ВСТУП

Процес проходження виробничої практики є суттєвим етапом підготовки студентів до реальної роботи за обраною спеціальністю. Він дає можливість застосувати теоретичні знання в практичній діяльності, а також здобути важливі професійні навички, необхідні для подальшої кар'єри. Практика дозволяє оцінити власний рівень підготовки, перевірити правильність вибору професійного напряму і глибше зрозуміти специфіку роботи в конкретній галузі.

Протягом практики студенти мають змогу вирішувати реальні робочі завдання, працювати в команді, приймати самостійні рішення та обґрунтовувати свої висновки. Хоча на початковому етапі молодим фахівцям не часто доручають виконання складних проектів, участь у повсякденних завданнях сприяє розвитку їх професійних навичок. Це дає можливість швидше адаптуватися до робочих умов і розвивати вміння працювати в колективі.

Особлива увага приділяється навчанню таких компетенцій, як дотримання корпоративних стандартів, розвиток навичок ефективної комунікації та взаємодії з керівниками і колегами. Вирішення практичних завдань допомагає студентам краще усвідомити вимоги до професіоналів у їхній сфері діяльності.

Процес організації практики базується на чинних нормах законодавства, зокрема на законах України «Про вищу освіту» [1], «Про фахову передвищу освіту» [2], а також на наказі Міністерства освіти та науки України №1006 від 21.09.2021 [3]. Ці нормативні акти визначають стандарти підготовки студентів за спеціальністю 121 «Інженерія програмного забезпечення» і закріплюють обов'язковість проходження практики для всіх студентів, які успішно освоїли навчальний план.

Виробнича практика проводиться на базі Відокремленого структурного підрозділу «Фаховий коледж ракетно-космічного машинобудування Дніпровського національного університету імені Олеся Гончара».

1 Правила охорони праці та техніки безпеки на підприємстві

1.1 Аналіз небезпечних і шкідливих виробничих чинників проєктованого технологічного процесу, об’єкту, системи або пристрою

Процес розробки програмного продукту є комплексним та вимагає багатьох етапів, що включають проектування системної архітектури, програмування, тестування і підготовку документації. Всі ці роботи виконуються в спеціально облаштованих приміщеннях, які мають необхідне обладнання для ефективної діяльності. Важливою частиною розробки є правильна організація робочого простору, оскільки при роботі з комп'ютерною технікою можуть виникати фізичні та психофізіологічні ризики для працівників.

Основною небезпекою є електричні ризики. Використання електричних пристроїв, таких як комп'ютери, зарядні пристрої і мережеві адаптери, пов'язане з можливими небезпеками, включаючи коротке замикання або несправності обладнання, що може призвести до травм. Тому важливо постійно перевіряти стан електричних мереж та пристроїв і вживати заходів для їх захисту від можливих проблем.

Шум та вібрація, що виникають при роботі охолоджувальних систем і вентиляторів у комп'ютерних системах, можуть спричиняти дискомфорт при тривалому впливі. У великих офісах, де використовуються кілька пристроїв одночасно, рівень шуму та вібрацій може стати відчутним і впливати на комфорт працівників. Тому при плануванні робочого простору важливо враховувати ці фактори, використовуючи технології, що знижують рівень шуму та системи охолодження з низьким рівнем вібрацій.

Метеорологічні умови і освітлення на робочому місці мають прямий вплив на ефективність праці. Недостатнє або надмірне освітлення може викликати втомленість та зорове напруження, що знижує продуктивність. Відсутність належної вентиляції може підвищити температуру в приміщенні, що погіршує самопочуття працівників і знижує їх ефективність. Тому важливо створити оптимальні умови освітлення та вентиляції для забезпечення комфортної роботи [4].

1.2 Інженерно-технічні заходи з охорони праці

Створення безпечних умов праці при розробці програмного забезпечення є важливим аспектом, що забезпечує не лише здоров'я працівників, але й ефективність їхньої роботи. Ключовим напрямком є застосування автоматизованих систем для контролю технологічних процесів, що дозволяє оперативно виявляти аварійні ситуації на ранніх етапах. Такі технології здійснюють моніторинг електричних систем, апаратного забезпечення і інших технічних засобів, що дозволяє забезпечити їхню надійну та безпечну експлуатацію. Крім того, до важливих складових захисту належать сигналізація, засоби аварійного вимкнення і захисту, що дозволяють швидко реагувати на можливі загрози.

Також важливо забезпечити належні метеорологічні умови в робочих приміщеннях. Для цього використовуються кондиціонери, які контролюють температуру та вологість повітря, що сприяє комфортній атмосфері для працівників. Очищення повітря від пилу і шкідливих часток здійснюється за допомогою спеціалізованих фільтрів та пиловловлювачів. Це дозволяє знизити ризики захворювань дихальних шляхів і алергічних реакцій, створюючи сприятливі умови для здоров’я працівників.

Особливою увагою потребує зниження рівня електромагнітних випромінювань, які можуть негативно впливати на здоров’я співробітників. Для цього використовуються низьковипромінюючі монітори та оптимізується розташування комп'ютерного обладнання. Це дозволяє знизити негативний вплив електромагнітних полів на організм. Також важливо зменшувати рівень шуму і вібрацій, що виникають від охолоджувальних систем. Для цього застосовуються безшумні вентилятори та спеціальні звукоізоляційні матеріали.

Організація робочого простору програміста повинна враховувати не лише технічні аспекти, але й ергономічні вимоги. Використання регульованих меблів, таких як столи і крісла, зменшує навантаження на опорно-руховий апарат та сприяє правильній поставі. Освітлення має відповідати санітарним вимогам для захисту зору, а також повинно бути передбачене аварійне освітлення для випадку відключення електроенергії. Усі ці заходи забезпечують не лише комфортні умови праці, але й сприяють високій продуктивності працівників [5].

1.3 Пожежна профілактика

У процесі розробки програмного забезпечення надзвичайно важливо забезпечити належний рівень пожежної безпеки в робочих приміщеннях. Одним із основних факторів ризику є використання електричних пристроїв і проводки, які можуть стати причиною пожежі через перегрів або коротке замикання. Для зменшення цих загроз потрібно регулярно перевіряти стан електричних мереж та використовувати блоки живлення з функцією захисту від короткого замикання. Також слід застосовувати кабелі з негорючими матеріалами, що мінімізує ризик загоряння.

Ще однією важливою складовою є наявність системи пожежної сигналізації. Встановлення димових датчиків дозволяє своєчасно виявити початок загоряння і передати сигнал на пульт охорони, що дає можливість швидко реагувати на небезпеку та запобігти її поширенню. Ці заходи є невід'ємною частиною забезпечення безпеки в приміщеннях, де використовуються великі обсяги електронного обладнання.

Додатково для підвищення рівня пожежної безпеки в офісах варто оснащувати приміщення вогнегасниками, здатними ефективно гасити пожежі в електричному обладнанні під напругою. Вуглекислотні і порошкові вогнегасники мають бути доступними в приміщеннях, де зберігається техніка. Для площ до 50 м² рекомендовано наявність принаймні двох вогнегасників. Також необхідно мати протипожежні покривала для оперативного гасіння малих загорянь, що можуть виникнути під час роботи.

Крім технічних засобів безпеки, не менш важливо розробити чітко визначені евакуаційні плани. Вони повинні включати позначення виходів та шляхів евакуації, що дозволяє співробітникам швидко і безпечно залишити приміщення у випадку надзвичайної ситуації. Крім того, має бути налаштована система аварійного освітлення, яка дозволяє забезпечити видимість під час відключення електропостачання. Проведення тренінгів для співробітників щодо основ пожежної безпеки та правильного використання вогнегасників значно знижує ризики і підвищує рівень готовності до непередбачених ситуацій [6].

