідно виконати наступні кроки:

1) завантажити модуль зі сторінки drupal.org/project/[модуль], впевнившись, що версія модуля відповідає версії системи;

2) розпакувати файли з архіву;

3) ознайомитись з файлами *README.txt* та *INSTALL.txt*, які містять інформацію про можливості модуля, його інсталяцію та налаштування;

4) скопіювати папку модуля в системну папку додаткових модулів *sites/all/modules*;

5) перейти на сторінку «Управління → Створення сайту → Модулі» та включити встановлений модуль;

6) виконати для модуля необхідне налаштування;

7) перейти на сторінку «Управління → Користувачі → Права доступу» та визначити права доступу до модуля.

Модуль CMS Drupal складається з двох файлів, які мають міститись у /sites/all/modules: ім’ямодуля.info та ім’ямодуля.module. Другий файл містить код модуля, а перший – інформацію про модуль: ім’я,

опис, версію ядра, які підтримує модуль, а також список модулів, необхідних для роботи даного модуля, та пакет, до якого відноситься модуль.

Приклади плагінів, які використовуються для розширення функціональності ядра CMS Wordpress:

– WordPress Database Backup – дає змогу зробити бекап бази даних;

– MobilePress – створює мобільну версію сайту;

– All in One SEO Pack – дозволяє виконувати пошукову оптимізацію;

– Akismet – допомагає боротися зі спам-коментарями;

– Simple Tags – дозволяє працювати з тегами;

– Related Posts – дозволяє виводити наприкінці кожної публікації список схожих статей.

Для інсталяції плагіну в Wordpress необхідно: завантажити файли обраного плагіну і завантажити їх у папку /public_html/wp-content/plugins/Wordpress. Після цього можливість активації плагіну з'явиться у адміністративній частині (пункт меню «Модулі»). Слід натиснути на кнопку «Активувати» та увімкнути плагін.

Плагін Wordpress складається з файлу (або файлів) PHP, а також може додатково містити файли JavaScript, CSS, зображення тощо.

На початку файлу PHP необхідно розмістити стандартний інформаційний заголовок (назва плагіну, адреса веб-сторінки з описом плагіну, короткий опис плагіну, номер версії плагіну, ім'я автора та його веб-сторінка) та інформацію про ліцензію.

Для розширення функціональності CMS Joomla можна зокрема використати наступні модулі:

– mod_syndicate – відображає посилання на RSS-стрічку поточної сторінки;

– mod_wrapper – створює у вказаній позиції вікно, в якому відображається сторінка;

– mod_poll – виводить в обрану позицію голосування;

– mod_latestnews – виводить список останніх опублікованих матеріалів;

– mod_related_items – порівнює ключові слова у поточному матеріалі та знаходить у базі даних матеріали з ключовими словами, що співпадають, після чого виводить список схожих матеріалів;

– mod_stats – відображає статистику сайту.

Для інсталяції модуля в Joomla необхідно виконати наступні кроки:

1) завантажити модуль;

2) зайти в адміністративний центр та перейти в «Розширення» => «Менеджер модулів»;

3) натиснути на кнопку «Створити» на панелі інструментів та обрати необхідний модуль;

4) налаштувати параметри модуля та натиснути «Зберегти».

Модуль CMS Joomla складається мінімально з двох файлів: mod_назвамодуля.xml та mod_назвамодуля.php, які за замовчуванням розташовуються в папці «modules» на сервері.

Файл mod_назвамодуля.xml – багатофункціональний файл, який містить загальні відомості про модуль – назву, опис, авторство, версію, відомості для інсталювання/деінсталювання модуля – список файлів та шляхи їх розміщення, список параметрів для конфігурації модуля.

Файл mod_назвамодуля.php містить програмний код модуля.

7.3 Завдання на лабораторну роботу

7.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

7.3.2 Ознайомитись з існуючими модулями розширення функціональності для обраного фреймворку.

7.3.3 Проаналізувати вимоги до функціональності, що визначені в оформленій документації на ПЗ.

7.3.4 Визначити необхідні модулі розширення функціональності.

7.3.5 Застосувати обрані модулі для вебзастосунку.

7.3.6 Оформити звіт з роботи.

7.3.7 Відповісти на контрольні питання.

7.4 Зміст звіту

7.4.1 Тема та мета роботи.

7.4.2 Тема, обрана для проєктування.

