

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

РЕВКАЧ АНДРІЙ МАКСИМОВИЧ

Допускається до захисту:

завідувач кафедри

інформаційних технологій

д-р техн. наук, доцент

Тетяна НЕСКОРОДЄВА

« » 2022р.

**РОЗРОБКА ВІДЖЕТУ СПИСКУ СПРАВ З ВИКОРИСТАННЯМ
МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ**

Спеціальність 122 «Комп'ютерні науки»

Кваліфікаційна (бакалаврська) робота

Керівник:

Павло РИМАР, старший викладач
кафедри інформаційних технологій

Оцінка: / /
(бали за шкалою ЄКТС / за національною шкалою)

Голова ЕК:
(підпис)

Вінниця – 2022

АНОТАЦІЯ

Ревкач А.М. Розробка віджету списку справ з використанням мікросервісної архітектури. Спеціальність 122 «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2022.

Основною метою виконання кваліфікаційної роботи є створення мульти платформеного додатку для запису своїх завдань на день і з легким інтерфейсом.

У вступі наведено актуальність розробки даного додатку. Перший розділ містить постановку задачі та огляд аналогів. Другий розділ містить інформацію про використані технології. В третьому розділі наведено основні можливості роботи з додатком.

Ключові слова: Website, Server Side Rendering.

ABSTRACT

Revkach AM Development of a widget TO-DO LIST using microservice architecture. Specialty 122 "Computer Science". Vasyl Stus Donetsk National University, Vinnytsia, 2022.

The main purpose of the qualification work is to create a multi-platform application for recording your tasks on a daily basis and with an easy interface.

The introduction shows the relevance of the development of this supplement. The first section contains the problem statement and an overview of analogues. The second section contains information about the technologies used. The third section presents the main features of working with the application.

Keywords: Website, Server Side Rendering.

ЗМІСТ

ВСТУП	4
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ З РОЗРОБКИ ЗАСТОСУНКУ	5
1.1 Постановка задачі	5
1.2 Огляд існуючих аналогів.....	5
Висновок до розділу 1.....	7
РОЗДІЛ 2. МЕТОДИ ТА ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ ДОДАТКУ	8
2.1 Інструменти	8
2.2 Мови програмування	11
Висновок до розділу 2	20
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ	21
3.1 Опис дизайну користувача.....	21
3.2 Програмна реалізація.....	23
Висновок до розділу 3	38
ВИСНОВКИ.....	39
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	40

ВСТУП

В сучасному світі ми стаємо все більш і більш завантажені різними завданнями, тому нам потрібні додатки, які зможуть допомогти не забути всі справи які нам потрібно виконати. З кожним роком збільшується кількість технологій, змінюються професії і наш темп життя стає набагато швидшим, і людям потрібно бути продуктивнішими. Для цього потрібно планувати свій день і не забувати виконувати ті завдання, які ми собі поставили. Через це виникла ідея створити легкий і мультиплатформений додаток для запису наших завдань на дошку і додати сповіщення по часу, щоб ми не забули, що хотіли зробити. Також в житті кожній людині доводилось зіткнутись з проблемою не зробленого завдання. Тому, проаналізувавши ринок всіх подібних додатків, стало зрозуміло, що безкоштовного і легкого додатку немає.

Мета роботи полягає у розробці спеціального програмного додатку, який дозволить користувачу записувати завдання, які йому необхідно буде виконати протягом певного часу.

Задачами дослідження є:

- Огляд існуючих додатків, проведення порівняльної характеристики.
- Застосування необхідних методів та технологій для розробки функціоналу додатку.
- Розробка додатку.

Об'єкт дослідження: Додаток – дошка завдань.

Предмет дослідження: процес розробки дошки завдань.

Структура роботи: кваліфікаційна (бакалаврська) робота складається зі вступу, трьох розділів, висновків, списку використаних джерел.

РОЗДІЛ 1

ПОСТАНОВКА ЗАДАЧІ З РОЗРОБКИ ЗАСТОСУНКУ

1.1 Постановка задачі

Необхідно розробити програмне забезпечення, яке буде підтримувати функції для запису необхідних завдань, встановлення часу на їх виконання, нагадування про закінчення терміну виконання. Додаток повинен підтримувати наступні функції:

1. Надавати необхідні можливості для роботи.
2. User-friendly інтерфейс.
3. Підтримується локальне сховище.
4. Чуйний, на всіх пристроях.
5. Теми: користувачі можуть вибирати між різними темами.

1.2 Огляд існуючих аналогів

Todoist – веб-сервіс та набір програмного забезпечення для керування задачами. Вони можуть також мати замітки з файлами різних типів. Задачі можна розміщувати в проєкти, сортувати за фільтрами, присвоювати їм мітки, редагувати та експортувати.



Рисунок 1.1 – Логотип «todoist»

Trello – хмарне програмне забезпечення для керування проєктами невеликих груп. Було розроблене Fog Creek Software. Trello використовує парадигму для керування проєктами, відому як канбан. Хоча на початковому етапі ця технологія використовувалася компанією Тойота в 1980-х роках для керування процесом поставок запчастин.



Рисунок 1.2 – Логотип «Trello»

Jira – спеціальний інструмент для керування проєктами, який дозволяє оптимізувати роботу команди розробників. Принцип роботи схожий на роботу диспетчера завдань в комп'ютері: з його допомогою можна слідкувати за проєктами, які знаходяться в стадії розробки, а також контролювати ресурси (кількість співробітників).



Рисунок 1.3 – Логотип «Jira»

На основі альтернативних програм сформовано таблицю з порівняльними характеристиками:

Таблиця 1.1 – Альтернативні програми

	Jira	Todoist	Trello
Безкоштовно	-	-	-
User-Friendly	-	+	-
Мульти платформеність	-	+	-
Простий дизайн	-	-	-

На основі цього можна зробити висновок, що у всіх аналогів є свої недоліки.

Висновок до розділу 1

У цьому розділі було визначено перелік конкурентів і зрозуміли, що немає простого і безкоштовного додатку на всі платформи, а також зручного у користуванні. Також зрозуміли, що попит на даний додаток дуже великий тому, що багато людей цим користується.

РОЗДІЛ 2

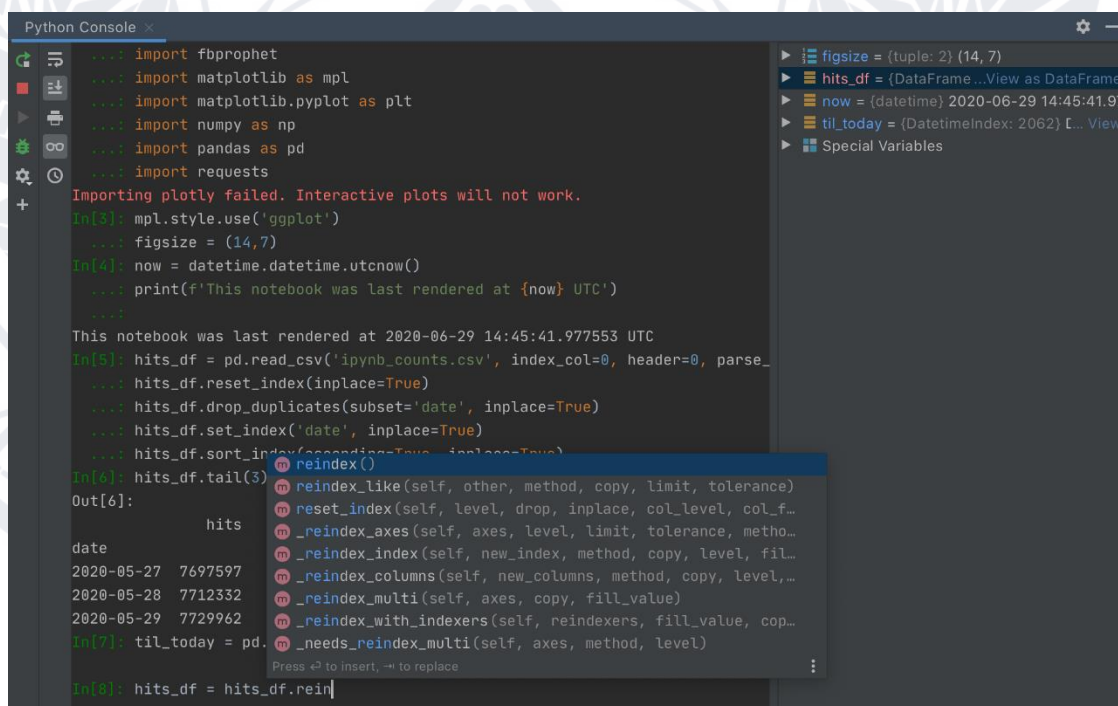
МЕТОДИ ТА ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ ДОДАТКУ

2.1 Інструменти

На сьогоднішній день у сфері програмування є велика кількість методів та технологій, за допомогою яких можна створювати необхідне програмне забезпечення: десктоп, мобільні та веб-додатки. Кожна компанія пропонує своє програмне забезпечення для вирішення тих чи інших задач. Розглянемо декілька з них.

PyCharm

PyCharm – це інтелектуальна IDE для професійної розробки, яка використовує мову програмування Python для написання програмного коду. Середовище розробки забезпечує точне автодоповнення, пошук та виправлення помилок, навігація по коду та інше. PyCharm розроблена компанією JetBrains[2] на основі IntelliJ IDEA. [3]



```
Python Console x
import fbprophet
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import requests

Importing plotly failed. Interactive plots will not work.
In[3]: mpl.style.use('ggplot')
...: figsize = (14,7)
In[4]: now = datetime.datetime.utcnow()
...: print(f'This notebook was last rendered at {now} UTC')
...:
This notebook was last rendered at 2020-06-29 14:45:41.977553 UTC
In[5]: hits_df = pd.read_csv('ipynb_counts.csv', index_col=0, header=0, parse_
...: hits_df.reset_index(inplace=True)
...: hits_df.drop_duplicates(subset='date', inplace=True)
...: hits_df.set_index('date', inplace=True)
...: hits_df.sort_index(inplace=True)
In[6]: hits_df.tail(3)
Out[6]:
             hits
date
2020-05-27  7697597
2020-05-28  7712332
2020-05-29  7729962
In[7]: til_today = pd.
In[8]: hits_df = hits_df.rein
```

Рисунок 2.1 – Приклад вікна розробки PyCharm

Переваги даного IDE – є величезна кількість плагінів які можемо підключити для роботи, це дає змогу не змінюючи IDE продовжити розробку.

WebStorm

Webstorm – IDE від компанії JetBrains для розробки веб-сайтів та редагування html, css та javascript-коду. За його допомогою можна забезпечити швидку навігацію по файлам та генерацію повідомлень про проблеми в програмному коді в режимі реального часу.

