МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Розробка програмного забезпечення»

Група: 4РП-05

Дипломний проект

здобувача освіти денної форми навчання
РП.05.17.000.ДП

Пекельника
Данила
Олександровича

м. Одеса
2022 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОДЕСЬКОГО НАЦІОНАЛЬНОГО ТЕХНОЛОГІЧНОГО УНІВЕРСИТЕТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**
Освітня програма: «**Розробка програмного забезпечення»**
Група: **4РП-05**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) на тему:

Розробка методології впровадження тестування серверного коду за рахунок використання коштів Swagger і XUnit

Проектний матеріал складається з пояснювальної записки на 46 сторінках та графічного (презентаційного) матеріалу на 10 аркушах (слайдах).

Дипломник _____ (Пекельник Д.О.)
Керівник _____ (Медведєв А.О.)

Консультанти:
з економічної частини _____ (Копайгородська Т.Г.)
з охорони праці_____ (Чорновол Н.І.)
з дотримання вимог ЄСКД _____ (Петрашова В.І.)
старший консультант _____ (Скорнякова О.В.)

До захисту допущений
Голова циклової комісії _____ (Скорнякова О.В.)
Завідувач відділення _____ (Суліма Ю.Ю.)

Захист «____»_____ 2022 р. Протокол ДКК №_____
Оцінка ДКК _____

Секретар ДКК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНАХТ»

Відділення **комп'ютерних систем** Комісія **КТ та ПІ**

Спеціальність **121 «Інженерія програмного забезпечення»**

Освітня програма **«Розробка програмного забезпечення»**

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР_____

_____

"_____"_____2022 р.

З А В Д А Н Н Я

на дипломний проект (роботу)

Здобувачеві (здобувачці) освіти Пекельник Данило Олександрович

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка методології впровадження тестування серверного коду за рахунок використання коштів Swagger і XUnit

затверджена наказом по коледжу від "30" грудня 2021 р. № 306-А2-ОД

2. Термін здачі закінченого проекту (роботи) _____

3. Вихідні данні до проекту (роботи) Microsoft Visual Studio, .Net, C#, MS SQL Server, JavaScript LINQ, API, HTTP, Swagger, XUnit, Microsoft Edge, Postman, MySQL, СУБД

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

1. Принципи тестування програмного продукту, побудованого з використанням Docker – контейнера. 2. Економічний розрахунок. 3. Охорона праці.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Презентація (10 слайдів)

6. Консультанти по проекту (роботі), із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний	Медведєв А.О.		
Економічний	Копайгородська Т.Г.		
Охорона праці	Чорновол Н.І.		
Нормоконтроль	Петрашова В.І.		
Старший консультант	Скорнякова О.В.		

7. Дата видачі завдання _____

Керівник _____

(підпис)

Завдання прийняв до виконання _____

(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	Розділ 1. Принципи тестування програмного продукту		
	побудованого з використанням Docker-контейнера		
2	Розділ 2. Економічний розрахунок		
3	Розділ 3. Охорона праці		
4	Розробка презентації до дипломної роботи		
5	Чистове оформлення пояснювальної записки		
6	Підготовка доповіді до захисту		
7	Отримання рецензії, відповіді на зауваження		
	Рецензента		
8	Захист роботи		

Дипломник _____

(підпис)

Керівник _____

(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП 05.17.000 ДП ПЗ	*Дипломний проект*		
А4			*РП 05.17.000 ДП ПЗ*	*Пояснювальна*	*1*	
				записка		
				Презентація		

Ізм.	Арк.	№ докум.	Підпис	Дата	*РП 05.17.000*		
Розробив		Пекельник Д.О			Розробка методології впровадження тестування серверного коду за рахунок використання коштів Swagger і XUnit	Літ.	Аркуш 3 / Аркушів
Перевірив		Медведєв А.О.				Н Д П	
Н. Контр.		Петрашова В.І				ВСП «ОТФК ОНТУ» гр.РП-05	
Затверд.		Скорнякова О.В.					

ЗМІСТ

ВСТУП

Розробка програмного забезпечення швидко змінюється і прогресує, те саме відбувається із забезпеченням якості. Тестувальники ПЗ постійно стикаються з новими проблемами, для вирішення яких потрібне глибоке розуміння новітніх методів QA та процедур тестування. Щоб забезпечити високу якість продукту, QA-інженери повинні мати необхідні знання та досвід.

Тестування - один із найважливіших елементів циклу розробки. Воно буде найефективнішим, якщо виконується від початку процесу. Завдяки ранньому та безперервному тестуванню можна заощадити час та гроші, які пізніше підуть на виправлення помилок. Найкращі сучасні практики QA сприяють оптимізації та покращенню процесів тестування, підвищують якість та продуктивність. Проте тестувальники все-таки можуть зіткнутися з деякими ключовими проблемами та слабкими місцями тестування QA.

Хороша agile-команда розробників буде чуйно реагувати на зміни вимог та використовувати ті методи та процеси, які дозволять вносити виправлення без шкоди для якості проекту. QA-інженери також повинні вміти адаптуватися та ретельно вести звіти про тестування. Ці записи допоможуть іншим членам команди знайти оптимальний спосіб внесення необхідних змін.

Оскільки сценарії тестування повинні відповідати кінцевим цілям, важливо переконатися, що вони чітко сформульовані наперед. Якщо команда тестування не має чіткого уявлення про кінцевий результат, то може вийти зовсім

Щоб уникнути цієї проблеми, зацікавлені сторони мають чітко визначити рамки проекту, свої вимоги та очікування до початку роботи. Також необхідно підтримувати постійний зв'язок між командою проекту та зацікавленими сторонами. Щоб отримати високоякісний продукт, який буде працювати, як задумано, всі неясності та суперечності в очікуваннях та вимогах повинні бути усунені якомога раніше. Критерії приймання,

сформульовані замовником, та використання концепції 3 Amigos сприяють успіху проекту.

МЕТА ДИПЛОМНОГО ПРОЕКТУ

Мета проекту - вивчити процес автоматизаційного тестування запитів та скласти процес тестування АПІ за рахунок коштів Swagger, а також реалізувати покриття тестами проекту реалізованого з використанням мікросервісної архітектури, для досягнення цієї мети необхідно:

1. Реалізувати методи та запити для настроювання середовища Swagger.
2. Грамотно налаштувати середовище тестування та бібліотеки тестування та розгорнути їх у проекті
3. До зазначених бібліотек тестування підключити шар покриття, який детально відображатиме інформацію про код протестованого
4. Протестувати запити на АПІ за допомогою Swagger
5. Скласти Docker - файли для запуску процесу побудови контейнерів.

1 ПРИНЦИПИ ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ, ПОБУДОВАНОГО З ВИКОРИСТАННЯМ DOCKER – КОНТЕЙНЕРА

1.1Розбір класу посилань при побудові тестів. Концепція XUnit

Один із зручних підходів передбачає використання імітованих (або пробних) об'єктів, які емулюють функціональність реальних об'єктів із проекту, але специфічним та керованим чином. Імітовані об'єкти дозволяють звузити фокус тестів, перевіряючи лише ту функціональність, яка цікавить [1].

Поширена практика поміщати всі тестові проекти окремі папки. Це стосується і структури папок на диску, і папок у проекті. Практика поширена завдяки зручності. Також ім'я проекту і простір імен повністю повторюють тестований модуль з додаванням в кінці слова Tests (зазвичай через точку). Для простоти пошуку тестів всі тести, які стосуються деякого класу, поміщаються у його клас із тестами. Тобто виходить пара з оригінального класу та класу з тестами. Поділ між юніт, інтеграційними, навантажувальними відбувається або за категоріями, або за логікою основної системи. Це означає, що якщо інтеграційні тести не можна віднести до одного класу (зазвичай, це так), то вони виносяться в окрему збірку, присвячену функціону, що тестується. Або, наприклад, тести навантаження можуть тестувати роботу деякого методу одного класу. І тут тест розміщується у парному класі [2].