1.4 Заходи з ергономіки

Однією з головних причин травм на робочому місці є психофізіологічні фактори, такі як стрес, перевантаження та неправильні реакції на зовнішні стимули. Для зменшення їхнього впливу на працівників під час розробки програмного забезпечення застосовано ряд заходів, спрямованих на забезпечення безпеки і зручності. Важливим компонентом є ергономіка робочого простору: застосування меблів з можливістю регулювання, правильне розташування моніторів та м'яке освітлення, що допомагає знижувати фізичне навантаження і сприяє збереженню концентрації.

Також велике значення має організація простору без відволікань, таких як зайві предмети або надмірний шум. Це дозволяє створити умови для високої продуктивності. Важливо, щоб усі необхідні інструменти, такі як клавіатура, миша та інші аксесуари, були завжди під рукою, що допомагає уникати зайвих рухів і знижує втому.

Забезпечення комфортних умов праці значно знижує рівень стресу, що, в свою чергу, позитивно впливає на продуктивність та здоров'я співробітників. Не менш важливою є організація регулярних перерв для фізичних вправ, які сприяють зняттю напруги і підтримці фізичного та психічного здоров'я. Крім того, необхідно проводити навчання співробітників щодо ергономіки і правильного використання робочих місць, що допомагає мінімізувати негативний вплив стресових ситуацій на їхню діяльність.

У загальному підсумку, добре організоване робоче середовище сприяє підвищенню ефективності праці та зниженню ймовірності виникнення травм. Врахування ергономічних аспектів, зниження рівня стресу і створення комфортних фізичних умов є важливими складовими успішної роботи та розробки програмного забезпечення [7].

2 Характеристика підприємства

2.1 Загальна характеристика навчального закладу

Коледж розпочав свою діяльність 25 жовтня 1944 року, коли був заснований як Дніпропетровський автомеханічний технікум згідно з постановою Народного Комісара середнього машинобудування СРСР від 31 липня 1944 року. Його метою була підготовка кадрів для майбутнього автомобільного заводу в Дніпропетровську.

Першим директором був Дмитро Іванович Шандра, а завідуючою навчальною частиною – Віра Харлампієвна Шаповалова, які організували приймальну комісію. Спочатку технікум працював в школі №81, а пізніше отримав приміщення ремісничого училища.

У 1946 році було створено 11 навчальних кабінетів, а також відкрито перші навчально-виробничі майстерні. До 1947 року підготовка проводилася за спеціальностями: ковальське і ливарне виробництво, холодна обробка металів, інструментальне виробництво та збирання автомашин. У 1947 році відбувся перший випуск спеціалістів.

31 липня 1947 року технікум очолив Назарій Омелянович Клебанський. Було відкрито вечірнє відділення і введено спеціальність „Електрообладнання промислових підприємств”. З 1951 року технікум змінив профіль на підготовку фахівців у галузі ракетно-космічної техніки.

Після реконструкції в 1951 році навчальні будівлі були оснащені актовим залом, електротехнічною лабораторією та спеціалізованими кабінетами. Також були введені нові спеціальності: двигуни літальних апаратів, корпуси літальних апаратів, прилади і системи управління.

У 1962 році технікум переїхав до нового корпусу на вулиці Більшовицькій. До кінця 1960-х років були побудовані навчально-виробничий корпус та гуртожиток, створено кабінети і лабораторії для спеціальності „Експлуатація та наладка верстатів з програмним управлінням”.

З 1970-х років у технікумі активно впроваджувалися новітні технології: навчальні електронні обчислювальні машини, програмовані верстати і лабораторії автоматизації. У 1981 році було завершено будівництво нового навчального корпусу площею понад 3000 м².

У 1991 році технікум отримав нову назву — Дніпропетровський технікум ракетно-космічного машинобудування, та увійшов до складу Дніпропетровського національного університету. З 1997 року навчальний заклад здобув статус коледжу.

На сьогодні коледж продовжує підготовку фахівців у галузі машинобудування, ракетно-космічної техніки і суміжних спеціальностей, активно запроваджує інноваційні освітні технології та співпрацює з провідними підприємствами галузі [8].

2.2 Організаційна структура підприємства

Організаційно-штатна структура закладу зображено на рисунку 2.2.1.

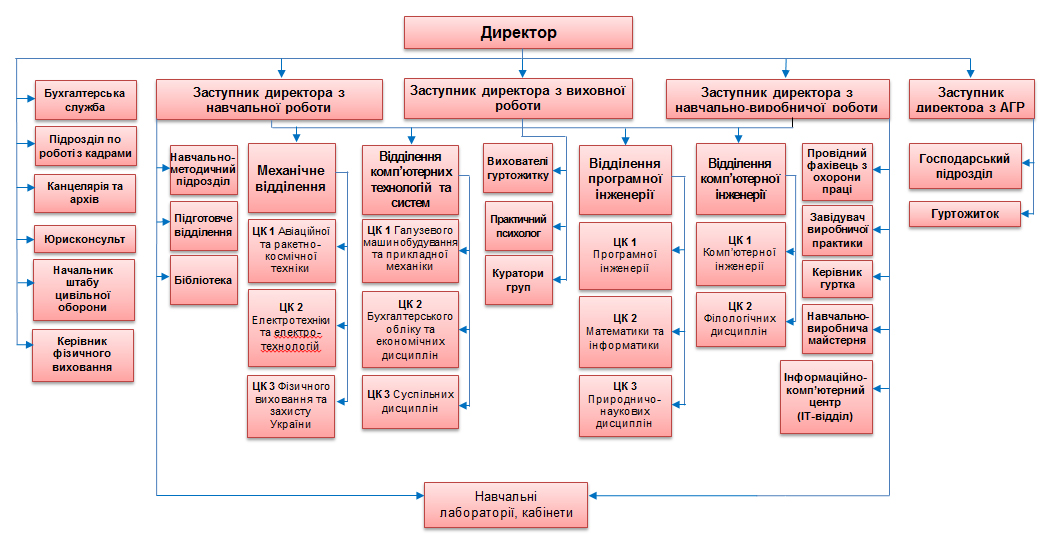


Рисунок 2.2.1 – Організаційно-штатна структура закладу

2.3 Обов’язки посадових осіб закладу

2.3.1 Посадові обов’язки директора коледжу

Директор навчального закладу є головною посадовою особою, відповідальною за стратегічне управління і координацію діяльності навчального закладу, а також за його успішну роботу перед засновниками та державними органами. Основне завдання директора — створення умов для високоякісного навчання, виховання і професійного розвитку студентів та колективу, відповідно до державних стандартів освіти.

Однією з основних функцій є організація роботи педагогічного колективу, яка включає призначення викладачів, завідувачів відділень, заступників, а також формування кадрової політики. Директор має забезпечити відповідність освітніх програм сучасним вимогам, впроваджувати інноваційні технології, підтримувати партнерські зв'язки з іншими навчальними закладами, підприємствами і міжнародними організаціями.

Директор відповідає за складання бюджету, розподіл фінансових ресурсів, організацію тендерів, контроль за виконанням кошторису та розвиток матеріально-технічної бази закладу. Його завданням є забезпечення належного рівня інфраструктури, зокрема, оснащення навчальних аудиторій, лабораторій, майстерень і спортивних залів сучасним обладнанням.

Окрім адміністративних обов'язків, директор виконує роль представника навчального закладу у зовнішньому середовищі: бере участь у нарадах, конференціях, форумах, встановлює ділові контакти, відстоює інтереси студентів та колективу в державних органах влади. Він також організовує моніторинг якості освіти, аналізує результати роботи закладу і впроваджує корективи для покращення показників.

Особлива увага приділяється питанням безпеки студентів та працівників, розробці заходів із запобігання надзвичайним ситуаціям, забезпеченню охорони праці, а також створенню сприятливого морально-психологічного клімату в колективі. Директор несе повну відповідальність за виконання законодавчих норм у сфері освіти, підготовку звітної документації, а також організацію підготовки закладу до акредитаційних перевірок і аудиту [9].