Забезпечує процес відладки javascript-коду та надає широкий діапазон можливостей: знаходження точки зупинки в html та javascript, налаштування параметрів, тестування синтаксису в режимі реального часу. Підтримується на великій кількості платформ. Є можливість редагувати файли и автоматично проводити синхронізацію їх за вимогою під час віддаленої роботи або зберігання.

Підтримується функція контролю версій та попереднього варіанту коду та фіксування всіх дій. Завдяки цьому можна переглядати історію зміни програмного коду та повертатися до раніше створених версій.

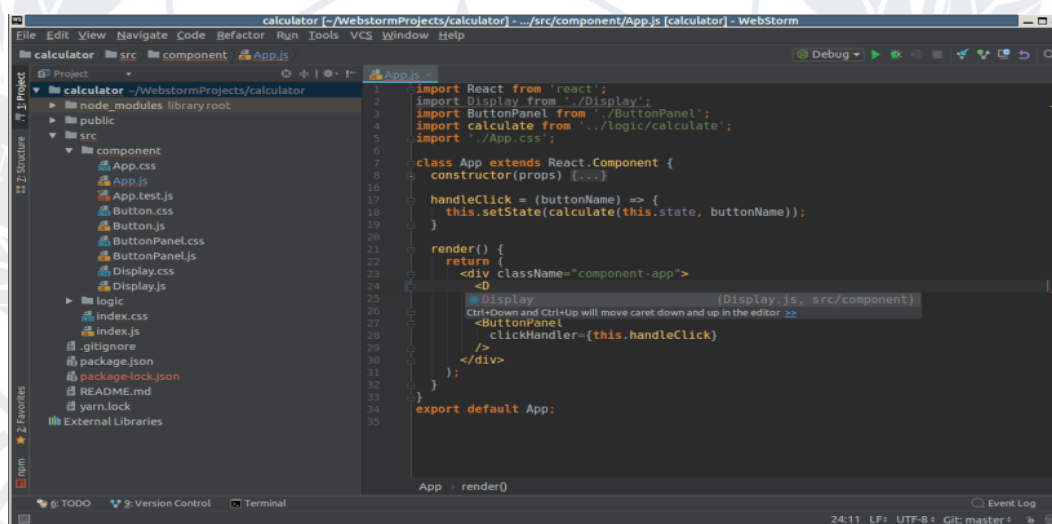


Рисунок 2.2 – Проект у WebStorm

Visual Studio Code

Visual Studio Code – один з популярних редакторів для написання програмного коду. Використовується для розробки веб-додатків та хмарних систем. Підтримується більшістю операційних систем.[4]

Спроектован за допомогою Electron.js. Є можливість для написання коду на багатьох мовах програмування. Вбудовано велику кількість додаткових фіч, які можна використовувати під час розробки. Можна налаштовувати сторонні додатки. Автоматично налаштована інтеграція з Github. Для вивчення потрібен низький поріг входу. Є можливість кастомізувати налаштування під потреби користувача.



Рисунок 2.3 – Логотип Visual Studio Code

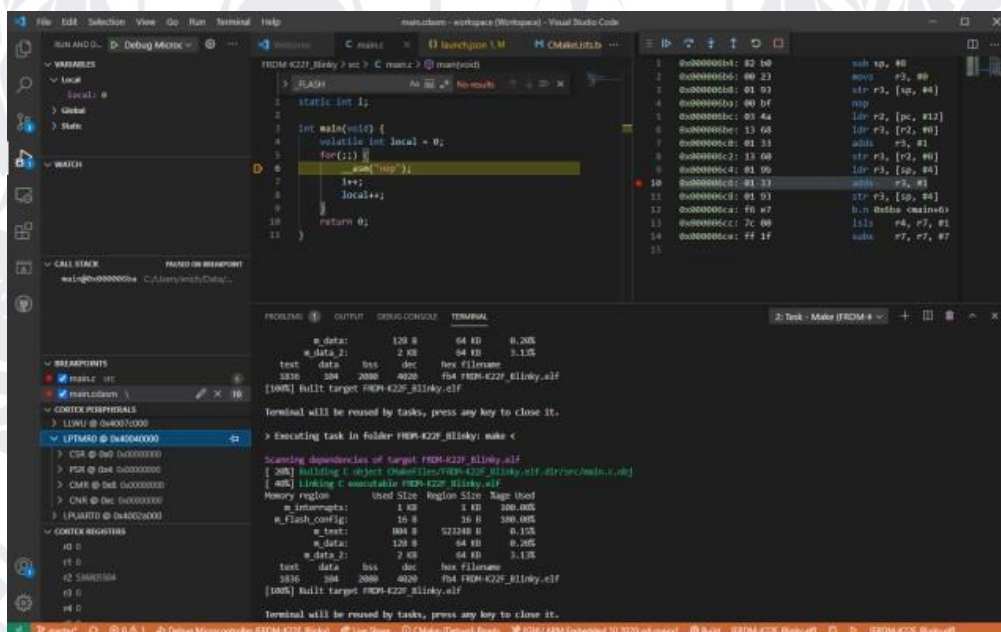


Рисунок 2.4 – Інтерфейс розробки Visual Studio Code

Відповідно до характеристик, це доволі зручний інструмент для розробки різного роду додатків та систем. На його основі розроблено додаток.

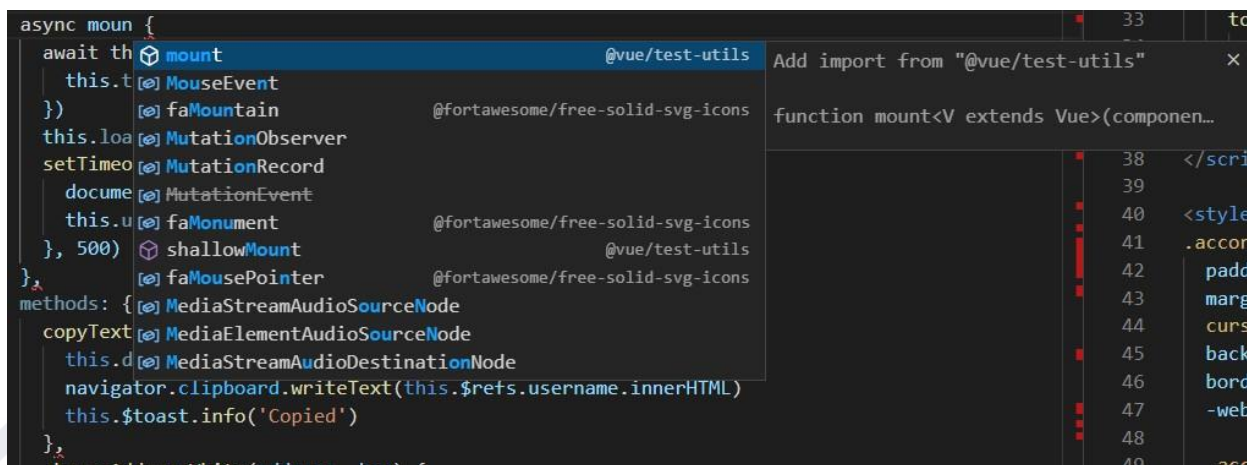


Рисунок 2.5 – Автозавершення відповідно до коду із бібліотек та framework's

2.2 Мови програмування

В якості мови програмування використовувалася JavaScript, для серверної частини використовувався Python з додатком Django. Для поставлення на хост і легкого розвертання використовувалися інструменти DevOps такі як: AWS, Jenkins.

JavaScript – інамічна об'єктно-орієнтовна прототипна мова програмування. В сучасному світі розробки веб-додатків без JS не обійтись.

Взагалі для створення веб-сторінок використовують декілька технологій, такі як HTML, CSS і JS .

Також JS може підключати до себе різні плагіни, які дадуть змогу збільшити інструментарій для розробки веб-додатків.



Рисунок 2.6 – Логотип мови програмування JavaScript

Python – скриптова мова програмування, яку можна використовувати майже скрізь, в роботі вона була використана для написання серверної частини. Дана мова програмування також може використовувати різні додатки та плагіни для серверної частини.



Рисунок 2.7 – Логотип мови програмування Python

AWS – сама поширена в світі хмарна платформа, яка дає більше 200 повнофункціонуючих сервісів для центрів обробки даних по всьому світу. В роботі використовувалося для резервації віртуальної машини і хостингу для сервісу.

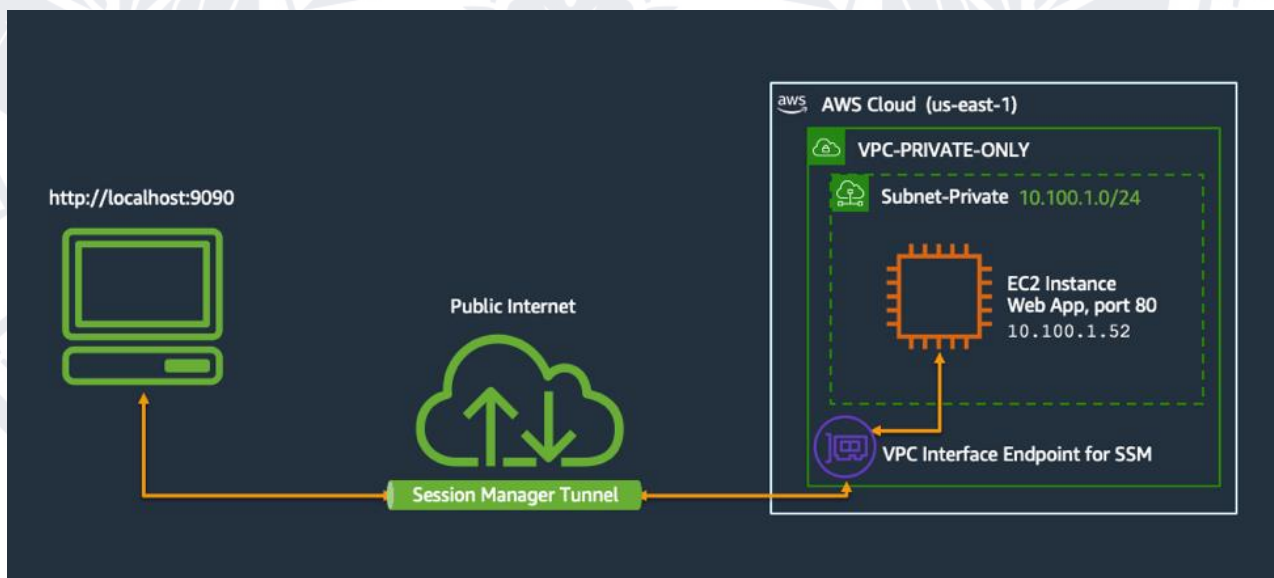


Рисунок 2.8 – Схема мережі AWS EC2

Jenkins – інструмент, який дуже популярний в сфері DevOps, його використовують для автоматизації процесів. Другими словами CI/CD .