Структура самих випробувань будь-якого типу відповідає стилю AAA. Систематизуємо вищесказане:

Таблиця 1.1 - Структура проектування тестів.

Розташування проекту	в папке Tests
Ім'я тестового проекту	[ProjectName].Tests
Простір імен	[Namespace].Tests
Ім'я класу із тестами	[Class]Tests
Пара 1-1	Виходячи з тестованого класу та класу з тестами
Найменування юніт-тестів	BDD (мануал)
Стиль тестів	AAA (Arrange-Act-Assert)

1.2 Проектування тестів у проекті з використанням XUnit

Для початку, необхідно розуміння архітектури типу моделі-перегляду або моделі-уявлення-презентатора без використання повномасштабної структури. Архітектура написання тестування серверного коду, що реалізується виходячи з вимог бізнес-архітектури, передбачає, що юніт-тестування ui-компонентів складно реалізується при неправильному посиланні між об'єктами проекту. Є способи обійти це, але маршрут, що задається архітектурою бізнес-моделі, об'єднує складні компоненти окремих моделей запиту, тому в даному завдання використання обходу архітектури не є доцільним. Зазвичай це буде дуже складно підтримувати тести, більше додаткових дій щодо моделі обслуговування, з якими можна не взаємодіяти.

Для початку побудови архітектури тестування необхідно виконати такі кроки:

1. Грамотне розуміння вибірки функціональності, яку необхідно протестувати, у класі "контролер" або "презентатор".

2. Перевірка логіки потрібних класів.

3. Оновлення існуючого класу за рахунок приховування класу usercontrol (подання) за інтерфейсом і змусити диспетчер або ведучий поговорити з поданням через інтерфейс.

4. Обґрунтування уявлень у своїх тестах.

Рисунок 1.1 – Структура системи тестування.

Як правило, юніт тестування застосовується до відносно невеликих програмних елементів, наприклад, до підпрограм або методів, що асоціюються з об'єктами. За такого підходу випробувач аналізує програмний код й у отримання тестових даних використовує знання структуру компонента. Наприклад, з аналізу коду можна визначити, скільки контрольних тестів потрібно виконати для того, щоб у процесі тестування всі оператори виконалися принаймні один раз.

Недоліки тестування "білої скриньки":

1. Кількість незалежних маршрутів може бути дуже великою.

2. Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

3. У програмі може бути пропущені деякі маршрути.

4. Не можна знайти помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип "білої скриньки" дозволяє врахувати особливості програмних помилок:

1. Кількість помилок мінімальна в "центрі" та максимально на "периферії" програми.

2. Попередні припущення щодо ймовірності потоку управління або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабко.

3. При записі алгоритму ПЗ як тексту мовою програмування можливе внесення типових помилок трансляції (синтаксичних і семантичних).

4. Деякі результати у програмі залежить не від вихідних даних, як від внутрішніх станів програми.

Кожна з цих причин є аргументом щодо тестування за принципом "білого ящика". Тести "чорної скриньки" не зможуть реагувати на помилки таких типів.

1.3 Застосування функціонального тестування

Функціональне тестування - один з основних видів незалежного тестування програмного забезпечення, спрямований на перевірку реалізованих функціональних вимог [3]. Інакше кажучи, фахівці за функціональним тестуванням визначають, чи вирішує розроблене ПЗ завдання, заради яких воно було створено, чи задовольняє воно потреби замовника/користувача. Серед основних вимог до ПЗ зазвичай виділяють функціональну придатність, точність, здатність до взаємодії, відповідність стандартам та правилам, захищеність.

Величезне значення має регресійне функціональне тестування, яке проводиться з метою перевірити, чи не впливають нові функції, поліпшення і виправлені дефекти на існуючу функціональність продукту.

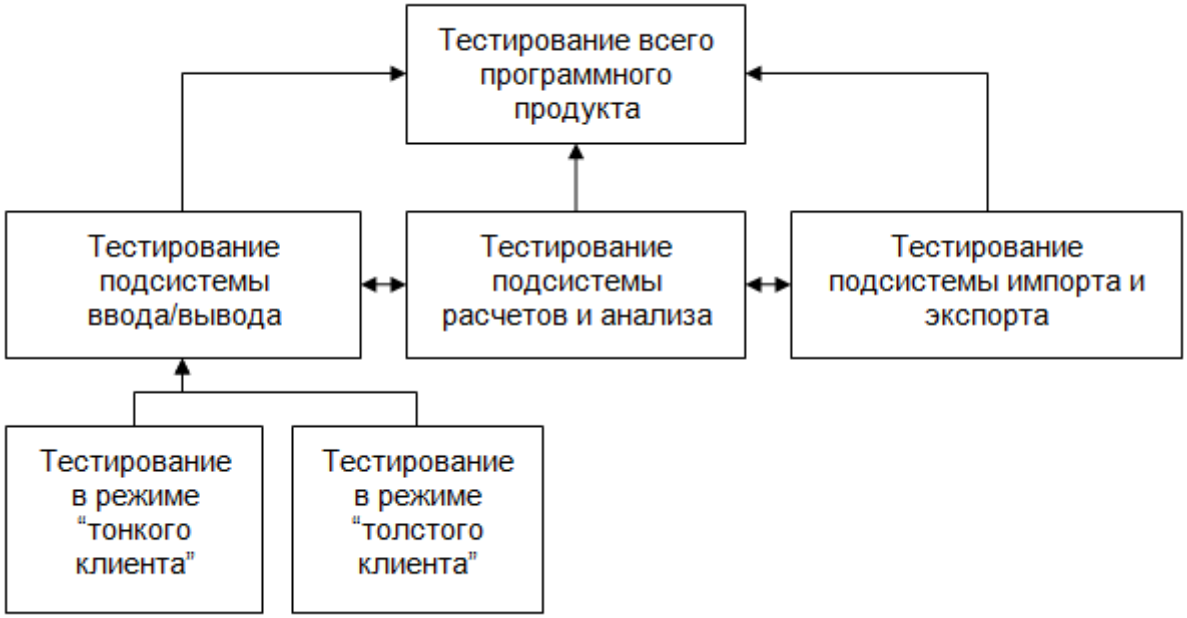

Рисунок 1.2 Настройки тестування продукту на проектах.

Як правило, функціональне тестування програмного забезпечення поділяється на:

1. компонентне
2. інтеграційне
3. системне
4. приймальне
5. регресійне
6. автоматизовано регресійне
7. Дослідницьке тестування
8. функціональне тестування сумісності

Завдяки такому детальному поділу розробники можуть перевірити функціональність як окремих компонентів, і всієї системи. Перевірки можуть проводитися вручну, а також за допомогою спеціалізованого ПЗ для тестування, яке виконує тестові сценарії через зручний програміст інтерфейс.

1.4 Розгортання unit – тестів у докер контейнері

Для початку, розгортання тестів необхідно описати модуль, який розпочне етап побудови.

```
# syntax=docker/dockerfile:1

FROM openjdk:16-alpine3.13 as base

WORKDIR /app

COPY .mvn/ .mvn
COPY mvnw pom.xml ./
RUN ./mvnw dependency:go-offline
COPY src ./src

FROM base as test
CMD ["./mvnw", "test"]

FROM base as development
CMD ["./mvnw", "dockerfile:run", "-Dspring-boot.run.profiles=mysql",
"-Dspring-boot.run.jvmArguments='-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:8000'"]

FROM base as build
RUN ./mvnw package

FROM openjdk:11-jre-slim as production
EXPOSE 8080

COPY --from=build /app/target/spring-petclinic-*.jar /spring-petclinic.jar

CMD ["CSharp", "-Djava.security.egd=file:/dev/./urandom", "-net", "/angular-composite.project.net"]

```

Рисунок 1.3 – Опис команда для запуску тестів у контейнері

Таблиця 1.2: Опис структури файлу запиту на контейнер

Назва запиту до контейнера	**Дія**
FROM openjdk:16-alpine3.13 as base	Вибірка проекту на контейнері
WORKDIR /app	Директива папки
COPY .mvn/ .mvn COPY mvnw pom.xml ./ RUN ./mvnw dependency:go-offline COPY src ./src	Вибірка з структури хабу для запиту на робочі групи.
FROM base as test CMD ["./mvnw", "test"]	Вибірка папки, яка включає тести.
FROM base as development CMD ["./mvnw", "dockerfile:run", "-Dspring-boot.run.profiles=mysql", "-Dspring-boot.run.jvmArguments='-agentlib:jdwp= transport=dt_socket, server=y,suspend=n,address=*:8000'"]	Підключення до бази даних вказаним шляхом, для доступу до даних.
FROM base as build RUN ./mvnw package	Запуск процесу побудови проекту.
FROM openjdk:11-jre-slim as production EXPOSE 8080	Відкриття порту 8080 на контейнер, у цих типах контейнерів він використовується за замовчуванням.
COPY --from=build /app/target/spring-petclinic-*.jar /spring-petclinic.jar	Підвантаження результату в бібліотеку.

