Командна розробка ІТ-проектів

Шпортько О. В.
Лотюк Ю. Г.
Богут О. М.

Лабораторний практикум
для студентів денної та заочної форм навчання

Освітньої програми	Інженерія програмного забезпечення Інтернету речей
Рівня вищої освіти	першого (бакалаврського)
За спеціальністю	121 Інженерія програмного забезпечення
Галузі знань	12 Інформаційні технології

Рівне 2023

Міністерство освіти і науки України

Приватний вищий навчальний заклад
«Міжнародний економіко-гуманітарний університет
імені академіка Степана Дем'янчука»

Факультет кібернетики

Кафедра інформаційних систем та обчислювальних методів

Командна розробка IT-проектів

Лабораторний практикум
для студентів денної та заочної форм навчання

Освітньої програми Інженерія програмного забезпечення Інтернету речей

Рівня вищої освіти першого (бакалаврського)

За спеціальністю 121 Інженерія програмного забезпечення

Галузі знань 12 Інформаційні технології

Рівне 2023

УДК 004.422/004.424
ББК 32.973-01
Ш 84

Друкується за рішенням Навчально-методичної комісії Приватного вищого навчального закладу "Міжнародний економіко-гуманітарний університет імені академіка Степана Дем’янчука" (протокол № 10 від 20.06.2023 р.).

Командна розробка ІТ-проектів. Лабораторний практикум для студентів денної та заочної форм навчання освітньої програми «Інженерія програмного забезпечення Інтернету речей» першого (бакалаврського) рівня вищої освіти за спеціальністю 121 Інженерія програмного забезпечення галузі знань 12 Інформаційні технології / уклад. О. В. Шпортько, Ю. Г. Лотюк, О. М. Богут. Рівне: ПВНЗ "МЕГУ ім. акад. С. Дем’янчука", 2023. 65 с.

Укладачі: О. В. Шпортько, кандидат технічних наук, доцент, доцент кафедри інформаційних систем та обчислювальних методів Приватного вищого навчального закладу "Міжнародний економіко-гуманітарний університет імені академіка Степана Дем’янчука";
Ю. Г. Лотюк, кандидат педагогічних наук, доцент, завідувач кафедри інформаційних систем та обчислювальних методів Приватного вищого навчального закладу «Міжнародний економіко-гуманітарний університет імені академіка Степана Дем’янчука»;
О. М. Богут, старший викладач кафедри інформаційних систем та обчислювальних методів ПВНЗ «Міжнародний економіко-гуманітарний університет імені академіка Степана Дем’янчука».

Рецензенти: А. Я. Бомба, доктор технічних наук, професор, професор кафедри комп’ютерних наук та прикладної математики Національного університету водного господарства та природокористування;
П. С. Янчук, кандидат фізико-математичних наук, доцент, професор кафедри інформаційних систем та обчислювальних методів Приватного вищого навчального закладу "Міжнародний економіко-гуманітарний університет імені академіка Степана Дем’янчука".

Відповідальний за випуск Ю. Г. Лотюк, кандидат педагогічних наук, доцент, завідувач кафедри інформаційних систем та обчислювальних методів Приватного вищого навчального закладу "Міжнародний економіко-гуманітарний університет імені академіка Степана Дем’янчука".

Лабораторний практикум розроблений у відповідності з практичною частиною робочої програми дисципліни «Командна розробка ІТ-проектів» для студентів денної та заочної форм навчання освітньої програми «Інженерія програмного забезпечення Інтернету речей» першого (бакалаврського) рівня вищої освіти за спеціальністю 121 Інженерія програмного забезпечення. Містить впорядкований набір завдань і вказівок до виконання лабораторних робіт та самостійного опрацювання матеріалу дисципліни. Основна увага приділяється вивченню основ організації командної розробки та використанню систем контролю версіями програмних проектів в ІТ-індустрії. Призначений для студентів, викладачів та всіх, хто прагне оволодіти підходами командної розробки ІТ-проектів.

Затверджені кафедрою інформаційних систем та обчислювальних методів Приватного вищого навчального закладу "МЕГУ імені академіка Степана Дем’янчука" (протокол № 11 від 13.06.2023 р.).

Схвалені навчально-методичною комісією факультету кібернетики Приватного вищого навчального закладу "МЕГУ імені академіка Степана Дем’янчука" (протокол № 11 від 13.06.2023 р.).

ББК 32.973-01

Зміст

Передмова

На сьогоднішній день важко уявити собі діяльність сучасного суспільства без комп'ютерної техніки, локальних та глобальних мереж і відповідного програмного забезпечення. З комп'ютерами ми стикаємося всюди: у банку та відділі бухгалтерії, в пенсійному фонді та податковій інспекції, у паспортному столі та на залізничному вокзалі, в ЖЕКу та Укртелекомі, в редакціях газет та друкарнях, в рекламних агенціях та конструкторських бюро. Високі темпи накопичення інформації, вимоги мобільності, оперативності поширення та надійності зберігання даних, швидкості та достовірності їх обробки спонукають розвиток апаратних засобів, системних та прикладних оболонок і пакетів. На сьогоднішній день 97 % населення світу користується мобільними телефонами, а кількість комп'ютерів у світі перевищує 1361 мільйонів і продовжує зростати. При цьому приблизно один раз в півтора року подвоюються показники основних технічних характеристик апаратних засобів, один раз в два-три роки змінюються покоління програмного забезпечення і один раз на п'ять-сім років змінюється база стандартів, інтерфейсів та протоколів. Разом з тим, темп кількісного зростання інформаційних систем значно перевищує темп підготовки спеціалістів, здатних ефективно працювати з ними.

В цих умовах кожен випускник ВНЗ з галузі інформаційних технологій, що має використовувати комп’ютерну техніку в своїй професійній діяльності для розробки, експлуатації та супроводження інформаційних систем, повинен не лише володіти елементарними навиками використання програмного забезпечення (на сьогодні це повинна вміти кожна освічена людина), а й розуміти принципи організації командної розробки та вміти застосовувати системи контролю версіями програмних проектів в IT-індустрії. Формуванню саме таких фахівців має сприяти вся дисципліна та даний лабораторний практикум зокрема. Після опанування матеріалу практикуму студент повинен вміти формувати команди та керувати їх роботою в процесі реалізації програмних проектів і створювати та використовувати репозиторії на локальних та мережевих ресурсах за допомогою систем керування версіями.

Даний лабораторний практикум містить впорядкований набір завдань і рекомендацій до виконання лабораторних робіт та самостійного опрацювання матеріалу дисципліни. Виконання завдань кожної лабораторної роботи вимагає використання знань, умінь та навиків,

здобутих під час виконання попередніх робіт, що, з одного боку, дозволяє автоматично повторювати вивчений матеріал, а з іншого – сприяє системності його викладу.

На початку кожної лабораторної роботи, з метою актуалізації знань, отриманих під час лекційних занять, наведено відповідні контрольні запитання. Відлагодження програм та перевірка різних тестових випадків виносяться на самостійне опрацювання. За результатами виконання кожної лабораторної роботи студент оформляє письмовий звіт, що складається з наступних розділів:

- теми та мети лабораторної роботи;
- відповідей на контрольні запитання лабораторної роботи;
- скрінів та коментарів до виконання основних завдань лабораторної роботи;
- висновків стосовно знань, умінь та навичок, здобутих при виконанні лабораторної роботи.

Лабораторна робота № 1

Тема. Вивчення специфікації вимог до програмного продукту. Розбиття реалізації проекту на спринти та завдання. Закріплення завдань за членами команди.

Контрольні запитання

1. Для чого перед реалізацією проекту вивчають специфікацію вимог до програмного продукту?
2. Які основні розділи специфікації вимог?
3. Що таке спринт? Яка середня тривалість спринта?
4. Який основний результат закінчення спринта?
5. За яким принципом обирають завдання для чергового спринта?
6. Від чого залежить кількість членів команди для реалізації IT-проекту?

Хід роботи

1. Вивчіть наведену нижче специфікацію вимог для розробки інформаційної системи продуктового магазину.
2. Виділіть окремі завдання для реалізації цього проекту (не менше десяти). Оцініть складність виконання кожного завдання.
3. Визначте кількість виконавців проекту, якщо орієнтовний час його реалізації має становити три місяці / півроку / рік.
4. Розподіліть виділені вами завдання по спринтах для різних термфінів виконання.
5. Перелік виділених завдань з зазначенням їх складності, розподіл завдань по спринтах та відповіді на контрольні питання здайте як звіт з лабораторної роботи в систему дистанційного навчання Moodle.

Специфікація вимог до програмного продукту (Software Requirements Specification, SRS)

для продуктового магазину «Фуршет»

ВСТУП

Призначення

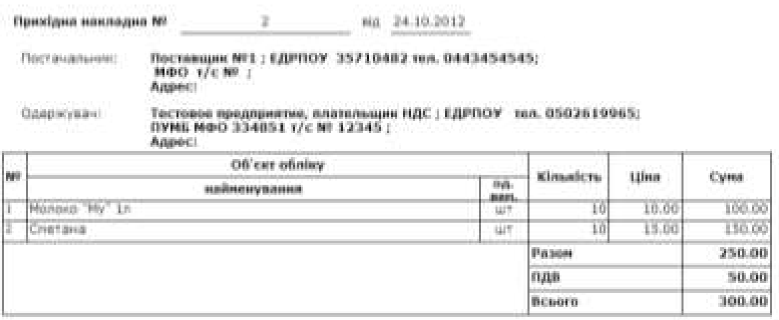
Ця специфікація вимог до програмного продукту описує функціональні та нефункційні вимоги до випуску 1.0 ProdCir. Цей документ призначений для розробників, які будуть реалізовувати і перевіряти роботу програмного продукту. Крім спеціально визначених випадків, всі вказані тут вимоги мають високий пріоритет і закріплені до випуску 1.0.

Обсяг проекту і функції продукту

ProdCir – це програмний продукт, який дасть змогу продуктовому магазину обліковувати постачання та продаж товарів. ProdCir планується, як мережевий програмний продукт, тобто він має забезпечувати одночасну незалежну роботу в мережі багатьох користувачів. В розділі «Функції системи» детально описані всі функції, часткова або повна реалізація яких передбачена в цьому випуску продукту.

Посилання

1. Український продуктовий онлайн-магазин «Банан» – http://banan.cn.ua/.
2. Американський продуктовий онлайн-магазин “Walmart” – http://www.walmart.com/.
3. Німецький продуктовий онлайн-магазин “Samtog Shop” – http://www.samtogshop.com/.
4. Накладна обігу товарів (рис. 1).



Прихідна накладна № 2 від 24.10.2012

Постачальник: Поставщик №1 ; ЕДРПОУ 35710482 тел. 0443454545; МФО т/с № ; Адрес:

Одержувач: Тестовое предприятие, плательщик НДС ; ЕДРПОУ тел. 0502619965; ПУМБ МФО 334851 т/с № 12345 ; Адрес:

№	Об'єкт обліку		Кількість	Ціна	Сума
	найменування	од. вим.			
1	Молоко "Му" 1л	шт	10	10.00	100.00
2	Сметана	шт	10	15.00	150.00
			Разом		250.00
			ПДВ		50.00
			Всього		300.00

Рис. 1. Типова накладна продуктового магазину

ЗАГАЛЬНИЙ ОПИС

Перспективи продукту

ProdCir дозволить користувачам обліковувати постачальників, клієнтів, працівників та товари продуктового магазину, оформлювати квартальні звіти, організовувати закупки, аналізувати фінансові показники. З допомогою зручних форм інформацію можна буде легко додавати та редагувати. Також можна розвинути програмний продукт і забезпечити з його допомогою нарахування заробітної плати працівникам магазину.

Основні функції програмного продукту:

Експорт-імпорт даних;

Авторизація користувачів;
Розрахунок вартості замовлення та його оформлення;
Облік співробітників та їх закріплень за відділами;
Оформлення звітів;
Нарахування заробітної плати.

Класи користувачів та їх характеристики

Клас користувача	*Характеристика користувача*
Менеджер	Менеджер – це користувач, який керує базою постачальників, клієнтів та співробітників. *Список можливостей:* 1. Додавання інформації про нових клієнтів, співробітників, постачальників. 2. Редагування інформації. 3. Оформлення замовлень на продукцію.
Консультант	Консультант – це користувач, який володіє інформацією про особливості роботи ProdCir та на вимогу клієнта надає йому необхідну інформацію. *Список можливостей:* 1. Пошук запитуваної користувачем інформації в системі ProdCir. 2. Зв'язок з клієнтом. 3. Зв'язок з менеджером у випадку відсутності відповіді на запитувану інформацію.
Бухгалтер	Бухгалтер – це користувач, який відповідає за контроль над фінансовим обігом та за вчасне формування звітності. *Список можливостей:* 1. Нарахування заробітної плати. 2. Доступ до всіх грошових транзакцій.