2.3.2 Посадові обов’язки заступника директора з навчальної роботи коледжу

Заступник директора з навчальної роботи є ключовою посадовою особою, відповідальною за організацію навчального процесу, його якість та відповідність актуальним освітнім стандартам. Основне завдання цієї посадової особи — розробка, впровадження і контроль за виконанням навчальних планів, програм та розкладів занять.

Заступник відповідає за координацію діяльності викладачів, сприяє підвищенню їх кваліфікації, організовує методичні семінари, конференції і педагогічні ради. Важливою частиною роботи є впровадження сучасних методів навчання, таких як інтерактивні заняття, проектний підхід та дистанційні технології. Заступник активно підтримує інновації, залучає викладачів до створення електронних навчальних матеріалів, організовує курси для підвищення кваліфікації.

Однією з основних функцій є контроль за відвідуваністю і успішністю студентів. Заступник аналізує навчальні результати, вживає заходів для поліпшення академічної підготовки, дає рекомендації щодо усунення прогалин у знаннях. Він також координує взаємодію між студентами, викладачами та батьками з питань навчальної діяльності.

Крім того, важливим напрямом є підготовка до атестацій, акредитацій і ліцензування спеціальностей. Заступник забезпечує якісне складання документації та створює умови для проходження закладом всіх необхідних перевірок. Ця посадова особа також займається моніторингом освітніх тенденцій і адаптацією навчальних програм до потреб ринку праці [10].

2.3.3 Посадові обов’язки заступника директора з навчально-виробничої роботи коледжу

Заступник директора з навчально-виробничої роботи є відповідальним за взаємодію між навчальним процесом та виробничими практиками, організацію навчальних і виробничих процесів в межах коледжу. Його завдання включають розробку програм виробничої практики, взаємодію з підприємствами та організаціями для забезпечення студентів місцями для практики.

Основною функцією є контроль за проведенням навчальних і виробничих практик, а також моніторингом їх ефективності та результативності. Заступник активно співпрацює з підприємствами, розвиваючи нові можливості для працевлаштування випускників і покращення практичного досвіду студентів.

Важливою частиною його роботи є організація атестацій та сертифікацій для студентів, проведення моніторингу результатів виробничих практик і аналіз потреб у вдосконаленні навчальних планів. Заступник також займається вдосконаленням навчальних та виробничих програм відповідно до вимог ринку праці [11].

2.3.4 Посадові обов’язки заступника директора з адміністративно-господарської роботи коледжу

Заступник директора з адміністративно-господарської роботи відповідає за збереження і розвиток матеріально-технічної інфраструктури навчального закладу, гарантування відповідності умов для навчання та роботи чинним стандартам безпеки, санітарії і технічним вимогам. Головною метою є забезпечення оптимальних умов для учнів та працівників, з урахуванням усіх аспектів їх діяльності.

Основною функцією є моніторинг стану будівель, лабораторій, аудиторій, спортивних об'єктів і гуртожитків, а також координація роботи всіх підрозділів, які відповідають за збереження закладу в належному стані. Заступник організовує роботу технічного персоналу, що включає спеціалістів з прибирання, охорони, обслуговування електрики та сантехніки.

Завдання також включають складання планів щодо ремонтів і контролю за їх виконанням, організацію закупівлі необхідних матеріалів, а також забезпечення енергоефективності та раціонального використання ресурсів. Особлива увага приділяється забезпеченню безпеки, включаючи навчання з питань пожежної безпеки і технічних стандартів.

У числі інших обов’язків — організація роботи з інвентаризації майна, підготовка документації для перевірок та аудитів, співпраця з підрядниками [12].

2.3.5 Посадові обов’язки завідувача відділенням коледжу

Завідувач відділенням здійснює керівництво навчальним процесом і адміністративною діяльністю структурного підрозділу, відповідаючи за забезпечення якості навчання та ефективну роботу викладачів. Він організовує планування і реалізацію навчальних програм та розкладів занять.

Завідувач виконує функції контролю за успішністю студентів, організовує атестації, сприяє підвищенню дисципліни і якості знань серед учнів. Усі питання, що стосуються академічної заборгованості студентів, також знаходяться під його контролем, та він розробляє стратегії для покращення успішності.

Завідувач відділенням відповідає за організацію професійного розвитку викладачів, проводить семінари і курси підвищення кваліфікації для персоналу. Також він активно реалізує інноваційні методи навчання та організовує створення нових навчальних матеріалів.

Ця посадова особа забезпечує співпрацю з підприємствами і іншими установами для практичного навчання студентів, а також займається підготовкою професійних звітів та організацією заходів з профорієнтації [13].

2.3.6 Посадові обов’язки викладача коледжу

Викладач є важливим елементом освітнього процесу, який відповідає за організацію навчання, розвиток і виховання студентів. Його основне завдання — забезпечення навчання, яке сприяє розвитку ключових знань та навичок, необхідних студентам для професійної діяльності в майбутньому.

Викладач проводить лекції, семінари, практичні і лабораторні заняття, розробляє навчальні матеріали, організовує тести та завдання для самостійної роботи, а також займається оцінюванням знань студентів. Важливим аспектом його діяльності є індивідуальний підхід до студентів, пояснення складних тем і підтримка їх мотивації до навчання.

Однією з важливих функцій викладача є оцінювання студентських знань через різні форми: контрольні роботи, тести, заліки та екзамени. Викладач бере участь у розробці навчальних планів і програм, а також інтегрує новітні методи та інформаційні технології в навчальний процес.

Важливою частиною роботи викладача є виховний процес, що включає формування у студентів відповідальності, поваги до законів і моральних принципів, а також участь у позанавчальних заходах. Викладач також надає консультації студентам з освітніх та особистих питань.

Регулярне підвищення кваліфікації через участь у семінарах, тренінгах і конференціях є важливою складовою його професійного розвитку. Його діяльність має безпосередній вплив на рівень підготовки студентів та на престиж навчального закладу [14].

3 Практика на робочому місці

3.1 Календарні терміни виконання завдань

TEXT зображено у таблиці 3.1.1.

Тблиця 3.1.1 – Графік проходження практики

|  |  |
| --- | --- |
| Тиждень | Виконана робота |
| 1 | – проведено аналіз небезпечних та шкідливих чинників проєктованого технологічного процесу, об’єкту, системи та пристроїв; – пройдено інструктаж з інженерно-технічних заходів з охорони праці; – пройдено інструктаж з пожежної профілактики; – пройдено інструктаж із заходів з ергономіки; – ознайомлено з діяльністю коледжу; – ознайомлено із структурою коледжу; – ознайомлено з обов’язками посадових осіб коледжу; – пройдено інструктаж з інтрументами розробки (Vim, Clang, CMake); – аналіз алгоритмів необхідних для написання проекту; – розробка необхідних утиліт для роботи з файлами; – відлагодження та тестування перших написаних компонентів проекту. |
| 2 | – аналіз помилок пам’яті (buffer overflow, memory leaks); – освоєння принципів побудови власних бібліотек; – вивчення стандартів офорлмення коду clang-format; – розробка основної логіки проекту; – інтеграція утиліт з основною програмою. |
| 3 | – тестування та відлагодження інтегрованих компонентів; – аналіз та виправлення помилок, виявлених під час тестування; – перевірка функціональності на різних етапах реалізації; – початкова перевірка проекту на відповідність вимогам; – завершення основної розробки програмного забезпечення; – остаточне тестування та виправлення помилок. |
| 4 | – підготовка звіту про виконану роботу; – підготовка проекту до захисту; – проведення фінальних перевірок відповідності проекту вимогам; – написання документації до проєкту; – оформлення звіту за результатами проходження практики. |

3.2 Характеристика технічного і програмного обладнання робочого місця

Під час проходження практики використовується ноутбук Dell Latitude 5490, який має наступні технічні характеристики:

– процесор: intel core i5-8250u;

– дисплей: 14-дюймовий дисплей full hd;

– об’єм оперативної пам’яті: 8 гб;

– об’єм пам’яті дисків: ssd 256 гб;

– графічна карта: intel uhd graphics 620.