CMD ["CSharp", "-Djava.security.egd=file:/dev/./urandom", "-net", "/angular-composite.project.net"]	Деплоймент даних.

Результат запиту виконання тесту, через дані контейнера:

```
$ docker run -it --rm --name .net-builder data-container
[INFO] Scanning for projects...
[INFO]
[INFO] ------------< angular-composite.project.net >------------
[INFO] Building petclinic 2.4.2
...

[INFO] Results:
[INFO]
[WARNING] Tests run: 40, Failures: 0, Errors: 0, Skipped: 1
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:22 min
```

Рисунок 1.4 – Результат запиту на тести.

1.5 Тестування та рефакторинг

До запуску програми в виробництво, коли воно стане доступним користувачам, важливо переконатися, що дане додаток функціонує, як і повинно, що в ньому немає помилок. Для перевірки додатку ми можемо використовувати різні схеми і механізми тестування. Одним з таких механізмів є юніт-тести.

Юніт-тести дозволяють швидко і автоматично протестувати окремі компоненти програми незалежно від іншої його частини. Не завжди юніт-тести можуть покрити весь код програми, але тим не менше вони дозволяють істотно зменшити кількість помилок вже на етапі розробки.

Немає необхідності тестувати код використовуваного фреймворка або використовуваних залежностей. Тестувати треба тільки той код, який був написаний.

Треба відзначити, що в цілому концепція юніт-тестів не є непорушною вимогою до веб-розробці, та й взагалі до розробки. Є думка, що юніт-тести обов'язково повинні покривати весь код проекту, хтось вважає, що юніт-тести можна використовувати переважно для особливо складних моментів у коді програми, якоїсь складної логіки. Деякі не використовують юніт-тести.

Але тим не менше юніт-тести несуть потенційні переваги при розробці, до яких слід віднести не тільки власне перевірку результату і тестування коду, але і інші, як наприклад, написання слабосвязанних компонентів відповідно до принципів SOLID. Адже щоб тестувати компоненти програми незалежно один від одного, нам треба, щоб вони були слабо зв'язаної. А подібне побудова додатки в подальшому може позитивно позначитися на його подальшій модифікації і підтримки.

Більшість юніт-тестів так чи інакше мають ряд таких ознак:

Тестування невеликих ділянок коду ("юнітів")

Для створення юніт-тестів вибираються невеликі ділянки коду, які треба протестувати. Тестований ділянку, як правило, повинен бути менше класу. У більшості випадків тестується окремий метод класу або навіть частина функціоналу методу. Упор на невеликі ділянки дозволяє досить швидко писати прості тести.

Одного разу написаний код нерідко читають багато разів, тому важливо писати зрозумілий код. Особливо це важливо в юніт-тестах, де в разі невдачі при тестуванні розробник повинен швидко прочитати вихідний код і зрозуміти в чому проблема і як її виправити. А використання невеликих ділянок коду значно спрощує подібну роботу.

Тестування в ізоляції від решти коду

При тестуванні важливо ізолювати тестований код від решти програми, з якою він взаємодіє, щоб потім чітко визначити можливість помилок саме в цьому ізольованому коді. Що спрощує і підвищує контроль над окремими компонентами програми.

Автоматизація тестів

Створення юніт-тестів для невеликих ділянок коду веде до того, що кількість цих юніт-тестів стає дуже велике. Якщо процес отримання результатів і проведення тестів не автоматизовано, то це може привести до непродуктивної витраті робочого часу і зниження продуктивності.

Тому важливо, щоб результати юніт-тестів представляли собою просте рішення, яке означає, пройдений тест чи ні. Для автоматизації процесу розробники зазвичай звертаються до фреймворками юніт-тестування.

Тестування тільки загальнодоступних кінцевих точок

Навіть невеликі зміни в класі можуть привести до невдачі багатьох юніт-тестів, оскільки реалізація використовуваного класу змінилася. Тому при написанні юніт-тестів обмежуються тільки загальнодоступними кінцевими точками, що дозволяє ізолювати юніт-тести від багатьох деталей внутрішньої реалізації компонента. В результаті зменшується ймовірність, що зміни в класах можуть привести до провалу юніт-тестів.

Фреймворки тестування

Для написання юніт-тестів можна створювати весь необхідний функціонал, використовувати якісь свої способи тестування, однак, як правило, для цього застосовуються спеціальні фреймворки. Деякі з них:

xUnit.net: фреймворк тестування для платформи .NET. Найбільш популярний фреймворк для роботи саме з .NET Core і ASP.NET Core

MS Test: фреймворк юніт-тестування від компанії Microsoft, який за замовчуванням включений в Visual Studio і який також можна використовувати з .NET Core

NUnit: портований фреймворк з JUnit для платформи .NET

Дані фреймворки надають нескладний API, який дозволяє швидко написати і автоматично перевірити тести.Разработка через тестирование **(Test-Driven-Development)**

Окремо варто сказати про концепцію TDD або розробка через тестування. TDD представляє процес застосування юніт-тестів, при якому

спочатку пишуться тести, а потім вже програмний код, достатній для виконання цих тестів.

Використання TDD дозволяє знизити кількість потенційних багів в додатку. Створюючи тести перед написанням коду, ми тим самим описуємо спосіб поведінки майбутніх компонентів, не зв'язуючи себе при цьому з конкретною реалізацією цих тестованих компонентів (тим більше що реалізація на момент створення тесту ще не існує). Таким чином, тести допомагають оформити і описати API майбутніх компонентів [9].

1.6 Огляд Swagger

Swagger – це фреймворк специфікації RESTful API. Його принадність полягає в тому, що він дає можливість не тільки інтерактивно переглядати специфікацію, але й надсилати запити - так званий Swagger UI, ось так це виглядає:

Swagger має ряд наступних переваг:

Підтримка специфікації API для генерації коду, згенерованого клієнтського та серверного скелетного коду, може прискорити розробку та прискорити тестування

- Документи можуть бути використані для внутрішнього огляду проекту API
- Зручно для тестерів, щоб зрозуміти API
- Підтримка API для автоматичної генерації синхронних онлайн-документів
- Документи можуть бути опубліковані як частина документації клієнта

Одним словом, він простий у використанні та дуже високий. Нижче коротко опишемо, як швидко створити інструмент документації на основі Node та Swagger UI API локально.