Середовище функціонування

Програмний продукт є крос-платформеним, тому встановлюється і експлуатується в ОС Windows 2000 / XP / Vista / Win7 / Win8, а також в ОС сімейства Linux. Як сервер баз даних використовується Oracle, який може бути встановлений як на Windows, так і на Unix-подібних ОС.

Обмеження проектування та реалізації

1. Система повинна використовувати поточну версію корпоративного стандарту процесора бази даних Oracle.
2. Для забезпечення крос-платформеності основні сценарії повинні бути реалізовані на Java.

3. Інтерфейс та довідка програмного продукту повинні повністю підтримувати українську мову.

Документація користувача

1. Покрокова ілюстрована інструкція для встановлення програмного продукту на різних ОС.
2. Система повинна підтримувати ієрархічну систему довідки з доступом до мережі, яка описує та ілюструє всі функції системи.
3. Програмний продукт повинен реалізувати гнучку, диференційовану систему оповіщення та опису помилок.
4. Програмний продукт повинен через розумні проміжки часу повідомляти користувача про поточний стан системи.

ХАРАКТЕРИСТИКИ СИСТЕМИ

✓ *Вхід в систему*

Користувач повинен ввести зареєстровані раніше логін і пароль для того, щоб ввійти в систему як менеджер, бухгалтер або касир. Відповідно до виконуваних ролей повинен надаватись доступ до необхідної інформації, а також можливість її редагування.

Пріоритет – середній.

Послідовність «дія-відгук»

Дія: Користувач вводить необхідні дані, заповнюючи типову форму.

Відгук: Система надає доступ до певних даних.

Функціональні вимоги

1. Система повинна перевіряти, чи заповнені всі обов'язкові поля.
2. Система повинна перевіряти поля на коректність введених даних.
3. Система повинна співставляти введені дані з зареєстрованими в БД.

✓ *Введення нової інформації про постачальника, клієнта або співробітника*

Введення інформації передбачає заповнення всіх необхідних полів тієї чи іншої категорії.

Пріоритет – високий.

Послідовність «дія-відгук»

Дія : Користувач вводить дані, використовуючи типовий інтерфейс.

Відгук: Система формує і додає новий запис до вже існуючої бази даних.

Функціональні вимоги

1. Система повинна перевіряти, чи заповнені всі обов'язкові поля.
2. Система повинна перевіряти поля на коректність введених даних.

✓ *Формування звітів*

Користувач може сформувати необхідний звіт за вказаний проміжок часу відносно певних транзакцій. Для цього необхідно сформувати певний запит до бази даних і відредагувати, якщо це необхідно, результат.

Пріоритет – середній.

Послідовність «дія-відгук»

Дія: Користувач вводить необхідний запит.

Відгук: Система повертає запитувану інформацію.

Функціональні вимоги

1. Система повинна перевіряти, чи заповнені всі обов'язкові поля.
2. Система повинна перевіряти поля на коректність введених даних.
3. Система повинна відображати статус виконуваної операції.

ВИМОГИ ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Користувацькі інтерфейси

1. Система повинна забезпечувати посилання на довідку в кожному вікні програми для пояснення, як користуватися цим вікном.
2. Система повинна постійно відображати поточний статус системи.
3. Система повинна виділяти поля, обов'язкові для заповнення.
4. Система повинна перевіряти коректність введених даних.
5. Кожне вікно програми повинне відповідати єдиному стандарту стилю та вигляду.

Апаратні інтерфейси

1. Принтер для друку звітів.
2. Зчитувач штрих-кодів товарів.

Програмні інтерфейси

1. Експорт/імпорт даних між програмою та базою даних.
2. Зв'язок між програмою та онлайн-довідкою в браузері.

Комунікаційні інтерфейси

1. Синхронізований інтерфейс обміну електронними формами між програмою та СУБД.
2. Шифровані версії протоколів для передачі особистої інформації абонента та даних економічного характеру.

НЕФУНКЦІЙНІ ВИМОГИ

Вимоги продуктивності

1. Система повинна обслуговувати багатьох користувачів в одиничний проміжок часу без втрат продуктивності.
2. Система повинна виводити користувачу повідомлення про підтвердження операції не більше ніж через 3 секунди після того, як користувач завершує її введення.

Вимоги надійності

1. Система повинна вести журнал обліку всіх операцій для запобігання втрат даних користувача.
2. Система повинна бути стійкою до збоїв зовнішніх систем (СУБД), повинен бути реалізований механізм відкату транзакцій.

Вимоги безпеки

1. Всі мережеві операції, що включають фінансову чи особисту інформацію, повинні бути зашифровані.
2. Система повинна завершувати поточний сеанс роботи у випадку тривалої бездіяльності користувача для убезпечення втрат його особистих даних.
3. Система повинна оцінювати надійність паролю при створенні облікового запису та повідомляти користувача про це.

Атрибути якості програмного продукту

1. Система повинна надавати користувачу швидкий доступ до потрібних йому даних.
2. Система повинна мінімізувати ризики втрати даних.
3. Система повинна бути зрозумілою в користуванні для всіх груп користувачів.

ІНШІ ВИМОГИ

Вимоги інтернаціоналізації

1. Система повинна забезпечити можливість вибору мови інтерфейсу для кожного користувача.

Вимоги бази даних

1. Система повинна дозволяти доступ до баз даних для внесення змін тільки менеджеру та бухгалтеру.
2. База даних повинна зберігати інформацію про історію покупок кожного зареєстрованого клієнта.

Лабораторна робота № 2

Тема. Використання систем супроводу виконання завдань в процесі командної розробки невеликих програмних проектів.

Контрольні запитання

1. Які основні методології розробки ПЗ ви знаєте? На які команди вони орієнтовані?
2. До якої методології належить метод Scrum? У чому проявляється ітеративність та інкрементність методу Scrum?
3. Для чого по методу Scrum програмні проекти розбиваються на спринти?
4. З яких етапів складається кожен спринт? Яка їх гранична тривалість?
5. Яку з розглянутих систем супроводу виконання завдань в процесі командної розробки програмних проектів доцільно, на вашу думку, використовувати в Scrum-проектах?

Завдання

Виконайте наведені нижче завдання. В місцях, позначених ✎, вставте знімки системи супроводу виконання завдань у файл звіту. Після виконання завдань дайте у файлі звіту письмові відповіді на контрольні питання.

Хід роботи

I. Використання систем супроводу виконання завдань в локальній мережі

1. Завантажте інсталяцію та встановіть на власному ПК чи ноутбуці програму ToDoList (наприклад, з web-сторінки https://soft.mydiv.net/win/download-ToDoList.html). В процесі інсталяції оберіть зручну для вас мову інтерфейсу та дозвольте редагувати список завдань по мережі.
2. Завантажте встановлену програму. Створіть новий список завдань по спринту (для кожного окремого спринта в процесі розробки програмного проекту (в нашому випадку – електронного магазину) створюється окремий список завдань). Зверніть увагу, що у верхній частині вікна програми задаються параметри відбору завдань, а у нижній – описується обране завдання/підзавдання.
3. Ознайомтеся з прикладами постановки окремих завдань з розробки програмних проектів на https://www.aniart.com.ua/blog/tz-sample-base-on-useing-cases-laravel/.
4. Реалізуйте окремі завдання для одного з спринтів, як на рис. 1.

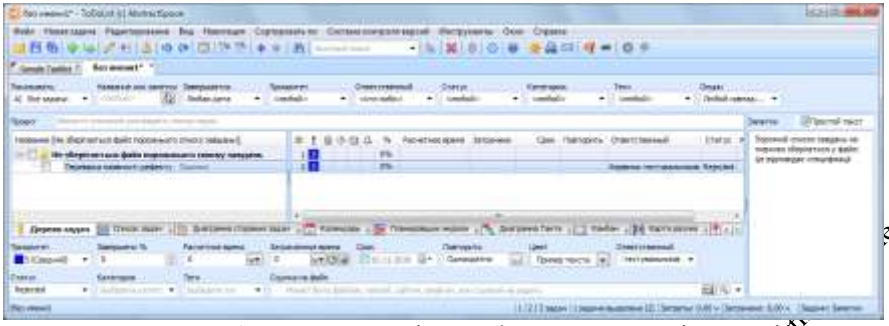


Рис. 1. Приклад трекінгу дефекту в програмі ToDoList

5. Підзавдання цих завдань мають відображати етапи їх виконання. . Завершіть виконання окремих завдань. . Заблокуйте редагування одного з завдань. . Відберіть лише невиконані завдання по конкретному виконавцю.
6. **Додаткове завдання на 1 бал до рейтингу.** Збережіть ваш файл зі списком завдань в загальнодоступній мережевій папці. Відкрийте файл завдань одногрупника і спробуйте одночасно з ним редагувати завдання з цього файлу.

ІІ. Використання систем обліку завдань в Інтернеті

7. Зайдіть на сайт https://todoist.com/, зареєструйтеся на ньому та увійдіть у свій обліковий запис.
8. Створіть новий проект з розробки програмних проектів (для кожного спринта з розробки в нашому випадку – електронного магазину) створюється окремий проект). Використовуючи список додаткових можливостей проекту (рис. 2), відкрийте до нього загальний доступ та запросіть у виконавці своїх одногрупників.

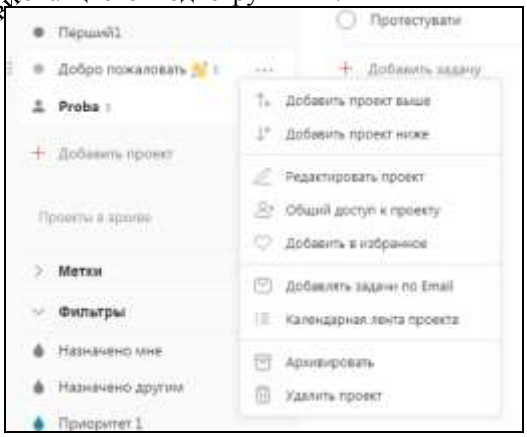


Рис. 2. Список додаткових можливостей проекту

Реалізуйте окремі завдання для одного з спринтів. Відповідальним за завдання/підзавдання має бути конкретний виконавець з сформованого раніше списку. Різні завдання повинні мати різні життєві цикли.

9. Виконайте завдання, передані вам одногрупниками.
10. Завершіть виконання окремих завдань. . Відберіть лише невиконані завдання по конкретному виконавцю.
11. Зробіть висновки про переваги і недоліки використання систем супроводу виконання завдань в локальних мережах та Інтернеті для розробки програмних проектів.

Звіт за результатами виконання лабораторної роботи здайте через Moodle, а якщо це не вдасться – надішліть безпосередньо викладачу.

Лабораторна робота № 3

Тема. Використання професійних систем супроводу командної розробки програмних проектів.

Контрольні запитання

1. Чому для входу в акаунт JIRA доцільно використати google-авторизацію?
2. Що таке беклог у Scrum?
3. З чого складається спринт? Які основні параметри спринта?
4. Які три основні стани задач спринта? Які два способи зміни стану задачі використовуються в Jira? Які типи задач використовуються в Jira?
5. Як переглянути власні поточні задачі до виконання у всіх проектах?

Завдання.

Виконайте наведені нижче завдання. В місцях, позначених ✎, вставте знімки системи супроводу виконання завдань у файл звіту. Після виконання завдань дайте у файлі звіту письмові відповіді на контрольні питання.

Теоретичні відомості

Ознайомтеся з призначенням системи керування Agile-проектами Jira на сайті https://uk.wikipedia.org/wiki/Atlassian_JIRA.

Хід роботи

1. Створіть свій акаунт (обліковий запис) з реальними даними на сайті gmail, якщо ви цього не зробили раніше.
2. На сайті розробника системи керування Agile-проектами JIRA https://www.atlassian.com/ru/software/jira увійдіть до свого акаунту, **використовуючи Google-авторизацію.**
3. Серед запропонованих компанією Atlassian програмних продуктів оберіть Jira Software.
4. Використовуючи пункт меню *Проекты – Создать проект*, розпочніть створення проекту нового покоління. Дайте змістовну назву своєму програмному проекту (вона має містити ваше прізвище і предметну область) та вкажіть ключ-абревіатуру без пробілів великими латинськими літерами. **Оберіть шаблон проекту на основі технології Scrum**, як на рис. 1, та натисніть кнопку *Создать*.