3.3 Засоби дистанційного навчання в малих групах

Під час проходження практики були застосовані методи організації роботи малих груп у дистанційному форматі для ефективного навчання та співпраці між учасниками.

Для організації роботи малих груп було вибрано наступні інструменти:

– платформа для обговорень: slack — для забезпечення комунікації між учасниками групи та обміну інформацією в реальному часі;

– система керування версіями коду: github — для зберігання, обміну та колективної роботи над проектами та кодом.

Організація малих груп дозволила досягти наступних результатів:

– покращення залучення учасників до навчального процесу, забезпечуючи більшу активність;

– швидке виявлення проблем у розумінні матеріалу завдяки взаємодії з одногрупниками;

– розвиток командної роботи та обміну знаннями серед учасників групи.

Основними факторами успіху в роботі малих груп стали ефективна комунікація між учасниками, використання сучасних технологій і регулярний зворотний зв’язок від викладачів. Це дозволило покращити якість освітнього процесу та спростило вирішення практичних завдань під час практики.

3.3.1 Біль детальніше про Slack

Slack — це популярна платформа для комунікації і співпраці, що забезпечує реальний час для організації робочих процесів в командах. Вона дозволяє ефективно організовувати обговорення, обмінюватися файлами та проводити відеоконференції, що підвищує продуктивність команди. Slack інтегрується з багатьма інструментами, такими як Google Drive, GitHub, Trello і інші, що робить її потужним інструментом для командної роботи.

Платформа також забезпечує зручне використання тематичних каналів для обміну інформацією, що дозволяє зберігати важливі дані в структурованому вигляді, доступному для всіх учасників проекту [15].

Основні функції Slack:

– обмін повідомленнями в реальному часі;

– створення каналів для різних проектів та тем;

– інтеграція з іншими сервісами (google drive, github, trello);

– організація відео- та голосових дзвінків;

– завантаження документів і файлів;

– пошук по історії повідомлень;

– реакції на повідомлення через смайли;

– автоматизація через боти та команди.

Переваги використання Slack для учасників:

– доступ до спеціалізованих каналів для обговорення проектів;

– можливість обміну файлами та документацією;

– організація відео- та голосових дзвінків для швидких обговорень;

– забезпечення швидкої та ефективної комунікації серед учасників;

– інтеграція з іншими інструментами для покращення роботи над проектами;

– можливість задавати питання та отримувати зворотний зв'язок від колег та керівників;

– доступ до історії повідомлень для пошуку потрібної інформації.

3.3.2 Біль детальніше про GitHub

GitHub — це популярна платформа для зберігання коду, що підтримує систему контролю версій Git [16]. Вона дає змогу розробникам зберігати проекти в репозиторіях, співпрацювати з колегами, обговорювати зміни в коді та інтегруватися з іншими інструментами для автоматизації розробки. GitHub забезпечує зручну командну роботу завдяки таким функціям, як створення pull requests, перегляд коду і моніторинг змін та версій [17]. Платформа активно використовується в навчальних закладах і організаціях для навчання студентів основам програмування та контролю версій.

Основні функції GitHub:

– створення і керування репозиторіями для збереження коду [18];

– контроль версій проекту та історії змін [19];

– створення та перегляд pull requests для колективної роботи [20];

– проведення обговорень і коментування коду;

– інтеграція з ci/cd системами для автоматизованого тестування;

– відстеження задач і помилок через issues [21].

У середовищі GitHub студенти можуть:

– зберігати і управляти своїми проектами;

– працювати з іншими студентами над спільними репозиторіями;

– користуватися інструментами для контролю версій і зміни коду;

– коментувати та переглядати код інших учасників;

– відновлювати попередні версії проекту завдяки історії змін;

– налаштовувати автоматичне тестування та перевірку коду;

– спільно працювати над документацією і проектними матеріалами.

4 ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

4.1 Програмне забезпечення

4.1.1 Ubuntu

Ubuntu — це одна з найбільш відомих операційних систем на основі Linux, яка набуває популярності серед розробників завдяки своїй простоті і великій кількості інструментів для програмування. Ця система включає безліч бібліотек, компіляторів та інших необхідних утиліт, що робить її ідеальною для створення програмного забезпечення. [22].

4.1.2 Vim

Vim — це надзвичайно ефективний текстовий редактор, що використовується для програмування, редагування файлів і налаштування конфігурацій. Цей редактор побудований на основі старого Vi та спроектований для оптимізації процесу роботи з кодом через використання команд і макросів. Завдяки численним плагінам та можливості інтеграції з іншими інструментами, Vim є потужним інструментом, що забезпечує високу швидкість і гнучкість роботи. [23].

Функціональні можливості Vim включають:

– підтримка численних мов програмування, таких як c, c++, python, java, javascript, ruby і багатьох інших [24];

– можливість тонкої настройки через конфігураційні файли (vimrc) [25];

– інтеграція з системами контролю версій (git) через плагіни [26];

– використання регулярних виразів для пошуку і заміни тексту [27];

– розширена функціональність для роботи з макросами та командним редагуванням [24];

– підтримка обробки великих файлів [24];

– можливість роботи з кількома файлами одночасно завдяки використанню вкладок [24];

– інтеграція з компіляторами та інструментами для налагодження коду [24].

4.1.3 Clang

Clang — це компілятор для мов C, C++ та Objective-C, який є частиною проекту LLVM. Він відомий своєю швидкістю компіляції і розвиненими інструментами для діагностики помилок, оптимізації коду та інших задач, що важливі для розробників. [28].

4.1.4 CMake

CMake — це кросплатформенний інструмент, який автоматизує процес побудови програмного забезпечення. Він дозволяє генерувати файли компіляції, враховуючи особливості різних операційних систем, що робить його ідеальним вибором для великих і складних проектів. [29].

4.2 Опис проєкту

Метою цього проєкту є створення системи управління файлами, яка включатиме в себе набір функцій для збереження, перевірки та організації файлів в певних каталогах, а також для обробки помилок, що можуть виникнути в процесі роботи з файлами. Система має працювати на мові програмування C і включати в себе модулі для роботи з метаданими та утилітами для обробки файлів.

4.3 Постановка завдання

Модуль управління файлами (file\_storage):

а) створення функцій для роботи з файлами, таких як перевірка наявності файлів, збереження нових файлів, створення директорій та видалення файлів:

1) функції повинні забезпечувати унікальні імена для файлів при їх збереженні, щоб уникнути перезапису існуючих файлів;

2) модуль має включати обробку помилок, що можуть виникнути під час роботи з файлами, таких як відсутність файлів або прав на доступ.

б) модуль обробки помилок (file\_storage\_errors):

1) реалізація функцій для обробки та виведення помилок при виконанні операцій з файлами, наприклад, неуспішне створення директорії чи відсутність потрібного файлу.

в) модуль метаданих (metadata):

1) створення функцій для обробки метаданих файлів, наприклад, зчитування розміру, дати створення, модифікації та інших атрибутів файлів.

г) утиліти (utils):

1) реалізація допоміжних функцій для полегшення роботи з файлами та метаданими, наприклад, генерація унікальних імен для файлів чи обробка різних форматів даних.

д) тести (tests):

1) реалізація набору тестів для перевірки коректності роботи всіх модулів;

2) тести повинні перевіряти основні функціональні можливості, такі як створення/видалення файлів, перевірка коректності збереження файлів з унікальними іменами, а також правильність обробки помилок.