Рисунок 2.9 – Логотип Jenkins

GitHub – інструмент для зберігання коду та його змін. Дуже ефективний інструмент без якого в сучасному світі ІТ не обійтись.

Плюси DevOps технологій в сучасному ІТ

На сьогоднішній день світ ІТ дуже змінився, з'явилося дуже багато нових технологій та методологій. Одна з них це DevOps. Дані технології спрощують розробку продукту та допомагають розробникам зберегти свій час.

DevOps (англ. development і operations) – практика, призначена для поєднання взаємодії розробників із фахівцями інформаційно-технологічного обслуговування та зближення їхніх робочих процесів одне з одним. Ґрунтується на думці про те, що існує взаємозалежність між розробкою та використанням програмного забезпечення та на меті допомоги організації має швидше створювати та оновлювати програмні продукти та послуги.

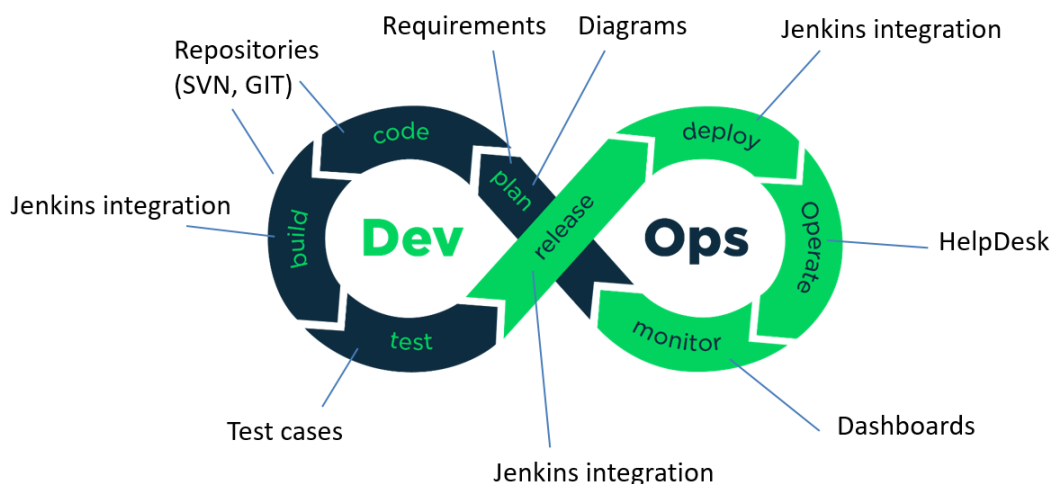


Рисунок 2.10 – Схема методології DevOps

В даному проекті було використано дві основні технології: Jenkins і AWS. Jenkins - відкритий інструмент для безперервної інтеграції, написаний на Java. Проект був розроблений проектом Hudson, після конфлікту з Oracle, який заявив про свої права на торговельну марку Hudson і так створив її в грудні 2010 року. Jenkins розробляє частину процесу розробки ПЗ, яка не потребує участі (у таких, як неперервна інтеграція), та розширює технічні можливості розробників команд з неперервною доставкою продукту.

Використовувався для автоматизації розгортання продукту на хмарний сервер та автоматичних тестів коду на сервері .

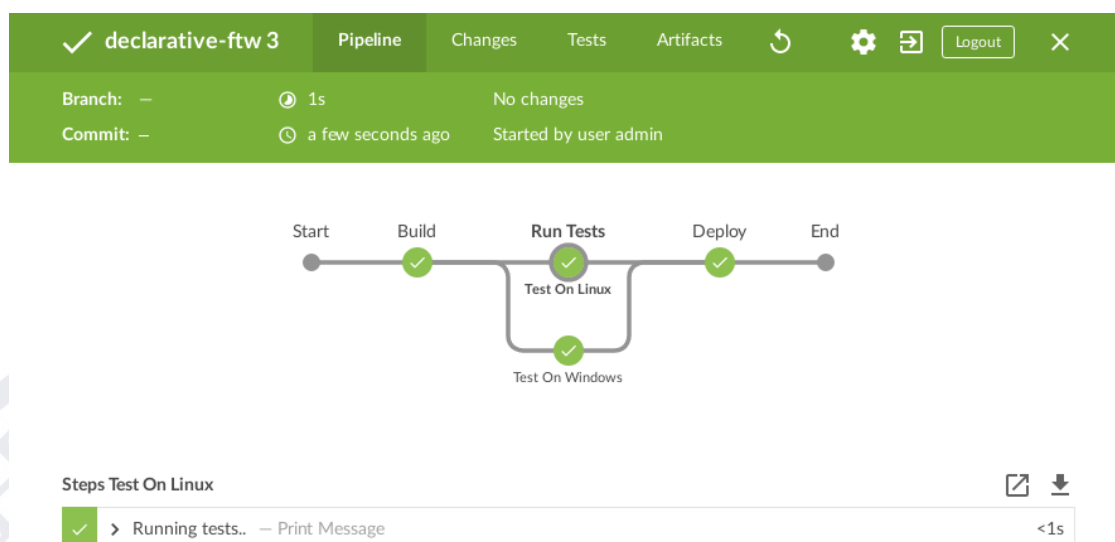


Рисунок 2.11 – Екран запуску пайплайну Jenkins

Великим плюсом є те що Джоби (скрипти для автоматизації) можна переносити з сервера на сервер і це не зламає автоматизаційний процес. Також сервер Jenkins дуже легко інстальювати на любую систему.

Jenkins pipeline

Jenkins Pipeline – набір плагінів, що дозволяє визначити життєвий цикл складання та доставки програми як код. Він є Groovy-скриптом з використанням Jenkins Pipeline DSL і зберігається стандартно в системі контролю версій. Існує два способи опису пайплайнів – скриптовий та декларативний.

Scripted:

```
node {
    stage('Example') {
        try {
            sh 'exit 1'
        }
        catch (exc) {
            throw exc}}}

```

Declarative:

```

pipeline {
    agent any
    stages {
        stage("Stage name") {
            steps {}
        }
    }
}

```

Вони обидва мають структуру, але в скриптовому вона вільна – достатньо вказати, на якому слейві запускатись (node), та стадію складання (stage), а також написати Groovy-код для запуску атомарних степів.

Декларативний пайплайн визначено жорсткіше, і, відповідно, його структура читається краще.

Розглянемо докладніше декларативний пайплайн.

1. У структурі має бути визначена директива pipeline.
2. Також потрібно визначити, на якому агенті (agent) буде запущено складання.
3. Далі йде визначення stages, які будуть у пайплайні, і обов'язково має бути конкретний стейдж під назвою stage("name"). Якщо імені немає, тест впаде в runtime з помилкою "Додайте ім'я стейджу".
4. Обов'язково має бути директива steps, у якій містяться атомарні кроки складання. Наприклад, ви можете вивести в консоль Hello.

```

pipeline {
    agent any
    stages {
        stage("Stage name") {
            steps {
                echo "Hello work"
            }
        }
    }
}

```

Більше подобається декларативний вид пайплайну тим, що він дозволяє визначити дії після кожного стейджу або, наприклад, після всього збирання. Його рекомендовано використовувати під час опису пайплайнів на верхньому рівні.

```

pipeline {
    stages {
        stage("Post stage") {
            post { // определяет действия по завершении стейджа
                success { // триггером исполнения секции является состояние сборки
                    archiveArtifacts artifacts: '**/target/*'
                }
            }
        }
        post { // после всей сборки
            cleanup {
                cleanWs()
            }
        }
    }
}

```

Якщо складання та стейдж завершилися успішно, можна зберегти артефакти або почистити workspace після складання. Якщо за таких умов використовувався б скриптовий пайплайн, довелося б за цим «флоу» стежити самостійно, додаючи обробку винятків, умови тощо.

AWS – хмарний сервіс, який має понад 200 сервісів для допомоги розробці проекту



Рисунок 2.12 – Логотип AWS

Переваги AWS

- всі сервіси мають API, які дозволяють інженерам DevOps дотримуватися принципів IaC та використовувати Terraform для проектування та розгортання постійної IT-інфраструктури для клієнтів.
- більшість сервісів мають SLA, що є величезною перевагою для стартапів, оскільки вони можуть значно скоротити витрати на підтримку
- Для роботи з AWS створено величезну різноманітність інструментів, які мають докладні посібники з цього питання.

Недоліки

- Незважаючи на те, що AWS наголошує на економічній ефективності, він досить дорогий у порівнянні з GCP або Azure. У якийсь момент свого зростання компанії розуміють, що створення та підтримка власної локальної хмари буде менш дорогим вибором, ніж продовжувати оплачувати рахунки AWS, що постійно зростають.
- Платформа має досить круту криву навчання, і освоїти всі необхідні послуги самостійно навряд чи можливо. Наприклад, щоб побудувати дійсно надійну, безпечну та відмовостійку IT-інфраструктуру, компанії доведеться з'ясувати використання маршрутизації та політики IAM – або найняти сертифікованого AWS партнера, щоб налаштувати його для них.

- Amazon API-інтерфейсів багато, так, але вони далекі від досконалості. Наприклад, AWS не надає SES в азіатському регіоні, але його API дозволяє користувачеві відправляти запит на створення ресурсу SES, який буде активний до скасування через час очікування запиту.

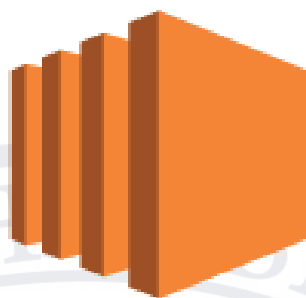
Для даного проекту було створено приватну мережу VPC і сервер EC2.

AWS VPC – це сервіс, що дозволяє створювати приватні ізольовані мережі у публічній хмарі Amazon.



Рисунок 2.13 – Логотип AWS VPC

AWS EC2 – є одним із сервісів Amazon Web Services, що дозволяє користувачеві орендувати віртуальний сервер, які називаються інстанс. Для запуску віртуальних серверів використовуються попередньо налаштовані образи, що скорочує час завантаження нового сервера.



Amazon
EC2

Рисунок 2.14 – Логотип EC2 AWS

Висновок до розділу 2

В цьому описано всі технології, які використовувалися для подальшої розробки продукту. Далі буде описано сам процес розробки.

РОЗДІЛ 3

ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Опис дизайну користувача

Головною метою створення даного додатку було створення простого інтерфейсу для користувача та сучасний дизайн. Для цього було зроблено декілька тем на вибір для користувача.