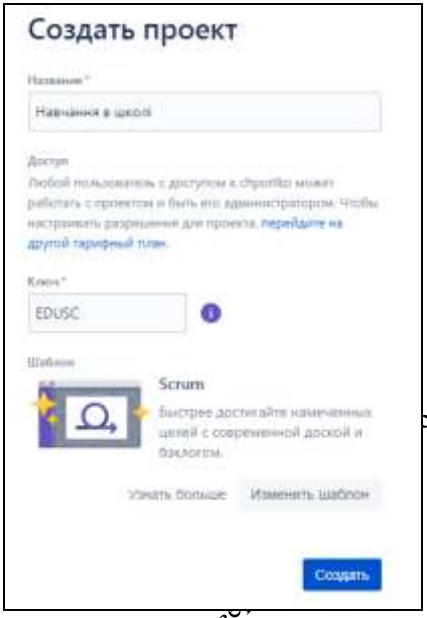
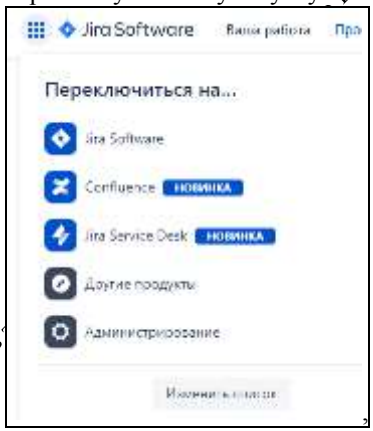


Рис. 1. Вікно створення нового проекту за технологією Scrum

5. Зареєструйте виконавців вашого проекту. Для цього в меню додатків у верхньому лівому кутку вашого сайта, яке виглядає, приблизно, так:



перейдіть до програми *Администрирование* та на сторінці *Пользователи* запросіть до виконання одногрупників, вказавши їх електронні пошти. Перегляньте список запрошених виконавців. У своїй поштовій скриньці прийміть запрошення на виконання аналогічних проектів одногрупників. Поверніться до програмного продукту Jira Software, а у ньому – до свого проекту.

6. На сторінці вашого проекту перейдіть до розділу **Бэклог** та створіть там декілька задач (не менше десяти), які у ньому потрібно реалізувати. Як позначаються створені задачі? Для 2-3 задач, використовуючи панель справа, призначте виконавців (рис. 2).

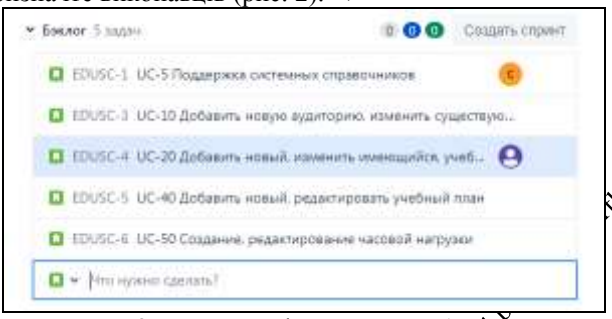


Рис. 2. Приклад сформованого бекклогу

7. Перетягніть у перший спринт вашого проекту не менше трьох найважливіших задач. Створіть безпосередньо в ньому ще дві додаткові задачі (рис. 3).

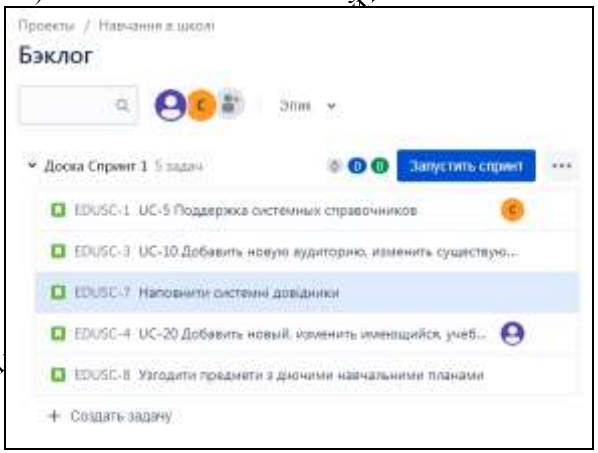


Рис. 3. Приклад сформованого списку задач спринта

8. Запустіть перший двотижневий спринт.
9. Перейдіть на дошку розпочатого спринта. Призначте виконавців трьом задачам. Розпочніть виконання двох задач, перетягнувши їх в панель *В работе*. Для ще однієї задачі встановіть такий же статус. Де вона тепер відображається?
10. Використовуючи верхню частину дошки, віднайдіть можливість швидкого відбору задач по виконавцю та повернення до відображення всіх задач.

11. Призначте окремим задачам епіки та story point. Як ці призначення використовуються в *Дорожній карті*? . Віднайдіть задачу з найдовшою історією.
12. Умовно виконайте більшість задач першого спринта, перемістивши їх до розділу *Готово*.
13. Завершіть перший спринт, перемістивши невиконані задачі до другого спринта. . Що тепер відображається в беклозі?
14. Використовуючи розділ *Ваша работа*, прийміть участь у виконанні задач проектів колег.
15. Віднайдіть виконані раніше задачі та завершені спринти.
16. Віднайдіть та використайте інші корисні можливості Jira.

Звіт за результатами виконання лабораторної роботи здайте через Moodle, а якщо це не вдасться – надішліть безпосередньо викладачу.

Лабораторна робота № 4

Тема. Основи роботи з локальними репозиторіями в СКВ Git.

Контрольні запитання.

1. Як створити новий репозиторій? Які три області контролює Git в репозиторії?
2. Які файли розміщуються в області індексів репозиторію?
3. Які команди Git переносять файли в індекс та вилучають з нього?
4. Що розміщується в області комітів? Як записати файли з області індексів в коміт?
5. Як напряму записати файли з робочої області в коміт без використання індексу?

Завдання.

Виконайте наведені нижче завдання, запропоновані студентом Мариничем Іваном з групи ПІ-20:

Introduction

In this lab, you will learn some essential Git skills and perform some practical exercises to help you get comfortable with the Git workflow.

Prerequisites

- Basic knowledge of the command line interface.
- Git installed on your local machine.
- A GitHub account.
- Must be created folder, called “tasks”, where should be stored each screenshot of your terminal after completing each task in a directory called “task_1” for example. The folder should be added separately one-by-one with its commit and certain message

Tasks

Task 1: Clone a Repository

1. Choose a repository on GitHub that you want to work with.
2. Clone the repository to your local machine using the `git clone` command.

Task 2: Create a New Branch

1. Create a new branch using the `git branch` command. Give the branch a descriptive name that indicates what changes you will make.
2. Switch to the new branch using the `git checkout` command.

Task 3: Make Changes and Commit

1. Make some changes to the code in the repository.
2. Stage the changes using the `git add` command.
3. Commit the changes using the `git commit` command with a descriptive message that summarizes the changes you made.

Task 4: Push Changes to Remote

1. Push the changes to the remote repository using the `git push` command. Make sure to push to the branch you created in Task 2.
2. Create a pull request on GitHub to merge your changes into the master branch.

Task 5: Review and Merge Changes

1. Have a peer review your code changes.
2. If there are any issues, make the necessary changes and repeat Tasks 3 and 4.
3. If the code changes are approved, merge the changes into the master branch using the pull request on GitHub.

Task 6: Fetch and Rebase Changes

1. Fetch changes from the master branch using the `git fetch` command.
2. Switch to the master branch using the `git checkout` command.
3. Rebase the master branch with your changes using the `git rebase` command.
4. Resolve any conflicts that arise during the rebase process.
5. Push the rebased changes to the remote master branch using the `git push` command.

Task 7: Review and Merge Rebased Changes

1. Have a peer review your rebased changes.
2. If there are any issues, make the necessary changes and repeat Tasks 3 and 4.
3. If the code changes are approved, merge the rebased changes into the master branch using the pull request on GitHub.

Task 8: Stashing Changes

1. Create a new branch from the main branch and make some changes to a file.
2. Use **`git stash`** to temporarily store your changes.
3. Switch back to the main branch and make some other changes to the same file.
4. Use **`git stash apply`** to apply your stashed changes to the file.
5. Resolve any conflicts if necessary.

Task 9: Reflogging

1. Use **`git reflog`** to view the commit history for your repository.
2. Find a commit that you want to restore.
3. Use **`git reset --hard <commit-hash>`** to restore the repository to the state it was in at the chosen commit.

Task 10: Diff changes

1. Make changes to a file and commit them to the repository.
2. Make additional changes to the same file and save them without committing.
3. Use **`git diff`** to compare the changes you made to the file since the last commit.
4. Use **`git diff <commit-hash>`** to compare the changes you made to the file between the last commit and a previous commit.

Task 11: Discarding changes

1. Use **`git log`** to view the commit history for your repository.
2. Find the hash of a previous commit that you want to revert to.
3. Use **`git reset --hard <commit-hash>`** to revert your repository to the state it was in at the chosen commit.

Conclusion

Congratulations! You have completed the Git Skills lab. You should now have a better understanding of how to work with Git and how to collaborate with others using Git. Keep practicing and exploring Git to continue improving your skills.

Звіт за результатами виконання лабораторної роботи здайте через Moodle, а якщо це не вдасться – надішліть безпосередньо викладачу.

Лабораторна робота № 5

Тема. Використання аліасів та тегів для керування різними версіями файлів за допомогою СКВ Git. Дослідження внутрішньої структури Git-репозиторію.

Контрольні запитання.

1. У чому різниця між аліасом і тегом? Як створити і знищити кожен з цих об'єктів?
2. Які способи визначення тегів репозиторію ви знаєте?
3. Чим відрізняється дерево коміту від вмісту поточної директорії? Що ще міститься в коміті, крім посилання на його дерево?
4. Чим відрізняється дерево директорії в репозиторію від самої директорії?
5. На які *blob*и (binary large object) містить посилання кожне дерево репозиторію? Де в репозиторію віднайти незмінені файли поточного коміту?
6. Як створити гілку, ототожнену з гілкою віддаленого репозиторію?
7. Чим *git pull* відрізняється від *git fetch*?

Теоретичні відомості

Призначення, приклади застосування та особливості використання аліасів (псевдонімів) та тегів (міток) наведені, наприклад, у фрагменті книги

Chacon S, Straub B. Pro Git. Second Edition. – Apress, 2014. – С. 56-62, посилання на яку є на сторінці дисципліни навчальної платформи університету

Хід роботи

Пройдіть навчальний курс на сайті GitHowTo (https://githowto.com/). Для цього після переходу на початкову сторінку сайту оберіть в правому верхньому кутку українську мову інтерфейсу, натисніть кнопку **Почати** та послідовно виконайте всі дії, передбачені цим курсом. Команди, наведені на сайті, слід синхронно виконувати в GitBash **в директорії, назва якої співпадає з вашим прізвищем**. У звіт за результатами виконання лабораторної роботи вставте знімки програми Git Bash, підписані своїми назвами, **після** таких етапів виконання (спочатку вказується номер етапу, а після дефіса – номер пункту):

11-02. Зберігання аліасу (результат виконання команди git config –list).
12-02. Повернення до останньої версії в гілці.
13-02. Створення тегу та перехід до попередньої версії.
13-03. Перехід за назвою тегу.
13-04. Перегляд тегів.
14-04. Скасування змін у файлі.
15-03. Скасування змін в індексі.
16-03. Створення коміту з скасуванням небажаних змін.
16-04. Форматований перегляд історії комітів.
17-05. Перегляд історії всіх комітів.
18-01. Видалення тегу та комітів.
20-02. Результат переміщення файлів.
21-01. Результат відкриття *index.html*.

22-03. Перегляд директорії об’єктів службової папки.
22-05. Теги репозиторію та їх вміст.
23-02. Вміст останнього коміту в репозиторію.
23-03. Перегляд дерева останнього коміту в репозиторію.
23-04. Перегляд дерева каталогу в репозиторію.
23-05. Перегляд вмісту файла в репозиторію.
24-02. Створення коміту в новій гілці.
27-01. Перегляд графу гілок комітів.
28-01. Результат злиття гілок.
32-01. Результат злиття після конфлікту.
34-01. Застосування перебазування.
37-02. Результат клонування репозиторію.
39-01. Відомості про віддалений репозиторій.
40-01. Список всіх доступних гілок.
42-01. Перенесення комітів з віддаленого репозиторію без злиття.
43-02. Врахування змін з віддаленого репозиторію.
45-01. Ототожнення нової гілки з гілкою віддаленого репозиторію.
48-01. Відправка змін в «чистий» репозиторій.
50-01. Копія репозиторію на віддаленому сервері.

