



**Тема: «Модифікація застосунку доповненої реальності з використанням
Git-системи контролю версій»**

Викладач: Олександр А. Блажко,
доцент кафедри ІС Одеської політехніки, blazhko@ieee.org

Мета роботи: придбання навичок у модифікації програмного *JavaScript*-коду простих мульти-маркерних *WebAR*-застосунків з використанням *Git*-системи контролю версій.

План

1 Теоретичні відомості	1
1.1 Особливості редагування файлів на <i>GitHub</i>	1
1.2 Початок роботи з <i>Git</i> -клієнтом системи контролю версій	3
1.3 Огляд команд <i>Bash</i> -оболонки	4
1.4 Налаштування <i>Git</i> -клієнту	5
1.5 Безпечна робота з <i>Git</i> -репозиторієм	6
1.6 Основи роботи з локальним <i>Git</i> -репозиторієм	12
1.7 Створення <i>WebAR</i> -застосунку з використанням <i>Pattern</i> -маркерів	15
2 Завдання для виконання	20
Контрольні запитання	23

1 Теоретичні відомості

1.1 Особливості редагування файлів на веб-сервісі *GitHub*

Веб-сервіс *GitHub* традиційно дозволяє створювати файли, перш за все орієнтовані на Веб-мову розмітки (*markup*), наприклад, *HTML* – набір анотацій до тексту, які є інструкціями стосовно структури тексту чи його відображення на екрані. Але велика кількість таких анотацій, складно запам'ятовуваних людиною, призвела до створення полегшених мов розмітки даних, наприклад, *Markdown*, яку створено з ухилом на легкість у читанні та зручність у публікації з подальшим перетворенням її на звичайний *HTML*-формат без необхідності пам'ятати, записувати *HTML*-теги у звичайному текстовому редакторі.

Враховуючи вищевикладене, веб-сервіс *GitHub* дозволяє створювати текстові файли з розширенням *MD* (*MarkDown*), в якому можуть міститися спеціальні символи, які оброблюються мовою розмітки *MarkDown*, вказуючи на певне форматування частин тексту, наприклад, жирний шрифт, курсив, форматування таблиці та багато іншого.

За замовченням при створенні репозиторію в ньому автоматично створюється файл *README.md*, який найчастіше використовується для докладнішого опису проекту репозиторію.

Наведемо декілька прикладів *Markdown*-форматування:

- заголовок 1-го рівня – один символ # (решітка) з символом пробілу зліва від рядка тексту, наприклад, # Заголовок1;
- заголовок 2-го рівня – два символи # (решітка) з символом пробілу зліва від рядка тексту;
- заголовок будь-якого рівня (1-6) – підводна кількість символу # (решітка) з символом пробілу зліва від рядка тексту;
- курсивний шрифт – один символ «зірочка» або «підкреслення» без символу пробілу ліворуч та праворуч від рядка тексту, наприклад, *Курсивний шрифт*
- жирний шрифт – два символи «зірочка» або «підкреслення» без символу пробілу ліворуч та праворуч від рядка тексту, наприклад, **Жирний шрифт**
- перелічення – символ + (плюс) або - (мінус) перед кожним пунктом перелічення рядків тексту, наприклад, + Пункт1 + Пункт2 + Пункт 3
- абзац (параграф) – один та більше порожніх рядків;
- рисунок – ![Альтернативний текст](URL-шлях-до-файлу-зображення), де альтернативний текст можна пропустити, а URL-шлях взяти як копію шляху розміщення файлу зображення безпосередньо у самому репозиторію;
- таблиця – символ | для розділення стовпчиків таблиці, при цьому шапка таблиці відокремлюється від основних рядків таблиці спеціальним рядком з символами |- як роздільниками стовпчиків таблиці, наприклад:

Стовпчик1	Стовпчик2
-----	-----
Значення1	Значення2

Докладніше про спеціальні символи форматування тексту *MD*-формату можна дізнатися з документу [1] зі списку літератури.

На рисунку 1 наведено приклади *MD*-форматування

```

1  ## WebAR-буклет «Перший проєкт механічного комп'ютера»
2  За малюнком з книги Сідні Падуа
3  *«Неймовірні пригоди Лавлейс та Бєббіджа.
4  Майже правдива історія першого комп'ютера».*
5  Буклет розробено як наочний приклад виконання лабораторної роботи
6  "Розробка маркерних WebAR-застосунків з використанням простого WebAR-конструктору
7
8  Містить 6 маркерів.
9
10 Посилання на AR-інформаційну основу (підкладку) буклету - https://github.com/infosystemdepartment/BC\_Computer/blob/main/BC\_Computer\_BookLet.pdf
11 Створено з використанням маркерного WebAR-конструктора - https://ar.gamehub.od.ua/
12
13 **Тренери:**
14 - Олександр Блажко, доцент кафедри інформаційних систем Національного університету
15 - Олег Савенюк, бакалаврант кафедри інформаційних систем, oleg.saveniuk@gmail.com

```

WebAR-буклет «Перший проєкт механічного комп'ютера»

За малюнком з книги Сідні Падуа «Неймовірні пригоди Лавлейс та Бєббіджа. Майже правдива історія першого комп'ютера». Буклет розробено як наочний приклад виконання лабораторної роботи "Розробка маркерних WebAR-застосунків з використанням простого WebAR-конструктора на GitHub-репозиторії"

Містить 6 маркерів.

Посилання на AR-інформаційну основу (підкладку) буклету - https://github.com/infosystemdepartment/BC_Computer/blob/main/BC_Computer_BookLet.pdf
WebAR-конструктора - <https://ar.gamehub.od.ua/>

Тренери:

- Олександр Блажко, доцент кафедри інформаційних систем Національного університету
- Олег Савенюк, бакалаврант кафедри інформаційних систем, oleg.saveniuk@gmail.com

Рис. 1 – Приклади MD-форматування

1.2 Початок роботи з *Git*-клієнтом системи контролю версій

З 2005 року свого народження *GIT* як програмна система контролю версій файлів, перш за все, підтримує розробників ядра ОС *Linux*. Але інші ІТ-команди також стали її використовувати для підтримки своїх проєктів зі створення електронних ресурсів.

Докладніше про *GIT* можна дізнатися з документу [2] зі списку літератури.

За посиланням <https://git-scm.com/downloads> можна отримати інсталяційний пакет з програмним забезпеченням *Git*-клієнту.

Програмний пакет *Git*-клієнта містить три базові програми:

- *Git Bash* – керування *Git*-репозиторієм через командний рядок з вбудованою оболонкою *Bash* команд ОС *Linux*;
- *Git CMD* - керування *Git*-репозиторієм через командний рядок ОС *Windows*;
- *Git GUI* - керування *Git*-репозиторієм через графічний інтерфейс.

1.3 Огляд команд *Bash*-оболонки

Нижче наведено декілька базових команд огляду файлової системи:

- *pwd* (*print working directory* - надрукувати робочий каталог) - виводить повний шлях від кореневого каталогу до поточного робочого каталогу;
- *whoami* - виводить ім'я користувача, який запустив оболонку командного рядку;
- *ls* - виводить вміст поточного каталогу файлової системи, або додатково вказаного каталогу;
 - *cd назва_каталогу* - змінює поточний каталог на новий каталог за назвою (для переходу на батьківський каталог використовуються символи .., наприклад, *cd ..*);
 - *less назва_файлу* – перегляд вмісту файлу за вказаною назвою (для завершення перегляду файлу використовується клавіша *q*);
 - *rm назва_файлу* – видалення файлу;
 - *rm назва_каталогу -rf* – видалення каталогу з файлами.

