Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Кузбасский государственный технический университет
имени Т. Ф. Горбачева»

Кафедра информатики и информационных систем

Составитель
Юлия Сергеевна Дементьева

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В ПРОФЕССИОНАЛЬНОЙ ДЕЯТЕЛЬНОСТИ

Методические материалы к лабораторным работам

Рекомендовано цикловой методической комиссией
специальности Информационные системы
и программирование в качестве электронного издания
для использования в образовательном процессе

Кемерово 2026

Рецензенты:

Семенова О. С. – к.т.н., преподаватель кафедры информатики и информационных систем ФГБОУ ВО «Кузбасский государственный технический университет имени Т. Ф. Горбачева».

Дементьева Юлия Сергеевна

Информационные технологии в профессиональной деятельности : методические материалы к лабораторным работам для обучающихся специальности СПО 09.02.11, специализации «Разработка и управление программным обеспечением» / Кузбасский государственный технический университет имени Т. Ф. Горбачева, Кафедра информатики и информационных систем ; составитель Ю. С. Дементьева. – Кемерово : КузГТУ, 2026. – 1 файл (748 КБ). – Текст : электронный.

Приведено содержание лабораторных работ, порядок их оформления, а также материал, необходимый для успешного изучения дисциплины. Назначение издания – помощь обучающимся в получении знаний по дисциплине «Информационные технологии в профессиональной деятельности» и организация лабораторных работ.

© Кузбасский государственный
технический университет
имени Т. Ф. Горбачева, 2026
© Дементьева Ю. С., составление, 2026

Оглавление

Лабораторная работа №1. Генерация кода и автотестов

Цель работы: изучить возможности современных интеллектуальных инструментов и нейросетевых систем для генерации программного кода и автоматического создания тестов, а также научиться применять такие инструменты в профессиональной деятельности разработчика.

Теоретическая часть

В последние годы активно развиваются технологии автоматической генерации программного кода с использованием методов машинного обучения и искусственного интеллекта. Такие системы анализируют текстовые описания задачи и на их основе формируют программный код на различных языках программирования.

Подобные инструменты могут значительно ускорить процесс разработки программного обеспечения, помочь начинающим программистам быстрее освоить язык программирования и автоматизировать рутинные задачи.

Наиболее распространённые возможности систем генерации кода:

- создание программ по текстовому описанию задачи;
- генерация отдельных функций или алгоритмов;
- автоматическое исправление ошибок в коде;
- объяснение существующего программного кода;
- создание тестов для проверки работы программы.

Среди инструментов, которые могут использоваться в профессиональной деятельности, можно выделить:

- GigaChat – российская нейросетевая система, способная помогать в генерации текстов и программного кода;
- CodeGeeX – открытая модель для генерации программного кода;
- TabbyML – открытый помощник для генерации и автодополнения программного кода;
- Codeium – инструмент автодополнения и генерации кода, интегрируемый в различные среды разработки.

Использование подобных инструментов особенно актуально в профессиональной деятельности программистов, аналитиков данных и специалистов по информационным технологиям.

Автоматическое тестирование

Тестирование является важной частью разработки программного обеспечения. Оно позволяет проверить корректность работы программы и обнаружить ошибки на ранних этапах разработки.

Тестирование программного обеспечения – это процесс проверки корректности работы программы и выявления ошибок. Тестирование позволяет убедиться, что программа выполняет поставленные задачи правильно и стабильно работает в различных условиях.

Существует два основных подхода к тестированию программ:

- ручное тестирование;
- автоматизированное тестирование.

Ручное тестирование выполняется разработчиком или пользователем, который проверяет программу, выполняя различные действия и анализируя результаты работы.

Автоматизированное тестирование выполняется с помощью специальных программных средств – тестов, которые автоматически проверяют корректность работы функций и алгоритмов.

Автоматические тесты – это программы, которые проверяют правильность работы других программ. Они позволяют быстро проверить большое количество функций без участия человека.

Основные преимущества автоматического тестирования:

- ускорение процесса проверки программ;
- повышение надёжности программного продукта;
- возможность многократного повторного тестирования;
- снижение вероятности появления ошибок в новых версиях программы.

В языке Python для создания автотестов часто используется стандартный модуль unittest.

Пример функции и автоматического теста:

```
def multiply(a, b):
    return a * b
```

Пример теста для этой функции:

```
import unittest
from main import multiply

class TestMultiply(unittest.TestCase):
    def test_positive_numbers(self):
        self.assertEqual(multiply(3, 4), 12)
```

```
    def test_zero(self):
        self.assertEqual(multiply(5, 0), 0)
    def test_negative_numbers(self):
        self.assertEqual(multiply(-2, 3), -6)

if __name__ == "__main__":
    unittest.main()
```

Каждый тест проверяет определённый сценарий работы функции. Если результат работы функции отличается от ожидаемого, тест считается не пройденным.

Использование нейросетей для создания тестов

Современные интеллектуальные системы способны автоматически генерировать тесты на основе исходного кода программы. Разработчик может предоставить программе функцию или описание задачи, после чего система предложит набор тестов для проверки различных сценариев работы программы.

Такие тесты обычно включают:

- проверку стандартных случаев работы функции;
- проверку граничных значений;
- проверку обработки ошибок;
- проверку нестандартных ситуаций.

Однако важно понимать, что результаты генерации кода и тестов необходимо проверять и анализировать. Нейросетевые системы могут допускать ошибки, поэтому окончательное решение всегда принимает разработчик.

Практическая часть

Задание 1. Выбрать одну из нейросетевых систем или инструментов генерации кода (например, GigaChat, CodeGeeX или другой аналогичный инструмент). Опишите почему вы выбрали данную модель.