Після виконання завдань дайте у файлі звіту письмові відповіді на контрольні питання. Звіт за результатами виконання лабораторної роботи здайте через Moodle, а якщо це не вдасться – надішліть безпосередньо викладачу.

Лабораторна робота № 6

Тема. Використання комітів та гілок для керування різними версіями файлів за допомогою СКВ Git.

Контрольні запитання.

1. Які три ієрархічні структури контролює Git? Коли формується кожна з цих структур?
2. Чим відрізняються дії команд git commit, git commit –amend та git commit –a?
3. Яка команда створює нову гілку? Переходить в нову гілку? Створює і переходить в нову гілку?
4. Де видаляють файли команди **rm** та **git rm**?
5. Які дії потрібно виконати після усунення конфліктів у вмістах файлів?

Теоретичні відомості

В процесі виконання лабораторної роботи рекомендується використовувати такі команди (в круглих дужках вказується момент часу в лекції Романа Романовського, а в кутових – сторінка з книги ProGit, де висвітлюються відповідні команди чи параметри):

Загальні налаштування:

1. Встановлення імені користувача та адреси його електронної пошти:
   - **git config --global user .name <ім'я без лапок>**;
   - **git config --global user.email <адреса email>** (system – діє для всіх користувачів; global – діє для всіх проектів обраного користувача, без них – тільки для активного проекту <22>).
2. Перевірка встановлених параметрів:
   - **git config –-list (q-вихід з режиму перегляду) (1:27:05).**
   - **git config user.name – видає ім'я користувача. Інші параметри – аналогічно.**

Робота з файлами і каталогами:

1. cd <disk>:, cd <dir>/ – перейти на вказаний диск чи в папку.
2. ls – вивести вміст поточного каталогу, при чому файли і каталоги виглядають по різному, –l – вивід в стовбець, -a – вивід з прихованими файлами, -la – разом. dir – вивести всі об'єкти в однаковому форматі.

3. | - результати команди зліва передати команді справа. Наприклад, ls | grep <текст без лапок> - виводить лише файли, в назві яких є потрібний текст.
4. cat <ім'я> - вивести вміст вказаного файла на екран (1:33:30).
5. explorer . – відкрити поточний каталог в провіднику.
6. mkdir <dir> – створити каталог.
7. touch <file> – створити файл. Якщо в підкаталозі, то sub/<file> (зворотній слеш). Назви з пробілами потрібно брати в подвійні лапки.
8. echo <текст> > <file> – вивести текст в файл . Дописати: >> . Текст може бути без лапок.
9. cp <звідки> <куди> – скопіювати файл.
10. mv <звідки> <куди> - перейменувати/перемістити файл (1:05:15).
11. rm – видалити файл, rm –r – видалити каталог, rm –rf – видалити без підтвердження. Якщо не вказати каталог – то може очистити весь диск (55:00).
12. clear – очистити консоль.

Робота з комітами репозиторію:

1. git cat-file –t <hash> – вивести тип файла; -p – вивести дані про коміт.

Робота з локальним репозиторієм:

1. **git init - створити новий репозиторій. При цьому створюється прихований каталог git з визначеною структурою (56:00).**
2. **git status - видає несинхронізовані файли (58:08),** -s – скорочений перегляд.
3. **git add <назва> - переміщує файл до підготовлених для commit. git add *.txt (59:28), git add * - переміщує всі видимі файли, git add . – переміщує також приховані файли (1:03:10), git add –f <каталог>/ - додати до синхронізації каталог, навіть якщо він – ігнорований об'єкт (1:09:10).**
4. **git commit –m <повідомлення>– відправити файли в локальний репозиторій з вказаним коментарем. Можна без –m <повідомлення> (59:50), --amend – переписати останній коміт,**

-a – створити коміт з усіх відстежуваних файлів в обхід індекса <39> (якщо файл ще не відстежується, то спочатку має бути add).

5. **git mv** <звідки> <куди> – перейменовує в індексі <41>.
6. touch .gitignore – створюється файл, вміст якого ігнорується від синхронізації (1:06:40). У цьому файлі вводяться назви файлів, маски файлів, **назви каталогів вказуються з** /, які не піддаються для commit (1.07.24)
7. notepad .gitignore – відредагувати файл ігнорування синхронізації
8. **git rm – видалити файл з індексу та каталогу, помітивши для вилучення з наступного коміту, --cached – видалити з індексу, повернувши в основний каталог <40>.**
9. **git diff – вивести зміни поточного каталогу відносно індекса, git diff master – показує зміни поточної версії відносно комітів (1:24:17), git diff –staged – вивести зміни індекса відносно останнього коміту <35>.**
10. **git show [коміт]** – вивести зміни, зроблені вказаним комітом.
11. **git restore <file> – відновити файл з індекса в активний каталог.**
12. **git reset – відновити стан індекса з активного коміту , --hard – відновити індекс та робочий каталог <267>, --soft – перемістити лише «голову» <265>, <коміт> – перейти до вказаного коміту <264>, HEAD <file> – вилучити файл з переліку індексованих (відмінити його запис в коміт) <49>.**
13. **git checkout [коміт] <file>** – відновити файл з поточного чи вказаного коміту в активний каталог.
14. **git revert [коміт]** – відмінити зміни, внесені вказаним комітом.
15. **git log – показати історію всіх commit в зворотній часовій послідовності (1:10:20)**, номер коміту при посиланнях може складатися з кількох до 40 символів, -p – показує зміни відносно попереднього коміту, -<N> - обмежує кількість комітів для виведення <43>, --graph – показує галуження гілок, --stat – показує скорочену статистику по змінах в комітах.
16. history – переглянути всі попередні команди (1:09:51).

17. git help <команда > - вивести розгорнуту довідку про команду, <команда> -h – вивести скорочену версію довідки.

Хід роботи

Виконайте наведені нижче завдання. Відповідні команди Git **з тією ж нумерацією** наведіть у звіті. В місцях, позначених 📎, вставте знімки програми Git Bash. Після виконання завдань дайте у файлі звіту письмові відповіді на контрольні питання.

1. Завантажте та встановіть на своєму ПК Git (не GitHub) версії не нижче 2.25 (краще з сайту https://gitforwindows.org/).
2. Завантажте консольну програму Git Bash.
3. Визначте версію встановленої програми.
4. Встановіть ім’я користувача та пароль для ідентифікації комітів.
5. Переконайтеся, що ваші параметри встановлено.
6. Створіть каталог для виконання завдань лабораторної роботи та зайдіить у неї.
7. Створіть в папці лабораторної локальний репозиторій.
8. Створіть текстовий файл **MyFamily**. Введіть в нього своє ПІБ. Виведіть вміст цього файлу в консоль. 📎
9. Створіть перший коміт з цим файлом та запам’ятайте його ідентифікатор.
10. Створіть гілку **Father**.
11. Доповніть файл **MyFamily** ПІБ батька та його батьків. Виведіть вміст цього файлу в консоль. 📎
12. Створіть другий коміт з цим файлом та запам’ятайте його ідентифікатор.
13. Створіть гілку **Mother**.
14. Перейдіть в цій гілці до першого коміту як в індексі, так і в активному каталозі.
15. Доповніть файл **MyFamily** ПІБ мами та її батьків. Виведіть вміст цього файлу в консоль. 📎
16. Створіть третій коміт з цим файлом та запам’ятайте його ідентифікатор.
17. Поверніться в гілку **master**.

18. Перейдіть в цій гілці до першого коміту як в індексі, так і в активному каталозі.
19. Злийте файл цього коміту з гілкою **Father**. Чи виник конфлікт при злитті? Виведіть вміст цього файлу в консоль.
20. Злийте файл цього коміту з гілкою **Mother**. Відредагуйте файл **MyFamily**, в якому виник конфлікт. Виведіть вміст цього файлу в консоль.
21. Створіть четвертий коміт з цим файлом. Виведіть інформацію про всі коміти.

Після виконання завдань дайте у файлі звіту письмові відповіді на контрольні питання. Звіт за результатами виконання лабораторної роботи здайте через Moodle, а якщо це не вдасться – надішліть безпосередньо викладачу.

Лабораторна робота № 7

Тема. Командна розробка програмних проектів з використанням web-сервісу GitHub та протоколів http/https.

Контрольні запитання.

1. Які гілки, крім **master**, найчастіше створюються в комерційних IT-проектах? Для чого вони використовуються?
2. Для чого використовується ім'я користувача та пароль в консолі Git?
3. Чому для обміну даними між локальним та глобальним репозиторіями на сьогодні найчастіше використовується протокол https? Які ще протоколи і коли застосовуються з цією метою?
4. Для яких команд Git вказуються параметри ідентифікації на сервісі GitHub? Коли це відбувається? Як забезпечити повторне введення цих параметрів?
5. Як забезпечити коректне відображення кирилиці у текстових файлах, збережених на серверах GitHub?

Теоретичні відомості

Посилання на теоретичні відомості для виконання лабораторної роботи "Внесення змін в репозиторії GitHub за допомогою запитів (Fork + Pull request)" міститься на сайті навчального середовища Moodle на сторінці дисципліни в розділі лекційних матеріалів.

Хід роботи

Виконайте наведені нижче завдання. **Відповідні команди Git та GitHub з тією ж нумерацією** наведіть у звіті. В місцях, позначених ✍, вставте знімки програми Git Bash або web-сервісу GitHub. Після виконання завдань дайте у файлі звіту письмові відповіді на контрольні питання.

1. Зареєструйтеся на сайті GitHub.com, якщо ви цього не зробили раніше.
2. Підтвердіть реєстрацію з вашої електронної пошти.
3. Відкоригуйте дані профілю, завантаживши вашу фотографію.
4. На цьому ж сайті GitHub.com вийдіть з вашого облікового запису та зайдіть в обліковий запис репозиторіїв спеціальності ІПЗ в університеті під логіном **IPZMEGU** з паролем **ipzmegu1999**. Для реєстрації нового пристрою, з якого відбувається вхід в репозиторій, введіть код підтвердження, який міститься у листі на пошті **IPZMEGU@gmail.com** (пароль входу **ipzmegu1999**).
5. Активізуйте загальний репозиторій потоку MF**-20XX**, де XX вказує на дві останні цифри поточного року.
6. Використовуючи надані вам права адміністратора, створіть у цьому репозиторії гілку (як на рис. 2), назва якої співпадає з вашим прізвищем, записаним англійською мовою. ✍

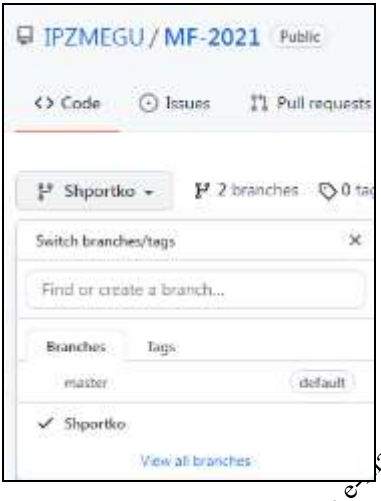


Рис. 1. Створення нової гілки в репозиторії

7. В параметрах загального репозиторію потоку ознайомтеся з правилами роботи з гілками (⚙) та забезпечте, щоб правило заборони запису змін в гілку без погодження поширювалося і на вашу гілку (рис. 2).

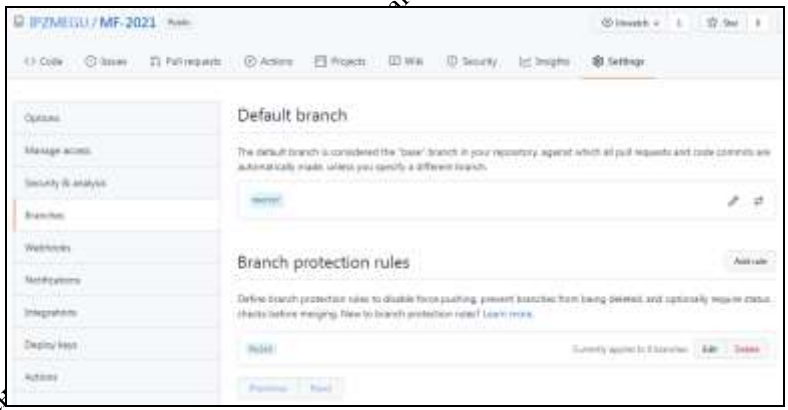


Рис. 1. Застосування правила заборони запису змін в гілку без погодження адміністраторів проекту

8. На цьому ж сайті GitHub.com вийдіть з облікового запису репозиторіїв вашої спеціальності в університеті та повторно зайдіть в ваш обліковий запис.
9. Зробіть собі копію репозиторію потоку. Для цього спочатку віднайдіть репозиторій потоку, використовуючи не лише його назву, а й назву

облікового запису **IPZMEGU**, а потім перейдіть на сторінку віднайденого репозиторію і натисніть на ній кнопку **Fork**.