Більшість команд *Bash*-оболонки пропонують розширений функціонал через так звані прапорці або ключі – аргументи, що керують роботою команди або вказують додаткові значення та позначаються через дефіс для уникнення двозначності.

В таблиці 1 наведено приклади таких ключів для команди *ls*

Таблиця 1 - Параметри команди *ls*

Ключ	Опис ключа
<i>-a</i>	відображаються файли, назви яких починаються із крапки - « приховані » файли
<i>-c</i>	відображається час останньої зміни файлу
<i>-l</i>	відображається список файлів у даному форматі
<i>-R</i>	відображається вміст підкаталогів
<i>-S</i>	файли впорядковуються відповідно до розміру
<i>-t</i>	файли впорядковуються у відповідності часу останньої зміни
<i>-u</i>	файли впорядковуються у відповідності часу останнього доступу

Ключі можна використовувати сумісно, наприклад:

ls -lS

Git Bash пропонує наступні редактори текстових файлів для ОС *Linux*:

- *ex* – подробиці на [https://uk.wikipedia.org/wiki/Ex_\(Unix\)](https://uk.wikipedia.org/wiki/Ex_(Unix))
- *vi* – подробиці на <https://uk.wikipedia.org/wiki/Vi>;
- *nano* – подробиці на <https://uk.wikipedia.org/wiki/Nano>

Але також можна проводити редагування всіх файлів, використовуючи текстові редактори, які встановлені в ОС за межами *Git-Bash*.

1.4 Налаштування *Git*-клієнту

1.4.1 Налаштування *Git*-середовища

Налаштування *Git*-клієнту починається з налаштування ідентифікатора користувача, для якого зареєстровано обліковий запис на *GitHub*.

Для цього необхідно виконати наступні команди:

```
git config --global user.name "назва GitHub-користувача"  
git config --global user.email E-mail GitHub-користувача
```

Приклад виконання команд

```
git config --global user.name "oleksandrblazhko2"  
git config --global user.email blazhko@op.edu.ua
```

Для отримання значень вказаних параметрів можна виконати наступну команду:

```
git config user.name  
git config user.email
```

1.4.2 Створення нового *Git*-репозиторію

Для створення нового *Git*-репозиторію на локальному комп'ютері необхідно:

- 1) створити каталог, наприклад, командою *mkdir назва_каталогу*;
- 2) перейти до каталогу, наприклад, командою *cd назва_каталогу*;
- 3) ініціалізувати системний каталог з назвою *.git* через наступну команду:

```
git init
```

Команда створює системний *.git*-каталог, який є прихованим у стилі ОС *Linux*, тому його можна побачити лише через наступну команду:

```
ls -a
```

1.4.3 Створення копії існуючого *GitHub*-репозиторію

В лабораторній роботі №1 під час створення *GitHub*-репозиторію *WebAR*-застосунку на веб-сервісі *GitHub* використовувалась операція відгалуження *Fork*, яка копіювала зміст існуючого *GitHub*-репозиторію в новий *GitHub*-репозиторій.

В *Git*-системі подібна операція копіювання виконується через спеціальну команду клонування – *clone*.

Для отримання *GitHub*-репозиторію на локальний комп'ютер необхідно виконати його клонування через наступну команду:

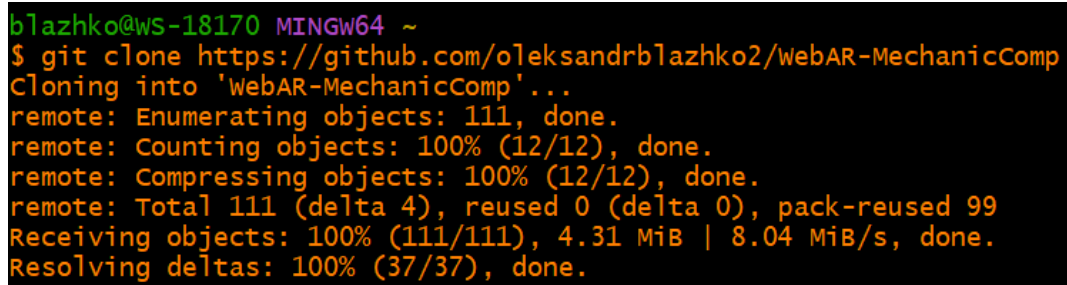
```
git clone <повна адреса GitHub-репозиторію>
```

Найпростіший, швидкий засіб отримати копію – використати *https*-адресу *GitHub*-репозиторію, яку можна взяти як командний рядок з Веб-навігатора.

Приклад виконання команди:

```
git clone https://github.com/oleksandrblazhko2/WebAR-MechanicComp
```

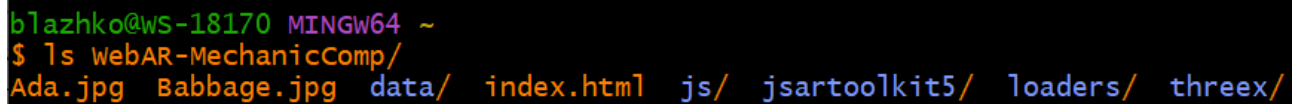
Результат виконання команд представлено на рисунку 2.



```
blazhko@ws-18170 MINGW64 ~
$ git clone https://github.com/oleksandrblazhko2/webAR-MechanicComp
Cloning into 'webAR-MechanicComp'...
remote: Enumerating objects: 111, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 111 (delta 4), reused 0 (delta 0), pack-reused 99
Receiving objects: 100% (111/111), 4.31 MiB | 8.04 MiB/s, done.
Resolving deltas: 100% (37/37), done.
```

Рис. 2 – Фрагмент екрану з результатом виконання команди клонування

Після завершення клонування можна переглянути зміст створеного каталогу з *Git*-репозиторієм, виконавши команду *ls WebAR-MechanicComp*, як показано на рисунку 3.



```
blazhko@ws-18170 MINGW64 ~
$ ls webAR-MechanicComp/
Ada.jpg Babbage.jpg data/ index.html js/ jsartoolkit5/ loaders/ threex/
```

Рис. 3 – Фрагмент екрану зі змістом каталогу з *Git*-репозиторієм

1.5 Безпечна робота з *Git*-репозиторієм

1.5.1 Налаштування режиму безпечного редагування *Git*-репозиторію

Більшість архітектур програмного забезпечення використовує клієнт-серверну взаємодію програмних компонент, в якій виконуються основні дії:

- програма-клієнт надсилає інформаційні запити до програми-серверу;
- програма-сервер оброблює запит від програми-клієнта;
- програма-сервер надсилає відповідь на запит програми-клієнту.

Всі діє забезпечуються обміном мережових пакетів з даними-запитами та даними-відповідями. При цьому канал зв'язку найчастіше є відкритим, незахищеним від прослуховування або втручання зовні (рисунки 4). Це може призвести до небезпечного втручання в роботу системи з боку злоумисників, наприклад:

- порушення конфіденційності даних через прослуховування каналу;
- порушення цілісності даних через перехоплення пакетів та їх несанкціоновану зміну.



Рис. 4 – Відкрита архітектура клієнт-серверної взаємодії

Всі засоби захисту використовують окремі або комбінацію наступних концептуальних сутностей:

- «те, що знаю», наприклад, секретний пароль;
- «те, чим володію», наприклад, банківська картка;
- «те, чим я є», наприклад біометричні показники людини.