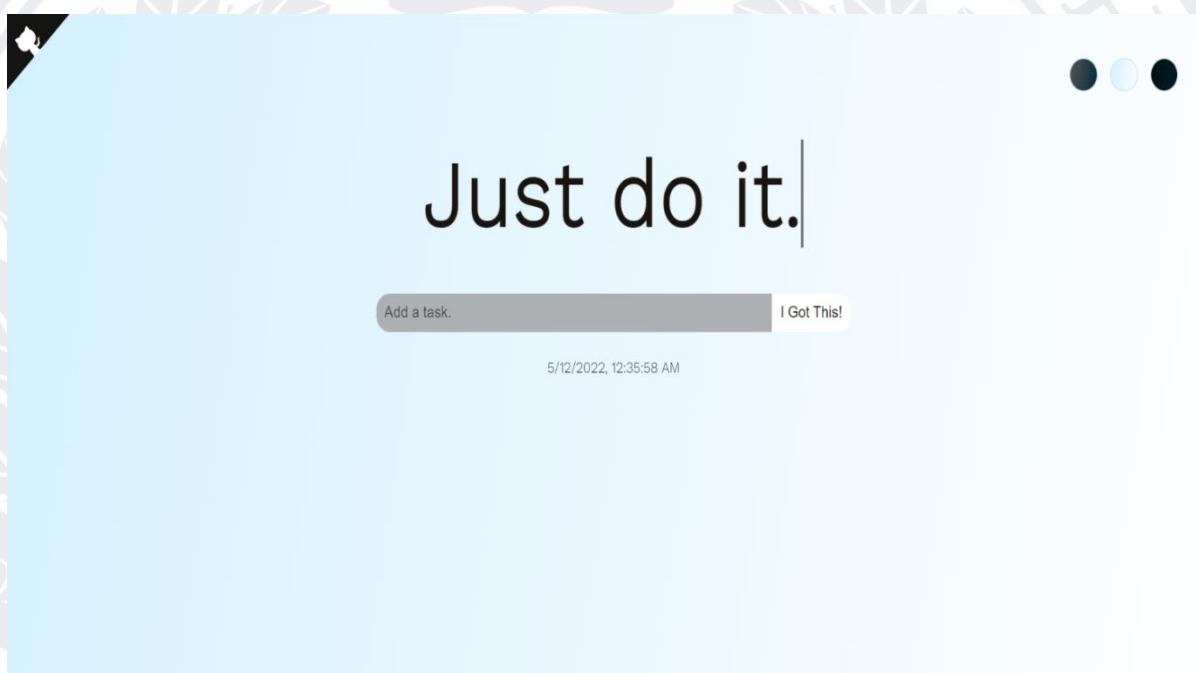


Рисунок 3.1 – Світла тема додатку

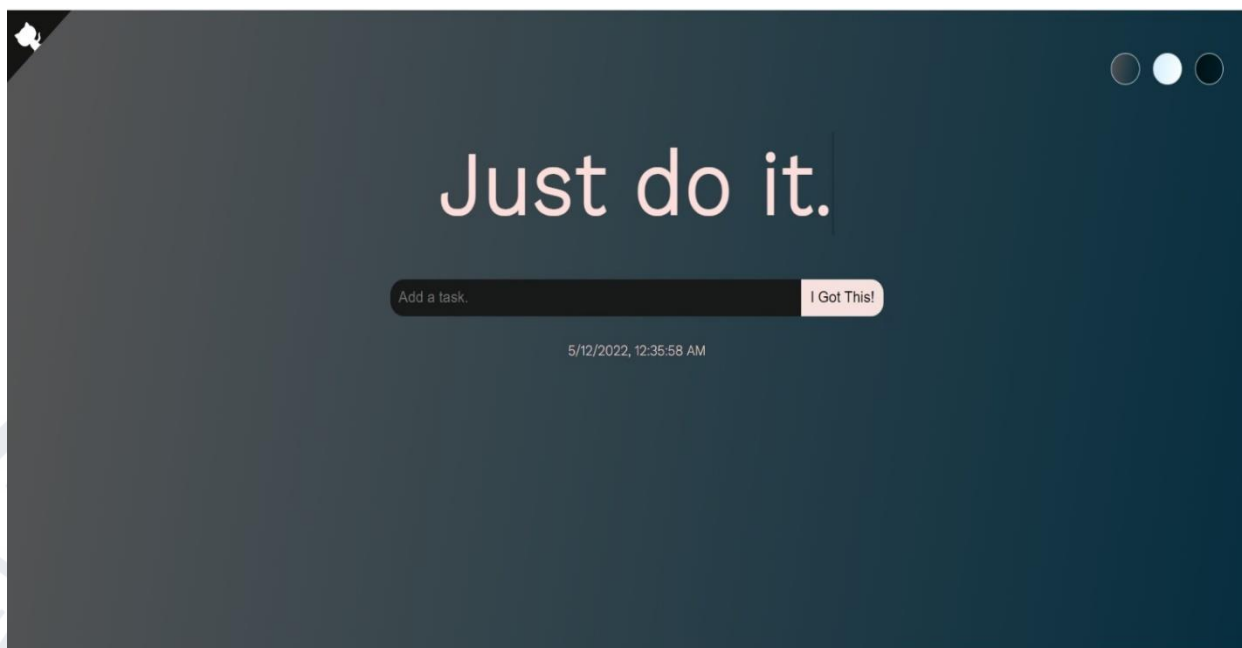


Рисунок 3.2 – Сіра тема додатку

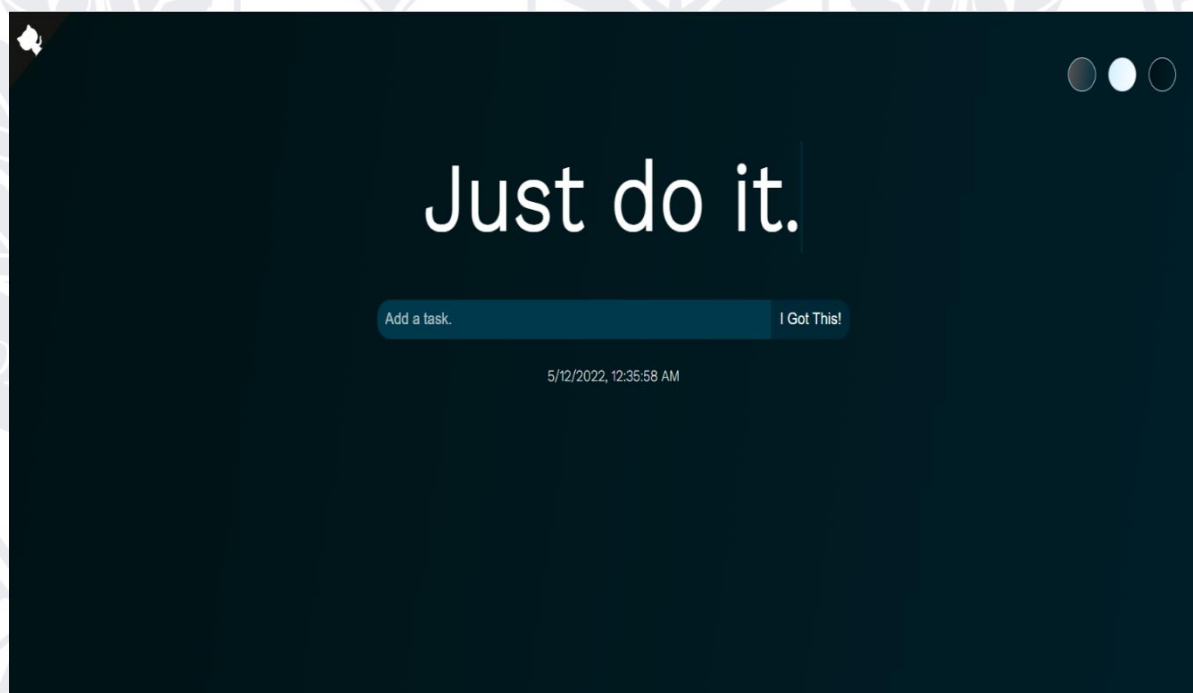


Рисунок 3.3 – Темна тема додатку

Проблему з сучасним і красивим дизайном було вирішено. Також було добавлено посилання на гіт розробника, щоб можна було слідкувати за всіма новинками та апдейтами. Створено максимально легке керування та інтуїтивні кнопки. За допомогою них ми можемо добавляти та видаляти наші завдання на день. Було створено прив'язку до реального часу та створено під додаток який

буде відправляти вам нагадування на телефон або комп'ютер.

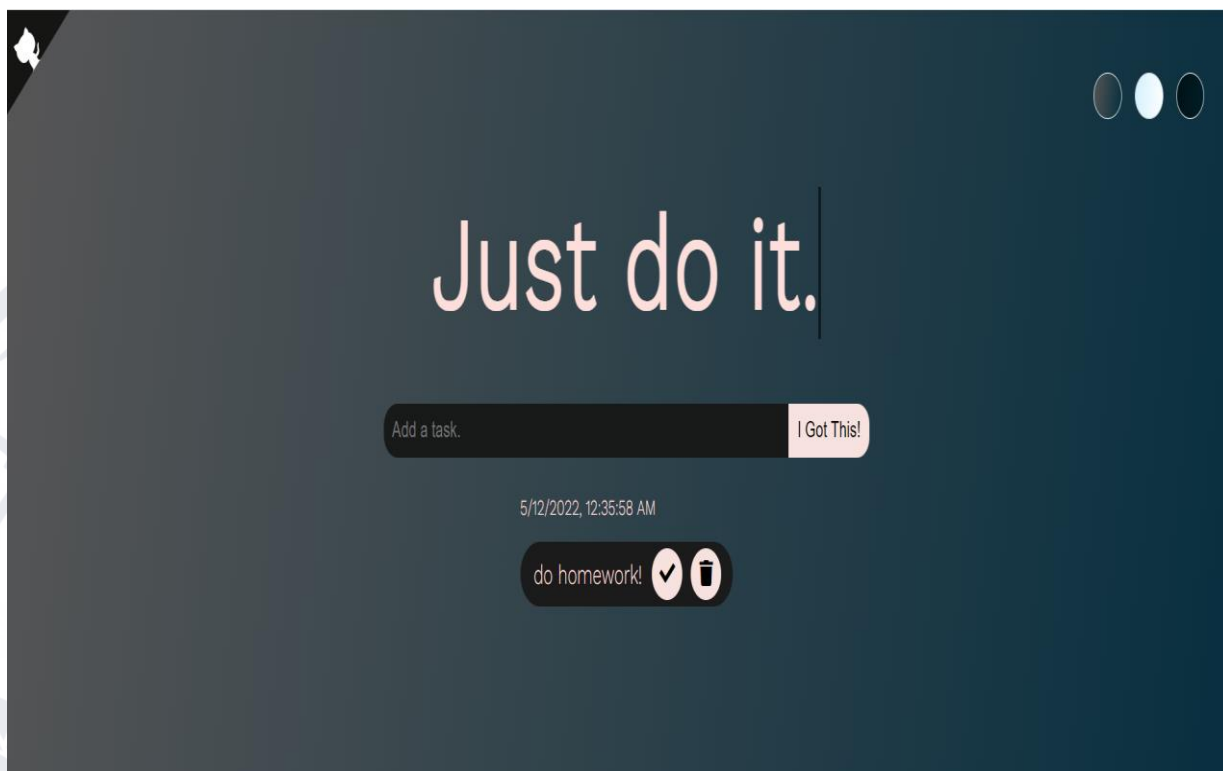


Рисунок 3.4 – Демонстрація роботи додатку

3.2 Програмна реалізація

На рисунку нижче наведено архітектуру розроблюваного проєкту.