10. Якщо ви цього не зробили раніше, то завантажте та встановіть на своєму ПК Git (не GitHub) версії не нижче 2.25 (краще з сайту https://gitforwindows.org/).
11. Завантажте консольну програму Git Bash.
12. Якщо ви цього не зробили раніше, то встановіть ім'я користувача та пароль для ідентифікації комітів.
13. Переконайтеся, що ваші параметри встановлено.
14. Створіть директорію для виконання завдань лабораторної роботи та зайдіть у неї.
15. Клонуйте у цю директорію репозиторій потоку по протоколу https з **вашої копії репозиторію потоку**.
16. Перейдіть в директорію склонованого репозиторію. Перейдіть у ній в гілку, назва якої співпадає з вашим прізвищем.
17. Скопіюйте у цю директорію **засобами Git Bash** файл **MyFamily.txt** з контрольної роботи.
18. Виведіть вміст цього файлу в консоль.
19. Проіндексуйте внесені зміни та створіть коміт з цим файлом.
20. Перенесіть внесені зміни з локального репозиторію у віддалений репозиторій **вашої копії репозиторію потоку** по протоколу https, використовуючи створену вами гілки.
21. На сайті GitHub.com у вашому обліковому записі на сторінці копії репозиторію потоку переконайтеся в наявності файла **MyFamily.txt** та в його коректному вмісті і створіть пропозицію змін (**New pull request**) до репозиторію потоку. При цьому забезпечте узгодження даних саме між гілками, назва яких співпадає з вашим прізвищем.
22. На цьому ж сайті GitHub.com вийдіть з вашого облікового запису та повторно зайдіть в обліковий запис **IPZMEGU** репозиторіїв вашої спеціальності.
23. Уявляючи себе в ролі адміністратора IT-проекту, перегляньте запропоновані зміни вашої пропозиції (на вкладці **pull request**), ліквідуйте можливі конфлікти (🡒) та обов'язково дайте на ваш **pull request** схвальний відгук (обравши посилання **Add your review** у нижній частині вікна запиту, як на рис. 3, та встановивши перемикач **Approve** перед натисненням кнопки **Submit review** у вікні відгуку, як на рис.4).

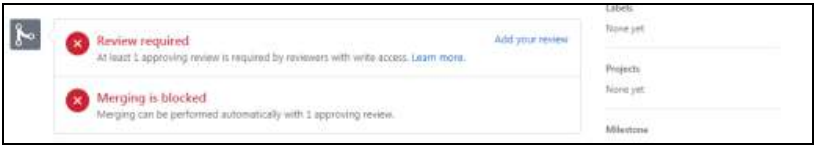


Рис. 3. Перехід до формування відгуку на запит корегування

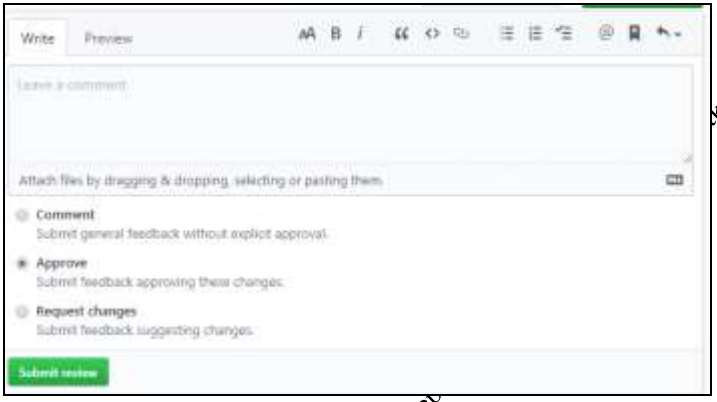


Рис. 4. Формування відгуку на запит корегування

24. Злийте пропозиції вашого **pull request** з гілкою **master** (🡅). Переконайтеся, що в гілці **master** репозиторію потоку є файла MyFamily.txt, а в ньому – відомості про вас і ваших родичів.

Після виконання завдань дайте у файлі звіту письмові відповіді на контрольні питання. Звіт за результатами виконання лабораторної роботи здайте через Moodle, а якщо це не вдасться – надішліть безпосередньо викладачу.

Лабораторна робота № 8

Тема. Розробка програмних проектів з використанням web-сервісу GitHub та протоколу SSH.

Контрольні запитання

1. Які переваги з'єднання по протоколу SSH?
2. Чим з'єднання по протоколу SSH відрізняється від з'єднання по протоколу HTTPS?
3. Для чого генеруються два ключі SSH? Хто і коли ці ключі використовує?
4. Як віддалений сервер виконує ідентифікацію по протоколу SSH?
5. Навіщо слід періодично переглядати ключі SSH на віддаленому сервері?

Теоретичні відомості

Посилання на теоретичні відомості для виконання лабораторної роботи "Налаштування SSH-ключів для роботи з віддаленими репозиторіями" міститься на сайті навчального середовища Moodle на сторінці дисципліни в розділі лекційних матеріалів.

Хід роботи

Виконайте наведені нижче завдання. **Відповідні команди Git та GitHub в місцях, позначених ✍, з тією ж нумерацією** наведіть у звіті. Результати виконання цих команд наведіть у звіті знімками програми Git Bash або web-сервісу GitHub. Після виконання завдань вкажіть у файлі звіту письмові відповіді на контрольні питання.

1. Зайдіть у ваш обліковий запис на сайті GitHub.com. Якщо назва облікового запису не містить ваше прізвище, то створіть новий обліковий запис з виконанням цієї вимоги, підтвердіть його дієвість з електронної пошти та зайдіть в цей обліковий запис.
2. Відкоригуйте дані профілю, завантаживши вашу фотографію. ✍
3. Створіть у вашому обліковому записі репозиторій **hello-world** та виконайте в ньому всі кроки навчального курсу https://guides.github.com/activities/hello-world/. Для кроків №№ 3, 4, 5 додатково виконайте ✍.
4. На локальному ПК завантажте Git Bash, створіть папку для клонованої копії репозиторію **hello-world** та зайдіть в неї.
5. **Створіть на локальному ПК нову пару SSH-ключів. Перейменуйте ці ключі так, щоб вони містили ваше прізвище латиницею.** ✍
6. **Додайте згенеровані вами ключі в SSH-агент на ПК.** Виведіть перелік встановлених ключів. ✍
7. Додайте згенерований вами **публічний** ключ у ваш обліковий запис на GitHub. ✍

8. Клонуйте репозиторій **hello-world** у вашу локальну папку на ПК з використанням протоколу SSH.
9. Внесіть зміни у файл про вас в **локальному** репозиторії **hello-world**.
10. **Створіть коміт з внесеними змінами**.
11. **Відправте внесені зміни у ваш репозиторій на сайті GitHub.com**.
12. **Відобразіть внесені зміни на цьому сайті**.

Після виконання завдань дайте у файлі звіту письмові відповіді на контрольні питання. Звіт за результатами виконання лабораторної роботи здайте через Moodle, а якщо це не вдасться – надішліть безпосередньо викладачу.

Лабораторна робота № 9

Тема. Командна розробка невеликих програмних проектів з використанням середовища Visual Studio та web-сервісу GitHub.

Контрольні запитання.

1. Скільки віддалених репозиторії найчастіше створюють для невеликих програмних проектів? Чому?
2. Яка мінімальна кількість гілок створюється у невеликих програмних проектах? Які обмеження і як накладаються на ці гілки?
3. Як склонувати проект у Visual Studio? Які коміти при цьому переносяться на локальний ПК?
4. У якому файлі зберігаються маски файлів, які не індексуються Git? Як його налаштувати в проектах Visual Studio?
5. Як в Visual Studio перенести локально зроблені коміти в глобальний репозиторій? Як отримати нові коміти інших розробників?

Теоретичні відомості

В невелики програмних проектах над їх розробками, як правило, працють декілька учасників. **Всі вони мають доступ до одного репозиторія, але кожен з них працює в своїй гілці.** Гілка **master** використовується замовником. Важливі ж зміни, зроблені окремим програмістом, додаються в гілку **developer** за допомогою запитів на злиття **Pull request**.

Відеоуроки Олександра Макеєва для вивчення можливостей використання Visual Studio в командній розробці розміщені за посиланнями:

https://www.youtube.com/watch?v=-dJXyWhUrNA – Урок № 1. Вступ
https://www.youtube.com/watch?v=I6zHmBN9OxY – Урок № 2. Ініціалізація репозиторія
https://www.youtube.com/watch?v=gggMmbZWjcg – Урок № 3. Створення комітів
https://www.youtube.com/watch?v=RMeoU_BSu9c – Урок № 4. Віддалені репозиторії
https://www.youtube.com/watch?v=FQhVbY5zGGQ – У рок № 5. Вирішення конфліктів злиття
https://www.youtube.com/watch?v=FQhVbY5zGGQ – Урок № 6. Видалення комітів
https://www.youtube.com/watch?v=3ntEDZCbPI8 – Урок № 7. Галуження (теорія)
https://www.youtube.com/watch?v=qeaDb4Hu0Vs – Урок № 8. Галуження (практика)
https://www.youtube.com/watch?v=EdAskZXjcsA – Урок № 9. Переміщення комітів

Посилання на теоретичні відомості для виконання лабораторної роботи “Внесення змін в репозиторії GitHub за допомогою запитів” міститься на сайті навчального середовища Moodle на сторінці дисципліни в розділі лекційних матеріалів.

Хід роботи

Виконайте наведені нижче завдання. **Відповідні команди Visual Studio та GitHub з тією ж нумерацією** наведіть у звіті. В місцях, позначених ☞, вставте знімки середовища розробки Visual Studio чи web-сервісу GitHub. Після виконання завдань дайте у файлі звіту письмові відповіді на контрольні питання.

1. **Якщо ви цього не зробили раніше**, то зареєструйтеся на сайті GitHub.com, підтвердіть реєстрацію з вашої електронної пошти, відкоригуйте дані профілю, завантаживши вашу фотографію.
2. На цьому ж сайті GitHub.com зайдіть в обліковий запис репозиторіїв спеціальності ІПЗ в університеті під логіном **IPZMEGU** з паролем **ipzmegu1999**. Якщо пристрій, з якого здійснюється вхід, новий для цього облікового запису, то для реєстрації такого пристрою введіть код підтвердження, який міститься у листі на пошті **IPZMEGU@gmail.com** (пароль входу **ipzmegu1999**).
3. Переконайтеся в наявності, а при відсутності – створіть загальний репозиторій потоку **VS-20XX**, де XX вказує на дві останні цифри поточного року.
4. Використовуючи надані вам права адміністратора, створіть у цьому репозиторії гілку (як на рис. 1), назва якої співпадає з вашим прізвищем, записаним англійською мовою.

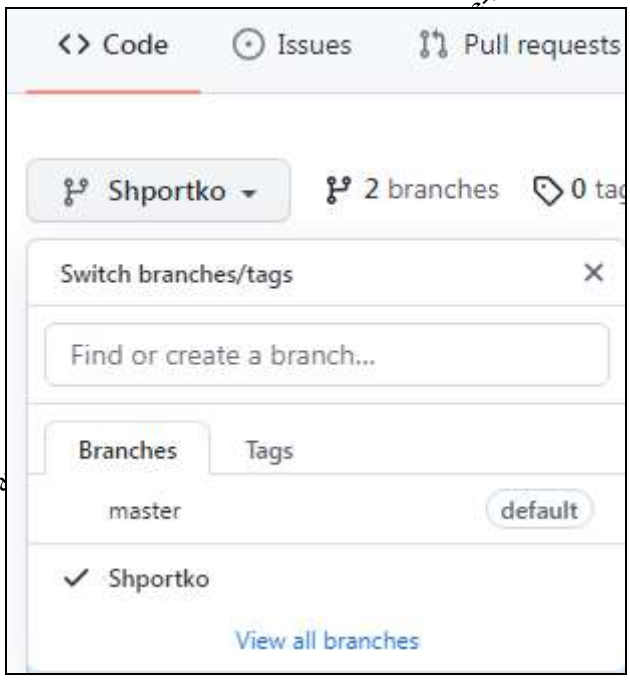


Рис. 1. Створення нової гілки в проекті

5. Використовуючи надані вам права адміністратора, з метою внесення змін, додайте свій обліковий запис в перелік користувачів репозиторію. Для цього в параметрах (**Settings**) репозиторію потоку **VS-20XX** перейдіть на вкладку **Manage access**, натисніть кнопку **People,** знайдіть

свій обліковий запис та підтвердіть його додавання до користувачів (**Invite colaborator**).