4.4 Структура проєкту

Вміст файлу PRACTICE\_PROJECT\_STRUCTURE.md відображено у листингу 4.4.1.

Лістинг 4.4.1 – Вміст PRACTICE\_PROJECT\_STRUCTURE.md

├── CMakeLists.txt # Main build configuration file  
├── build # Directory for build output  
├── include # Header files  
│ ├── file\_storage.h # Header for file\_storage module  
│ ├── file\_storage\_errors.h # Header for error handling module  
│ ├── metadata.h # Header for metadata module  
│ └── utils.h # Header for utility functions  
├── src # Source code files  
│ ├── file\_storage.c # Implementation of file\_storage functions  
│ ├── metadata.c # Implementation of metadata functions  
│ └── utils.c # Implementation of utility functions  
└── tests # Test files  
 ├── file\_storage\_test.c # Tests for file\_storage module  
 ├── metadata\_test.c # Tests for metadata module  
 └── utils\_test.c # Tests for utility functions

4.5 Вимоги до реалізації

Використовувати мову програмування C:

– застосувати принципи модульності та організації коду в окремі функції та модулі;

– забезпечити обробку помилок та коректну взаємодію між різними модулями;

– написати тестування для перевірки основних функціональних можливостей системи.

4.6 Хід виконання роботи

Вміст файлу PRACTICE\_PROJECT\_STRUCTURE.md відображено у листингу 4.6.2.

Лістинг 4.6.2 – Вміст PRACTICE\_PROJECT\_STRUCTURE.md

├── CMakeLists.txt # Main build configuration file  
├── build # Directory for build output  
├── include # Header files  
│ ├── file\_storage.h # Header for file\_storage module  
│ ├── file\_storage\_errors.h # Header for error handling module  
│ ├── metadata.h # Header for metadata module  
│ └── utils.h # Header for utility functions  
├── src # Source code files  
│ ├── file\_storage.c # Implementation of file\_storage functions  
│ ├── metadata.c # Implementation of metadata functions  
│ └── utils.c # Implementation of utility functions  
└── tests # Test files  
 ├── file\_storage\_test.c # Tests for file\_storage module  
 ├── metadata\_test.c # Tests for metadata module  
 └── utils\_test.c # Tests for utility functions

Вміст файлу CMakeLists.txt відображено у листингу 4.6.3.

Лістинг 4.6.3 – Вміст CMakeLists.txt

cmake\_minimum\_required(VERSION 3.10)  
project(FileStorageProject C)  
  
set(CMAKE\_C\_STANDARD 99)  
  
include\_directories(include)  
  
include\_directories("/opt/homebrew/opt/libsodium/include")  
  
link\_directories("/opt/homebrew/opt/libsodium/lib")  
  
add\_subdirectory(src)  
add\_subdirectory(tests)  
  
enable\_testing()

Вміст файлу utils.h відображено у листингу 4.6.4.

Лістинг 4.6.4 – Вміст utils.h

#ifndef UTILS\_H  
#define UTILS\_H  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <time.h>  
  
// directory  
int is\_directory(const char \*dirpath);  
  
// file  
int file\_exists(const char \*filepath);  
size\_t get\_file\_size(const char \*filepath);  
int remove\_file(const char \*filepath);  
int copy\_file(const char \*source, const char \*target);  
char\* get\_file\_extension(const char \*filepath);  
  
#endif // UTILS\_H

Вміст файлу uuid.h відображено у листингу 4.6.5.

Лістинг 4.6.5 – Вміст uuid.h

#ifndef UUID\_H  
#define UUID\_H  
  
#include <sodium.h>  
  
void generate\_uuid\_v4(char \*uuid) {  
 static int initialized = 0;  
  
 if (initialized == 0) {  
 if (sodium\_init() < 0) {  
 perror("libsodium initialization failed");  
 return;  
 }  
 initialized = 1;  
 }  
  
 unsigned char random\_bytes[16];  
  
 randombytes\_buf(random\_bytes, sizeof(random\_bytes));  
  
 random\_bytes[6] = (random\_bytes[6] & 0x0f) | 0x40;  
 random\_bytes[8] = (random\_bytes[8] & 0x3f) | 0x80;  
  
 snprintf(  
 uuid, 37, "%02x%02x%02x%02x-%02x%02x-%02x%02x-%02x%02x-%02x%02x%02x%02x",  
 random\_bytes[0], random\_bytes[1], random\_bytes[2], random\_bytes[3],  
 random\_bytes[4], random\_bytes[5], random\_bytes[6], random\_bytes[7],  
 random\_bytes[8], random\_bytes[9], random\_bytes[10], random\_bytes[11],  
 random\_bytes[12], random\_bytes[13], random\_bytes[14], random\_bytes[15]);  
}  
  
#endif UUID\_H

Вміст файлу file\_storage.h відображено у листингу 4.6.6.

Лістинг 4.6.6 – Вміст file\_storage.h

#ifndef FILE\_STORAGE\_H  
#define FILE\_STORAGE\_H  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#define MAX\_PATH\_LEN 256  
#define BUFFER\_SIZE 1024  
#define MAX\_FILE\_SIZE 5242880 // 5 MB  
  
int validate\_file(const char \*file\_path);  
int create\_directory(const char \*directory\_name);  
char\* generate\_unique\_path(const char \*directory, const char \*file\_name);  
int save\_file(const char \*source\_path, const char \*target\_path);  
  
#endif // FILE\_STORAGE\_H

Вміст файлу metadata.h відображено у листингу 4.6.7.

Лістинг 4.6.7 – Вміст metadata.h

#ifndef METADATA\_H  
#define METADATA\_H  
  
#include <stddef.h>  
#include <time.h>  
  
#define MAX\_FILENAME\_LEN 256  
#define MAX\_UUID\_LEN 37  
  
typedef struct {  
 char uuid[MAX\_UUID\_LEN];  
 char filename[MAX\_FILENAME\_LEN];  
 size\_t size;  
 time\_t created\_at;  
 time\_t modified\_at;  
} FileMetadata;  
  
int save\_metadata(const char \*filepath, const FileMetadata \*metadata);  
int load\_metadata(const char \*filepath, FileMetadata \*metadata);  
int update\_metadata(const char \*filepath, const FileMetadata \*metadata);   
void print\_metadata(const FileMetadata \*metadata);  
  
#endif // METADATA\_H

Вміст файлу file\_storage\_errors.h відображено у листингу 4.6.8.

Лістинг 4.6.8 – Вміст file\_storage\_errors.h

#ifndef FILE\_STORAGE\_ERRORS\_H  
#define FILE\_STORAGE\_ERRORS\_H  
  
#define FILE\_ERROR\_STATUS -1  
#define FILE\_EMPTY\_OR\_NOT\_EXIST -2  
#define FILE\_TOO\_LARGE -3  
  
#define DIR\_ERROR\_CREATE -1  
  
#define PATH\_TOO\_LONG "Path exceeds maximum length"  
#define MEMORY\_ALLOCATION\_ERROR "Memory allocation error"  
  
#define FILE\_ERROR\_OPEN\_SOURCE -1  
#define FILE\_ERROR\_OPEN\_TARGET -2  
#define FILE\_ERROR\_WRITE -3  
  
#endif // FILE\_STORAGE\_ERRORS\_H

Вміст файлу metadata\_test.c відображено у листингу 4.6.9.

