

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Куижева Саида Казбековна
Должность: Ректор
Дата подписания: 24.10.2026 00:21:02
Уникальный программный ключ:
71183e1134ef9cfa69b206d480271b3c1a975e6f

ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ РАБОТ

ПО ПМ.02 Разработка и интеграция модулей программного обеспечения

для специальности 09.02.11 Разработка и управление программным
обеспечением

МДК.02.01 Разработка программных модулей

Практическая работа № 1. «Разработка модуля для работы с файловой системой и регулярными выражениями»

Время: 1,5 часа (90 мин).

Цель: Освоить навигацию по файловой системе, чтение/запись файлов и применение регулярных выражений для обработки текста (согласно ПЗ4 и ПЗ5).

- **Задание:** Создайте консольное приложение, которое считывает текстовый файл `input.txt` (объемом 1000+ строк), извлекает из него все корректные email-адреса и номера телефонов, и записывает их в новый файл `output.txt` в отсортированном виде.
- **Технология выполнения:**
 1. **Подготовка (10 мин):** Создать проект, подготовить тестовый файл `input.txt` с «мусорным» текстом.
 2. **Чтение файла (15 мин):** Использовать библиотеки ввода-вывода (например, `System.IO` в C#, `fs` в `Node.js` или встроенные функции `Python`) для потокового чтения файла.
 3. **Регулярные выражения (30 мин):** Написать регулярные выражения для email (например, `[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}`) и телефона. Применить их к каждой строке.
 4. **Обработка данных (15 мин):** Собрать найденные данные в коллекцию (список/массив), удалить дубликаты и отсортировать.
 5. **Запись файла (10 мин):** Записать результат в `output.txt`.
 6. **Тестирование (10 мин):** Проверить корректность работы на граничных случаях (отсутствие данных, неверные форматы).

Практическая работа № 2. «Разработка модуля с использованием паттернов проектирования (Singleton / Factory)»

Время: 1,5 часа (90 мин).

Цель: Применить паттерны проектирования для создания масштабируемой архитектуры модуля (согласно ПЗ3).

- **Задание:** Разработать модуль конфигурации приложения. Использовать паттерн **Singleton** для гарантии того, что у приложения будет только один экземпляр конфигурации, и паттерн **Factory** для создания различных типов логгеров (`FileLogger`, `ConsoleLogger`).
- **Технология выполнения:**
 1. **Проектирование (15 мин):** Нарисовать UML-диаграмму классов (в тетради или ПО).
 2. **Реализация Singleton (20 мин):** Написать класс `ConfigManager` с приватным конструктором и статическим методом получения экземпляра. Убедиться в потокобезопасности (при необходимости).
 3. **Реализация Factory (30 мин):** Создать интерфейс `ILogger` и классы, его реализующие. Написать фабрику `LoggerFactory`, которая возвращает нужный логгер по строковому ключу.
 4. **Интеграция (15 мин):** Написать тестовый сценарий, который берет настройки из `Singleton` и логирует события через созданный `Factory` объект.

5. **Защита (10 мин):** Продемонстрировать, что при повторном вызове Singleton возвращается тот же объект (сравнить хеш-коды).

МДК.02.02 Осуществление интеграции программных модулей

Практическая работа № 3. «Создание REST API приложения с реализацией CRUD-операций»

Время: 1,5 часа (90 мин).

Цель: Освоить разработку серверного API для взаимодействия с клиентами (согласно ПЗ2).

- **Задание:** Создать простейший REST API для управления списком задач (To-Do list). Реализовать методы: GET (получить все), POST (создать), PUT (обновить), DELETE (удалить).
- **Технология выполнения:**
 1. **Настройка окружения (15 мин):** Инициализировать проект (например, Node.js + Express, Python + Flask или C# + **ASP.NET** Core Web API). Создать структуру данных (in-memory массив или простую БД).
 2. **Реализация GET (15 мин):** Создать эндпоинт `/api/tasks`, возвращающий JSON-массив всех задач.
 3. **Реализация POST (20 мин):** Настроить прием JSON-тела запроса, валидацию данных и добавление новой задачи в хранилище.
 4. **Реализация PUT и DELETE (20 мин):** Настроить маршрутизацию с параметрами (`/api/tasks/:id`) для обновления и удаления конкретной записи.
 5. **Тестирование (20 мин):** Использовать инструмент Postman или встроенный Swagger (OpenAPI) для отправки запросов и проверки корректности ответов (статус-коды 200, 201, 404).

Практическая работа № 4. «Конфигурация и контейнеризация REST API приложения»

Время: 1,5 часа (90 мин).

Цель: Научиться выносить настройки в конфигурационные файлы и упаковывать приложение в Docker-контейнер (согласно ПЗ12 и ПЗ13).

- **Задание:** Настроить конфигурацию ранее созданного REST API (порт, хост, данные подключения) через переменные окружения. Упаковать приложение в Docker-контейнер и запустить его.
- **Технология выполнения:**
 1. **Конфигурация (20 мин):** Вынести параметры в файл `.env`. Настроить чтение этих переменных в коде приложения.
 2. **Логирование (15 мин):** Добавить в приложение базовое логирование входящих запросов (URL, метод, время отклика).
 3. **Создание Dockerfile (25 мин):** Написать Dockerfile (выбор базового образа, копирование зависимостей, копирование кода, команда запуска).
 4. **Сборка и запуск (20 мин):** Выполнить команды `docker build` и `docker run`, пробросив порты. Проверить доступность API.
 5. **Анализ (10 мин):** Проверить логи контейнера (`docker logs`), убедиться, что приложение использует переданные переменные окружения.