6. В цих же параметрах репозиторію ознайомтеся з правилами роботи з гілками та забезпечте заборону запису змін в гілку **master** всім користувачам, крім старости чи керівника групи. Для цього створіть нове правило захисту гілок (його назва вводиться без пробілів, як на рис. 2).

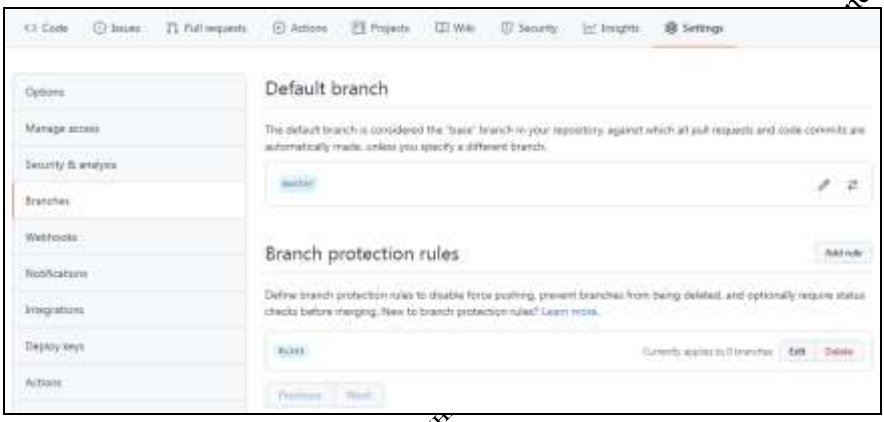


Рис. 2. Застосування правила заборони запису змін в гілку без погодження адміністраторів проекту

7. На цьому ж сайті GitHub.com вийдіть з облікового запису репозиторіїв вашої спеціальності в університеті та зайдіть в ваш обліковий запис.
8. Зробіть собі копію репозиторію потоку. Для цього спочатку віднайдіть репозиторій потоку, використовуючи не лише його назву, а й назву облікового запису **IPZMEGU** (рис. 3), а потім перейдіть на сторінку віднайденого репозиторію і натисніть на ній кнопку **Fork**.

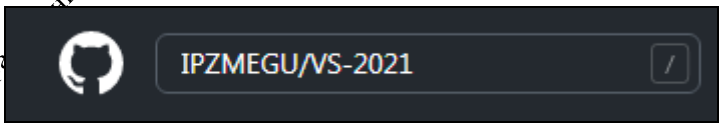


Рис. 3. Пошук репозиторію потоку з свого облікового запису

9. На локальному ПК в середовищі розробки Visual Studio **клонуйте (а не створюйте новий)** в обрану вами папку репозиторій лабораторної **з вашого облікового запису**, як на рисунку нижче і відкрийте з нього проект Windows Forms.

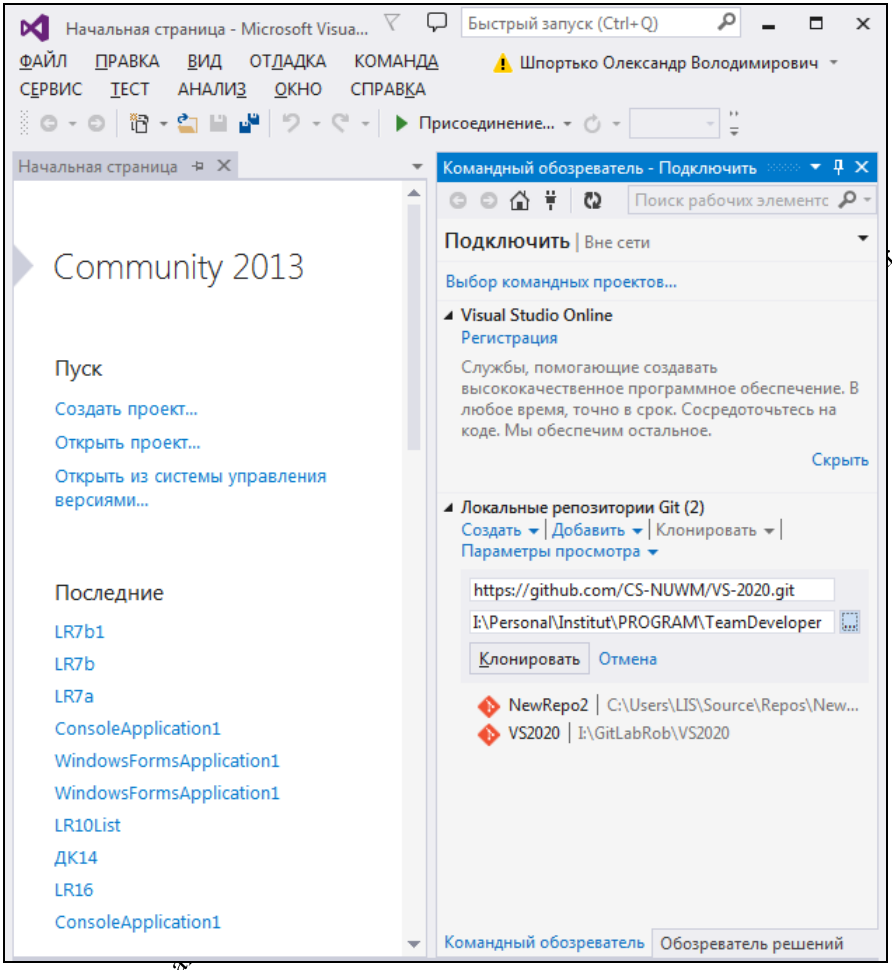


Рис. 4. Клонування репозиторію потоку з GitHub на власний ПК

10. Перейдіть в клонованому репозиторію у гілку, назва якої співпадає з вашим прізвищем, записаним англійською мовою.
11. Створіть в клонованому проекті кнопку, назва якої містить шифр вашої групи та ваш номер в журналі групи.
12. Забезпечте при натисненні цієї кнопки вивід однієї з Windows-форм, розроблених вами на попередніх лабораторних, а якщо це складно – то повідомлення з вашим улюбленим анекдотом.
13. Проіндексуйте зміни та створіть коміт і відправте його у ваш репозиторій.

14. Перенесіть внесені зміни з локального репозиторію у віддалений репозиторій **вашої копії репозиторію потоку** по протоколу https, використовуючи створену вами гілку.
15. На сайті GitHub.com у вашому обліковому записі на сторінці копії репозиторію потоку (рис. 5) переконайтеся в коректному вмісті файла коду форми і створіть пропозицію змін (**New pull request**) до репозиторію потоку. При цьому забезпечте узгодження даних саме між гілками, назва яких співпадає з вашим прізвищем.

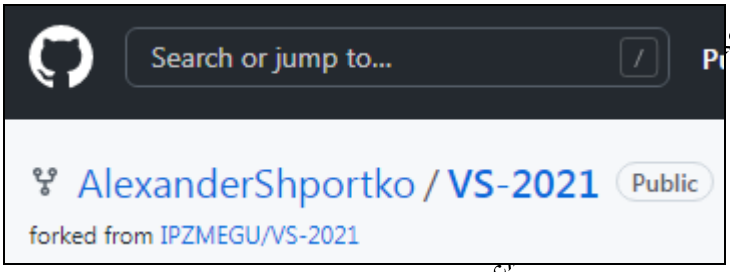


Рис. 5. Копія репозиторію потоку у власному обліковому записіку з GitHub на власний ПК

16. На цьому ж сайті GitHub.com вийдіть з вашого облікового запису та повторно зайдіть в обліковий запис **IPZMEGU** репозиторіїв вашої спеціальності.
17. Уявляючи себе в ролі адміністратора IT-проекту, перегляньте запропоновані зміни вашої пропозиції (на вкладці **pull request**), ліквідуйте можливі конфлікти () та обов'язково дайте на ваш **pull request** схвальний відгук (обравши посилання **Add your review** у нижній частині вікна запиту, як на рис. 6, встановивши перемикач **Approve** перед натисненням кнопки **Submit review** у вікні відгуку, як на рис. 7). **Прийміть в проект пропоновані вами зміни.**

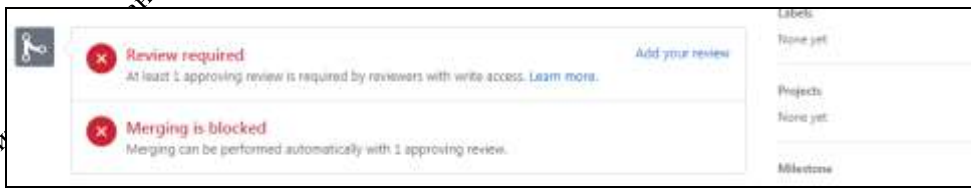


Рис. 6. Перехід до формування відгуку на запит корегування

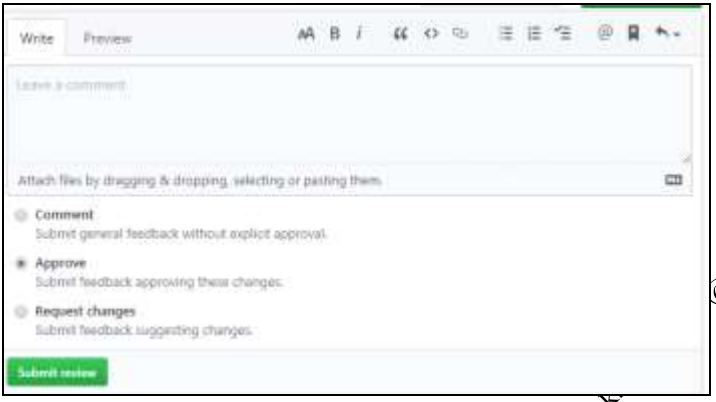


Рис.7. Формування відгуку на запит коригування

18. На сайті GitHub.com в обліковий запис **IPZMECU** репозиторіїв вашої спеціальності у вашій гілці створіть пропозицію змін (**New pull request**) до гілки **master**.
19. Уявляючи себе в ролі адміністратора, прийміть самостійно або за допомогою колег запропоновані вами зміни в гілку **master**.

Після виконання завдань дайте у файлі звіту письмові відповіді на контрольні питання. Звіт за результатами виконання лабораторної роботи здайте через Moodle, а якщо це не вдасться – надішліть безпосередньо викладачу.

Лабораторна робота № 10

Тема. Командна розробка великих програмних проектів з використанням СКВ Git, середовища розробки Visual Studio та web-сервісу GitHub.

Контрольні запитання.

1. Навіщо створювати кнопки для кожного студента в стартовій версії проекту **KR2 -20XX-1**?
2. Навіщо в GitHub створювати декілька облікових записів?
3. Чим обліковий запис GitHub відрізняється від репозиторію?
4. Чи синхронізується перехід між гілками в локальному і віддаленому репозиторіях?
5. Як забезпечити узгодження між гілками локального і віддаленого репозиторіїв?

Завдання.

Виконайте наведені нижче завдання. **Відповідні команди Git Bash, Visual Studio та GitHub з тією ж нумерацією** наведіть у звіті. В місцях, позначених ✍, вставте знімки програми Git Bash, середовища розробки Visual Studio чи web-сервісу GitHub.

Використання середовища розробки Visual Studio

1. **Завдання для старости:**
   - **створіть на локальному ПК в Visual Studio проект Windows Form KR2-20XX-1, де XX вказує на дві останні цифри поточного року;**
   - **створіть в початковій формі кнопки для кожного студента потоку. Підпишіть кнопки шифрами груп та порядковими номерами студенів в журналі;**
   - **створіть репозиторій та перенесіть його копію в обліковий запис IPZMEGU на сайт GitHub.com;**
   - **В параметрах репозиторію на сайті забезпечте заборону запису змін в гілку master всім користувачам, крім вас.** ✍
2. **Далі завдання для всіх: якщо ви цього не зробили раніше**, то зареєструйтеся на сайті GitHub.com, підтвердіть реєстрацію з вашої електронної пошти, відкоригуйте дані профілю, завантаживши вашу фотографію.

3. На цьому ж сайті GitHub.com зайдіть в обліковий запис репозиторіїв спеціальності ІПЗ в університеті під логіном **IPZMEGU** з паролем **ipzmegu1999**. Якщо пристрій, з якого здійснюється вхід, новий для цього облікового запису, то для реєстрації такого пристрою введіть код підтвердження, який міститься у листі на пошті **IPZMEGU@gmail.com** (пароль входу **ipzmegu1999**).
4. Використовуючи надані вам права адміністратора, створіть у репозиторії **KR2-20XX-1** гілку (як на рис. 1), назва якої співпадає з вашим прізвищем, записаним англійською мовою.