7.4.3 Вимоги до функціональності.

7.4.4 Опис використаних модулів.

7.4.5 Архітектура вебзастосунку після використання модулів.

7.4.6 Структура вебзастосунку після використання модулів.

7.4.7 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

7.5 Контрольні запитання

7.5.1 Що таке модуль?

7.5.2 Які модулі включає стандартний набір модулів обраного фреймворку?

7.5.3 З яких файлів складається модуль розширення функціональності?

7.5.4 Як інсталювати модуль розширення функціональності в проєкт при роботі із обраним фреймворком?

8 ЛАБОРАТОРНА РОБОТА № 8 ТЕСТУВАННЯ ТА АНАЛІЗ ЯКОСТІ ВЕБЗАСТОСУНКІВ

8.1 Мета роботи

Навчитися оцінювати якість вебзастосунків на основі проведення тестування.

8.2 Короткі теоретичні відомості

Тестування ПЗ (Software Testing) – це процес технічного дослідження, який виконується на вимогу замовників, і призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись.

Програмний дефект (баг) – помилка, вада або дефект в програмі або системі, що викликає в ній неправильний або неочікуваний результат або неочікувану поведінку. Термін зазвичай використовується стосовно помилок, котрі виявляються на стадії роботи програми, на відміну від помилок проєктування чи синтаксичних помилок. «Баги» локалізуються та виправляються у процесі тестування та доробки програми.

Різні системи баг-трекінгу пропонують різні шляхи опису **серйозності** та **пріоритету** баг репорту, незмінним залишається лише сенс, що вкладається в ці поняття.

Серйозність (severity) – це атрибут, що характеризує вплив дефекту на працездатність програми.

Пріоритет (priority) – це атрибут, який вказує на черговість виконання завдання або усунення дефекту. Можна сказати, що це інструмент менеджера з планування робіт. Чим вище пріоритет, тим швидше потрібно виправити дефект.

Градація серйозності дефекту може бути наступною:

– блокуюча (blocker). Блокуюча помилка, що приводить застосунок в неробочий стан, в результаті якого подальша робота з системою або її ключовими функціями стає неможлива. Вирішення проблеми необхідне для подальшого функціонування системи;

– критична (critical). Критична помилка, неправильно працює ключова бізнес логіка, діра в системі безпеки, проблема, яка призвела до тимчасового падіння сервера або приводить в неробочий стан деяку частину системи, без можливості вирішення проблеми, використовуючи інші вхідні точки. Вирішення проблеми необхідно для подальшої роботи з ключовими функціями системи;

– значна (major). Значна помилка, частина основної бізнес логіки працює некоректно. Помилка не критична або є можливість для роботи з тестованою функцією, використовуючи інші вхідні точки;

– незначна (minor). Незначна помилка, що не порушує бізнес-логіку частини програми, що тестується, очевидна проблема користувацького інтерфейсу;

– тривіальна (trivial). Тривіальна помилка, не стосується бізнес-логіки за стосунку. Загалом, така помилка не надає ніякого впливу на загальну якість продукту. Зазвичай, малопомітна у користувацькому інтерфейсі, наприклад проблема сторонніх бібліотек або сервісів.

Градація пріоритету є наступною:

– високий (high). Помилка повинна бути виправлена якомога швидше, тому що її наявність є критичною для проєкту;

– середній (medium). Помилка повинна бути виправлена, її наявність не є критичною, але вимагає обов'язкового рішення;

– низький (low). Помилка повинна бути виправлена, її наявність не є критичною, і не вимагає термінового рішення.

Вид тестування сфокусований на конкретну мету тестування, яка може бути перевіркою функції, виконуваної компонентом, або системи в цілому. Мета тестування може бути спрямована як на перевірку елементів нефункціонального тестування (надійність, зручність використання), структури, архітектури компонентів або системи в цілому, так і на елементи залежно від змін у системі (перевірка виправлення конкретного дефекту (повторне тестування або перевірка ненавмисних змін).

Залежно від мети сам процес тестування повинен бути організований відповідним чином. Отже, можна визначити 4 види тестування ПЗ:

– функціональне тестування;

– нефункціональне тестування;

– структурне тестування;

– тестування змін.

Функціональне тестування – тестування ПЗ, головною метою якого є перевірка реалізованості функціональних вимог додатка, тобто здатність додатка в заданих критеріях вирішувати поставлені завдання.