Наприклад, коли ми хочемо отримати доступ до *GitHub*-серверу, нам необхідно ввести ім'я облікового запису, а також пароль як секретний рядок, який підтвердить нас як справжніх власників цього облікового запису. Такий спосіб використовує сутність безпеки «що знаю». Але зломисник, прослуховуючи відкритий канал, може перехопити пароль і в подальшому від нашого імені працювати з *GitHub*-репозиторіями серверу.

1.5.2 Налаштування режиму безпечного редагування *Git*-репозиторію

Для захисту відкритого каналу використовуються безпечні (*Secure*) мережеві протоколи обміну мережевими пакетами між програмою-клієнтом та програмою-сервером.

Сьогодні веб-шлях до більшості веб-сторінок починається назви мережевого протоколу *HTTPS* (*HyperText Transfer Protocol Secure*), який забезпечує шифрований (таємний) обмін між програмою-клієнтом та програмою-сервером (*Web-сервером*) завдяки крокам криптографічного протоколу:

- програма-сервер використовує програмну систему з відкритим ключем та закритим ключем (секретним);
- коли програма-клієнт надсилає перший запит до програми-серверу, сервер перш за все, надсилає програмі-клієнту свій відкритий ключ;
- програма-клієнт використовує відкритий ключ для створення додаткового ключа шифрування та цим ключем в подальшому шифруються всі пакети, починаючи з пакету з секретним паролем програми-клієнта.

Але сьогодні секретність паролю часто підпадає під сумнів:

- пароль можна створити дуже простим для підбору зловмисником;
- пароль можна комусь випадково показати;
- пароль хтось вкраде.

Тому в сучасних системах використовують засоби, які використовують щонайменше дві з вказаних сутностей безпеки, створюючи так звану двох фазову перевірку справжності програми-клієнта або двох факторну аутентифікацію (*two-factor authentication, 2FA*), яка передає пароль до програми-сервера.

Одним із засобів забезпечення *2FA* є використання *SSH* (*Secure Shell*, «безпечна оболонка») – мережевого протоколу встановлення безпечного з'єднання між клієнтом та сервером через підтвердження справжності програми-клієнта з використанням сутності безпеки «те, що маю». Протокол також використовує два ключі: відкритий та закритий.

Але для безпечної роботи з *Git* використовується відкритий ключ програми-клієнта, який необхідно розмістити для постійного зберігання на *GitHub*-сервері, як показано на рисунку 5.

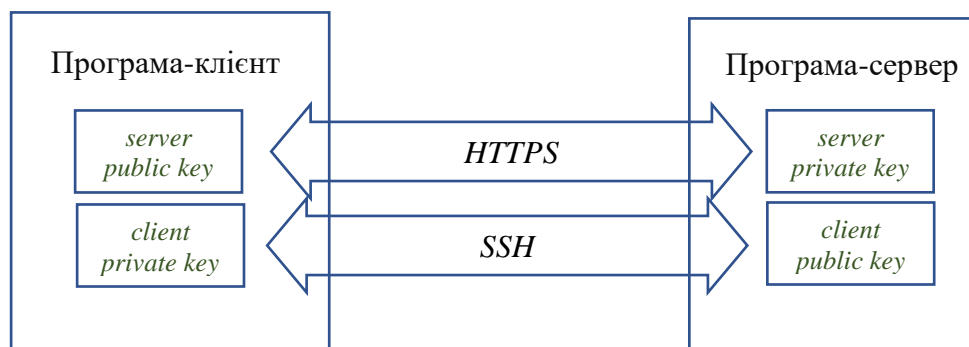


Рис. 5 – Особливості використання відкритих ключів програми-клієнта та програми-сервера задля безпечної роботи з *GitHub*-репозиторіями.

Для створення ключів *Git* пропонує команду *ssh-keygen*, як показано на рисунку 6:

- 1) пропонується вибрати назву файлу з ключами або визначити його за замовчуванням;
- 2) пропонується створити каталог з ключами або визначити за замовчуванням;
- 3) пропонується вказати пароль для безпечного зберігання файлу з секретним ключем через його шифрування, або відмовитись, якщо вказати пустий рядок паролю.


```

blazhko@WS-4853 MINGW64 ~ (master)
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/blazhko/.ssh/id_rsa):
Created directory '/c/Users/blazhko/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/blazhko/.ssh/id_rsa
Your public key has been saved in /c/Users/blazhko/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:Td8qiQHlJPPsBnCrQonaBYK7K1Pwe2iPTQvVozLofd8 blazhko@WS-4853
The key's randomart image is:
+---[RSA 3072]-----+
| o o . o |
| o..o + = |
|. + + + . |
|= . o = . o . . |
|o+o o o S . . . |
|..oo . . o . . |
| ooo+ o . o . |
|+.+*.+. . . |
|.ooo=. . . E |
+-----[SHA256]-----+
blazhko@WS-4853 MINGW64 ~ (master)

```

Рис. 6 – Фрагмент екрану з виконання команди *ssh-keygen*

Команда *ssh-keygen* створює два ключі: відкритий (*public*) ключ та закритий або секретний (*private*) ключ.

Зміст створених ключів можна переглянути в каталозі, як показано на рисунку 6.

Для доступу до відкритого ключа необхідно відкрити файл *id_rsa.pub*, наприклад використовуючи команду *less*, як показано на рисунку 7.

```

blazhko@WS-4853 MINGW64 ~ (master)
$ less /c/Users/blazhko/.ssh/id_rsa

blazhko@WS-4853 MINGW64 ~ (master)
$ less /c/Users/blazhko/.ssh/id_rsa.pub

```

Рис. 7 – Фрагмент екрану з файлами створених ключів

Приклад змісту ключа наведено на рисунку 8. Для завершення перегляду файлу можна натиснути клавішу *q*.

```

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGCmyrcLFNNkp7BeDRoqXw147gUVCxbXiV
w6MUDRrT0gwvdugM904hN+OZsGQKgP+PLROh1JlBtITGrjTa5tVxFJRcd2w++wvdVa4wVGgL
490Clwly114dziee516of+XmVA8IS7LgHJp3FHAdT1iBPiTL4pDPuzsfzasqZUZJq8XwSnKa
YhhQcv0EKrw3gSP1yErLoz9ukvWEjzgcA0pucDr/J7yzfYJJxC6f/1/ypb1cymN7U1m23u2m
revGm3Lkz+8KHxhA6e1Ru0QmQJCP0SVY0qjYV0xYkJjD3soDgrgwI2JAN4L+bx0/09/zZTEa
pZRqTydFx+HUJXG+DIw8KMCi75GGLZ5u1vEquC97XaYHjtFk37Ody8cRFYf9rq1tLv86RcJo
mSPziCZGyCYDuBmRfEuARat82ko8kLR+86+tbjjuu1lyT+SgCBpJ8QuEP7otBd8zpdhEdwc=
@WS-18170
/c/Users/blazhko/.ssh/id_rsa.pub (END)

```

Рис. 8 – Фрагмент екрану зі змістом файлу відкритого ключа

Зміст відкритого ключа необхідно скопіювати та розмістити на сервері з веб-сервісом *GitHub*. Швидко отримати доступ до сторінки з розміщення відкритого ключа можна через:

- посилання на сторінку <https://github.com/settings/keys>
- або через кроки, показані на рисунку 9.