1.7 Документація до тестування проекту

1.7.1 Створення проекту та впровадження тестування

Для реалізації запуску середовища Swagger необхідно зробити такі дії:

Задати залежність від проекту і налаштувати їх для запуску тестованого середовища:

```
48          app.UseEndpoints(endpoints =>
49          {
50              endpoints.MapHealthChecksUI();
51
52              endpoints.MapControllerRoute(
53                  name: "default",
54                  pattern: "{controller=Home}/{action=Index}/{id?}");
55          });
```

Рисунок 1.5 – Завдання ендпоінтів для створення роутингу.

```
0 references | 0 changes | 0 authors, 0 changes
public void ConfigureServices(IServiceCollection services)
{
    services.AddTransient<LoggingDelegatingHandler>();

    services.AddHttpClient<ICatalogService, CatalogService>(c =>
        c.BaseAddress = new Uri(Configuration["ApiSettings:CatalogUrl"]))
        .AddHttpMessageHandler<LoggingDelegatingHandler>()
        .AddPolicyHandler(GetRetryPolicy())
        .AddPolicyHandler(GetCircuitBreakerPolicy());

    services.AddHttpClient<IBasketService, BasketService>(c =>
        c.BaseAddress = new Uri(Configuration["ApiSettings:BasketUrl"]))
        .AddHttpMessageHandler<LoggingDelegatingHandler>()
        .AddPolicyHandler(GetRetryPolicy())
        .AddPolicyHandler(GetCircuitBreakerPolicy());

    services.AddHttpClient<IOrderService, OrderService>(c =>
        c.BaseAddress = new Uri(Configuration["ApiSettings:OrderingUrl"]))
        .AddHttpMessageHandler<LoggingDelegatingHandler>()
        .AddPolicyHandler(GetRetryPolicy())
        .AddPolicyHandler(GetCircuitBreakerPolicy());

    services.AddControllers();
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new OpenApiInfo { Title = "Shopping.Aggregator", Version = "v1" });
    });

    services.AddHealthChecks()
        .AddUrlGroup(new Uri($"{Configuration["ApiSettings:CatalogUrl"]}/swagger/index.html"), "Catalog.API", HealthStatus.Degraded)
        .AddUrlGroup(new Uri($"{Configuration["ApiSettings:BasketUrl"]}/swagger/index.html"), "Basket.API", HealthStatus.Degraded)
        .AddUrlGroup(new Uri($"{Configuration["ApiSettings:OrderingUrl"]}/swagger/index.html"), "Ordering.API", HealthStatus.Degraded);
}
```

Рисунок 1.6 – Приклад налаштування середовища для одного контейнера.

У цьому налаштуванні середовища йде встановлення залежностей інтерфейсів і самих сервісів для підтримки застосування залежностей контейнера.

1.7.2 Налаштування Maven для запуску проекту

Для початку необхідно додати до проекту Maven залежність Swagger'a. Оскільки ми будемо підключати Swagger до RESTEasy, то додамо відповідну залежність.

```
95   <dependency>
96       <groupId>io.swagger</groupId>
97       <artifactId>swagger-jaxrs</artifactId>
98       <version>1.5.6</version>
99   </dependency>
```

Рисунок 1.7 – Додавання залежності

На час написання роботи актуальною версією є 1.5.6.

Потрібно врахувати, що Swagger має legacy Maven репозиторій. У lagecy репозиторію groupId=getDataId().

Далі нам необхідно підключити до проекту необхідних слухачів (listeners), щоб Swagger міг автоматично визначати анотації та генерувати документацію.

Налаштування за допомогою нащадка класу getCatalogData().

Налаштування буде виглядати так:

```
public class SampleApplication : Application
{
  public Set<Class<?>> getClasses()
  {
        Set < Class <?>> resources = new HashSet();
        resources.add(io.swagger..ApiListingResource.class);
        resources.add(io.swagger..SwaggerSerializers.class);
        return resources;
  }
}
```

Рисунок 1.8 – Ініціалізація отримання середовища.

Для початку опишемо інструкції, які є обов'язковими для правильного документування та коректного відображення REST-сервісів на Swagger-UI.

@Api

Для того, щоб swagger визначив, що в класі знаходяться REST-сервіси, цей клас має бути позначений інструкцією @Api. У параметрах даної анотації можна вказати назву розділу, в якому будуть REST'и в UI, і вказати опис даного розділу.

Наприклад:

```
@Api(value = "Account", description = "APIs for working with users")
public class AccountRestService {...}
@ApiOperation
```

Анотацію @ApiOperation необхідно вказувати над методом сервісу REST. Також у її параметрах можна вказати опис сервісу.

```
@Path("login")
@POST
@ApiOperation(value = "Login")
public Response login() {...}
```

Для явної вказівки хідерів, які є обов'язковими для конкретного сервісу, можна використовувати інструкцію @ApiImplicitParam

```
@Path("logout")
@POST
@ApiOperation(value = "Logout")
@ApiImplicitParams({
    @ApiImplicitParam(name = "Authorization", paramType = "header",
     dataType = "string",required = true,
     defaultValue = "Token <paste_token_here>")
})
public Response logout() {...}
```

Для явної вказівки об'єкта відповіді можна використовувати інструкцію @ApiResponse. Це буде корисно, якщо як відповідь від сервера REST повертає об'єкт Responce.

```
@Path("login")
@POST
@ApiOperation(value = "Login")
@ApiResponses(value = {
  @ApiResponse(code = 200, message = "OK", response = Token.class)
})
public Response login() {...}
```

1.7.3 Запуск Swagger для тестування відпрацювання АПІ

Відпрацювання отримання середовища обробки АПІ:

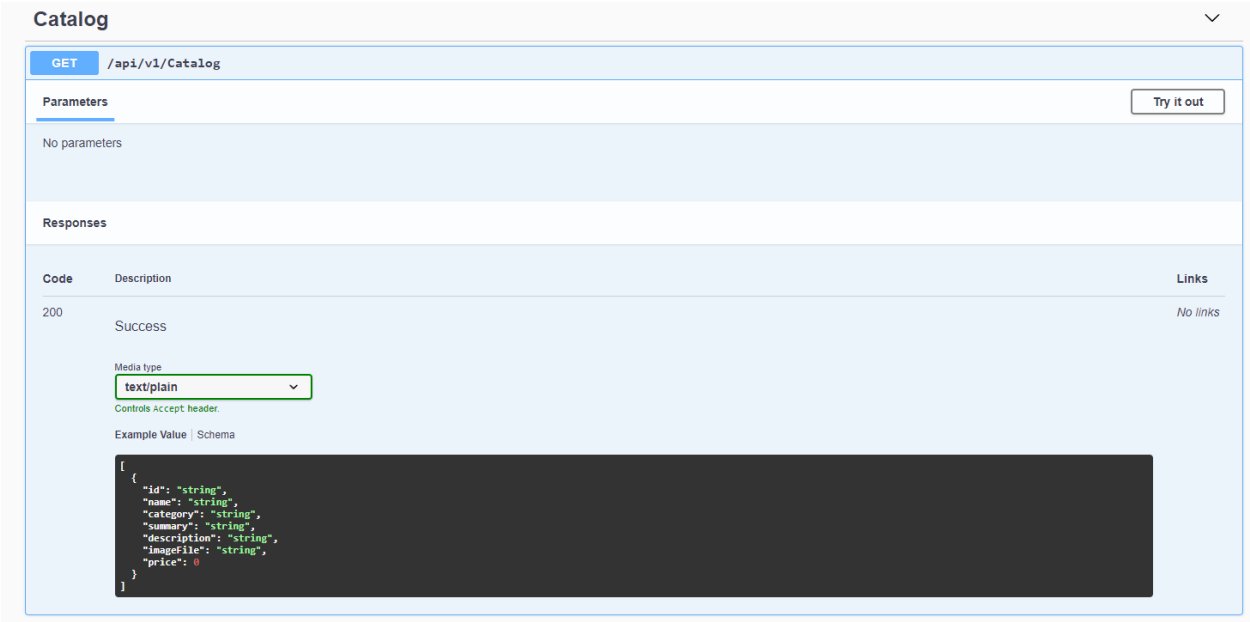

Рисунок 1.8 – Отримання каталогу

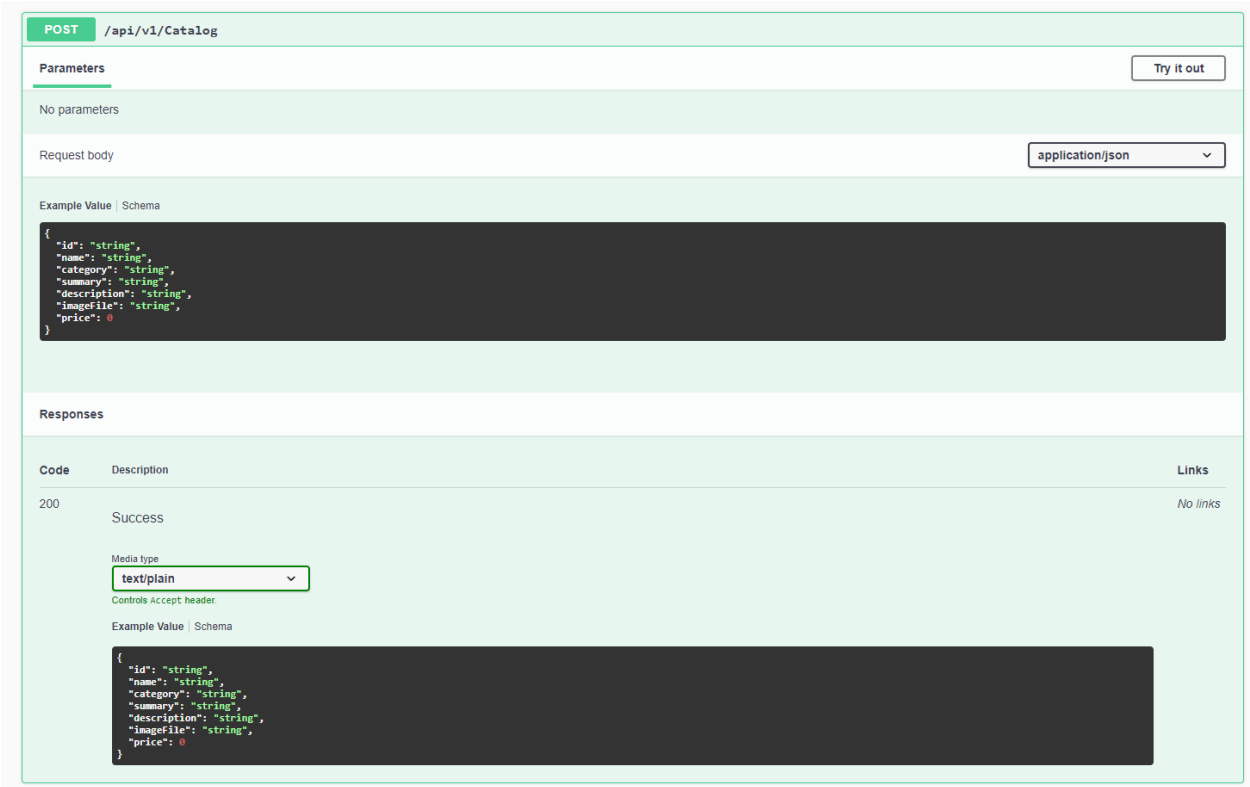

Рисунок 1.9 – Отработка POST-запроса

Для відображення результатів як приклад, що тестується, був обраний наступний шаблон обробки АПІ:

Рисунок 1.10 – Структура запиту, що надсилається

Щоб переконатися, що контейнер правильно відпрацьовує, подивимося на його побудову в консолі:

Рисунок 1.11 – Побудова контейнерів у консолі

1.7.4 Unit – тестування на основі XUnit

Для реалізації юніт тестування необхідно зробити таку схему:

1. Додати проект та зробити його залежним від тестованого проекту
2. Реалізувати конкретний клас тестування
3. Додати інструмент CodeCoverage для покриття коду

Додавання бібліотеки тестування до проекту:

```
<PropertyGroup>
  <TargetFramework>netcoreapp3.1</TargetFramework>
  <GenerateAssemblyConfigurationAttribute>false</GenerateAssemblyConfigurationAttribute>
  <GenerateAssemblyProductAttribute>false</GenerateAssemblyProductAttribute>
  <GenerateAssemblyTitleAttribute>false</GenerateAssemblyTitleAttribute>
  <GenerateAssemblyDescriptionAttribute>false</GenerateAssemblyDescriptionAttribute>
  <GenerateAssemblyCompanyAttribute>false</GenerateAssemblyCompanyAttribute>
  <GenerateAssemblyCopyrightAttribute>false</GenerateAssemblyCopyrightAttribute>
  <GenerateAssemblyFileVersionAttribute>false</GenerateAssemblyFileVersionAttribute>
  <GenerateAssemblyInformationalVersionAttribute>false</GenerateAssemblyInformationalVersionAttribute>
  <NeutralLanguage>en-US</NeutralLanguage>
  <ApplicationIcon />
  <OutputTypeEx>library</OutputTypeEx>
  <StartupObject />
  <IsTestProject>true</IsTestProject>
</PropertyGroup>

<PropertyGroup Condition="'$(Configuration)|$(Platform)'=='Debug|AnyCPU'">
  <NoWarn>1701;1702;1705;SA1652;SA1636;SA1124;SA1600;SA1602;SA1413</NoWarn>
  <TreatWarningsAsErrors>false</TreatWarningsAsErrors>
  <WarningsAsErrors />
</PropertyGroup>

<PropertyGroup Label="Configuration" Condition="'$(Configuration)|$(Platform)'=='Debug|AnyCPU'">
  <OutputType>Library</OutputType>
</PropertyGroup>

<ItemGroup>
  <ExcludeFromStyleCop Include="$(IntermediateOutputPath)\**\*.cs">
    <Visible>False</Visible>
  </ExcludeFromStyleCop>
</ItemGroup>
```

```
<ItemGroup>
  <Compile Remove="Coverage\**" />
  <EmbeddedResource Remove="Coverage\**" />
  <None Remove="Coverage\**" />
  <None Remove="Coverage.xml" />
</ItemGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.EntityFrameworkCore" Version="3.1.0" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.InMemory" Version="3.1.0" />
  <PackageReference Include="System.Runtime.Serialization.Primitives" Version="4.3.0" />
  <PackageReference Include="Moq" Version="4.13.1" />
  <PackageReference Include="xunit" Version="2.4.1" />
  <PackageReference Include="xunit.runner.visualstudio" Version="2.4.1">
    <PrivateAssets>all</PrivateAssets>
    <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
  </PackageReference>
  <PackageReference Include="OpenCover" Version="4.7.922" />
  <PackageReference Include="ReportGenerator" Version="4.3.6" />
</ItemGroup>

<ItemGroup>
  <ProjectReference Include="..\DataAccessCore\DataAccessCore.csproj" />
</ItemGroup>
```

Рисунок 1.12 – Структура файлу бібліотеки тестування проекту, що додається

Після, приклад реалізації тестованих модулів проекту:

```
private readonly GlobalRepository repo;

private readonly Mock<IUserStore<ApplicationUser>> mockStore;

private readonly Mock<IGlobalRepository> mockRepo;

private readonly Mock<ICopyService> mockICopyService;

private readonly CopyService mockCopyService;

private readonly Mock<DataContext> mockContext;

private readonly Mock<UserManager<ApplicationUser>> mockUserManager;

private readonly UserManager<ApplicationUser> userManager;

// Flag: Has Dispose already been called?
private bool disposed;

0 references | 0 changes | 0 authors, 0 changes
public GlobalRepositoryFixture()
{
    var mockOptions = new DbContextOptions<DataContext>();
    this.mockContext = new Mock<DataContext>(mockOptions);
    this.mockStore = new Mock<IUserStore<ApplicationUser>>();
    this.mockUserManager = new Mock<UserManager<ApplicationUser>>(this.mockStore.Object, null, null, null, null, null, null, null, null);
    this.userManager = this.mockUserManager.Object;
    this.mockRepo = new Mock<IGlobalRepository>();
    this.mockICopyService = new Mock<ICopyService>();
    this.repo = new GlobalRepository(this.mockContext.Object, this.userManager);
    this.mockCopyService = new CopyService(this.mockContext.Object, this.repo);
}
```