Елементи функціонального тестування:

– підготовка тестових даних виходячи з документації;

– бізнес-вимоги, як частина функціонального тестування;

– отримання результатів на основі специфікації;

– проходження тест-кейсів;

– аналіз фактичних і очікуваних результатів.

Переваги функціонального тестування:

– в рамках тестування проходить безпосереднє використання системи;

– тестування, як правило, проводиться в умовах близьких до реальних.

Недоліки функціонального тестування:

– існує ймовірність пропустити деяку кількість помилок логіки ПЗ під час перевірки функціоналу програми.

Нефункціональне тестування ПЗ – в першу чергу перевірка на відповідність нефункціональним вимогам:

– зручність (загалом проводиться оцінка зручності для користувачів);

– масштабованість;

– продуктивність (працездатність програми при різних навантаженнях);

– безпека (захист даних, захист даних програми, стійкість від зламу);

– сумісність (сумісність та можливість переносу програми для й під різні оточення, платформи, тощо);

– надійність (поведінка системи при різних непередбачених ситуаціях, здатність обробки нестандартних дій користувача).

У нефункціональному тестуванні можна виділити наступні типи тестувань:

– тестування стабільності програми – виявлення відмов системи під час використання;

– юзабіліті тестування – дослідження зручності використання;

– тестування ефективності – перевірка необхідних обсягів кодів, та ресурсів необхідних програмі для виконання окремої функції;

– тестування ремонтопридатності – визначає, наскільки легко підтримувати працездатність системи;

– перевірка сумісності (portability testing) – тестування доступності перенесення окремого компонента або всієї програми;

– тестування «пра-витоків» (baseline testing) – перевірка документації та специфікації, за якою будуть написані тест-кейси. До цього підвиду тестування можна віднести і тестування вимог;

– приймальне тестування – перевірка продукту на відповідність критеріям готовності;

– тестування документації – перевірка створеної в рамках тестування документації;

– тестування витривалості системи – тестування системи при високому навантаженні протягом тривалого періоду часу з метою вивчення її поведінки;

– навантажувальне тестування – як правило, проводиться з метою визначення поведінки під очікуваним рівнем навантаження;

– тестування продуктивності – перевірка швидкості роботи ПЗ або його окремих функцій;

– тестування сумісності – тестування в різних середовищах: апаратна частина, програмна частина, тощо;

– тестування безпеки – перевірка застосунку на безпечність та захищеність;

– об'ємне тестування – тестування з використанням баз даних певного розміру;

– стрес тестування – це тестування в обмежених умовах;

– тестування швидкості відновлення – визначення швидкості відновлення системи;

– тестування локалізації, інтернаціоналізація – перевірка на відповідність мовним, культурним та релігійним нормам.

Структурне тестування спрямоване на тестування структури системи або компонента. Цей вид тестування, як правило, відносять до тестування «білого» і «чорного» ящиків, так як перевіряється, що відбувається всередині системи або програми.

Методи структурного тестування:

– рядкове покриття – перевірка застосування всіх операторів у програмі на використання (хоча б один раз);

– покриття шляху – перевірка для виконання критеріїв охоплення кожного логічного шляху через програму;

– покриття рішення – перевіряє, чи має кожна умова розгалуження для програми істинні (true) або помилкові (false) значення;

– покриття умови – перевірка стану покриття для умовних і не умовних гілок.

Переваги структурного тестування:

– можливість виявити і видалити «зайвий» код;

– можливість виявлення потенційних помилок на ранній стадії;

– забезпечує більш ретельне тестування ПЗ;

– не вимагає високих витрат людино-годин.

Недоліки структурного тестування:

– вимагає знання коду та інструментів тестування.

Тестування змін – проводиться для зрозуміння різниці та кордонів між поняттями «регресійне тестування» та «повторне тестування».

Регресійне тестування (regression testing) проводиться з метою перевірки працездатності існуючого функціоналу і відсутності сторонніх помилок після внесення поправок чи доповнень в систему.

Повторне тестування (pretesting) – проводиться для підтвердження виправлення помилки і роботи даного функціоналу.

Види тестування розрізняють за часом проведення:

– альфа-тестування:

– димове тестування (smoke testing);

– тестування нової функції (new feature testing);

– регресійне тестування;

– приймальне тестування;

– бета-тестування.