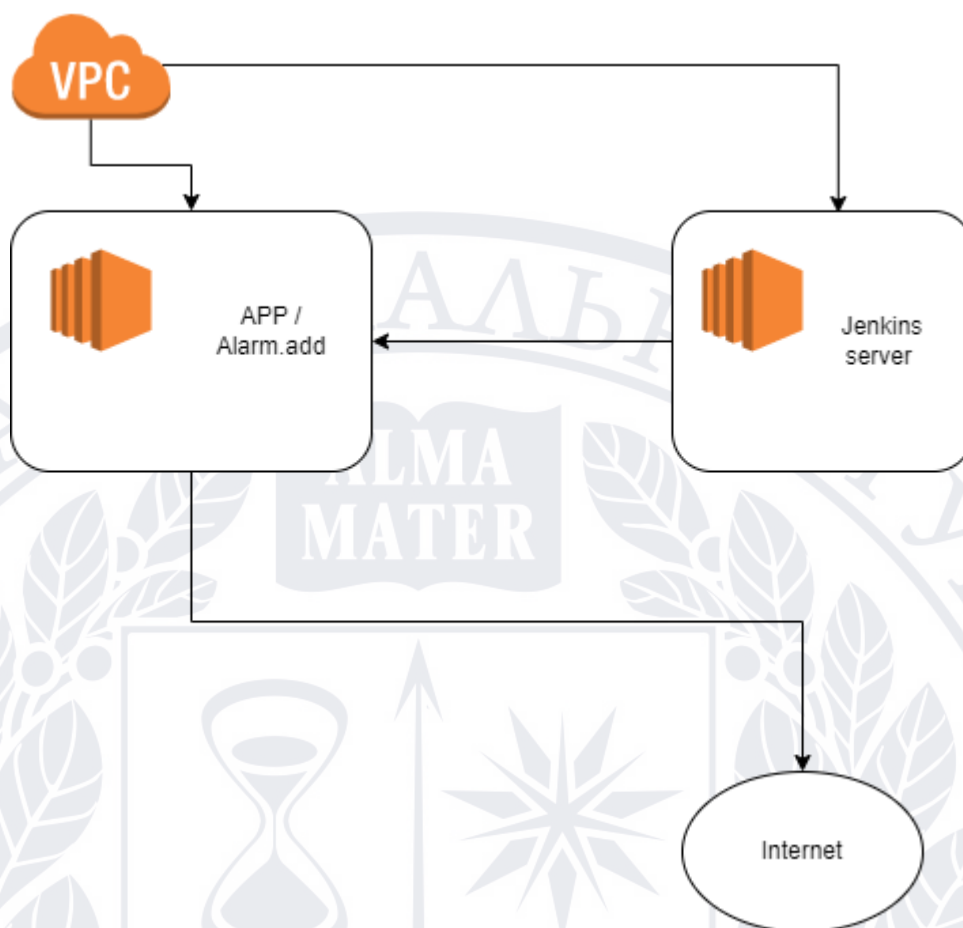


Рисунок 3.5 – Схема архітектури проекту

1. Створення архітектури на хмарному сервісі AWS
2. Верстка дизайну
3. Створення пайплайнів на Jenkins сервері

Перше, що потрібно створити для роботи додатку, це AWS VPC. Для прикладу було обрано шістнадцяту підмережу 10.50.0.0, яка включає $256 \times 256 = 65536$ адрес. До речі, ми можемо вибрати параметр Tenancy:

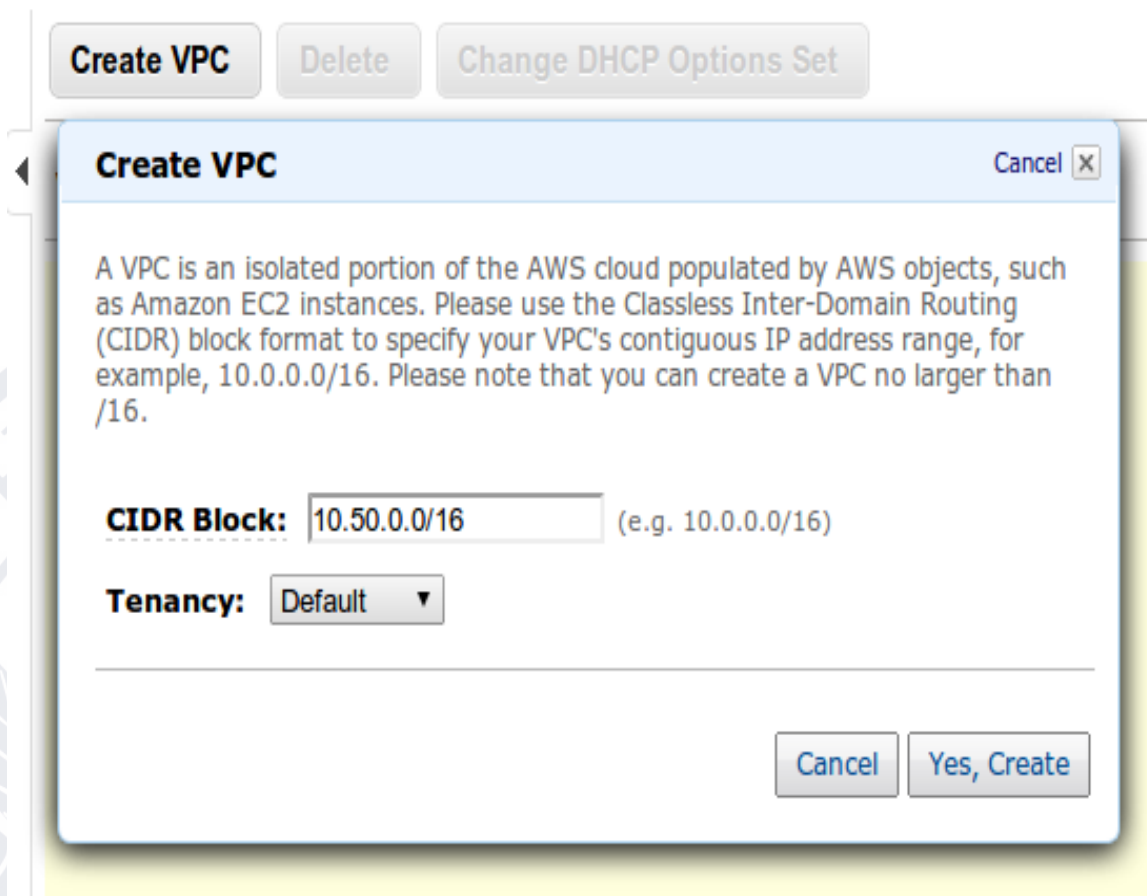


Рисунок 3.6 – Створення своєї мережі

Subnetworks

Створили хмару, тепер потрібно створити підмережі. Наша хмара прив'язана до регіону і може мати безліч підмереж, а самі підмережі вже прив'язані до регіонів. Також підмережі бувають:

- приватні – не мають прямого доступу до Інтернету, лише через публічні мережі та NAT
- публічні – можуть мати доступ до Інтернету через Elastic IP та Internet Gateway

Отже, було створено в одній зоні 2 підмережі: приватну та публічну, а в іншому тільки приватну, щоб вийшло так:

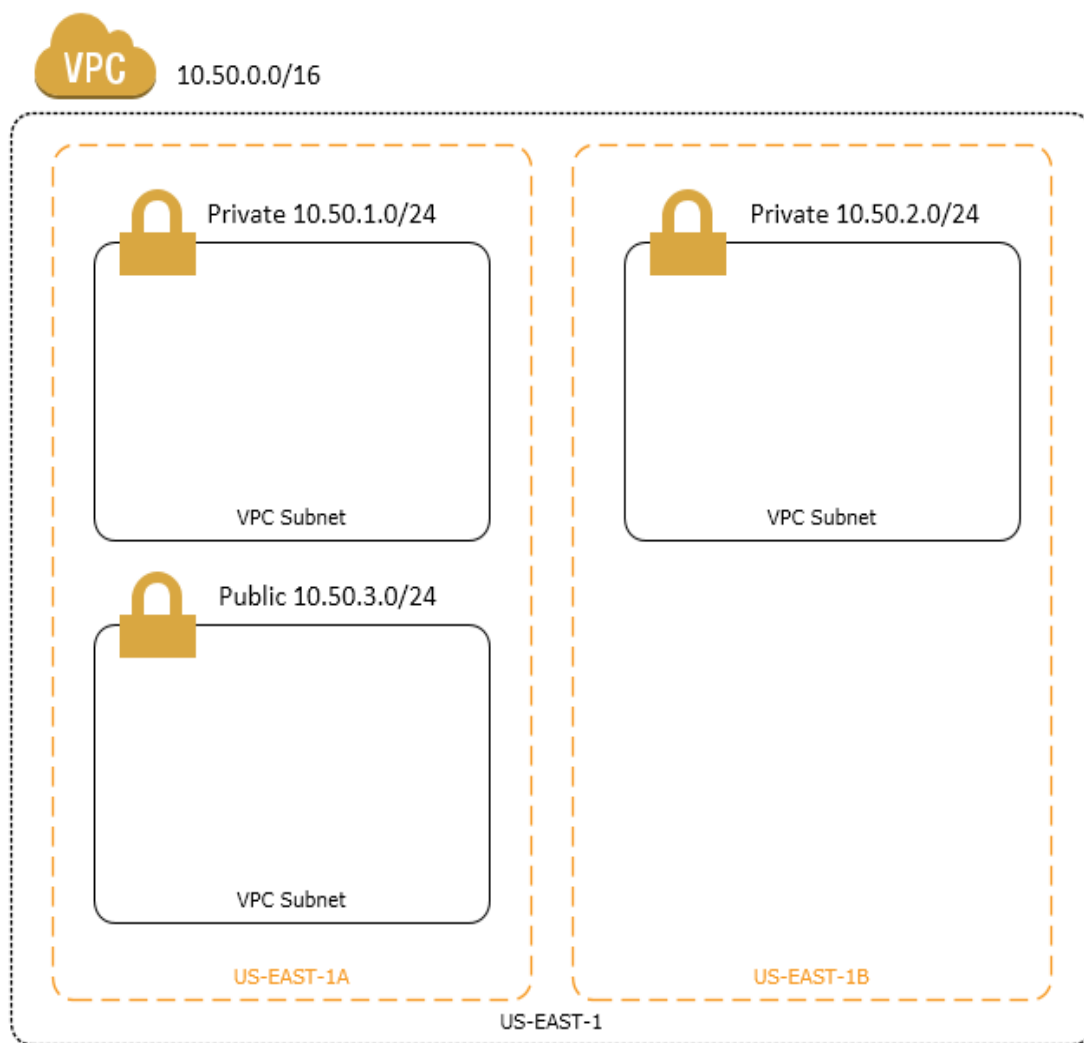


Рисунок 3.7 – Схема мережі без доступу в інтернет

Після створення мережі та підмереж потрібно відкрити доступ в інтернет для нашого додатку. Для цього використовуємо Internet Gateway. Internet Gateway – це сервіс AWS для транслявання трафіку в інтернет.