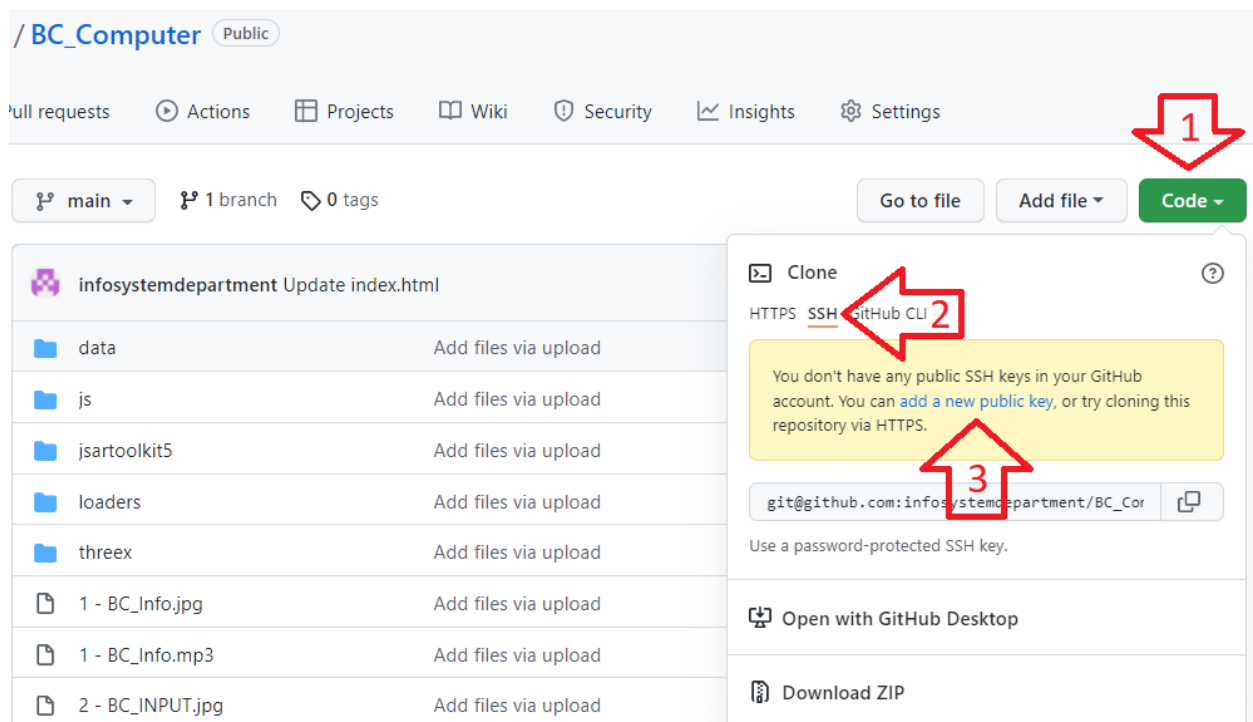


Рис. 9 – Фрагмент екрану з послідовністю дій зі старту процесу розміщення відкритого ключу

Після цього буде виконано перехід у розділ «*SSH and GPG keys*», як показано на рисунку 10.

Необхідно скопіювати увесь зміст відкритого ключа, починаючи зі слова *ssh-rsa*, з файлу *id_rsa.pub* на вашому локальному комп'ютері, наприклад, через класичне виділення «мишею» та наступною комбінацією клавіш *Ctrl+C*. Для внесення змісту ключа можна використати класичну комбінацію клавіш *Ctrl+V*.

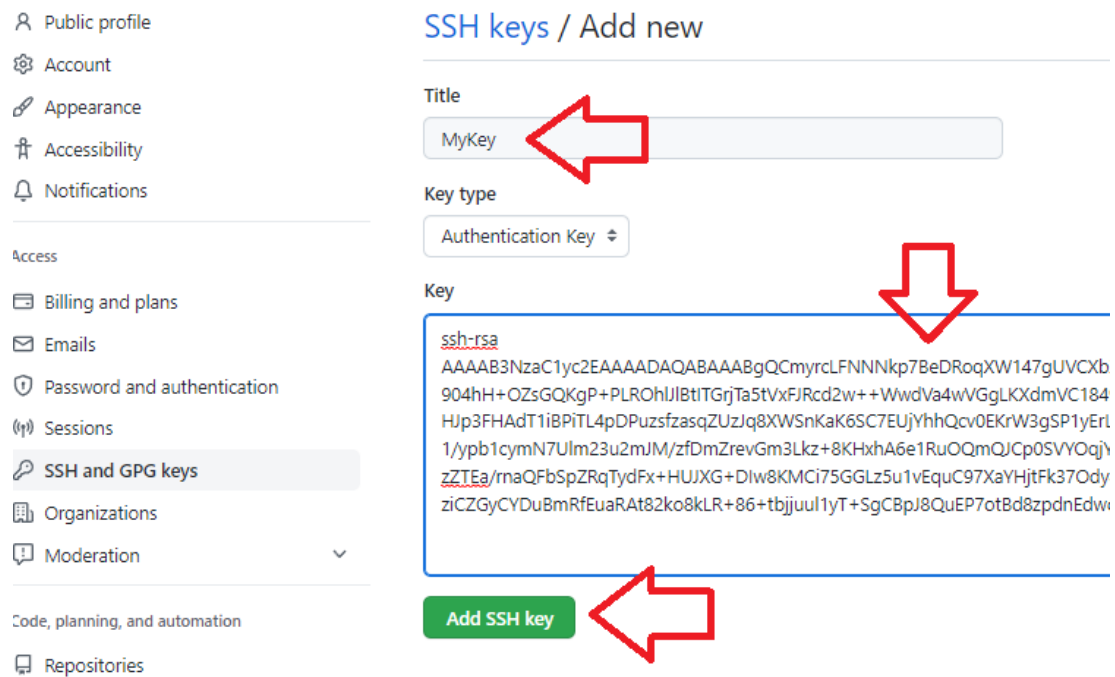


Рис. 10 – Фрагмент екрану із кроками по зберіганню змісту відкритого ключа.

Після реєстрації *SSH*-ключів можна виконувати безпечне редагування файлів у локальному репозиторію та пересилати змінені файли до *GitHub*-репозиторію без необхідності вказувати пароль користувача на веб-сервісі *GitHub*. Якщо раніше ви вже клонували репозиторій, його можна попередньо видалити командою *rm каталог -rf*, наприклад:

```
rm WebAR-MechanicComp -rf
```

Для клонування *GitHub*-репозиторію з подальшим його безпечним редагуванням необхідно виконати команду:

```
git clone git@github:користувач/репозиторій.git
```

Наприклад:

```
git clone git@github.com:oleksandrblazhko2/WebAR-MechanicComp.git
```

На рисунку 11 наведено фрагмент екрану з процесом безпечного клонування.

```
blazhko@ws-18170 MINGW64 ~
$ git clone git@github.com:oleksandrblazhko2/WebAR-MechanicComp.git
Cloning into 'WebAR-MechanicComp'...
The authenticity of host 'github.com (140.82.121.4)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqu.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
remote: Enumerating objects: 172, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 172 (delta 12), reused 22 (delta 9), pack-reused 145
Receiving objects: 100% (172/172), 4.33 MiB | 3.08 MiB/s, done.
Resolving deltas: 100% (64/64), done.
```

Рис. 11 – Фрагмент екрану з процесом безпечного клонування

Порівнюючи процес небезпечного клонування, який показано на рисунку 2, з процесом безпечного клонування з рисунку 11, можна побачити, що на початку процесу безпечного клонування з'являється повідомлення:

«This key is not known by any other names Are you sure you want to continue connecting (yes/no/[fingerprint])?»