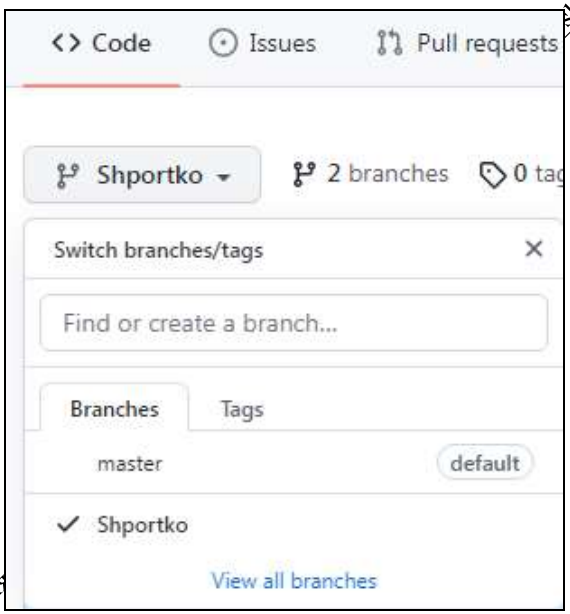


Рис. 1. Створення нової гілки в проекті

5. Використовуючи надані вам права адміністратора, з метою внесення змін, додайте свій обліковий запис в перелік користувачів репозиторію. Для цього в параметрах (**Settings**) репозиторію потоку **KR2 -20XX-1**перейдіть на вкладку **Manage access**, натисніть кнопку **People,** знайдіть свій обліковий запис та підтвердіть його додавання до користувачів (**Invite colaborator**).
6. На цьому ж сайті GitHub.com вийдіть з облікового запису репозиторіїв вашої спеціальності в університеті та зайдіть в ваш обліковий запис.

7. Зробіть собі копію репозиторію потоку. Для цього спочатку віднайдіть репозиторій потоку, використовуючи не лише його назву, а й назву облікового запису **IPZMEGU**, а потім перейдіть на сторінку віднайденого репозиторію і натисніть на ній кнопку **Fork**.
8. На локальному ПК в середовищі розробки Visual Studio **клонуйте (а не створюйте новий)** в обрану вами папку репозиторій **KR2 -20XX-1 з вашого облікового запису.**
9. Перейдіть в клонованому репозиторію у гілку, назва якої співпадає з вашим прізвищем, записаним англійською мовою.
10. Перейменуйте кнопку з вашим номером в журналі на ваше прізвище. Забезпечте при натисненні цієї вивід повідомлення з вашим улюбленим анекдотом з кнопкою і значком за допомогою методу **MessageBox.Show()**.
11. Проіндексуйте зміни та створіть коміт і відправте його у ваш репозиторій.
12. Перенесіть внесені зміни з локального репозиторію у віддалений репозиторій **вашої копії репозиторію потоку** по протоколу https, використовуючи створену вами гілку.
13. На сайті GitHub.com у вашому обліковому записі на сторінці копії репозиторію потоку переконайтеся в коректному вмісті файла коду форми і створіть пропозицію змін (**New pull request**) до репозиторію потоку. При цьому забезпечте узгодження даних саме між гілками, назви яких співпадають з вашим прізвищем.
14. На цьому ж сайті GitHub.com вийдіть з вашого облікового запису та повторно зайдіть в обліковий запис **IPZMEGU** репозиторіїв вашої спеціальності.
15. Уявляючи себе в ролі адміністратора IT-проекту, перегляньте запропоновані зміни вашої пропозиції (на вкладці **pull request**), ліквідуйте можливі конфлікти () та обов'язково дайте на ваш **pull request** схвальний відгук. **Прийміть в проект пропоновані вами зміни.**
16. На сайті GitHub.com в обліковий запис **IPZMEGU** репозиторіїв вашої спеціальності у вашій гілці створіть пропозицію змін (**New pull request**) до гілки **master**.
17. Уявляючи себе в ролі адміністратора, прийміть самостійно або за допомогою колег запропоновані вами зміни в гілку **master**.

Використання програми Git Bash

18. **Завдання для старости: створіть на сайті GitHub.com в обліковому записі IPZMEGU загальнодоступний проект KR2-20XX-2, де XX вказує на дві останні цифри поточного року;**
19. **Завдання для всіх:** створіть на власному ПК папку з вашим прізвищем і клонуйте засобами програми Git Bash проект **KR2-20XX-2**.
20. В третій рядок файла **ReadME.txt** введіть своє прізвище і після двокрапки – життєву мудрість, якої ви дотримуєтеся. Наявний текст, починаючи з третього рядка, перенесіть нижче.
21. Проіндексуйте внесені зміни засобами програми Git Bash.
22. Створіть коміт у вашій локальній папці за допомогою Git Bash.
23. Відправте зміни у віддалений репозиторій у головну його гілку.

Після виконання завдань дайте у файлі звіту письмові відповіді на контрольні питання. Звіт за результатами виконання лабораторної роботи здайте через Moodle, а якщо це не вдасться – надішліть безпосередньо викладачу.

Програма навчальної дисципліни

Змістовий модуль 1. Організація командної розробки програмних проектів в ІТ-індустрії

Тема 1. Поняття проекту, принципи підготовки успішного проекту

Проект як діяльність та документ. Мета та завдання проекту, аналіз цільових груп та їх потреб. Дерево проблем і рішень. SMART-принцип підготовки успішного проекту. Місце командної розробки в життєвому циклі програмних проектів. Методології розробки ПЗ.

Тема 2. Команди та командна робота

Формування команди (teambuilding). Групи і команди, відмінності між ними. Фактори, що стимулюють появу команди. Основні принципи формування команди. Найпоширеніші типи команд. Перевага і недоліки роботи команд. Формування й розвиток навичок командної роботи (teamskills), командного духу (teamspirit).

Командні процеси: розвиток команди, формування згуртованості і командних норм, груповий тиск, прийняття рішень в команді, конфлікти в команді. Командні цінності. Етапи розвитку команди та їх характеристика. Функціональні і командні ролі.

Тема 3. Командна реалізація проекту. Спринти проекту. Тайм-менеджмент в командній роботі

Специфіка проектної діяльності в ІТ-індустрії. Розбиття реалізації проекту на спринти. Задачі спринтів та їх виконавці.

Поняття та основні прийоми тайм-менеджменту. Матриця Ейзенхауера в плануванні розподілу часу на виконання завдань. Окремі методики керування часом. Методика GettingThingsDone у плануванні використання часу команди.

Змістовий модуль 2. Використання систем контролю версіями в процесі реалізації програмних проектів

Тема 4. Системи керування версіями програмного забезпечення: поняття, види та класифікація

Призначення систем керування версіями програмного забезпечення та передумови їх виникнення. Локальні, централізовані та

розподілені системи. Переваги та недоліки СКВ. Види та класифікація СКВ.

Тема 5. Використання системи керування версіями Git для ведення локальних репозиторіїв

Початок роботи з проектом. Щоденний цикл роботи з репозиторіями. Структура репозиторію. Версії проекту. Створення комітів. Перегляд списку комітів. Переміщення між комітами. Аліаси команд. Теги комітів.

Тема 6. Основи галуження в Git

Призначення галуження. Створення нових гілок. Гілки, як вказівки на коміти. Злиття комітів. Конфлікти та їх вирішення. Блокування змін. Перейменування та знищення гілок.

Тема 7. Віддалені репозиторії. Використання системи керування версіями Git для взаємодії з віддаленими репозиторіями

Віддалені репозиторії та основні сервіси для їх розміщення. Основи роботи з віддаленими репозиторіями GitHub та Bitbucket. Основні команди для роботи з віддаленими репозиторіями web-сервісу Bitbucket з використанням протоколів http/https. Командна розробка програмних проектів з використанням web-сервісу GitHub та протоколів http/https. Розробка програмних проектів з використанням web-сервісу GitHub та протоколу SSH.

Тема 8. Робота з віддаленими репозиторіями з використанням оболонок мов програмування

Командна розробка програмних проектів з використанням середовища Visual Studio та web-сервісу GitHub. Робота з комітами в Visual Studio. Галуження в Visual Studio. Додаткові утиліти для роботи з віддаленими репозиторіями з Visual Studio.

Приклади тестових питань для підсумкового контролю знань з дисципліни

Оберіть найпоширеніший різновид систем контролю версій

Децентралізовані;
Централізовані;
Локальні;
Приватні;
Публічні.

Які СКВ передбачають зберігання копій репозиторіїв на кожній робочій станції?

Децентралізовані;
Централізовані;
Локальні;
Приватні;
Публічні.

Які СКВ дозволяють працювати з репозиторіями навіть при відсутності підключення до Інтернету?

Децентралізовані;
Централізовані;
Глобальні;
Приватні;
Публічні.

Які СКВ використовують один екземпляр репозиторію?

Централізовані;
Децентралізовані;
Глобальні;
Приватні;
Публічні.

Оберіть найпоширеніший сервіс віддаленого хостингу репозиторіїв:

GitHub;
BitBucket;

Facebook;
Telegram;
Viber.

Оберіть другий за популярністю сервіс віддаленого хостингу репозиторіїв:

BitBucket;
GitHub;
Facebook;
Telegram;
Viber.

Яка з наведених команд зберігає зміни в локальній копії репозиторію?

commit;
reset;
push;
pull;
merge.

Яка з наведених команд відновлює файли з локального репозиторію?

reset;
commit;
push;
pull;
merge.

Яка з наведених команд зберігає зміни в глобальному репозиторію?

push;
reset;
commit;
pull;
merge.

Яка з наведених команд зчитує зміни з глобального репозиторію?

pull;

commit;

push;

reset;

merge.

Яка з наведених команд намагається поєднати зміни з двох гілок?

merge;

commit;

push;

pull;

reset.

Яку опцію задають в команді *git config* для формування конфігурації всіх користувачів?

--system;

--global;

--list;

alias;

додаткові параметри не задають.

Яку опцію задають в команді *git config* для формування конфігурації всіх проектів?

--global;

--system;

--list;

alias;

додаткові параметри не задають.

Яку опцію задають в команді *git config* для формування конфігурації поточного проекту?

додатковий параметр не задають;

--system;

--list;

alias;

--global.

Яка з конфігурацій git має найвищий пріоритет?

Конфігурація поточного проету;
Конфігурація всіх проектів користувача;
Конфігурація для всіх користувачів;
Конфігурація рпераційної системи;
Конфігурація MS Office.

Для чого в git використовується ім'я користувача і адреса його електронної пошти?

Для збереження в комітах;
Для з'єднання з локальним репозиторієм;
Для з'єднання з глобальним репозиторієм;
Для з'єднання з master-репозиторієм;
Для повідомлень про несанкціоновані підключення.

Для чого використовуються аліаси в git?

Для скороченого запису коман та їх частин;
Для ідентифікації комітів;
Для ідентифікації користувача;
Для задоволення власної значущості;
Для шифрування команд.

Оберіть правильний початок команди git для збереження аліасу в параметрах всіх проектів:

git config --global alias;
git config --system alias;
git config alias;
git history --global alias;
git log --system alias.

Оберіть команду git, яка видає ім'я користувача:

git config user.name;
git config --global user.name;
git config --system user.name;
git log --global user.name;
git params user.name.

Для чого використовується публічний SSH-ключ в git?

Для шифрування даних на сервері;
Для шифрування даних на локальному ПК;
Для дешифрування даних на сервері;
Для дешифрування даних на локальному ПК;
Для вхідних дверей.

Для чого використовується приватний SSH-ключ в git?

Для дешифрування даних на локальному ПК;
Для шифрування даних на сервері;
Для шифрування даних на локальнопу ПК;
Для дешифрування даних на сервері;
Для міжкімнатних дверей.

Яка опція *git config* дає змогу переглянути всі встановлені параметри?

--list;
--global;
--system;
ls;
dir.

Який символ використовується в git для посилань на каталог з параметрами користувача?

~;
!;
\;
/;
@.

Яка команда git виводить версію встановленого середовища?

git –version;
--version;
version;
git release;
release.

Яка команда git використовується для переходу на інший диск чи в каталог?

cd;
ls;
cat;
dir;
pwd.

Чим команда *git rm* відрізняється від команди *rm*?

Вилучає файл як в індексі, так і в робочому каталозі;
Вилучає файл в коміті, а не в робочому каталозі;
Вилучає файл в коміті, а не в індексі;
Вилучає файл в індексі, а не в робочому каталозі;
Вилучає файл як в робочому каталозі, так і в коміті.