Альфа-тестування (alpha testing) – це вид приймального тестування, яке зазвичай проводиться на пізній стадії розробки продукту і включає імітацію реального використання продукту штатними розробниками або командою тестувальників. Зазвичай альфа-тестування полягає в систематичній перевірці всіх функцій програми з використанням технік тестування «білого ящика» і «чорного ящика».

Бета-тестування (beta testing) – інтенсивне використання майже готової версії продукту з метою виявлення максимального числа помилок у його роботі для їх подальшого усунення перед остаточним виходом (релізом) продукту на ринок, до масового споживача. Бета-тестування являє собою реально працюючу версію програми з повним

функціоналом. Завдання бета-тестів – оцінити можливості і стабільність роботи програми з точки зору її майбутніх користувачів.

У таблиці 8.1 наведено порівняння альфа- та бета-тестування.

Таблиця 8.1 – Порівняння альфа- та бета-тестування

Альфа-тестування	Бета-тестування
Головна мета	
Підвищує якість продукту і забезпечує готовність до бета-тестування.	Підвищує якість продукту, інтегрує дані про клієнта в готовий продукт та забезпечує готовність до випуску.
Час проведення	
Ближче до кінця процесу розробки, коли продукт знаходиться в майже повністю працездатному стані.	Безпосередньо перед запуском.
Тривалість тестування	
Зазвичай дуже довго – протягом багатьох ітерацій (часто в 3-5 разів довше бета-тестування).	Зазвичай тільки кілька тижнів (іноді до декількох місяців) з невеликою кількістю основних ітерацій.
Перевіряє	
Майже виключно перевірка якості ПЗ.	Зазвичай включає в себе маркетинг, підтримку, документацію, якість і інжиніринг (в основному, всю групу продуктів).
Хто проводить	
Зазвичай виконується тестерами, розробниками. Фокусується на тестуванні, яке буде емулювати близько 80% клієнтів.	Випробувано в «реальному світі» з «реальними клієнтами» і зворотний зв'язок може охоплювати кожен елемент продукту.
Що підлягає виправленню	
Більшість відомих критичних проблем виправлені деякі функції можуть бути змінені або додані в результаті ранньої зворотного зв'язку.	Велика частина зібраного зворотного зв'язку розглядається та/або застосовується в майбутніх версіях продукту. Виконуються тільки важливі/критичні зміни.

Отримані результати	
Відмінне уявлення про те, як працює продукт і чи відповідає він критеріям дизайну (і чи «бета-готовий» він).	Уявлення про те, що клієнт думає про продукт, і про те, що він може відчувати, коли купує його.
Наступні кроки	
Проходження бета-тестування	Випуск продукту

Якість ПЗ – характеристика ПЗ, ступінь відповідності ПЗ вимогам. Частіше за все використовують визначення ISO 9001, згідно з яким якість – це «ступінь відповідності наявних характеристик вимогам».

Фактори якості – це нефункціональні вимоги до ПЗ, що відносяться до, наприклад, надійності та продуктивності програм.

Деякі з факторів якості:

Зрозумілість. Призначення ПЗ повинно бути зрозумілим з самої програми та документації.

Повнота. Всі необхідні частини програми повинні бути представлені та реалізовані.

Стислість. Відсутність надлишкової інформації та такої, що дублюється. Реалізація принципів DRY – Don't repeat yourself.

Можливість портування. Легкість в адаптації програми до інших умов: архітектури, платформи, операційної системи тощо.

Узгодженість. Вся документація та код повинні виконуватися за єдиними угодами, використовувати єдині формати та позначення.

Покриття тестуванням.

Зручність використання.

Надійність.

Безпечність.

Чек-лист – це документ, що описує, що має бути протестовано. При цьому чек-лист може бути абсолютно різного рівня деталізації. Наскільки детальним буде чек-лист залежить від вимог до звітності, рівня знання продукту співробітниками і складності продукту.

Чек-лист потрібен для:

– нагадування про необхідні тести;

– розподілу завдань за рівнем кваліфікації;

– збереження звітності та результатів тестування.

Чек-лист включає у себе:
– список перевірок (з необхідним ступенем деталізації);
– статус перевірок:
– збірка, на якій проводилося тестування;
– тестове оточення (якщо застосовується);
– тестувальник;
– результат перевірки.