Рисунок 1.13 – Побудова шаблону реалізації бібліотеки тестування

Приклад написання юніт-тестів виглядає так:

```
#region RegisterUser test complete

[Fact]
0 references | 0 changes | 0 authors, 0 changes
public async Task RegisterUserTest_Success()
{
    string password = "1234qwerty";
    string email = "wwp@gmail.com";
    var dummyUser = new ApplicationUser()
    {
        UserName = email,
        Email = email,
        EmailConfirmed = false
    };
    IdentityResult iResult = new IdentityResult();

    this.mockUserManager.Setup(x => x.CreateAsync(dummyUser, password))
        .Returns(Task.FromResult(iResult)).Verifiable();

    var result = await this.repo.RegisterUser(password, email);
}
```

Рисунок 1.14 – Приклад тест на автентифікацію користувача

Видача тестових даних:

```
ProblemDetails ⌄ {
    type                string
                        nullable: true
    title               string
                        nullable: true
    status              integer($int32)
                        nullable: true
    detail              string
                        nullable: true
    instance            string
                        nullable: true
}
```

Рисунок 1.15 – Видача тестових даних

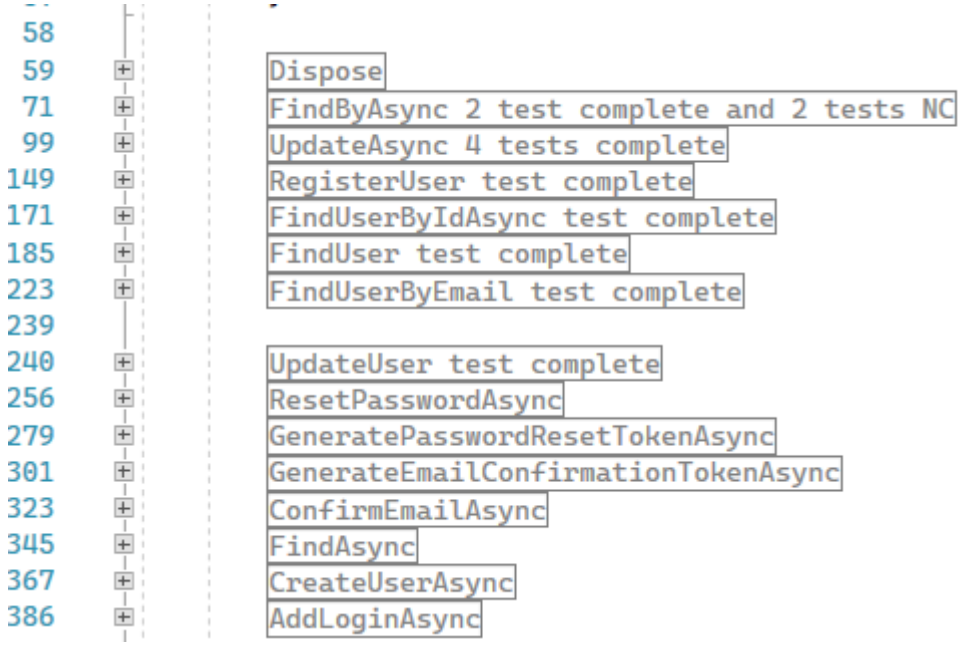

Рисунок 1.16 – Інші тести (приховані під регіоном для отримання зображення)

2 ЕКОНОМІЧНИЙ РОЗРАХУНОК

Метою даних розрахунків є обчислення вартості виконання науково-дослідної роботи «Розробка методології впровадження тестування серверного коду за рахунок використання коштів Swagger і XUnit». Основною метою є вивчення процесу автоматизаційного тестування запитів та складання процесу тестування АПІ за рахунок коштів Swagger, а також реалізація покриття тестами проекту реалізованого з використанням мікросервісної архітектури. Даний вид проекту відноситься до науково-дослідницької розробки. Оцінка якості розробленого проекту включає визначення трудомісткості і вартості його створення.

Розрахунок трудомісткості НДР здійснений в наступній послідовності:

1) Складений перелік всіх етапів і видів робіт, які необхідно виконати в ході даної НДР. Після узгодження з керівником проекту допущено виключення, доповнення, об'єднання окремих етапів і видів робіт;

2) По кожному виду робіт визначений кваліфікаційний рівень виконавців. Перелік етапів і робіт, що виконуються при проведенні НДР, приведений в таблиці 4.1.

Розподіл робіт по етапах і видах виконавців.

Таблиця 4.1.

Етап проведення НДР	**Вигляд робіт**	**Посада виконавця**
Розробка технічного завдання (ТЗ)	1.Складання і затвердження ТЗ для НДР по розробці «Розробка методології впровадження тестування серверного коду за рахунок використання коштів Swagger і XUnit».	Дипломник, керівник
Вибір напряму дослідження	1. Збір і вивчення науково-технічної літератури. 2. Формулювання можливих напрямів вирішення завдань, поставлених в	Дипломник керівник

	технічному завданні НДР і їх порівняльна оцінка. 3. Вибір напряму проведення досліджень 4. Розробка плану проведення досліджень для подальшої розробки.	
Теоретичні і експериментальні дослідження	1.Принципи тестування програмного продукту, побудованого з використанням doker-контейнера 2. Документація до тестування проекту	Дипломник керівник консультанти
Узагальнення і оцінка результатів досліджень	1. Узагальнення результатів 2. Оцінка повноти вирішення поставлених завдань. 3.Складання і оформлення звіту. Розгляд результатів проведеною НДР і прийняття результатів в цілому.	Дипломник керівник консультанти

Оцінка тривалості виконання робіт розраховується на основі вірогідних оцінок робіт, що задаються виконавцями.

Очікувана трудомісткість робіт.

Таблиця 4.2.

Вигляд роботи	**Очікуваний час виконання (дні)**
1. Складання і затвердження ТЗ для НДР «Розробка методології впровадження тестування серверного коду за рахунок використання коштів Swagger і XUnit»	1
2. Збір і вивчення науково – технічної літератури, технічної документації і інших матеріалів.	3
3. Формулювання можливих напрямів вирішення завдань, поставлених в технічному завданні НДР і їх порівняльна оцінка.	4
4. Розробка плану проведення досліджень для подальшої розробки.	2
5. Принципи тестування програмного продукту, побудованого з використанням doker-контейнера	5

5.1 Розбір класу посилань при побудові тестів. Концепція XUnit 5.2 Проектування тестів у проекті з використанням XUnit 5.3 Застосування функціонального тестування 5.4 Розгортання unit – тестів у докер контейнері 5.5 Тестування та рефакторинг 5.6 Огляд Swagger	
6. Документація до тестування проекту 6.1 Створення проекту та впровадження тестування 6.2 Налаштування Maven для запуску проекту 6.3 Запуск swagger для тестування відпрацювання АПІ 6.4 Unit – тестування на основі XUnit	6
Всього:	21

Розрахунок собівартості і ціни виконання НДР. Виходячи з особливостей створення науково – технічної продукції і її залежності від інтелектуальної праці, розрахунок собівартості і ціни виконання НДР включає наступні статті витрат: витрати на матеріали, основна і додаткова заробітна плата, відрахування до єдиного соціального фонду страхування, витрати на роботи, що виконуються сторонніми організаціями, і деякі інші.

1) Витрати на матеріали складають 165 грн.

2) До витрат «Основна заробітна плата» відносяться оплата праці виконавців, безпосередньо притягнених до її виконання. Розмір основної зарплати встановлюється виходячи з чисельності різних категорій виконавців, трудомісткості, що витрачається ними на виконання різних видів робіт, а також їх середньої заробітної плати (ставки) за один робочий день. Відповідно до статті 8 «Закону про Державний бюджет України на 2021» встановлено мінімальну заробітну плату у місячному розмірі з з 1 січня 2022 року - 6500 гривень; мінімальну погодинну тарифну ставку – 39,26 грн.