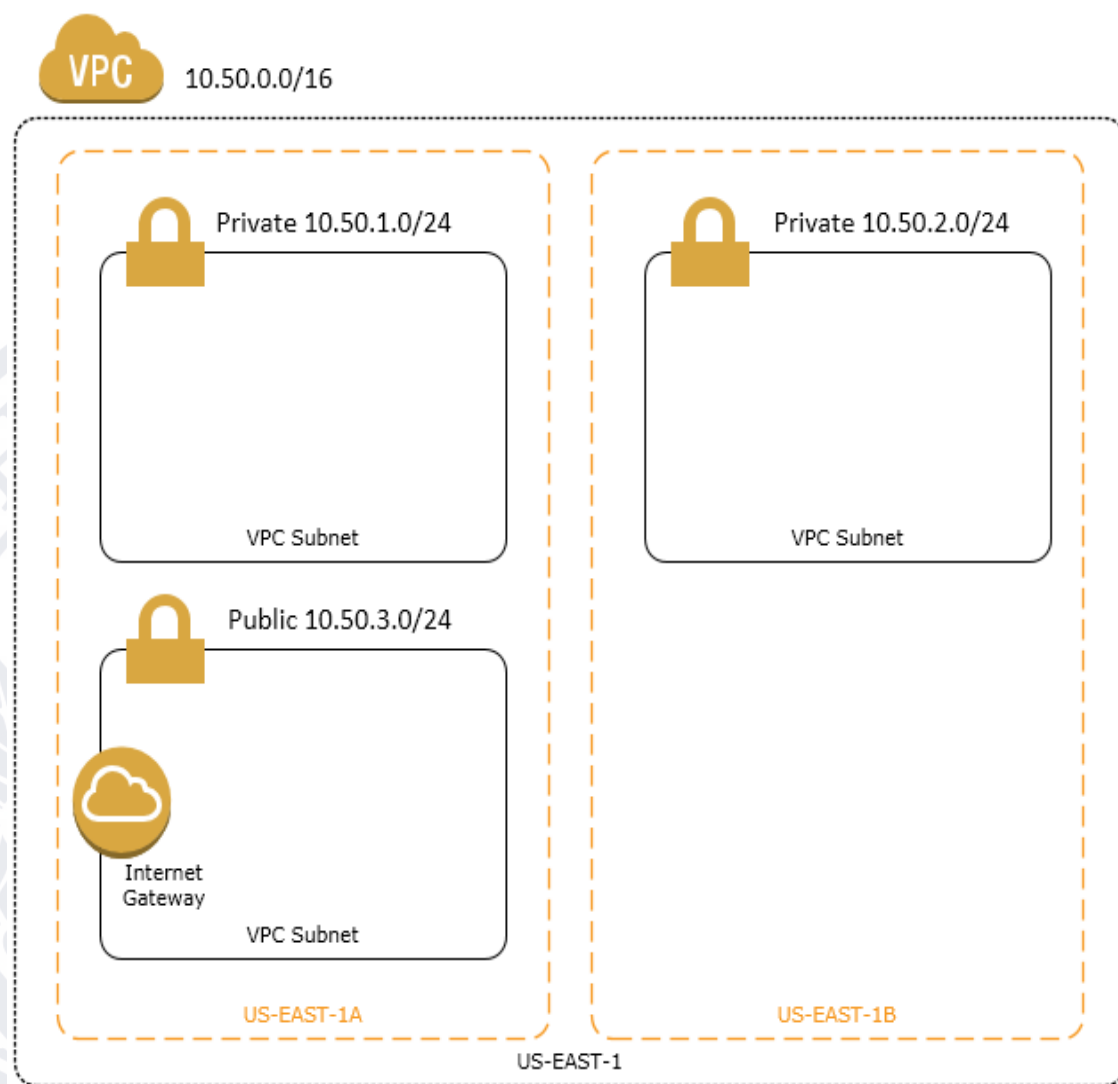


Рисунок 3.8 – Схема мережі з доступом в інтернет

Далі потрібно створити Route table. Route Table – сервіс який дозволяє відкрити трафік на входні порти або закрити їх. Після створення нашої мережі створюємо віртуальну машину для запуску наших додатків та Jenkins сервері. Екземпляр EC2 не що інше, як віртуальний сервер в Amazon Web Services термінології. Це означає Elastic Compute Cloud. Це веб-сервіс, де передплатник AWS може запитувати та надавати обчислювальний сервер у хмарі AWS. На вимогу екземпляр EC2 є пропозицією від AWS, де абонент/користувач може орендувати віртуальний сервер за годину та використовувати його для розгортання його/її власних додатків.

Примірник буде оплачуватись за годину з різними ставками залежно від типу обраного екземпляра. AWS надає кілька типів екземплярів для відповідних бізнес-потреб користувача. Перше, що потрібно зробити це зарезервувати машину.

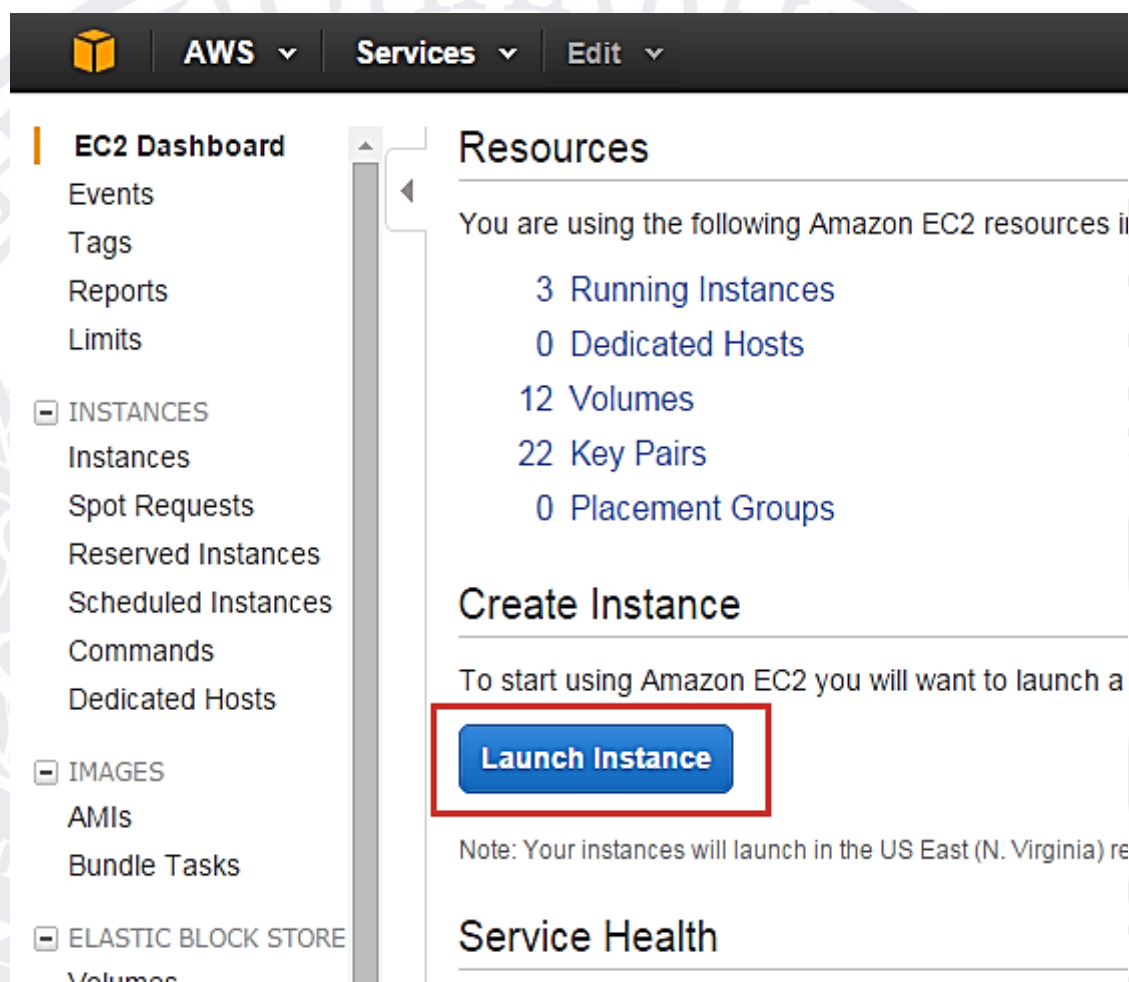


Рисунок 3.9 – Запуск віртуальної машини

Наступним кроком буде підключення віртуальної машини до нашої створеної мережі. Це дає змогу віртуальній машині перенаправляти трафік в інтернет. Також потрібно вибрати АМІ для нашої машини. АМІ – це код для створення нашої віртуальної машини, технологія швидкого створення (заздалегідь створені налаштування).