Яка команда виводить зміни файлів поточного каталогу відносно індекса?

git diff;
git show;
git rebase;
git status;
git config.

Яка команда виводить зміни, зроблені вказаним комітом?

git show;
git diff;
git rebase;
git status;
git config.

Яка команда застосовує зміни поточної гілки до вказаної гілки?

git rebase;
git diff;
git rebase;
git status;
git config.

Яка команда відновлює файл з індекса в поточний каталог?

git restore;
git reset;
git rebase;
git status;
git config.

Назви об'єктів репозиторію складаються з 40 символів якого коду?

SHA;
SSH;
DES;
RSA;
DDR.

Скільки символів містять назви файлів обєктів в репозиторії git?

38;
40;
8;
16;
32.

Гілка git - це фактично вказівка на

останній зроблений у ній коміт;
останній зроблений коміт;
тег;
аліас;
останній сформований індекс.

Яка команда використовується для відновлення файла з коміту в поточний каталог?

git checkout;
git restore;
git reset;
git rever;t
git rebase.

Що таке перебазування?

Це застосування у своїй гілці змін комітів поточної гілки до останнього коміту вказаної гілки;

Це зміна поточної гілки;

Це перенесення назв тегів на нові коміти;

Це зміна вказівки на глобальний репозиторій;

Це перенесення репозиторію в інший каталог.

Яка з зазначених команд після виконання створює один новий коміт?

git revert;

git rebase;

git restore;

git reset;

git checkout;

Яка з наведених команд створює нову гілку, починаючи з поточного коміту?

git checkout;

git branch;

git log;

git tag;

git merge.

Яка з наведених команд створює нову гілку, починаючи з вказаного тегом коміту?

git branch;

git checkout;

git log;

git tag;

git merge.

Яка з зазначених команд після виконання створює один новий коміт?

git merge;

git rebase;

git restore;

git reset;

git checkout.

Яка з наведених команд git використовується для переходу на іншу гілку?

git checkout;
git branch;
git tag;
git log;
git reset.

Яка команда git використовується для злиття з комітами іншої гілки?

git merge;
git rebase;
git log;
git branch;
git tag.

Яка з наведених команд git дасть змогу переглянути попередні введені команди?

history;
git history;
git log;
git lg;
git ln.

Яка з наведених команд виводить інформацію про коміти в зворотньому порядку?

git log;
git history;
history;
help;
git tag.

Які коміти не виводяться командою git log без параметрів?

З яких відбулося переміщення до попередніх комітів;
Коміти інших гілок;

Приховані коміти;
Зашифровані коміти;
Однакові коміти.

Які коміти з часом знищуються збірником сміття?

З яких відбулося переміщення до попередніх комітів;
Коміти інших гілок;
Приховані коміти;
Зашифровані коміти;
Однакові коміти.

Чим гілка git відрізняється від тегу?

Гілка переміщується на новий створений коміт, а тег – ні;
Гілка вказує на коміт, а тег - на каталог;
Гілка вказує на блоб, а тег - на коміт;
Це тотожні поняття;
Кожна гілка - це тег, але не кожен тег - це гілка.

Що знищується при видаленні тегу?

Посилання на коміт;
Відповідний коміт;
Гілка, в яку входить коміт, на яку вказує тег;
Посилання на гілку;
Посилання на цей тег в історії операцій.

Для звертання на батьківський коміт відносно тегу використовується символ ...

^;
&;
|;
<.

Яка команда git використовується для перегляду вмісту поточного каталогу з виділенням

ls;
cd;

cat;
dir;
pwd.

Яка команда git використовується для перегляду вмісту поточного каталогу без виділення

dir;
cd;
cat;
ls;
pwd.

Яка команда git використовується для виводу вмісту файла?

cat;
cd;
ls;
dir;
pwd.

В git для дописування рядка в текстовий файл після echo вказується символ

>>;
>;
>>>;
<<;
<.

Для чого використовується символ '|' в командах git?

Для передачі результату команди зліва команді справа;
Для позначення логічного OR;
Для позначення побітового OR;
Для передачі результату команди справа команді зліва;
Для відокремлення опцій від параметрів.

Яка команда git використовується для відбору рядків по шаблону?

grep;
dir;

ls;
filter;
filters.

Яка команда git створює новий каталог?

mkdir;
dir;
ls;
mv;
newdir.

В якому кодуванні слід зберігати текстові файли, щоб вони коректно відображалися в блокноті?

UTF-8;
ANSI;
Windows-1251;
KOI8-R;
KOI8-U.

Яка команда git створює новий файл?

touch;
file;
newfile;
pwd;
new.

Яка команда git використовується для видалення файлів?

rm;
del;
delete;
grep;
remove.

Яка команда git очищає консоль?

clear;
cls;
clrscr;
new;

start.

Яка команда git копіює файл?

cp;
copy;
clone;
dublicate;
replication.

Яка команда переміщує файл?

mv;
move;
rn;
rename;
ren.

Яку команду git слід використати першою для створення нового репозиторію?

rm -rt .git;
git init;
git repo .git;
git remote;
init.

Яка команда git виводить несинхронізовані файли?

git status;
git config;
git notsync;
echo;
echo off.

Яка команда переміщує до підготовлених для коміту всі видимі неігноровані файли?

git add *;
git commit –m;
git add;
git mv *;

git mv.

Яка з наведених команд переміщує до підготовлених всі файли, включаючи приховані?

git add .;
git add *;
git add *.*;
git add all;
git add … .

В якому файлі вказуються шаблони файлів і каталогів, які не підлягають синхронізації?

.gitignore;
gitignore;
.ignore;
ignore;
notsync/

Яка опція команди git commit дає змогу створити коміт з усіх відстежуваних файлів в обхід індекса?

-a;
--amend;
-m;
--hard;
--soft.

Чим команда git mv відрізняється від команди mv?

Переміщує файл в індексі, а не в робочому каталозі;
Переміщує файл в коміті, а не в робочому каталозі;
Переміщує файл в коміті, а не в індексі;
Переміщує файл як в індексі, так і в робочому каталозі;
Переміщує файл як в робочому каталозі, так і в коміті.

Про що виводить довідку команда git help?

Про зазначену команду;
Про конфлікт версій;
Про відвідування пар студентами;

Такої команди немає;
Про конфлікт комітів.

В git злиття передбачає ...

поєднання змін різних комітів;
поєднання комітів;
поєднання індекса з комітом;
поєднання робочого каталогу з індексом;
поєднання тегів.

Якщо внести зміни у файл, а потім – видалити внесені зміни, то команда git commit ...

не створить новий коміт;
створить новий коміт;
перепитає, чи створювати новий коміт;
відновить стан індексу;
поверне попередню версію файла.

Питання гарантованого рівня знань

1. Проект як діяльність та документ.
2. Мета та завдання проекту, аналіз цільових груп та їх потреб.
3. Місце командної розробки в життєвому циклі програмних проектів.
4. Методології командної розробки ПЗ.
5. Формування команди (teambuilding).
6. Групи і команди, відмінності між ними. Фактори, що стимулюють появу команд. Основні принципи формування команди.
7. Найпоширеніші типи команд. Перевага і недоліки роботи команд. Формування й розвиток навичок командної роботи (teamskills), командного духу (teamspirit).
8. Командні процеси: розвиток команди, формування згуртованості і командних норм, груповий тиск, прийняття рішень в команді, конфлікти в команді. Командні цінності.
9. Етапи розвитку команди та їх характеристика. Функціональні і командні ролі.
10. Специфіка проектної діяльності в IT-індустрії.
11. Розбиття реалізації проекту на спринти. Задачі спринтів та їх виконавці.
12. Поняття та основні прийоми тайм-менеджменту. Матриця Ейзенхауера в плануванні розподілу часу на виконання завдань. Окремі методики керування часом. Методика GettingThingsDone у плануванні використання часу команди.
13. Призначення систем керування версіями програмного забезпечення та передумови їх виникнення.
14. Локальні, централізовані та розподілені системи.
15. Переваги та недоліки СКВ. Види та класифікація СКВ.
16. Початок роботи з репозиторієм. Щоденний цикл роботи з репозиторіями.
17. Структура репозиторію. Версії проекту. Створення комітів. Перегляд списку комітів. Переміщення між комітами.
18. Аліаси команд. Теги комітів.
19. Призначення галуження. Створення нових гілок. Гілки, як вказівки на коміти. Перейменування та знищення гілок.
20. Злиття комітів. Конфлікти та їх вирішення. Блокування та розблокування змін.
21. Віддалені репозиторії та основні сервіси для їх розміщення.
22. Основи роботи з віддаленими репозиторіями GitHub.

23. Основи роботи з віддаленими репозиторіями Bitbucket.
24. Основні команди для роботи з віддаленими репозиторіями web-сервісу Bitbucket з використанням протоколів http/https.
25. Командна розробка програмних проектів з використанням web-сервісу GitHub та протоколів http/https.
26. Розробка програмних проектів з використанням web-сервісу GitHub та протоколу SSH.
27. Командна розробка програмних проектів з використанням середовища Visual Studio та web-сервісу GitHub.
28. Робота з комітами в Visual Studio.
29. Галуження в Visual Studio.
30. Додаткові утиліти для роботи з віддаленими репозиторіями з Visual Studio.

Рекомендована література

1. Chacon Scott, Straub Ben. Pro Git. 2-nd ed. Apress, 2014. 534 с. URL: https://git-scm.com/book/uk/v2.
2. Gitflow Workflow. Atlassian Git Tutorial. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow.
3. Introduction to Git workflows. URL: https://docs.gitlab.com/ee/topics/gitlab_flow.html.
4. Тулашвілі Ю. Й. Конспект лекцій з дисципліни «Командна розробка програмних проектів» для студентів напряму підготовки для студентів напряму підготовки 6.040302 «Інформатика». Рівне: НУВГП, 2015. URL: http://ep3.nuwm.edu.ua/2233/1/04-01-06.pdf.
5. Тулашвілі Ю. Й. Методичні вказівки до виконання лабораторних робіт з дисципліни «Командна розробка програмних проектів» для студентів напряму підготовки 6.040302 «Інформатика» денної форми навчання. Рівне: НУВГП, 2015. URL: http://ep3.nuwm.edu.ua/2236/1/04-01-05.pdf.
6. Романовський О. Г., Шаполова В. В., Квасник О. В., Гура Т. В. Психологія тімбілдингу : навчальний посібник / за заг. ред. Романовського О. Г., Калашникової С. В. Харків: «Друкарня Мадрид», 2017. 92 с. URL: http://repository.kpi.kharkov.ua/bitstream/KhPI-Press/36676/1/Book_2017_Romanovskyi_Psykholohiia_tymbildynhu.pdf.
7. Горбунова В. В. Психологія командотворення: ціннісно-рольовий підхід до формування та розвитку команд : монографія. Житомир: Вид-во ЖДУ ім. І.Франка, 2014. 380 с. URI: http://er.ucu.edu.ua/bitstream/handle/1/1044/Gorbunova_Psychology%20of%20teem%20building.pdf?sequence=1&isAllowed=y.
8. Алексенко О. В. Технології програмування та створення програмних продуктів : конспект лекцій. Суми: Сумський державний університет, 2013. 133 с. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/30254/1/Alekseenko_Programuvannja.pdf;jsessionid=D575D30AC6AACA2D325333552C613A680.
9. Арсеньєва О. П. Програмування мовою C# : Навчально-методичний посібник / О. П. Арсеньєва, Л. В. Соловей. – Харків: НТУ «ХПІ», 2019. – 104 с. – Англ. мовою.

Навчальне видання

Командна розробка ІТ-проектів. Лабораторний практикум для студентів денної та заочної форм навчання освітньої програми «Інженерія програмного забезпечення Інтернету речей» першого (бакалаврського) рівня вищої освіти за спеціальністю 121 Інженерія програмного забезпечення галузі знань 12 Інформаційні технології

Укладачі:
Олександр Володимирович Шпортько
Юрій Георгійович Лотюк
Олег Михайлович Богут

Відповідальний за випуск доц. Ю. Г. Лотюк
Комп'ютерна верстка О. В. Шпортька

Підписано до друку 20.06.2023.
Формат 60х84 1/16. Папір офсетний № 1.
Умовн. друк. арк. 1,57. Наклад 50 примірників.

Віддруковано засобами оперативної поліграфії
Приватного вищого навчального закладу
"Міжнародний економіко-гуманітарний університет
імені академіка Степана Дем’янчука".
м. Рівне, вул. С. Дем’янчука, 4.