Лістинг 4.6.9 – Вміст metadata\_test.c

#include <stdio.h>  
#include <assert.h>  
#include <string.h>  
#include <stdlib.h>  
#include <time.h>  
#include "metadata.h"  
  
#define TEST\_METADATA\_FILE "test\_metadata.dat"  
  
FileMetadata create\_test\_metadata() {  
 FileMetadata metadata;  
 strncpy(metadata.uuid, "123e4567-e89b-12d3-a456-426614174000", MAX\_UUID\_LEN);  
 strncpy(metadata.filename, "test\_file.txt", MAX\_FILENAME\_LEN);  
 metadata.size = 1234;  
 metadata.created\_at = time(NULL);  
 metadata.modified\_at = time(NULL);  
 return metadata;  
}  
  
void test\_save\_metadata() {  
 FileMetadata metadata = create\_test\_metadata();  
  
 int result = save\_metadata(TEST\_METADATA\_FILE, &metadata);  
 assert(result == 1);  
  
 printf("save\_metadata test passed\n");  
}  
  
void test\_load\_metadata() {  
 FileMetadata metadata = create\_test\_metadata();  
 save\_metadata(TEST\_METADATA\_FILE, &metadata);  
  
 FileMetadata loaded\_metadata;  
 int result = load\_metadata(TEST\_METADATA\_FILE, &loaded\_metadata);  
 assert(result == 1);  
 assert(strcmp(metadata.uuid, loaded\_metadata.uuid) == 0);  
 assert(strcmp(metadata.filename, loaded\_metadata.filename) == 0);  
 assert(metadata.size == loaded\_metadata.size);  
 assert(metadata.created\_at == loaded\_metadata.created\_at);  
 assert(metadata.modified\_at == loaded\_metadata.modified\_at);  
  
 printf("load\_metadata test passed\n");  
}  
  
void test\_update\_metadata() {  
 FileMetadata metadata = create\_test\_metadata();  
 save\_metadata(TEST\_METADATA\_FILE, &metadata);  
  
 FileMetadata updated\_metadata = metadata;  
 strncpy(updated\_metadata.filename, "updated\_file.txt", MAX\_FILENAME\_LEN);  
 updated\_metadata.size = 5678;  
 updated\_metadata.modified\_at = time(NULL);  
  
 int result = update\_metadata(TEST\_METADATA\_FILE, &updated\_metadata);  
 assert(result == 1);  
  
 FileMetadata loaded\_metadata;  
 load\_metadata(TEST\_METADATA\_FILE, &loaded\_metadata);  
 assert(strcmp(updated\_metadata.filename, loaded\_metadata.filename) == 0);  
 assert(updated\_metadata.size == loaded\_metadata.size);  
 assert(updated\_metadata.modified\_at == loaded\_metadata.modified\_at);  
  
 printf("update\_metadata test passed\n");  
}  
  
void cleanup\_metadata\_file() {  
 remove(TEST\_METADATA\_FILE);  
}  
  
void run\_metadata\_tests() {  
 test\_save\_metadata();  
 test\_load\_metadata();  
 test\_update\_metadata();  
  
 cleanup\_metadata\_file();  
}

Вміст файлу CMakeLists.txt відображено у листингу 4.6.10.

Лістинг 4.6.10 – Вміст CMakeLists.txt

add\_executable(file\_storage\_tests  
 file\_storage\_test.c  
 metadata\_test.c  
 utils\_test.c  
 main.c  
)  
  
target\_link\_libraries(file\_storage\_tests  
 file\_storage  
)  
  
add\_test(NAME file\_storage\_tests COMMAND file\_storage\_tests)

Вміст файлу utils\_test.c відображено у листингу 4.6.11.

Лістинг 4.6.11 – Вміст utils\_test.c

#include <stdio.h>  
#include <stdlib.h>  
#include <sys/stat.h>  
#include <unistd.h>  
#include <string.h>  
#include <assert.h>  
#include "utils.h"  
  
void test\_is\_directory() {  
 const char \*dirpath = "test\_dir";  
 mkdir(dirpath, 0755);  
  
 assert(is\_directory(dirpath) == 1);  
  
 rmdir(dirpath);  
}  
  
void test\_file\_exists() {  
 const char \*filepath = "test\_file.txt";  
 FILE \*file = fopen(filepath, "w");  
 fprintf(file, "Test data\n");  
 fclose(file);  
  
 assert(file\_exists(filepath) == 1);  
  
 remove(filepath);  
}  
  
void test\_get\_file\_size() {  
 const char \*filepath = "test\_file.txt";  
 FILE \*file = fopen(filepath, "w");  
 fprintf(file, "Test data\n");  
 fclose(file);  
  
 assert(get\_file\_size(filepath) == 10); // 10 characters  
  
 remove(filepath);  
}  
  
void test\_remove\_file() {  
 const char \*filepath = "test\_file.txt";  
 FILE \*file = fopen(filepath, "w");  
 fprintf(file, "Test data\n");  
 fclose(file);  
  
 assert(remove\_file(filepath) == 1);  
 assert(file\_exists(filepath) == 0); // File should be removed  
}  
  
void test\_copy\_file() {  
 const char \*source = "source\_file.txt";  
 FILE \*file = fopen(source, "w");  
 fprintf(file, "Test data\n");  
 fclose(file);  
  
 const char \*target = "target\_file.txt";  
   
 assert(copy\_file(source, target) == 1);  
 assert(file\_exists(target) == 1);  
 assert(get\_file\_size(source) == get\_file\_size(target));  
  
 remove(source);  
 remove(target);  
}  
  
void test\_get\_file\_extension() {  
 const char \*filepath1 = "file.txt";  
 assert(strcmp(get\_file\_extension(filepath1), "txt") == 0);  
  
 const char \*filepath2 = "file";  
 assert(get\_file\_extension(filepath2) == NULL);  
  
 const char \*filepath3 = ".gitignore";  
 assert(get\_file\_extension(filepath3) == NULL);  
}  
  
void run\_utils\_tests() {  
 test\_is\_directory();  
 test\_file\_exists();  
 test\_get\_file\_size();  
 test\_remove\_file();  
 test\_copy\_file();  
 test\_get\_file\_extension();  
}

Вміст файлу main.c відображено у листингу 4.6.12.

Лістинг 4.6.12 – Вміст main.c

#include <stdio.h>  
#include <assert.h>  
  
extern void run\_file\_storage\_tests();  
extern void run\_metadata\_tests();  
extern void run\_utils\_tests();  
  
int main() {  
 run\_file\_storage\_tests();  
 printf("File storage tests passed!\n");  
  
 run\_metadata\_tests();  
 printf("Metadata tests passed!\n");  
  
 run\_utils\_tests();  
 printf("Utils tests passed!\n");  
   
 printf("\nAll tests passed!\n");  
 return 0;  
}

Вміст файлу file\_storage\_test.c відображено у листингу 4.6.13.

Лістинг 4.6.13 – Вміст file\_storage\_test.c

#include "file\_storage.h"  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <dirent.h>  
#include <unistd.h>  
  
#define TEST\_DIRECTORY "test\_catalog"  
#define TEST\_TARGET\_TEST\_FILE\_PATH "target\_test\_file.txt"  
#define TEST\_SOURCE\_TEST\_FILE\_PATH "source\_test\_file.txt"  
  
void create\_test\_file(const char \*file\_name) {  
 FILE \*file = fopen(file\_name, "wb");  
 if (file) {  
 const char \*data = "Test data for file storage.";  
 fwrite(data, 1, strlen(data), file);  
 fclose(file);  
 }  
}  
  
int remove\_directory\_if\_empty(const char \*directory\_name) {  
 DIR \*dir = opendir(directory\_name);  
 if (dir) {  
 struct dirent \*entry;  
 int is\_empty = 1;  
 while ((entry = readdir(dir)) != NULL) {  
 if (entry->d\_name[0] != '.') {  
 is\_empty = 0;  
 break;  
 }  
 }  
 closedir(dir);  
  
 if (is\_empty == 1) {  
 if (rmdir(directory\_name) != 0) {  
 return -2; // Error removing directory  
 }  
 return 1;  
 }  
 } else {  
 return -1; // Error opening directory  
 }  
 return -3; // Directory is not empty  
}  
  
void run\_file\_storage\_tests() {  
 int result;  
  
 result = create\_directory(TEST\_DIRECTORY);  
 if (result != 1) {  
 return;  
 }  
  
 create\_test\_file(TEST\_SOURCE\_TEST\_FILE\_PATH);  
 result = validate\_file(TEST\_SOURCE\_TEST\_FILE\_PATH);  
 if (result != 1) {  
 remove(TEST\_SOURCE\_TEST\_FILE\_PATH);  
 return;  
 }  
  
 char \*unique\_path = generate\_unique\_path(TEST\_DIRECTORY, TEST\_TARGET\_TEST\_FILE\_PATH);  
 if (!unique\_path) {  
 remove(TEST\_SOURCE\_TEST\_FILE\_PATH);  
 return;  
 }  
  
 result = save\_file(TEST\_SOURCE\_TEST\_FILE\_PATH, unique\_path);  
  
 remove(TEST\_SOURCE\_TEST\_FILE\_PATH);  
  
 if (access(unique\_path, F\_OK) == 0) {  
 if (remove(unique\_path) != 0) {  
 perror("Error removing target file");  
 }  
 }  
  
 free(unique\_path);  
  