Середня зарплата за один робочий день для кожного виконавця визначена по формулі:

$$Зден= п.т.с. * 8;$$

де п.т.с – погодинна тарифна ставка, грн..;

8 – тривалість робочого дня,год.

Зден дипломника = 39.26*8 = 314,08 грн.

Зден керівника = 67.00 *8 = 536 грн.

Зден консультантів = 62.00 *8 = 496 грн.

Витрати на основну заробітну плату, НДР, що включаються в собівартість, приведені в таблиці 4.3.

Витрати на основну заробітну плату.

Таблиця 4.3.

Виконавець	**Погодинна тарифна ставка, грн**	**Денна ставка, грн**	**Трудомісткість робочих днів**	**Сума основної зарплати, грн**
Дипломник	39,26	314,08	22	6909,76
Керівник	67,00	536	1	536
Консультант по економічній частині	62,00	496	0,25	124
Консультант по охороні праці	62,00	496	0,25	124
Нормоконтроль	62,00	496	0,25	124
Всього (Зо)				7817,76

3) Витрати на додаткову заробітну плату визначаються у відсотках від основної. У наукових закладах додаткова заробітна плата складає 10-12% від основної заробітної плати.

$$Зд=11\%Зо;$$

$$Зд= 7817,76*0,11 = 859,95 \text{ грн}$$

4)До складу собівартості НДР включаються податки, збори і інші обов'язкові платежі, встановлені системою оподаткування що діє. Відрахування до єдиного соціального внеску складає:

$$Зєсв=0,22*(Зо+Зд);$$

$$Зєсв=0,22*(7817,76+859,95) = 1909,09 \text{ грн.}$$

5) До накладних витрат відносять витрати на управління і господарське обслуговування, що відноситься до всіх виконуваних НДР.. У наукових закладах накладні витрати складають 40 -120% від основної і додаткової заробітної плати.

$$Рнакл= (З0+Зд)*0,4;$$

$$Рнакл= (7817,76+859,95)* 0,4 = 3471,08 \text{ грн.}$$

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому НДР за формою, приведеною в таблиці 4.4.

Калькуляція планової собівартості

Таблиця 4.4.

Статті витрат	**Сума, грн.**
1. Матеріали	165,00
2. Основна заробітна плата	7817,76
3. Додаткова заробітна плата	859,95
4. Відрахування до єдиного соціального внеску	1909,09
5. Накладні витрати	3471,08
Планова собівартість (Спл)	14222,88

Плановий прибуток визначений по формулі:

Ппл = 0,1*Спл = 0,1*14222,88 = 1422,28 грн

Де 0,1 – норматив, який враховує граничний рівень рентабельності, встановлений чинним законодавством для науково-технічної продукції.

Договірна ціна визначається по формулі

Цнір = Спл + Ппл = 14222,88+1422,28 = 15645,16 грн.

Ціну реалізації встановлюємо з урахуванням ПДВ

ПДВ = 0,2*Цнір= 0,2*15645,16 = 3129,03 грн.

Звідси ціна реалізації становить:

Цр = Цнір + ПДВ Цр = 15645,16 + 3129,03 = 18774,19 грн.

3 ОХОРОНА ПРАЦІ

Соціально і законодавчо захищена людина зацікавлена в своїй праці, цінує свою роботу, яка дає їй змогу пристойно існувати, утримувати сім'ю, годувати і виховувати своїх дітей. Умови праці та економічні фактори безпосередньо впливають на продуктивність і якість праці. Отже, можна констатувати, що охорона праці є категорія економічна.

Безпека праці на підприємстві може бути на належному рівні тільки тоді, коли всебічно відповідає вимогам трудового законодавства, державним стандартам України, норм і правил, розроблених для збереження здоров'я працюючих. Важливе місце при цьому належить виконанню організаційних вимог з охорони праці, а також трудовій та виробничій дисципліні працюючих.

В розділі охорона праці дипломного проекту розглядається питання охорони праці програміста на стадії вирішення ним питань розробки методології впровадження тестування серверного коду за рахунок використання коштів Swagger і XUnit

1 Аналіз небезпечних та шкідливих чинників, що впливають на працівника

На операторів ПК і програмістів можуть мати вплив такі фізичні небезпечні і шкідливі виробничі фактори, як підвищений рівень шуму, підвищена температура зовнішнього середовища, недостатня освітленість робочої зони, електричний струм та інші. Тому на робочому місці програміста повинні бути створені умови для високопродуктивної праці..

2 Розробка заходів з охорони праці

2.1 Виробничі будівлі та приміщення.

Об'ємно-планувальні рішення будівель та приміщень, де експлуатуються відео дисплейні термінали мають відповідати вимогам ДСанПіН 3.3.2.007-98. Розміщення робочих місць з ВДТ ЕОМ і ПЕОМ у підвальних приміщеннях та на цокольних поверхах заборонено.

Площа на одне робоче місце для операторів повинна складати не менше 6,0 м^2, а об'єм – не менше 20,0м^3. Стіни повинні бути пофарбовані матовою краскою.

2.2 Мікроклімат робочої зони працівників, вентиляція.

У виробничих приміщеннях температура, відносна вологість і швидкість руху повітря на робочих місцях повинні відповідати санітарним нормам мікроклімату виробничих приміщень – ДСанПіН 3.3.2-007-98. Оптимальні параметри мікроклімату у виробничому приміщені повинні становити: температура повітря – 22-25^{0}С, відносна вологість – 40-60%, швидкість руху повітряних мас – 0,1-0,2 м/сек.

Для підтримки необхідних параметрів мікроклімату робоче приміщення оснащене системами опалення й кондиціювання.

У приміщеннях, де відбувається робота програміста вимоги до параметрів мікроклімату в цілому виконанні.

2.3 Освітлення робочого місця, шум, вібрація

Приміщення для роботи з ВДТ повинні мати природне та штучне освітлення, відповідно до ДБН В.2.5-28-2006. У приміщеннях, призначених для роботи з відео терміналами, доцільно, щоб вікна були орієнтовані на північ або північний захід. На вікнах повинні бути штора або жалюзі, що регулюють рівень освітленості і захищають від прямого влучення сонячних променів на робоче місце. При кольоровому оформленні виробничих і допоміжних приміщень необхідно враховувати орієнтацію їхніх вікон стосовно частин світу і використовувати гармонійне сполучення кольорів. Для стін і робочих поверхонь використовують мало насичені (основні) кольори, для невеликих помешкань або ділянок, що рідко потрапляють у поле зору працюючих, а також для створення контрастності – кольори середньої насиченості (допоміжні), для маленьких по площі поверхонь – насичені (акценти) – як функціональне фарбування. Стелі у всіх приміщеннях повинні бути білими. Поверхні устаткування в приміщеннях повинні бути матовими або

напівматовими, для виключення випадку відблисків світла в очі працюючого, а стіни бути пофарбованими фарбами пастельних тонів.

Для штучного освітлення у приміщенні використовуються люмінесцентні лампи типу ЛБ, які в порівнянні з лампами розжарювання мають ряд істотних переваг: за спектральним складом світла вони близькі до природного світла, мають підвищену світлову віддачу (у 2-5 разів вищу, ніж у ламп розжарювання); мають триваліший термін служби – до 10 тис годин.. Допускається застосування ламп розжарювання у світильниках місцевого освітлення.

2.4 Захист від дії шуму та вібрації.

У робочому приміщенні основними джерелами акустичних шумів є шуми ПЕОМ. Це коливання елементів електромеханічних пристроїв під впливом змінних магнітних полів тощо. Шум може викликати стомлення слуху й ослаблення звукового сприйняття, а також значне стомлення всього організму.

Оптимальні показники рівня шумів у робочих приміщеннях визначаються за ГОСТ 12.1.003-83. Припустимий рівень шуму при розумовій праці, що вимагає зосередженості – 50 дБ.

2.5 Електробезпека.