Це повідомлення вказує, що *GitHub*-сервер надсилає *Git*-клієнту свій відкритий ключ, про який клієнт ще нічого не знає та побоюється, що це може бути ключ від шахрайського серверу, тому запитує у користувача про наступні дії. Але будемо вважати, що *GitHub*-сервер є надійним сервером, тому на запитання вказуємо «yes». Перше успішне виконання клонування додасть у файл *.ssh/known_hosts* рядок з адресою серверу та його відкритий ключ, тому під час наступного з'єднання із сервером такого запитання вже не буде.

Але якщо виникне помилка *«fatal: Could not read from remote repository»* можна примусово записати відкритий ключ до файлу через команду:

```
ssh-keyscan -t rsa github.com >> .ssh/known_hosts
```

Після клонування репозиторію можна перейти до каталогу репозиторія через команду *cd каталог*, наприклад:

```
cd WebAR-MechanicComp
```

1.6 Основи роботи з локальним *Git*-репозиторієм

Git-репозиторій – це спеціальний каталог, який містить набір версій (знімків) звичайного каталогу файлів. В процесі редагування файлів у *Git*-репозиторії, як у звичайному каталозі, файл може бути додано, змінено або видалено. При цьому кожен файл може знаходитися в одному з двох станів:

- контрольований (*tracked*) файл – файл, який був в останньому знімку репозиторія, при цьому файл може бути не зміненим, зміненим або індексованим, коли про цей файл *Git* знає та може після його зміни передати ці зміни до *GitHub*-серверу;
- неконтрольований (*untracked*) файл - файл, який відсутній в останньому знімку і про нього *Git* не знає, тому не буде передавати його зміни до *GitHub*-серверу.

Звичайними засобами ОС всі оновлення файлів створюють неконтрольовані файли для *Git*-системи.

Після клонування репозиторію усі файли репозиторію є контрольованими.

Незалежно від операцій, виконаних над файлом, після їх завершення необхідно виконати наступні *Git*-дії:

- 1) почати контролювати файл у *Git*-репозиторії (zareestruvati fayli y *Git*-repozitorii) через наступну команду:

```
git add файл
```

примітка: для забезпечення швидкої реєстрації множини файлів замість назви файлу можна вказати символ крапка, наприклад, `git add .`

2) зафіксувати зміни у файлу, щоб вони з'явилися в останньому знімку, через команду `commit`:

```
git commit -m 'короткий опис змісту змін'
```

3) відправити останній знімок на віддалений *Git*-репозиторій через наступну команду:

```
git push
```

Перегляд статусу файлів, контрольованих або не контрольованих, можна виконати через наступну команду:

```
git status
```

Видалення файлів (зняття з реєстрації) виконується через наступну команду:

```
git rm файл
```

Отримання нових знімків з віддаленого *Git*-репозиторію виконується через наступну команду:

```
git pull
```

Створення знімків (версій) файлів каталогу визначає дуже важливу властивість *Git*-репозиторію – створення гілок (*branch*), до яких можна паралельно звертатися у разі необхідності. Кожна гілка може містити власні версії файлів, які було створено після різних команд `commit` в історичному часі.

Створення нової гілки виконується через наступну команду:

```
git branch назва
```

Наприклад:

```
git branch BookLet_with_pattern_marker
```

Щоб переключитися на іншу гілку необхідно виконати наступну команду:

```
git checkout назва
```

Після внесення змін у *Git*-гілку її знімок можна відправити на віддалений *Git*-репозиторій, наприклад, *GitHub*-репозиторій, з якого була створена копія, через наступну команду:

```
git push віддалена_гілка локальна_гілка
```

Якщо *Git*-репозиторій було створено через клонування, тоді можна вказати шаблонну назву віддаленої гілки як *origin*.

Наприклад, для відправлення поточного знімку гілки `webar_test2_with_README` на гілку *origin* віддаленого *Git*-репозиторію необхідно виконати:

```
git push origin BookLet_with_pattern_marker
```

На віддаленому *Git*-репозиторії автоматично буде створено нову гілку, якщо її там ще не було, інакше ця гілка буде оновлюватися.

Видалення гілки виконується через опцію *-D* наступної команди:

```
git branch -D назва
```

На рисунку 12 показано фрагмент екрану з *GitHub*-репозиторієм після створення нової гілки, коли можна переходити між двома гілками:

- основна, показана на рисунку як *master* (може мати назву *main*);
- нова, показана на рисунку як *BookLet_with_pattern_marker*.

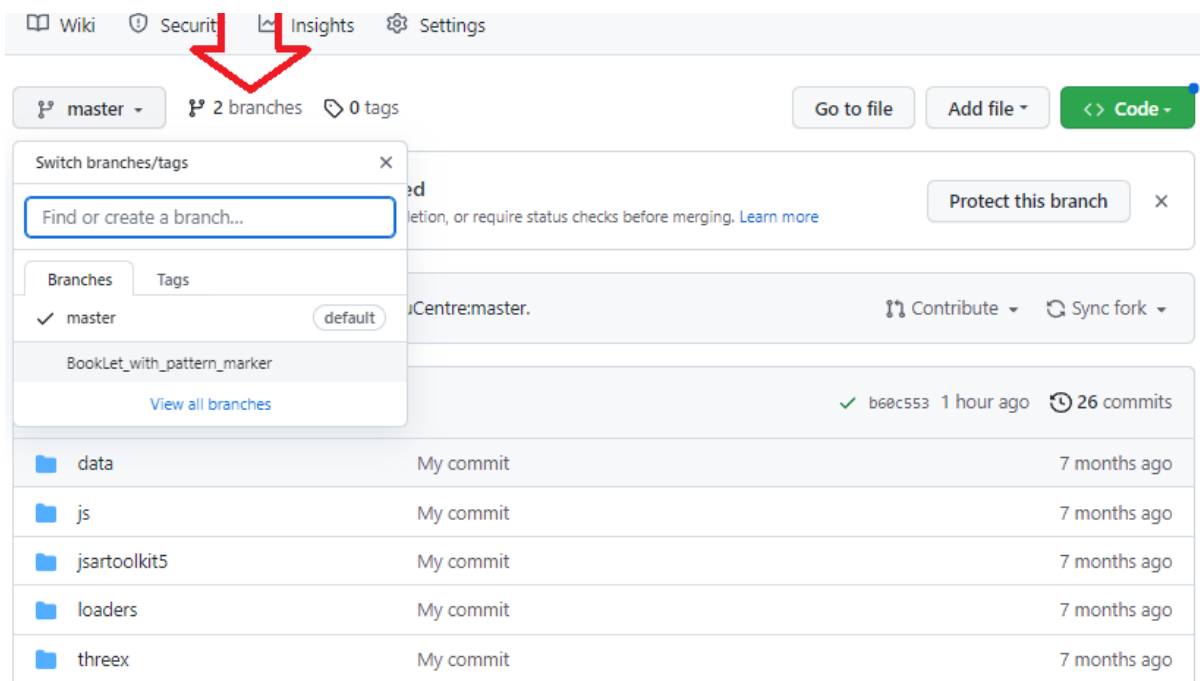


Рис. 12 – Фрагмент екрану з *GitHub*-репозиторієм після створення нової гілки

Нова гілка на *GitHub*-репозиторії буде мати нове посилання -

https://github.com/oleksandrblazhko2/WebAR-MechanicComp/tree/BookLet_with_pattern_marker

Для перегляду гілок на веб-сервісі *GitHub* можна перейти за посиланням *branch*, як показано на рисунку 13.