Приклад складання чек-листу для проходження тестування наведено на рисунку 8.1.

Вид перевірки	**Апаратна частина/Програмна частина/Платформа на базі яких проходило тестування**	**Апаратна частина/Програмна частина/Платформа на базі яких проходило тестування**	**Апаратна частина/Програмна частина/Платформа на базі яких проходило тестування**
Тест №1	Результати тестування	Результати тестування	Результати тестування
Тест №2	Результати тестування	Результати тестування	Результати тестування
…			
Тест №*N*	Результати тестування	Результати тестування	Результати тестування

Рисунок 8.1 – Приклад складання чек-листу

Приклад чек-листу після проходження тестування наведено у Додатку Д.

8.3 Завдання на лабораторну роботу

8.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

8.3.2 Розробити чек-листи для тестування верстки та функціонування вебзастосунку.

8.3.3 Провести тестування верстки.

8.3.4 Проаналізувати реалізованість функціональних вимог до ПЗ.

8.3.5 Провести тестування функціональності вебзастосунку.

8.3.6 Провести тестування стабільності.

8.3.7 Оформити звіт з роботи.

8.3.8 Відповісти на контрольні питання.

8.4 Зміст звіту

8.4.1 Тема та мета роботи.

8.4.2 Тема, обрана для проєктування.

8.4.4 Опис тестів, застосованих для аналізу якості вебзастосунку, чек-листи тестування та результати тестування.

8.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

8.5 Контрольні запитання

8.5.1 Що таке баг?

8.5.1 Як визначити серйозність та пріоритет багу?

8.5.2 Як можна класифікувати види тестування?

8.5.3 Що таке альфа- та бета-тестування?

8.5.4 Що таке статичне та динамічне тестування?

8.5.5 Які розрізняють фактори якості ПЗ?

8.5.6 Як відбувається тестування навантажувальної здатності?

8.5.7 Яким чином може проводитися перевірка ергономічності?

8.5.8 Чим відрізняється тестування за принципом «білого ящика» від тестування «чорного ящика»?

8.5.9 Як складається та що обов’язково має включати до себе чек-лист?

ЛІТЕРАТУРА

1. Wiegers K. Software Requirements. 3rd edition [Text] / K. Wiegers, J. Beatty. – Redmond : Microsoft Press, 2013. – 672 p.

2. Wiegers K. More About Software Requirements: Thorny Issues and Practical Advice [Text] / K. Wiegers. – Redmond : Microsoft Press, 2005. – 224 p.

3. Starck E. Agile Project Management QuickStart Guide: A Simplified Beginners Guide To Agile Project Management [Text] / E. Starck. – Albany : ClydeBank Media LLC, 2017. – 166 p.

4. Rothman J. Create Your Successful Agile Project [Text] / J. Rothman. – Raleigh : The Pragmatic Programmers, 2017. – 225 p.

5. Anderson D. Essential Kanban Condensed [Text] / D. Anderson, A. Carmichael. – Seattle : Lean Kanban University Press, 2016. – 92 p.

6. Rumpe B. Modeling with UML: Language, Concepts, Methods [Text] / B. Rumpe. – New York : Springer International Publishing, 2016. – 228 p.

7. Bank C. Web UI Design Patterns 2014. A Deeper Look At The Hottest Websites and Web Apps Today [Text] / C. Bank, W. Zuberi. – Gdansk : UXPin, 2014. – 195 p.

8. Faranello S. Practical UX Design [Text] / S. Faranello. – Birmingham : Packt Publishing, 2016. – 232 p.

9. Gothelf J. Lean UX: Designing Great Products with Agile Teams [Text] / J. Gothelf, J. Seiden. – Sebastopol : O'Reilly Media, 2017. – 208 p.

10. Kunjumohamed S. pring MVC: Designing Real-World Web Applications [Text] / S. Kunjumohamed, H. Sattari, A. Bretet, G. Warin. – Birmingham : Packt Publishing, 2016. – 962 p.

11. Wilson K. Spring MVC: Designing Real-World Web Applications [Text] / K. Wilson. – British Columbia : Leanpub, 2015. – 251 p.

12. Stauffer M. Laravel: Up and Running: A Framework for Building Modern PHP Apps [Text] / M. Stauffer. – Sebastopol : O'Reilly Media, 2016. – 795 p.