 DIR \*dir = opendir(TEST\_DIRECTORY);  
 if (dir) {  
 struct dirent \*entry;  
 while ((entry = readdir(dir)) != NULL) {  
 if (entry->d\_name[0] != '.') {  
 char file\_path[1024];  
 snprintf(file\_path, sizeof(file\_path), "%s/%s", TEST\_DIRECTORY, entry->d\_name);  
 if (remove(file\_path) != 0) {  
 perror("Error removing file in directory");  
 }  
 }  
 }  
 closedir(dir);  
 }  
  
 result = remove\_directory\_if\_empty(TEST\_DIRECTORY);  
}

Вміст файлу CMakeLists.txt відображено у листингу 4.6.14.

Лістинг 4.6.14 – Вміст CMakeLists.txt

add\_library(file\_storage STATIC  
 file\_storage.c  
 metadata.c  
 utils.c  
)

Вміст файлу utils.c відображено у листингу 4.6.15.

Лістинг 4.6.15 – Вміст utils.c

#include <stdio.h>  
#include <stdlib.h>  
#include <sys/stat.h>  
#include <unistd.h>  
#include <dirent.h>  
#include <string.h>  
  
int is\_directory(const char \*dirpath) {  
 struct stat statbuf;  
  
 if (stat(dirpath, &statbuf) != 0) {  
 return -1; // Path does not exist or error  
 }  
  
 return S\_ISDIR(statbuf.st\_mode) ? 1 : 0; // 1 if its a directory and 0 if if its not a directory  
}  
  
int file\_exists(const char \*filepath) {  
 struct stat statbuf;  
  
 if (stat(filepath, &statbuf) == 0) {  
 return 1;  
 } else {  
 return 0;  
 }  
}  
  
size\_t get\_file\_size(const char \*filepath) {  
 struct stat statbuf;  
  
 if (stat(filepath, &statbuf) == 0) {  
 return statbuf.st\_size;  
 } else {  
 return (size\_t)-1;  
 }  
}  
  
int remove\_file(const char \*filepath) {  
 if (file\_exists(filepath) == 0) {  
 return -1; // File does not exist  
 }  
  
 if (remove(filepath) == 0) {  
 return 1;  
 } else {  
 return -2; // Error while deleting file  
 }  
}  
  
int copy\_file(const char \*source, const char \*target) {  
 FILE \*source\_file = fopen(source, "rb");  
 if (source\_file == NULL) {  
 return -1; // Error opening source file  
 }  
  
 if (file\_exists(target)) {  
 return -2; // Target file is already exists  
 }  
  
 FILE \*target\_file = fopen(target, "wb");  
 if (target\_file == NULL) {  
 fclose(source\_file);  
 return -3; // Error opening target file  
 }  
  
 char buffer[1024];  
 size\_t bytes\_read;  
  
 while ((bytes\_read = fread(buffer, 1, sizeof(buffer), source\_file)) > 0) {  
 if (fwrite(buffer, 1, bytes\_read, target\_file) != bytes\_read) {  
 fclose(source\_file);  
 fclose(target\_file);  
 return -4; // Error writing to target file  
 }  
 }  
  
 if (ferror(source\_file)) {  
 fclose(source\_file);  
 fclose(target\_file);  
 return -5; // Error reading from source file  
 }  
  
 fclose(source\_file);  
 fclose(target\_file);  
  
 return 1;  
}  
  
const char\* get\_file\_extension(const char \*filepath) {  
 if (filepath == NULL || \*filepath == '\0') {  
 return NULL;  
 }  
  
 const char \*pos = strrchr(filepath, '.');  
 if (pos == NULL || pos == filepath) {  
 return NULL;  
 }  
  
 return pos + 1;  
}

Вміст файлу metadata.c відображено у листингу 4.6.16.

Лістинг 4.6.16 – Вміст metadata.c

#include <stdio.h>  
#include <string.h>  
#include "metadata.h"  
  
int save\_metadata(const char \*filepath, const FileMetadata \*metadata) {  
 if (filepath == NULL || metadata == NULL) {  
 perror("Invalid params");  
 return -1;  
 }  
  
 FILE \*file = fopen(filepath, "wb");  
 if (file == NULL) {  
 perror("Failed to open file");  
 return -1;  
 }  
  
 size\_t written = fwrite(metadata, sizeof(FileMetadata), 1, file);  
 if (written != 1) {  
 perror("Error writing to the file");  
 fclose(file);  
 return -1;  
 }  
  
 if (fclose(file) != 0) {  
 perror("File closure error");  
 return -1;  
 }  
  
 return 1;  
}  
  
int load\_metadata(const char \*filepath, FileMetadata \*metadata) {  
 if (filepath == NULL || metadata == NULL) {  
 perror("Invalid params");  
 return -1;  
 }  
  
 FILE \*file = fopen(filepath, "rb");  
 if (file == NULL) {  
 perror("Failed to open file");  
 return -1;  
 }  
  
 size\_t read = fread(metadata, sizeof(FileMetadata), 1, file);  
 if (read != 1) {  
 if (feof(file)) {  
 perror("File is empty");  
 } else {  
 perror("Error reading from the file");  
 }  
 fclose(file);  
 return -1;  
 }  
  
 if (fclose(file) != 0) {  
 perror("File closure error");  
 return -1;  
 }  
  
 return 1;  
}  
  
int update\_metadata(const char \*filepath, const FileMetadata \*metadata) {  
 if (filepath == NULL || metadata == NULL) {  
 perror("Invalid params");  
 return -1;  
 }  
  
 FILE \*file = fopen(filepath, "r+b");  
 if (file == NULL) {  
 perror("Failed to open file");  
 return -1;  
 }  
  
 FileMetadata current\_metadata;  
 size\_t read = fread(&current\_metadata, sizeof(FileMetadata), 1, file);  
 if (read != 1) {  
 if (feof(file)) {  
 fprintf(stderr, "File is empty or corrupted\n");  
 } else {  
 perror("Error reading from the file");  
 }  
 fclose(file);  
 return -1;  
 }  
  
 memcpy(&current\_metadata, metadata, sizeof(FileMetadata));  
  
 rewind(file);  
  
 size\_t written = fwrite(&current\_metadata, sizeof(FileMetadata), 1, file);  
 if (written != 1) {  
 perror("Error writing to the file");  
 fclose(file);  
 return -1;  
 }  
  
 if (fclose(file) != 0) {  
 perror("File closure error");  
 return -1;  
 }  
  
 return 1;  
}  
  
void print\_metadata(const FileMetadata \*metadata) {  
 if (metadata == NULL) {  
 perror("Invalid metadata");  
 return;  
 }  
  
 printf("File Metadata:\n");  
 printf(" UUID: %s\n", metadata->uuid);  
 printf(" Filename: %s\n", metadata->filename);  
 printf(" Size: %zu bytes\n", metadata->size);  
 printf(" Created at: %s", ctime(&metadata->created\_at));  
 printf(" Modified at: %s", ctime(&metadata->modified\_at));  
}

Вміст файлу file\_storage.c відображено у листингу 4.6.17.