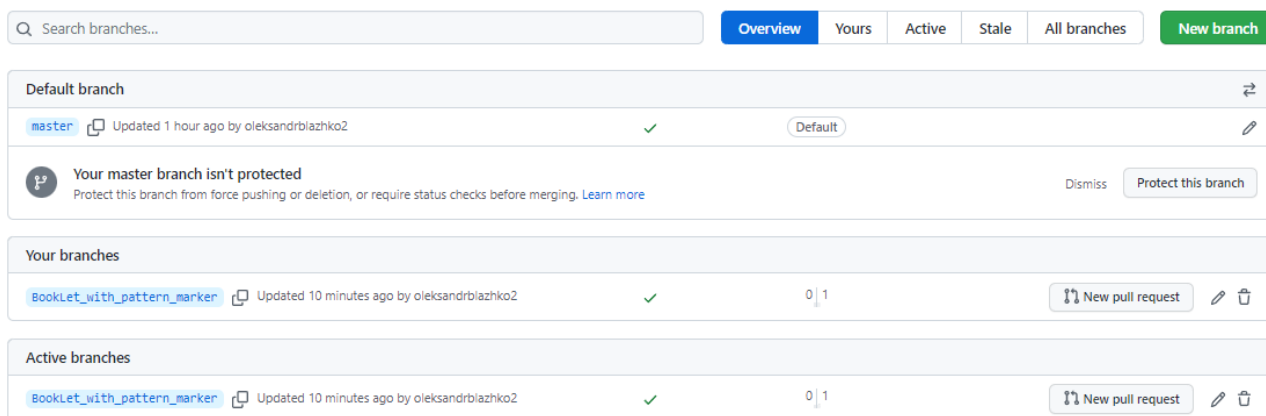


Рис. 13 – Фрагмент екрану з переліком гілок у *GitHub*-репозиторії

1.7 Створення *WebAR*-застосунку з використанням *Pattern*-маркерів

Pattern-маркер (шаблонний маркер) – маркер, який містить просте бітональне (двокольорове) зображення всередині кордону чорної квадратної смужки. Приклади маркерів представлено на рисунку 14.

Перевагою *Pattern*-маркеру над *BarCode*-маркером є можливість відобразити особливий рисунок, який буде пов'язано з об'єктом, та який, можливо, буде більш зрозумілий для користувача. Але така особливість *Pattern*-маркеру вимагає від програмного забезпечення роботи *WebAR*-застосунку виконувати додаткові дії з пошуку та розпізнавання маркерів серед переліку маркерів, які зберігаються у каталозі *WebAR*-застосунку. Але найпростіший маркер, яким є *BarCode*-маркер, розпізнається швидше та надійніше, ніж більш складний *Pattern*-маркер.



Рис. 14 – Приклади шаблонних маркерів

Pattern-маркер вперше було використано програмною бібліотекою *ARToolKit* – однією з найперших та популярних плоских систем *AR*-маркерів завдяки доступному вихідному коду. Подобиці використання *Pattern*-маркерів можна отримати з документу [3] списку літератури.

Для створення *Pattern*-маркеру можна скористатися посиланням - <https://jeromeetienne.github.io/AR.js/three.js/examples/marker-training/examples/generator.html>

Приклад зі створення *Pattern*-маркеру зображено на рисунку 15.

Для створення *Pattern*-маркеру необхідно:

1) завантажити зображення (кнопка «*Upload*») для наступного *Pattern*-маркеру, яке має властивості за наступними рекомендаціями:

- a) чорнобілий колір;
 - b) контрастність переходу між чорними та білими зонами;
 - c) розмір не більше 500x500 пікселів;
- 2) виставити розмір кордону навколо зображення = 0.9;
 - 3) виставити мінімальний розмір *Pattern*-маркеру, наприклад, 150px;
 - 4) отримати текстовий формат *Pattern*-маркеру з розширенням *patt* (кнопка «*Download Marker*»);
 - 5) отримати файл із зображенням *Pattern*-маркеру (кнопка «*Download Image*»).

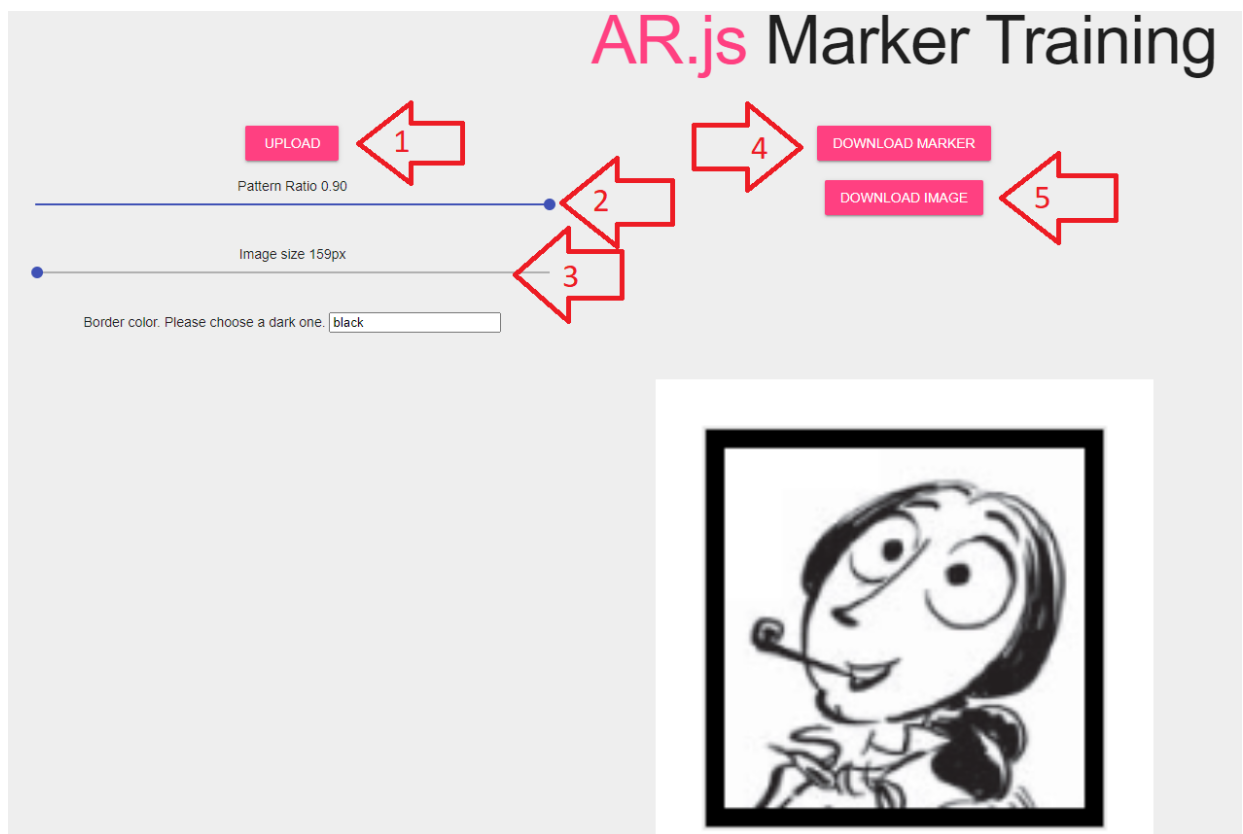


Рис. 15 – Фрагмент екрану зі створення *Pattern*-маркеру

В процесі створення маркеру програма *ARToolKit* перетворює зображення в текстовий файл з розширенням *patt*, в якому:

- пікселі кодуються числами відповідного кольору у вигляді матриці 16x16 пікселів;
- символ відображується у 4х варіантах проєкцій (горизонтально-пряма, горизонтально-зворотна, вертикально-пряма, вертикально-зворотна);
- для кожного варіанта проєкцій зберігається 3 варіанти освітленості окремих граней візерунку маркера.