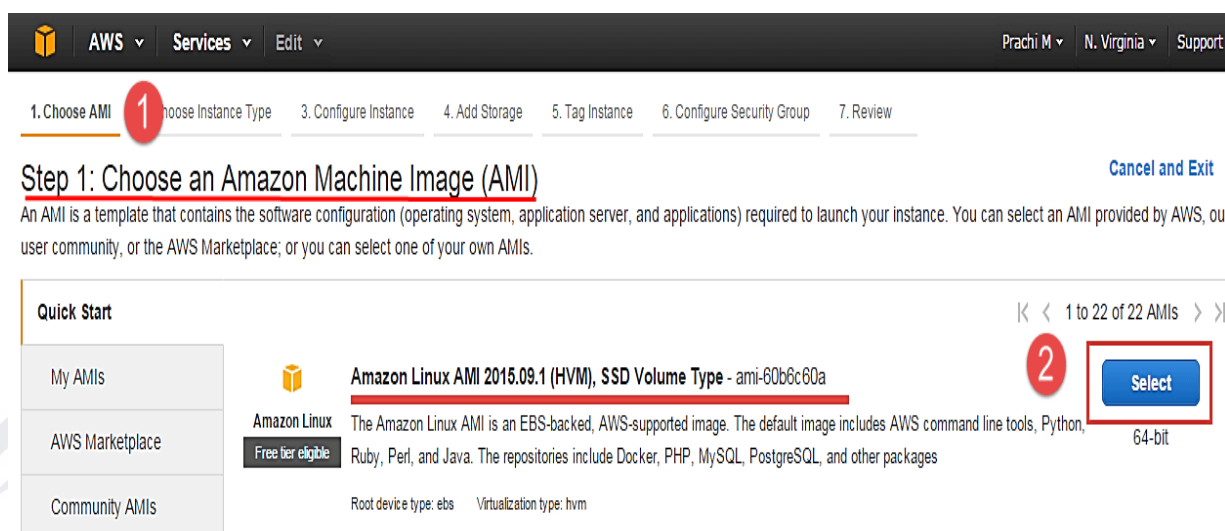


Рисунок 3.10 – Вибір типу віртуальної машини

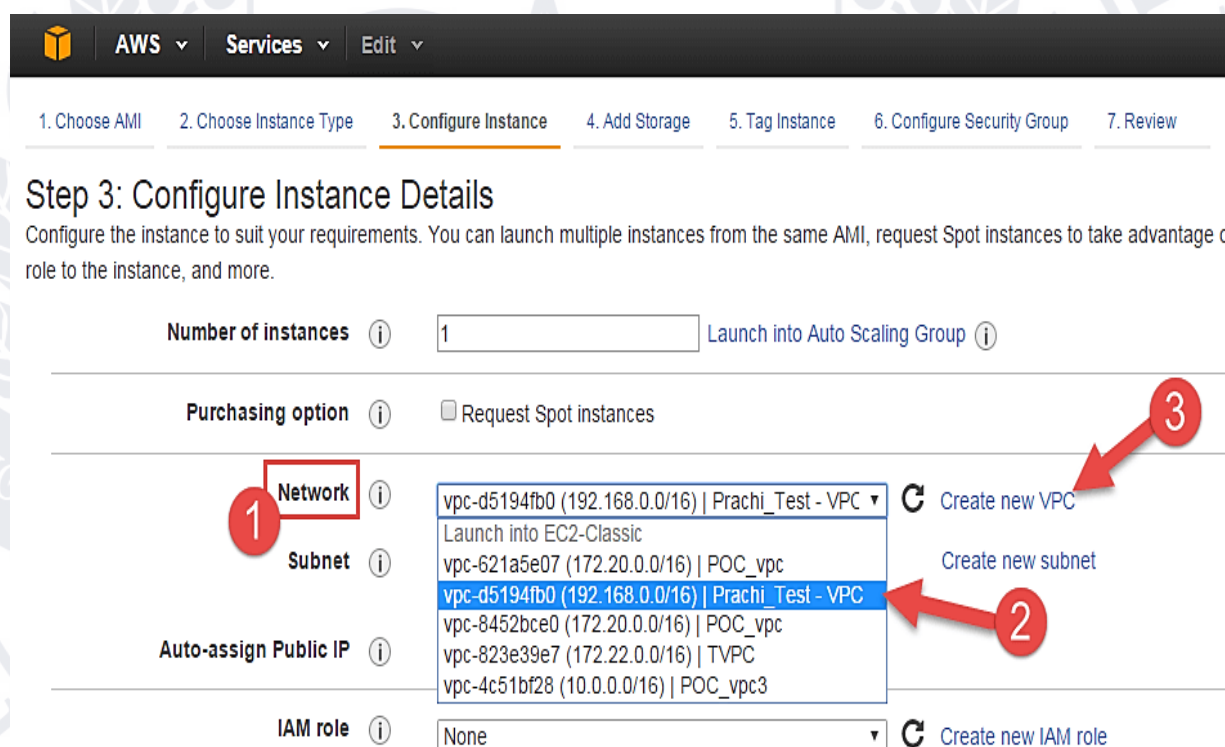


Рисунок 3.11 – Підключення віртуальної машини до мережі

Після всіх кроків ми маємо готовий сервер для додатку та Jenkins сервер. Наступна ціль – встановлення Jenkins на сервер. Це дозволяє автоматизувати частину процесу розробки програмного забезпечення, у якому обов'язково участь людини, забезпечуючи функції безперервної інтеграції. Працює у сервлет-контейнері, наприклад, Apache Tomcat. Підтримує інструменти системи керування версіями, включаючи AccuRev, CVS, Subversion, Git, Mercurial,

Perforce, Clearcase та RTC. Може збирати проекти за допомогою Apache Ant та Apache Maven, а також виконувати довільні сценарії оболонки та пакетні файли Windows. Складання може бути запущено різними способами, наприклад, за подією фіксації змін у системі керування версіями, за розкладом, за запитом на певний URL, після завершення іншого складання черги. Після встановлення Jenkins на сервер потрібно скачати пайплайни з github репозиторія і встановити їх .



Рисунок 3.12 – Запуск пайплайну

Jenkins pipeline – Jenkins Pipeline (або просто «Pipeline» з великої букви «P») – це набір плагінів, який підтримує впровадження та інтеграцію конвеєрів безперервної доставки в Jenkins. Це безвідказна система інтеграції коду та запуску його на віртуальній машині. Після встановлення ми запускаємо pipeline і Jenkins server який починає інсталювання додатку на сервер slave (віртуальна машина яка підключена під Jenkins master). Всі ресурси для інсталювання беруться з github репозиторія.

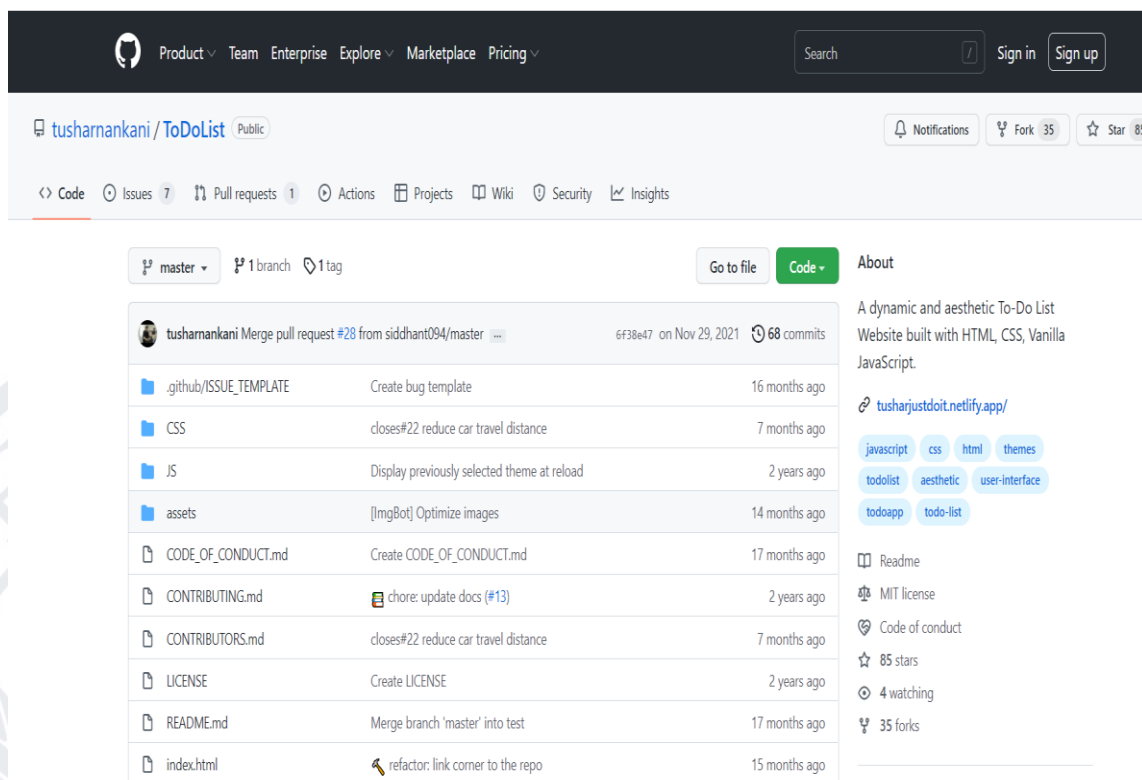


Рисунок 3.13 – Головна сторінка репозиторія

HTML — мова гіпертекстової розмітки, яка використовується для перегляду веб-сторінок у браузері. Використовуються теги для відображення різного виду інформації. Для стильового оформлення використовуються каскадні таблиці стилей.

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta name="theme-color" content="#062e3f">
7   <meta name="Description" content="A dynamic and aesthetic To-Do List WebApp.">
8
9   <!-- Google Font: Quick Sand -->
10  <link href="https://fonts.googleapis.com/css2?family=Work+Sans:wght@300&display=swap" rel="stylesheet">
11
12  <!-- font awesome (https://fontawesome.com) for basic icons; source: https://cdnjs.com/libraries/font-awesome -->
13  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.12.0-2/css/all.min.css" integrity="sha256-46r060N2LrChLb5zowXQ72/iKKHw/1AmygmHExk/o="
14
15  <link rel="shortcut icon" type="image/png" href="assets/favicon.png"/>
16  <link rel="stylesheet" href="CSS/main.css">
17  <link rel="stylesheet" href="CSS/corner.css">
18  <title>JUST DO IT</title>
19
20 </head>
21
22 <body>
23   <div id = "header">
24     <div class="flexrow-container">
25       <div class="standard-theme theme-selector"></div>
26       <div class="light-theme theme-selector"></div>
27       <div class="darker-theme theme-selector"></div>
28     </div>
29     <h1 id="title">Just do it.<div id="border"></div></h1>
30   </div>
31
32   <div id="form">
33     <form>
34       <input class="todo-input" type="text" placeholder="Add a task.">
35       <button class="todo-btn" type="submit">I Got This!</button>

```

Рисунок 3.14 – Файл HTML коду

CSS – каскадні таблиці стилей, які використовують правила стильового оформлення елементів на веб-сторінці. Застосовується для документів, які написані за допомогою html або йому подібних мов розмітки даних.