13. Bašić M. A Collection of Laravel Tutorials [Text] / M. Bašić. – British Columbia : Leanpub, 2017. – 208 p.

14. Portwood C. Yii Project Blueprints [Text] / C. Portwood. – Birmingham : Packt Publishing, 2014. – 320 p.

15. Asadi A. WordPress For Beginners. 7th edition [Text] / A. Asadi. –Bournemouth : Imagine Publishing, 2016. – 196 p.

16. Sabin-Wilson L. WordPress All-in-One For Dummies. 2nd edition [Text] / L. Sabin-Wilson. – New York : John Wiley & Sons, Inc., 2013. – 840 p.

17. Burge S. Joomla! 3 Explained: Your Step-by-Step Guide. 2nd edition [Text] / S. Burge. – Boston : Addison-Wesley Professional, 2014. – 448 p.

18. Chumley C. Mastering Drupal 8 [Text] / C. Chumley, W. Hurley. – Birmingham : Packt Publishing, 2017. – 456 p.

19. Zandstra M. PHP Objects, Patterns, and Practice. 5th edition [Text] / M. Zandstra. – New York : Apress, 2016. – 583 p.

20. Agarwal B. B. Software Engineering And Testing: An Introduction [Text] / B. B. Agarwal, M. Gupta, S. P. Tayal. – Burlington : Jones & Bartlett Publishers, 2009. – 515 p.

21. Watkins J. Testing IT: An Off-the-Shelf Software Testing Process. 2nd edition [Text] / J. Watkins, S. Mills. – Cambridge : Cambridge University Press, 2010. – 354 p.

ДОДАТОК А
ПРИКЛАДИ ТЕМ

№	Тема	Обов’язково до реалізації
1.	Інтернет-магазин фізичних товарів	– реєстрація та авторизація користувачів; – оформлення та опрацювання (адміністратором) замовлення; – вибір та опрацювання (адміністратором) виду доставки замовлення; – групування товарів за категоріями; – оновлення та поповнення асортименту товарів (адміністративна панель сайту).
2.	Інтернет-магазин цифрових товарів (послуг)	– реєстрація та авторизація користувачів; – оформлення та опрацювання (адміністратором) замовлення; – збереження та опрацювання (адміністратором) архіву попередніх замовлень для можливості повторного замовлення; – групування товарів за категоріями; – оновлення та поповнення асортименту товарів (адміністративна панель сайту).
3.	Сервіс з продажу транспортних квитків	– реєстрація та авторизація користувачів; – оформлення та опрацювання (адміністратором) замовлення; – можливість покупки квитків з/до проміжних станцій/зупинок; – оновлення маршрутів та коригування кількості вільних місць на маршрутах (адміністративна панель сайту).

4.	Планер (щоденник-органайзер)	– внесення інформації про майбутні плани у календар та уточнення часу; – ведення розділу «Календар»; – окреме ведення розділу «Зустрічі» (час, місце, вид зустрічі, учасники, нотатки); – ведення розділу «Контакти»; – нагадування (push-повідомлення) про важливі події.
5.	Інформаційний портал, присвячений кіно	– реєстрація та авторизація користувачів; – онлайн-перегляд трейлерів фільмів або фільмів-цілком; – виставлення оцінок та коментування улюблених кінострічок; – опрацювання (адміністратором) коментарів; – групування кінострічок за жанрами; – рекомендація нових кінострічок на основі переглянутих; – оновлення матеріалів порталу (адміністративна панель сайту).
6.	Фітнес-трекер	– внесення інформації про фізичну активність; – розроблення програм тренувань; – надання матеріалів про тренування (статті в Інтернеті, відео-уроки, тощо); – окреме ведення розділу «Щоденник харчування» та надання матеріалів про правильне харчування (статті в Інтернеті, відео-уроки, тощо); – нагадування (push-повідомлення) про внесення нових даних.