На рисунку 16 наведено фрагмент змісту текстового формату *Pattern*-маркеру з розширенням *patt* з двома проєкціями. Кожне число визначає значення пікселю в діапазоні від

Як було вказано раніше, якісний маркер повинен мати багато переходів між більш чорними пікселями (число наближається до 0) та білими пікселями (число наближається до 255).

[illegible]

Отриманий *patt*-файл необхідно завантажити у *GitHub*-репозиторій *WebAR*-застосунку, буде використано як математичну модель зображення *Pattern*-маркеру.

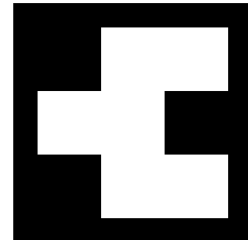


Рис. 17 – Приклад *WebAR*-буклету з розміщеним *Pattern*-маркеру

Якщо *WebAR*-застосунок вже було створено, але необхідно в ньому щось змінити, доцільно вносити зміни безпосередньо у файл *index.html*

Для редагування файлу *index.html* на веб-сервісі *GitHub* необхідно натиснути на його назву та в подальшому натиснути на символ як показано на рисунку 18.

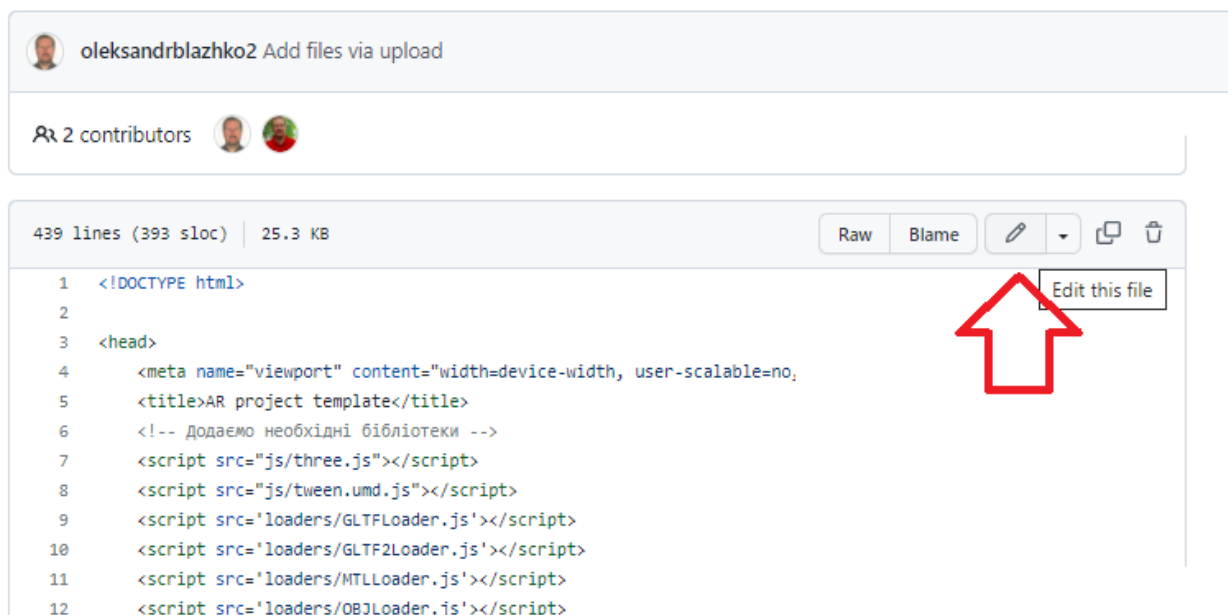


Рис. 18 – Фрагмент екрану з початком редагування файлу *index.html*

На рисунку 19 наведено фрагмент файлу *index.html* з привітальним повідомленням в рядках 30-32, яке можна змінити.

```

25 <div id="access" style="top: 0; left:
26     <div id="text-wrapper" style="top:
27         text-transform: uppercase;
28         user-select: none; pointer
29     ">
30     Press here
31     <br>
32     to enter the experience
33 </div>
34 </div>

```

Рис. 19 – Фрагмент програмного коду файлу *index.html* зі змістом текстового рядка привітального повідомлення *WebAR*-застосунку

З рисунку 20 видно основні дві зміни, які відбулися з файлом *index.html*:

- у рядку № 156 другий елемент масиву отримав значення назви текстового *patt*-файлу для *Pattern*-маркеру;
- у рядку № 160 другий елемент масиву отримав значення -1, як вказано у рекомендаціях.

```

153 // Масив імен patt-файлів з описом Pattern-маркерів.
154 // Якщо замість .patt було добавлено BarCode-маркер,
155 // на його місці у масив додається порожнє значення
156 const patternNames = ["" , "pattern-Ada-Image.patt"];
157 // Масив BarCode-маркерів
158 // Якщо замість BarCode-маркера було додано Pattern-маркер,
159 // на його місце у масив додається значення = -1
160 const patternBarcode = [1 , -1];
161 // Масив типів контенту для кожного маркеру, заповнюється значеннями:
162 // image(зображення), model(3D-модель), video(відео)
163 const modes = ["image" , "image"];
164 // Масив файлів 3d-моделей
165 const modelFiles = ["" , ""];
166 // Масив файлів зображень
167 const imageFiles = ["Babbage.jpg" , "Ada.jpg"];
168 // Масив відео-файлів
169 const videoFiles = ["" , ""];
170 // Масив аудіо-файлів
171 const audioFiles = ["" , ""];
172 // Масив опцій повтору аудіо та відео контенту, за замовченням = false
173 const repeatOptions = ["false" , "false"];

```

Рис. 20 – Фрагмент програмного коду файлу *index.html* зі змістом масивів опису маркерів та *AR*-контенту *WebAR*-застосунку

2 Завдання для виконання

Завдання лабораторної роботи пов'язане з модифікацією *WebAR*-застосунку, створеного у попередній лабораторній роботі.

2.1. Редагування файлів у *GitHub*-репозиторії

2.1.1 Створення файлу *README.md* короткого опису *WebAR*-проєкту

На сайті веб-сервісу *GitHub* в кореновому каталозі вашого репозиторія створити файл *README.md* з коротким описом проєкту *WebAR*-буклету.

Файл повинен містити наступні рядки з форматуванням:

- *WebAR*-буклет «Назва», де назва – назва пристрою (формат – заголовок 1-го рівня)
- Буклет створено як результат виконання лабораторної роботи з дисципліни «Операційні системи» (звичайний формат)
- Команда проєкту: (формат – жирний шрифт)
- студент(ка) ПІБ, група (формат - пункт переліку)
- викладач – Блажко О.А., доцент кафедри інформаційних систем Національного університету «Одеська політехніка» (формат - пункт переліку).

При фіксації змін вказати коментар «*Created by GitHub*».