Значення сили струму, що проходить через організм людини, залежить від напруги, під якою перебуває людина й від опору ділянки тіла, до якого прикладена ця напруга. Джерелом живлячої напруги є мережа змінного струму з напругою 229В, на яку поширюється ГОСТ 25861-83.

Основними причинами електротравматизму є:

- Випадковий дотик до струмоведучих частин, у результаті ведення робіт поблизу або на цих частинах;
- Несправність захисних засобів, якими потерпілий доторкався до струмоведучих частин;
- Помилкове прийняття устаткування, що перебуває під напругою, як відключеного;

- Несподіване виникнення напруги через ушкодження ізоляції там, де в нормальних умовах його бути не повинно;
- Контакт струмопровідного устаткування із проводом, що перебуває під напругою.

Для попередження поразок електричним струмом необхідно чітко й у повному обсязі виконувати правила провадження робіт і правил технічної експлуатації. Необхідно виключити можливість доступу оператора до частин устаткування, що працює під небезпечною напругою, до неізольованим частинам, призначеним для роботи при малій напрузі й не підключеним до захисного заземлення, а також підводити електроживлення до ПЕОМ від розетки за допомогою спеціальної вилки із заземлюючим контактом.

2.6 Організація робочого місця користувача ПК

Обладнання і організація робочого місця з ВДТ мають забезпечувати відповідність конструкцій всіх елементів робочого місця та їх взаємного розташування, ергономічним вимогам, з урахуванням характеру і особливостей трудової діяльності (ДСанПіН 3.3.2.-007-98).

Конструкція робочого місця й взаємне розташування всіх його елементів (сидіння, органи керування, засобу відображення інформації) відповідають антропометричним, фізіологічним і психологічним вимогам, а також характеру роботи. Конструкція робочих меблів дає можливість забезпечувати можливість індивідуального регулювання їх відповідно до потреб працівника для підтримки зручної пози. Робочий стіл повинен бути пофарбований матовою фарбою. Дисплей розташований так, що його верхній край перебуває на рівні очей, на відстані близько 70 см, що укладається в припустимі рамки від 60 до 90 см. Частота мерехтіння екрана дорівнює 100 Гц, що відповідає умові більше 70 Гц.

2.7 Психофізіологічне розвантаження робітників при роботі з ЕОМ

Для користувачів ВДТ існують загальні принципи збереження здоров’я, оскільки характерною особливістю їх роботи є розвиток психоемоційного стресу.

При проведенні сеансів психофізіологічного розвантаження рекомендується використовувати деякі елементи методу аутогенного тренування, який ґрунтується на свідомому застосуванні комплексу взаємопов'язаних прийомів психічної саморегуляції й виконанні нескладних фізичних вправ із словесним самонавіюванням. Головна увага при цьому приділяється набуванню й закріпленню навичок м'язового розслаблення (релаксації).

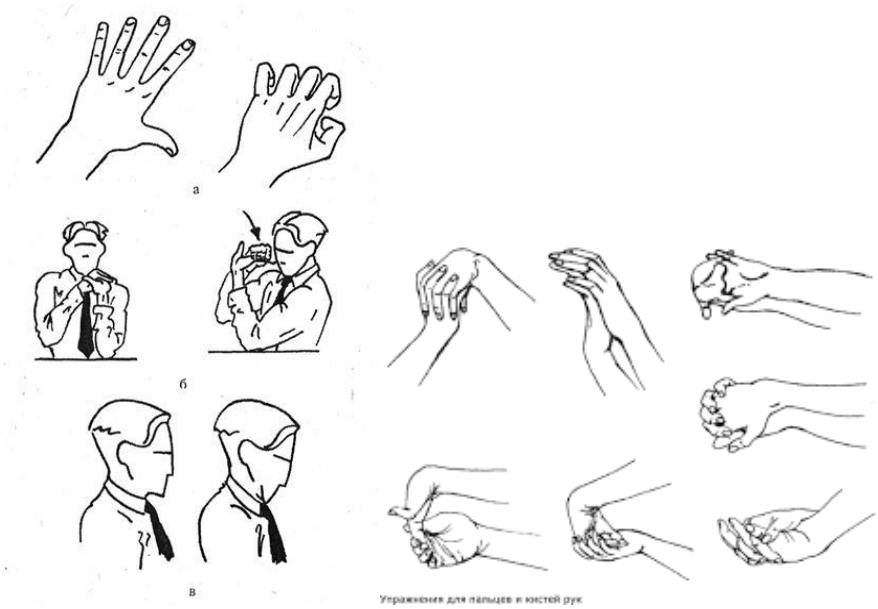

Рис. 3.1 - Фізичні вправи, які рекомендується проводити під час перерв у роботі для зняття напруження, що накопичується в м'язах кистей рук та шиї

Психофізіологічне розвантаження працівників, що виконують роботи із застосуванням ВДТ повинно проводитись у спеціально обладнаних приміщеннях (кімнатах психофізіологічного розвантаження) під час регламентованих перерв, або наприкінці робочого дня.

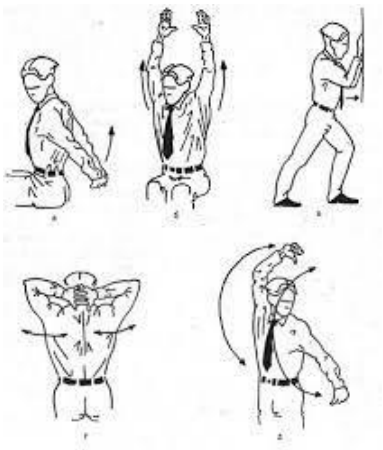

Рис. 3.2 - Фізичні вправи, які рекомендується проводити під час перерв у роботі для зняття напруження, що накопичується в м'язах рук, спини та ніг

Після сеансів психофізіологічного розвантаження у працівників знижується почуття стомленості, відзначається бадьорість, добрий настрій. Загальний стан значно поліпшується.

3 Пожежна безпека.

Під пожежною безпекою розуміють систему державних і суспільних заходів, спрямованих на охорону від вогню людей і власності. Пожежна безпека приміщень, що мають електричні мережі, регламентується ГОСТ 12.1.033-81, ГОСТ 12.1.004-85. Робота оператора ЕОМ повинна вестися в приміщенні, що відповідає категорії Д пожежної безпеки (негорючі речовини й матеріали в холодному стані.

Всі приміщення повинні бути забезпечені первинними засобами пожежегасіння: пожежним водопостачанням (пожежні крани ПК), пожежні щити з набором пожежного інструменту, вуглекислотними або порошковими вогнегасниками. У випадку виникнення пожежі необхідно відключити електроживлення, викликати по телефону 101 пожежну команду, евакуювати людей із приміщення відповідно до плану евакуації і приступити до ліквідації пожежі.

ВИСНОВКИ

У ході даної дипломної роботи було засвоєно та впроваджено такі структури тестування програмного продукту, розробленого на мікровервисній архітектурі:

1. Розроблено шаблон для запиту доступу на клієнтську частину пакета, що тестується з використанням інструменту Swagger. Це один із форматів написання документації до API. За фактом, це великий текстовий документ. Існує два можливі формати: json та yaml.

2. Поставлено та вирішено проблему тестування проекту.

3. Впроваджено та додано юніт тести, а також перевірено покриття програмного коду.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Работа с базой данных SQL Server с использованием C# [Електронний ресурс] https://o7planning.org/ru/10515/working-with-sql-server-database-using-csharp (29.05.2019)
2. Use Local Database File in C# Windows Application [Електронний ресурс] https://www.c-sharpcorner.com/UploadFile/7d3362/use-local-database-file-in-window-application-C-Sharp/ (29.05.2019)
3. Creating Local Database Using Microsoft SQL Server [Електронний ресурс]
4. C# Database Connection Tutorial with Example [Електронний ресурс] (29.05.2019) https://www.guru99.com/c-sharp-access-database.html
5. Албахари, Дж. C# 7.0. Карманный справочник. / Дж. Албахари. - М.: Диалектика, 2017. - 224 с.