Лістинг 4.6.17 – Вміст file\_storage.c

#include "file\_storage.h"  
#include "file\_storage\_errors.h"  
#include <sys/stat.h>  
#include <unistd.h>  
#include <fcntl.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
int validate\_file(const char \*file\_path) {  
 struct stat file\_info;  
   
 if (stat(file\_path, &file\_info) != 0) {  
 return FILE\_ERROR\_STATUS;  
 }  
  
 if (file\_info.st\_size <= 0) {  
 return FILE\_EMPTY\_OR\_NOT\_EXIST;  
 }  
  
 if (file\_info.st\_size > MAX\_FILE\_SIZE) {  
 return FILE\_TOO\_LARGE;  
 }  
  
 return 1;  
}  
  
int create\_directory(const char \*directory\_name) {  
 struct stat stat\_info;  
   
 if (stat(directory\_name, &stat\_info) == -1) {  
 if (mkdir(directory\_name, 0755) != 0) {  
 return DIR\_ERROR\_CREATE;  
 }  
 }  
 return 1;  
}  
  
char\* generate\_unique\_path(const char \*directory, const char \*file\_name) {  
 time\_t raw\_time = time(NULL);  
 char time\_str[20];  
   
 strftime(time\_str, sizeof(time\_str), "%Y%m%d\_%H%M%S", localtime(&raw\_time));  
  
 if (strlen(directory) + strlen(file\_name) + strlen(time\_str) + 2 > MAX\_PATH\_LEN) {  
 return PATH\_TOO\_LONG;  
 }  
  
 char \*unique\_path = (char\*)malloc(MAX\_PATH\_LEN);  
 if (!unique\_path) {  
 return MEMORY\_ALLOCATION\_ERROR;  
 }  
  
 snprintf(unique\_path, MAX\_PATH\_LEN, "%s/%s\_%s", directory, time\_str, file\_name);  
   
 return unique\_path;  
}  
  
int save\_file(const char \*source\_path, const char \*target\_path) {  
 FILE \*source\_file = fopen(source\_path, "rb");  
 if (!source\_file) {  
 return FILE\_ERROR\_OPEN\_SOURCE;  
 }  
  
 FILE \*target\_file = fopen(target\_path, "wb");  
 if (!target\_file) {  
 fclose(source\_file);  
 return FILE\_ERROR\_OPEN\_TARGET;  
 }  
  
 char buffer[BUFFER\_SIZE];  
 size\_t bytes;  
 while ((bytes = fread(buffer, 1, BUFFER\_SIZE, source\_file)) > 0) {  
 if (fwrite(buffer, 1, bytes, target\_file) != bytes) {  
 fclose(source\_file);  
 fclose(target\_file);  
 return FILE\_ERROR\_WRITE;  
 }  
 }  
  
 fclose(source\_file);  
 fclose(target\_file);  
  
 return 1;  
}

TEXT зображено на рисунку 4.6.1.

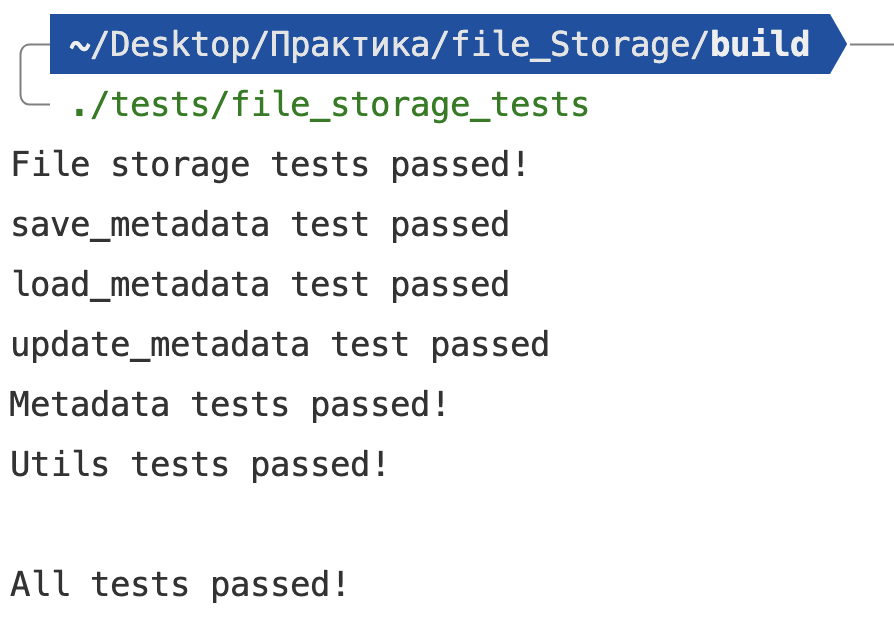


Рисунок 4.6.1 – TEXT

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про вищу освіту» https://zakon.rada.gov.ua/laws/show/1556-18

2. Закон України «Про фахову передвищу освіту» https://zakon.rada.gov.ua/laws/show/2745-19

3. Наказ МОН України №1006 від 21.09.2021 https://osvita.ua/legislation/faxova-peredvyshha-osvita/85816/

4. Вимоги до організації та обслуговування робочого місця https://studfile.net/preview/5437414/page:10/

5. Інженерно-технічні заходи з охорони праці https://studfile.net/preview/8968088/page:3/

6. Пожежна профілактика на робочому місці https://ohoronapraci.kiev.ua/article/news/pozezna-bezpeka-na-robocomu-misci

7. Заходи з ергономіки https://studfile.net/preview/9919354/page:9/

8. Загальна характеристика навчального закладу https://dkrkm.org.ua

9. Посадові обов’язки директора коледжу https://npu.edu.ua/images/file/viddil\_kadriv/dyrektor\_VPU.pdf

10. Посадові обов’язки заступника директора з навчальної роботи коледжу https://kkht.dp.ua/wp-content/uploads/2019/06/Заступника-директора-з-НР.pdf

11. Посадові обов’язки заступника директора з виробничої роботи https://hpslsamoilova.blogspot.com/p/blog-page\_11.html

12. Посадові обов’язки заступника директора з адміністративно-господарської роботи коледжу https://kkht.dp.ua/wp-content/uploads/2019/06/Поcадова-заступника-директора-з-АГР-2019.pdf

13. Посадові обов’язки завідувача відділенням коледжу https://profpressa.com/instructions/zaviduvach-viddilennia-u-koledzhi

14. Посадові обов’язки викладача коледжу https://npu.edu.ua/images/file/viddil\_kadriv/vykladach.pdf

15. Інтеграції Slack https://slack.com/integrations

16. GitHub https://github.com/home

17. GitHub Collaborating with pull requests https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests

18. GitHub Repositories https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/creating-a-new-repository

19. Git Version Control https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

20. GitHub Pull Requests https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests

21. GitHub Issues https://docs.github.com/en/github/managing-your-work-on-github/about-issues

22. Ubuntu https://ubuntu.com/about

23. Vim https://www.vim.org

24. Vim Documentation https://www.vim.org/docs.php

25. Vimrc Configuration https://vim.fandom.com/wiki/Vimrc

26. Vim Git Integration https://www.vimfromscratch.com/articles/using-git-from-vim

27. Vim Regular Expressions https://vim.fandom.com/wiki/Regular\_expressions

28. Clang Compiler https://clang.llvm.org/

29. CMake Documentation https://cmake.org/documentation/