2.1.2 Редагування файлу *index.html*

На сайті *GitHub* у файлі *index.html* змінити текстовий рядок привітального повідомлення *WebAR*-застосунку на наступний:

Натисніть на екран для отримання доступу до WebAR-буклету «Назва», де назва – назва пристрою. При фіксації змін вказати коментар «Changed by GitHub».

Після редагування необхідно провести тестування змін, перевіряючи появу нового привітального рядку *WebAR*-буклету.

2.2 Налаштування *Git*-клієнту

2.2.1 Встановлення *Git*-клієнту

2.2.1.1 Встановити на локальний комп'ютер *Git*-клієнт за посиланням <https://git-scm.com/downloads>

2.2.1.2 Запустити програму *Git Bash*.

2.2.1.3 Отримати значення повного шляху від кореневого каталогу до поточного робочого каталогу.

2.2.1.4 Створити знімок екрану з результатом попереднього пункту та зберегти його у файлі з назвою *2.2.1.4.jpg*

2.2.2 Налаштування *Git*-користувача

2.2.2.1 Налаштувати *Git*-змінні *global user.name* та *global user.email* у відповідності з вашим обліковим записом на *GitHub*.

2.2.2.2 Отримати на екран значення визначених *Git*-змінних.

2.2.2.3 Створити знімок екрану з результатом попереднього пункту та зберегти його у файлі з назвою *2.2.2.2.jpg*

2.3 Налаштування гілки *Git*-проєкту для подальшого безпечного редагування *GitHub*-репозиторію

2.3.1 Налаштування *SSH*-ключів

2.3.1.1 Створити відкритий та закритий *SSH*-ключі

2.3.1.2 Створити знімок екрану з результатом попереднього пункту та зберегти його у файлі з назвою *2.3.1.1.jpg*

2.3.1.3 Розташувати відкритий ключ у вашому обліковому записі веб-сервісу *GitHub*

2.3.1.4 Створити знімок екрану з результатом попереднього пункту та зберегти його у файлі з назвою *2.3.1.3.jpg*

2.3.2 Безпечне клонування *GitHub*-проєкту

2.3.2.1. Виконати безпечне клонування вашого *GitHub*-репозиторію на локальний комп'ютер.

2.3.2.2 Створити знімок екрану з результатом попереднього пункту та зберегти його у файлі з назвою *2.3.2.1.jpg*

2.3.3 Створення нової гілки проєкту

2.3.3.1 Створити гілку *Git*-репозиторію, в назві якої додатково до назви існуючого репозиторію наприкінці додати фразу *with_pattern_marker*

2.3.3.2 Переключити роботу з репозиторієм на створену гілку.

2.3.3.3 Створити знімок екрану з результатом попередніх двох пунктів та зберегти його у файлі з назвою *2.3.3.1.jpg*

2.3.4 Заміна одного маркеру *BarCode*-типу на маркер *Pattern*-типу

2.3.4.1 Для *BarCode*-маркеру *WebAR*-буклету, який описує роботу пристрою, підготувати будь-яке зображення, враховуючи надані раніше рекомендації.

2.3.4.2 Використовуючи файл із зображенням, створити файли із зображенням *Pattern*-маркеру та текстовий *patt*-файл *Pattern*-маркеру.

2.3.4.3 Переглянути зміст текстового *patt*-файлу *Pattern*-маркеру та зробити висновок щодо якості маркеру.

2.3.4.4 Створити знімок екрану із фрагментом зміст текстового формату *Pattern*-маркеру та зберегти його у файлі з назвою *2.3.4.3.jpg*

2.3.4.5 Розмістити текстовий *patt*-файл *Pattern*-маркеру у каталозі створеної гілки локального *Git*-репозиторію.

2.3.4.6 У створеній гілці локального *Git*-репозиторію внести зміни у код файлу *index.html*, замінивши опис відповідного *BarCode*-маркеру на *Pattern*-маркер. Зафіксувати зміни коментарем «*Created by Local Git*».

2.3.4.7 У файлі *WebAR*-буклету замінити зображення *BarCode*-маркеру на зображення *Pattern*-маркеру та розмістити новий файл з буклетом у *Git*-репозиторію.

2.3.4.8 Оновити зміни локального *Git*-репозиторія на віддаленому *GitHub*-репозиторії.

2.3.4.9 Створити знімок екрану з результатом попереднього пункту та зберегти його у файлі з назвою *2.3.4.8.jpg*

2.3.4.10 Оновити посилання для *GitHub Pages* з урахуванням назви створеної *Git*-гілки, змінивши гілку з *main* (або *master*) на нову гілку, щоб *WebAR*-застосунок було перенацілено на нову гілку.

2.4 Оформлення рішення для оцінювання викладачем

2.4.1 Перейти до вашого студентського *GitHub*-репозиторію, наданого викладачем, та створити в ньому каталог з назвою «*Laboratory-work-2*», при створенні якого одночасно створити файл *README.md*, вказавши роздільником символ /

2.4.2 У файлі *README.md* необхідно вказати посилання на нову гілку *GitHub*-репозиторія з новою версією *WebAR*-застосунку.

2.4.3 Завантажити до каталогу «*Laboratory-work-2*» всі файли зі створеними раніше знімками екранів.

Контрольні запитання

1. Для чого у *GitHub*-репозиторії може використовуватися файл *README.md*
2. Яка перевага *MD*-формату при створенні файлів у порівнянні з *HTML*-форматом?
3. Чим відрізняється програма *Git Bash* від програми *Git CMD*?
4. Яке призначення команд *pwd*, *whoami* при початку роботи з *Git*?
5. Що таке приховані файли і як вони можуть виглядати при перегляді у командному рядку програми *Git Bash*?
6. Для чого потрібні команда *init* при роботі з *Git*-репозиторієм та який каталог вона створює?
7. Що відбувається під час *Git*-клонування?
8. В чому різниця та схожість між поняттями *Fork* та *Clone* ?
9. Які проблеми можуть з'явитися при небезпечній роботі з *GitHub*?
10. Що надає мережевий *HTTPS*-протокол для роботи з *GitHub*?
11. Що надає мережевий *SSH*-протокол для роботи з *GitHub*?
12. Який ключ зберігається на *GitHub*-сервері?
13. Дайте визначення *Git*-репозиторію з точки зору каталогу файлів.
14. Що таке контрольовані файли у *Git*-репозиторії?
15. Що таке неконтрольовані файли у *Git*-репозиторії?
16. Для чого використовуються *Git*-гілки?
17. Яка перевага *Pattern*-маркеру у порівнянні з *BarCode*-маркером?
18. Яка перевага *BarCode*-маркеру у порівнянні із *Pattern*-маркером?
19. Для чого потрібна команда *push* при роботі з *Git*-репозиторієм?
20. Для чого потрібна команда *pull* при роботі з *Git*-репозиторієм?
21. Яку інформацію надає команда *status* при роботі з *Git*-репозиторієм?
22. Для чого потрібні команда *add* при роботі з *Git*-репозиторієм?
23. Для чого потрібні команда *commit* при роботі з *Git*-репозиторієм?

Література

1. GitHub Docs. Basic writing and formatting syntax. URL :
<https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax>
2. Pro Git book. URL: - <https://git-scm.com/book/uk/v2>
3. Олександр Блажко (2023) Лекція «Лабораторна робота №2 - Модифікація WebAR-застосунку з використанням Git-системи контролю версій». URL: -
<https://www.youtube.com/watch?v=br7k3nSZSjg>