7.	Сервіс для прослуховування музики	– реєстрація та авторизація користувачів; – онлайн-прослуховування музики; – виставлення оцінок улюблених музикантів/колективів або окремих альбомів; – групування аудіозаписів за категоріями (за виконавцями, альбомами, жанрами); – рекомендація нових аудіозаписів на основі оцінених; – оновлення матеріалів сайту (адміністративна панель сайту).
8.	Система колективного блогу	– реєстрація та авторизація користувачів; – перегляд та ознайомлення із публікаціями користувачів; – групування публікацій за ключовими словами (тегами) та/або за категоріями; – виставлення оцінок та коментування публікацій; – можливість створення голосувань (в форматі окремого виду публікацій або як частини публікацій); – опрацювання (адміністратором) коментарів; – рекомендація та виведення публікацій за рейтингом найбільш популярних; – адміністративна панель сайту.
9.	Інтернет-аукціон	– реєстрація та авторизація користувачів (користувач може бути або покупцем, або продавцем); – класичний аукціон (збільшення ставки від мінімальної ціни) з заданим кроком та таймером, для визна-

		чення переможця; – можливість миттєвої покупки за встановленою продавцем ціною; – групування товарів за категоріями; – моніторинг ходу торгів та вдалих угод (адміністративна панель сайту).
10.	Соціальний інтернет-сервіс для фотографів	– реєстрація та авторизація користувачів; – публікація власних фотоматеріалів; – захист авторських прав (захист від копіювання фотоматеріалів) користувачів; – перегляд та ознайомлення із роботами фотографів; – можливість створення власних колекцій з фотоматеріалів інших користувачів; – виставлення оцінок та коментування публікацій; – опрацювання (адміністратором) коментарів; – рекомендація та виведення фотографій за рейтингом найбільш популярних; – адміністративна панель сайту.
11.	Файлообмінник	– реєстрація та авторизація користувачів; – розміщення та редагування прав доступу до власних файлів на сайті; – можливість перегляду та ознайомлення із файлами з відкритим доступом; – групування файлів за типом (аудіозаписи, зображення, кіно, книги, тощо)

		– коментування файлами з відкритим доступом; – можливість відправлення миттєвого посилання на файл з режимом доступу: «за посиланням»; – адміністративна панель сайту.
12.	Сервіс бронювання готельних номерів/хостелів/житла	– реєстрація та авторизація користувачів; – оформлення та опрацювання (адміністратором) заявок на бронювання; – сортування пропозицій за найближчим часом до бажаного часу заселення та/або виселення; – підбір пропозицій на основі додаткових критеріїв (статус готелю, включений сніданок, наявність кухні, тощо) – виставлення оцінок та коментування; – оновлення пропозицій для бронювання (адміністративна панель сайту).

ДОДАТОК Б
ПРИКЛАД КАРТИ ІСТОРІЙ

На рисунку Б.1 наведено приклад представлення карти історій для проєкту «Інтернет-магазин».

Історія	Роль	Активність	Мета	Критерій прийнятності	Коментарі
Сайт					
Покупка товару	Покупець (авторизований користувач)	Пошук товару	Виведення списку товарів		
			Реалізація сортування товарів		Сортування за: назвою, ціною, популярністю
		Додавання товару до кошика	Реалізація «кошику покупця»		
		Вибір способу доставки	Користувач може обрати спосіб доставки товару	Кур'єрська доставка + 2 служби доставки	
		Оплата покупки	Можливість вибору оплати		Оплата накладеним платежем або на картку
Адміністративна панель керування сайтом					
Додавання нового товару	Адміністратор	Додавання товару до каталогу	Додавання опису товару		Заповнення обов'язкових полів (назва, ціна, тощо)
			Використання ключових слів (тегів)		

Рисунок Б.1 – Представлення карти історій для проєкту «Інтернет-магазин».

ДОДАТОК В
ПРИКЛАД ОПИСУ ПОТОКУ ПОДІЙ

У якості прикладу наведено потоки події прецеденту «Замовлення книги у бібліотеці».

На початку у вигляді таблиці представимо етапи основного потоку, альтернативних потоків та потоків помилок. Приклад наведено у таблиці В.1.

Таблиця В.1 – Етапи основного потоку, альтернативних та помилкових потоків

Основний потік	Альтернативні потоки	Потоки помилок
1. Бібліотекар просить надати абонемент.	А.1 У читача немає читацького квитка.	Е.1 Читач не пам’ятає, що вже отримував абонемент.
2. Читач надає абонемент.	А.2 У читача є заборгованості.	
3. Бібліотекар перевіряє дані за абонементом.	А.3 Читач не може надати повну інформацію про книгу.	
4. Бібліотекар запитує інформацію про книгу.	А.4 Не вдалось знайти книгу.	
5. Читач надає інформацію.	А.5 Немає вільного примірника книги.	
6. Бібліотекар знаходить книгу у каталозі.	А.6 Читач відмовляється надавати особисту інформацію.	
7. Бібліотекар видає екзем лір книги читачеві.	А.7 Читач вже попередньо отримував абонемент.	
8. Бібліотекар робить запис в реєстрі.		
9. Бібліотекар вносить зміни до каталогу.		