МДК.02.03 Поддержка и тестирование программных модулей

Практическая работа № 5. «Разработка модульных тестов (Unit Tests) методом белого ящика»

Время: 1,5 часа (90 мин).

Цель: Освоить написание модульных тестов и достижение покрытия кода (согласно ПЗ10 и ПЗ13).

- **Задание:** Написать функцию, вычисляющую скидку на товар в зависимости от суммы покупки (например, до 1000 – 0%, от 1000 до 5000 – 5%, свыше 5000 – 10%). Покрыть эту функцию модульными тестами (метод покрытия операторов и условий).
- **Технология выполнения:**
 1. **Написание функции (15 мин):** Реализовать логику на выбранном языке (Python, Java, C#).
 2. **Настройка фреймворка (15 мин):** Подключить библиотеку для тестирования (PyTest, JUnit или NUnit).
 3. **Тестирование операторов (20 мин):** Написать тесты, которые заставят выполняться каждую строку кода функции (сумма < 1000, сумма = 1000, сумма между 1000 и 5000, сумма = 5000, сумма > 5000).
 4. **Тестирование граничных условий (20 мин):** Написать тесты для граничных значений (0, отрицательные числа, максимально возможные значения).
 5. **Запуск и анализ (20 мин):** Запустить тесты, добиться 100% прохождения (Passed). Проанализировать отчет о покрытии кода (Code Coverage), если позволяет инструмент.

МДК.02.04 Математическое моделирование

Практическая работа № 6. «Решение задач линейного программирования симплексным методом»

Время: 1,5 часа (90 мин).

Цель: Освоить построение математической модели и решение оптимизационной задачи (согласно ПЗ2).

- **Задание:** Предприятие выпускает два вида продукции (А и Б). Известны затраты ресурсов на единицу продукции и общие запасы ресурсов. Необходимо составить математическую модель и найти план выпуска, максимизирующий прибыль.
- **Технология выполнения:**
 1. **Постановка задачи (15 мин):** Записать целевую функцию (максимизация прибыли) и систему ограничений (ресурсы, неотрицательность).
 2. **Приведение к канонической форме (15 мин):** Ввести дополнительные переменные для превращения неравенств в уравнения.
 3. **Построение симплекс-таблицы (20 мин):** Заполнить первую симплекс-таблицу.
 4. **Итерационный процесс (25 мин):** Выполнить шаги симплекс-метода (выбор разрешающего столбца и строки, пересчет таблицы) до достижения оптимального решения.
 5. **Проверка (15 мин):** Проверить решение с помощью инструментальных средств (например, надстройки «Поиск решения» в Excel или библиотеки `scipy.optimize` в Python).

Практическая работа № 7. «Решение нелинейных уравнений методом Ньютона»

Время: 1,5 часа (90 мин).

Цель: Реализовать численный метод для нахождения корней уравнений с заданной точностью (согласно ПЗЗ).

- **Задание:** Найти корень уравнения $f(x)=x^3-2x-5=0$ на отрезке $[2,3]$ с точностью $\varepsilon=0.0001$ методом Ньютона.
- **Технология выполнения:**
 1. **Анализ (15 мин):** Проверить условие сходимости метода (непрерывность и смена знака производной). Найти первую производную $f'(x)=3x^2-2$.
 2. **Выбор начального приближения (10 мин):** Выбрать x_0 (например, $x_0=2$ или $x_0=3$), исходя из условия $f(x_0) \cdot f'(x_0) > 0$.
 3. **Написание программы (35 мин):** Создать скрипт, который реализует итерационную формулу $x_{n+1} = x_n - f(x_n)/f'(x_n)$.
 4. **Запуск и контроль (20 мин):** Настроить вывод промежуточных значений ($x_n, f(x_n)$) на каждой итерации. Проверить, что программа останавливается, когда $|x_{n+1} - x_n| < \varepsilon$.
 5. **Вывод (10 мин):** Сравнить полученный результат с аналитическим или проверенным значением.

МДК.02.06 Безопасность программного обеспечения

Практическая работа № 8. «Хэширование паролей с использованием bcrypt и salt»

Время: 1,5 часа (90 мин).

Цель: Реализовать безопасное хранение паролей с использованием криптографической соли и адаптивных функций (согласно ПЗ4).

- **Задание:** Разработать модуль аутентификации (регистрация и вход), который хранит пароли не в открытом виде, а в виде хэша bcrypt с уникальной солью для каждого пользователя.
- **Технология выполнения:**
 1. **Установка библиотеки (10 мин):** Подключить библиотеку для работы с bcrypt (например, bcrypt в Python или bcryptjs в Node.js).
 2. **Функция регистрации (20 мин):** Написать функцию, которая принимает логин и пароль, генерирует соль (salt), вычисляет хэш и сохраняет в базу данных (или в файл) пару login -> hash.
 3. **Функция входа (20 мин):** Написать функцию, которая берет пароль от пользователя, извлекает сохраненный хэш из БД и использует встроенную функцию проверки (compare/verify).
 4. **Демонстрация (20 мин):** Зарегистрировать двух пользователей с одинаковыми паролями. Показать, что их хэши в базе данных различаются (благодаря случайной соли).
 5. **Защита (20 мин):** Попытаться подобрать пароль с помощью радужных таблиц (теоретически) и объяснить, почему использование соли делает это бесполезным.