```

1 .github-corner:hover .octo-arm {
2   animation: octocat-wave 560ms ease-in-out;
3 }
4
5 @keyframes octocat-wave {
6   0% {
7     transform: rotate(0deg);
8   }
9
10  20% {
11    transform: rotate(-25deg);
12  }
13
14  40% {
15    transform: rotate(10deg);
16  }
17
18  60% {
19    transform: rotate(-25deg);
20  }
21
22  80% {
23    transform: rotate(10deg);
24  }
25

```

Рисунок 3.15 – файл CSS коду

```
1  * {
2    margin: 0;
3    padding: 0;
4    box-sizing: border-box;
5  }
6
7  body {
8    /* #363532, rgb(88, 111, 112) */
9    align-items: center;
10   display: flex;
11   flex-direction: column;
12   font-family: 'Work Sans', sans-serif;
13   min-height: 100vh;
14   padding-top: 3%;
15 }
16
17 /* Body light or darker themes */
18 .standard {
19   background-image: linear-gradient(100deg, #575656, #062e3f);
20   color: #ffdfdb;
21   transition: 0.3s linear;
22   overflow: hidden;
23 }
24
25 .light {
26   background-image: linear-gradient(100deg, #d4f1ff, #ffffff);
27   color: #1a150e;
28   transition: 0.3s linear;
29 }
30
31 .darker {
32   background-image: linear-gradient(100deg, #001214, #001f29);
33   color: white;
34   transition: 0.3s linear;
```

Рисунок 3.16 – файл CSS коду

JS: Основні функції для роботи додатку.

```
1 // Selectors
2
3 const todoInput = document.querySelector('.todo-input');
4 const todoBtn = document.querySelector('.todo-btn');
5 const todoList = document.querySelector('.todo-list');
6 const standardTheme = document.querySelector('.standard-theme');
7 const lightTheme = document.querySelector('.light-theme');
8 const darkerTheme = document.querySelector('.darker-theme');
9
10
11 // Event Listeners
12
13 todoBtn.addEventListener('click', addToDo);
14 todoList.addEventListener('click', deletecheck);
15 document.addEventListener("DOMContentLoaded", getTodos);
16 standardTheme.addEventListener('click', () => changeTheme('standard'));
17 lightTheme.addEventListener('click', () => changeTheme('light'));
18 darkerTheme.addEventListener('click', () => changeTheme('darker'));
19
20 // Check if one theme has been set previously and apply it (or std theme if not found):
21 let savedTheme = localStorage.getItem('savedTheme');
22 savedTheme === null ?
23   changeTheme('standard')
24   : changeTheme(localStorage.getItem('savedTheme'));
25
26 // Functions;
27 function addToDo(event) {
28   // Prevents form from submitting / Prevents form from reloading;
29   event.preventDefault();
30 }
```

Рисунок 3.17 – Файл JS коду

```
34
35 // Create LI
36 const newToDo = document.createElement('li');
37 if (todoInput.value === '') {
38     alert("You must write something!");
39 }
40 else {
41     // newToDo.innerText = "hey";
42     newToDo.innerText = todoInput.value;
43     newToDo.classList.add('todo-item');
44     todoDiv.appendChild(newToDo);
45
46     // Adding to local storage;
47     savelocal(todoInput.value);
48
49     // check btn;
50     const checked = document.createElement('button');
51     checked.innerHTML = '<i class="fas fa-check"></i>';
52     checked.classList.add('check-btn', `${savedTheme}-button`);
53     todoDiv.appendChild(checked);
54     // delete btn;
55     const deleted = document.createElement('button');
56     deleted.innerHTML = '<i class="fas fa-trash"></i>';
57     deleted.classList.add('delete-btn', `${savedTheme}-button`);
58     todoDiv.appendChild(deleted);
59
60     // Append to list;
61     todoList.appendChild(todoDiv);
62
63     // Clearing the input;
64     todoInput.value = '';
65 }
66
67 }
68
```

Рисунок 3.18 – Файл JS коду

```
69
70 function deletecheck(event){
71
72     // console.log(event.target);
73     const item = event.target;
74
75     // delete
76     if(item.classList[0] === 'delete-btn')
77     {
78         // item.parentElement.remove();
79         // animation
80         item.parentElement.classList.add("fall");
81
82         //removing local todos;
83         removeLocalTodos(item.parentElement);
84
85         item.parentElement.addEventListener('transitionend', function(){
86             item.parentElement.remove();
87         })
88     }
89
90     // check
91     if(item.classList[0] === 'check-btn')
92     {
93         item.parentElement.classList.toggle("completed");
94     }
95
96
97 }
98
```

Рисунок 3.19 – Файл JS коду

```

117 function getTodos() {
118     //Check: if item/s are there;
119     let todos;
120     if(localStorage.getItem('todos') === null) {
121         todos = [];
122     }
123     else {
124         todos = JSON.parse(localStorage.getItem('todos'));
125     }
126
127     todos.forEach(function(todo) {
128         // toDo DIV;
129         const toDoDiv = document.createElement("div");
130         toDoDiv.classList.add("todo", `${savedTheme}-todo`);
131
132         // Create LI
133         const newToDo = document.createElement('li');
134
135         newToDo.innerText = todo;
136         newToDo.classList.add('todo-item');
137         toDoDiv.appendChild(newToDo);
138
139         // check btn;
140         const checked = document.createElement('button');
141         checked.innerHTML = '<i class="fas fa-check"></i>';
142         checked.classList.add("check-btn", `${savedTheme}-button`);
143         toDoDiv.appendChild(checked);
144         // delete btn;
145         const deleted = document.createElement('button');
146         deleted.innerHTML = '<i class="fas fa-trash"></i>';
147         deleted.classList.add("delete-btn", `${savedTheme}-button`);
148         toDoDiv.appendChild(deleted);
149
150         // Append to list;

```

Рисунок 3.20 – Файл JS коду


```

156 function removeLocalTodos(todo){
157     //Check: if item/s are there;
158     let todos;
159     if(localStorage.getItem('todos') === null) {
160         todos = [];
161     }
162     else {
163         todos = JSON.parse(localStorage.getItem('todos'));
164     }
165
166     const todoIndex = todos.indexOf(todo.children[0].innerText);
167     // console.log(todoIndex);
168     todos.splice(todoIndex, 1);
169     // console.log(todos);
170     localStorage.setItem('todos', JSON.stringify(todos));
171 }
172
173 // Change theme function:
174 function changeTheme(color) {
175     localStorage.setItem('savedTheme', color);
176     savedTheme = localStorage.getItem('savedTheme');
177
178     document.body.className = color;
179     // Change blinking cursor for darker theme:
180     color === 'darker' ?
181         document.getElementById('title').classList.add('darker-title')
182         : document.getElementById('title').classList.remove('darker-title');
183
184     document.querySelector('input').className = `${color}-input`;
185     // Change todo color without changing their status (completed or not):
186     document.querySelectorAll('.todo').forEach(todo => {
187         Array.from(todo.classList).some(item => item === 'completed') ?
188             todo.className = `todo ${color}-todo completed`
189             : todo.className = `todo ${color}-todo`;

```

Рисунок 3.21 – Файл JS коду

Висновок до розділу 3

В даному розділі було показано всі технології та інструменти які були застосовані для створення додатку. Саме за сучасних технологій та тенденцій ІТ світу ми можемо створювати легкі і ефективні додатки.

ВИСНОВКИ

В сучасному світі, де кожного дня у людини є дуже великий перелік задач, важко працювати без спеціального програмного забезпечення, яке дозволяє слідкувати за виконанням цих задач та пам'ятати, що треба зробити кожного дня. Існує велика кількість готових рішень, які можна використовувати в своїй роботі. Кожен з цих додатків має як свої переваги, так і недоліки.

В результаті виконаної роботи було розроблено додаток, який містить всі необхідні функції для роботи з задачами. Було створено сучасний та практичний дизайн для роботи користувача. Кожен користувач може налаштувати додаток для своїх потреб.

В роботі детально описано процес використання технології devops. Було створено невідказну та кросплатформену архітектуру. Додаток пройшов процес тестування в різних умовах. На етапі тестування були помилки, які виправлені. Були використанні такі інструменти та технології: AWS, Jenkins, python, css, html, js, github.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Trello - [Електронний ресурс] – Режим доступу до ресурсу:
<https://trello.com/ru>
2. Todoist – [Електронний ресурс] – Режим доступу до ресурсу:
<https://todoist.com/ru>
3. Python - [Електронний ресурс] – Режим доступу до ресурсу:
4. <https://www.python.org/>
5. Jenkins - [Електронний ресурс] – Режим доступу до ресурсу:
6. <https://www.jenkins.io/>
7. AWS - [Електронний ресурс] – Режим доступу до ресурсу:
8. <https://aws.amazon.com/>
9. AWS ec2 - [Електронний ресурс] – Режим доступу до ресурсу:
10. <https://aws.amazon.com/ru/ec2/features/>
11. AWS vpc - [Електронний ресурс] – Режим доступу до ресурсу:
12. <https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html>
13. Jenkins pipeline – [Електронний ресурс] – Режим доступу до ресурсу:
14. <https://www.jenkins.io/doc/book/pipeline/>
15. HTML - [Електронний ресурс] – Режим доступу до ресурсу:
<https://ru.wikipedia.org/wiki/HTML>
16. CSS - [Електронний ресурс] – Режим доступу до ресурсу:
17. <https://ru.wikipedia.org/wiki/CSS>
18. JavaScript - [Електронний ресурс] – Режим доступу до ресурсу:
19. <https://learn.javascript.ru/>
20. GitHub - [Електронний ресурс] – Режим доступу до ресурсу:
21. <https://github.com/>
22. PyCharm - [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.jetbrains.com/ru-ru/pycharm/>
23. WebStorm - [Електронний ресурс] – Режим доступу до ресурсу:
24. <https://www.jetbrains.com/ru-ru/webstorm/>

25. DevOps – [Електронний ресурс] – Режим доступу до ресурсу:
26. Django – [Електронний ресурс] – Режим доступу до ресурсу:
27. <https://www.djangoproject.com/>
28. Trello wiki – [Електронний ресурс] – Режим доступу до ресурсу:
29. <https://ru.wikipedia.org/wiki/Trello>
30. AWS wiki – [Електронний ресурс] – Режим доступу до ресурсу:
31. https://ru.wikipedia.org/wiki/Amazon_Web_Services
32. Jenkins wiki – [Електронний ресурс] – Режим доступу до ресурсу:
33. [https://ru.wikipedia.org/wiki/Jenkins_\(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5\)](https://ru.wikipedia.org/wiki/Jenkins_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5))