Далі наведено детальний опис етапів.

Етапи основного потоку подій.

1. Бібліотекар просить надати абонемент (читацький квиток) читача.

2. Читач надає абонемент.

А.1 У читача немає читацького квитка.

3. Бібліотекар перевіряє дані за абонементом.

А.2 У читача є заборгованості.

4. Бібліотекар запитує інформацію про книгу (назва та прізвище автора) у читача.

5. Читач надає інформацію про книгу для пошуку.

А.3 Читач не може надати повну інформацію про книгу.

6. Бібліотекар знаходить книгу у каталозі.

А.4 Не вдалось знайти книгу.

А.5 Немає вільного примірника книги.

7. Бібліотекар видає екземпляр книги читачеві.

8. Бібліотекар робить запис в реєстрі про: дату видачі, номер абонементу, очікувану дату повернення.

9. Бібліотекар вносить зміни до каталогу про наявність вільних екземплярів.

10. Прецедент завершено.

Етапи альтернативних потоків.

А.1 У читача немає читацького квитка.

1. Бібліотекар пропонує завести абонемент у бібліотеці

2. Читач погоджується та надає особисту інформацію.

А.6 Читач відмовляється надавати особисту інформацію.

3. Бібліотекар перевіряє чи немає читача із аналогічними даними.

А.7 Читач вже попередньо отримував абонемент.

4. Бібліотекар виписує абонемент.

5. Бібліотекар вносить дані нового читача до реєстру читачів бібліотеки.

6. Потік повертається до кроку 4 основного потоку.

А.2 У читача є заборгованості.

1. Бібліотекар відмовляє читачу в обслуговуванні.

2. Бібліотекар нагадує читачу про заборгованість.

3. Потік повертається до кроку 10 основного потоку.

А.3 Читач не може надати повну інформацію про книгу.

1. Бібліотекар запитує часткову інформацію про книгу

2. Бібліотекар виконує пошук за отриманою частковою інформацією про книгу.

3. Потік повертається до кроку 10 основного потоку.

А.4 Не вдалося знайти книгу.

1. Бібліотекар повідомляє про відсутність книги.

2. Бібліотекар пропонує здійснити пошук за частковою інформацією про книгу.

3. Потік переходить до альтернативного потоку А.3.

А.5 Немає вільного примірника книги.

1. Бібліотекар повідомляє, що вільного примірника немає

2. Бібліотекар перевіряє реєстр та каталог.

3. Бібліотекар повідомляє коли у найближчий час звільниться один із примірників.

4. Потік повертається до кроку 10 основного потоку.

А.6 Читач відмовляється надавати особисту інформацію.

1. Бібліотекар відмовляє в обслуговуванні читачу.

2. Потік повертається до кроку 10 основного потоку.

А.7 Читач вже попередньо отримував абонемент

1. Бібліотекар повідомляє, що такий читач вже існує.

Е.1 Читач не пам'ятає, що вже отримував абонемент.

Етапи потоків помилок.

Е.1 Читач не пам'ятає, що вже отримував абонемент.

1. Бібліотекар пропонує читачу перевірити абонемент вдома.

2. Потік повертається до кроку 10 основного потоку.

ДОДАТОК Д
ПРИКЛАД СКЛАДАННЯ ЧЕК-ЛИСТУ

Перевірка	**Windows**			**Android** *(Google Chrome 76.0.3809.111)*	**iOS** *(Safari 7.0)*
	Google Chrome (76.0.3809.100)	*Mozilla Firefox (68.0.2)*	*Microsoft Edge (44.17763.1.0)*		
Авторизація (користувач)	пройдено	пройдено	пройдено	пройдено	пройдено
Авторизація (адміністратор)	пройдено	пройдено	пройдено	пройдено	не пройдено (#bug42)
Зміна мови	пройдено	пройдено	пройдено	не пройдено (#bug124)	пройдено
Виведення «Домашньої сторінки»	пройдено	пройдено	не пройдено (#bug58)	пройдено	пройдено

Рисунок Д.1 – Приклад чек-листу