Задание 2. Сформулировать текстовое задание для генерации программного кода. Например:

- программа вычисления факториала числа;
- программа поиска максимального числа в списке;
- программа проверки, является ли число простым.

С помощью выбранного инструмента сгенерировать программный код для решения задачи.

Задание 3. Проанализировать полученный код:

- проверить корректность работы программы;

- объяснить назначение основных частей программы;
- определить возможные ошибки или недостатки.

Задание 4. С помощью нейросетевого инструмента сгенерировать автоматические тесты для созданной программы.

Задание 5. Запустить тесты и проверить, проходят ли они успешно. При необходимости исправить код программы или тесты.

Задание 6. Сравнить:

- код, написанный самостоятельно;
- код, созданный нейросетевой системой.

Сделать выводы о преимуществах и недостатках использования автоматической генерации кода.

Контрольные вопросы

1. Что такое автоматическая генерация программного кода?
2. Какие инструменты могут использоваться для генерации кода?
3. Что такое автоматическое тестирование программ?
4. Какие преимущества имеют автотесты по сравнению с ручным тестированием?
5. Почему важно проверять код, сгенерированный нейросетями?
6. Какие задачи можно автоматизировать с помощью интеллектуальных инструментов разработки?

Лабораторная работа №2. Промпт-инжиниринг и разработка документации проекта

Цель работы: изучить основы промпт-инжиниринга при работе с интеллектуальными системами, научиться формулировать корректные запросы для получения программного кода и текстовой информации, а также освоить использование нейросетевых инструментов для разработки документации программного проекта.

Теоретическая часть

Промпт-инжиниринг – это процесс разработки и оптимизации текстовых запросов (промптов), которые используются для взаимодействия с интеллектуальными системами и нейросетевыми моделями. От того, насколько правильно сформулирован запрос, зависит качество и точность полученного результата.

Промпт представляет собой текстовое описание задачи, которое передаётся системе искусственного интеллекта. В ответ система формирует текст, программный код, анализ данных или другие виды информации.

Промпт-инжиниринг используется для решения различных задач:

- генерация программного кода;
- создание документации;
- анализ данных;
- генерация текстов;
- автоматизация разработки программного обеспечения.

При работе с интеллектуальными системами важно формулировать запросы максимально ясно и подробно. Это позволяет получить более точный и полезный результат.

Основные принципы создания эффективных промптов:

- чёткое описание задачи;
- указание контекста и условий;
- указание формата ответа;
- использование примеров;
- уточнение ограничений и требований.

Например, менее эффективный запрос: «Напиши программу сортировки».

Более эффективный запрос: «Напиши программу на Python, которая сортирует список чисел методом пузырьковой сортировки и выводит результат на экран».

Современные системы искусственного интеллекта, такие как GigaChat, CodeGeeX и TabbyML, могут использоваться для генерации программного кода, объяснения алгоритмов и создания текстовой документации.

Разработка документации программного проекта

Документация является важной частью любого программного проекта. Она помогает разработчикам, пользователям и администраторам понимать структуру программы, её назначение и способы использования.

Основные виды документации программного обеспечения:

- техническая документация;
- пользовательская документация;
- документация разработчика;
- описание архитектуры системы;
- инструкции по установке и эксплуатации.

К технической документации относятся:

- описание функциональности программы;
- структура проекта;
- описание модулей и функций;
- описание алгоритмов;
- примеры использования программы.

Современные инструменты на основе искусственного интеллекта позволяют значительно ускорить процесс подготовки документации. Они могут автоматически создавать описание функций, комментарии к коду, инструкции по использованию программы и структуру проекта.

Однако важно понимать, что созданная автоматически документация должна проверяться разработчиком и при необходимости редактироваться. Это необходимо для обеспечения точности и полноты информации.

Практическая часть

Задание 1. Выбрать два нейросетевых инструмента для работы (например, GigaChat, CodeGeeX, TabbyML или другой аналогичный инструмент).

Задание 2. Сформулировать несколько различных промптов для генерации программного кода. Например:

- программа сортировки списка;
- программа поиска максимального элемента массива;

- программа вычисления среднего значения чисел.

Сравнить полученные результаты и определить, какие формулировки промптов дают более точный и полезный результат и в какой модели. Составьте сравнительную таблицу.

Задание 3. Создать улучшенную версию промпта, которая позволит получить более качественный программный код.

Задание 4. С помощью выбранного инструмента сгенерировать описание программы, включающее:

- назначение программы;
- описание основных функций;
- пример использования программы.

Задание 5. Разработать краткую документацию для созданного программного проекта. Документация должна содержать:

- название проекта;
- описание программы;
- структуру проекта;
- описание основных функций;
- инструкцию по запуску программы.

Задание 6. Проанализировать полученную документацию и при необходимости отредактировать её для улучшения структуры и понятности.

Контрольные вопросы

1. Что такое промпт-инжиниринг?
2. Что представляет собой промпт при работе с интеллектуальными системами?
3. Какие принципы помогают формулировать эффективные запросы?
4. Какие виды документации используются в программных проектах?
5. Какие преимущества даёт использование интеллектуальных инструментов при разработке документации?
6. Почему важно проверять результаты, полученные от нейросетевых систем?

Лабораторная работа №3. Сравнительный анализ работы ИИ-инструментов

Цель работы: изучить возможности различных интеллектуальных инструментов, предназначенных для генерации текстов, программного кода и аналитической информации, провести сравнительный анализ их работы и научиться оценивать эффективность применения нейросетевых систем в профессиональной деятельности специалиста в области информационных технологий.

Теоретическая часть

В последние годы технологии искусственного интеллекта активно внедряются в различные сферы профессиональной деятельности. Особенно заметно это в области информационных технологий, где интеллектуальные системы помогают разработчикам писать программный код, анализировать данные, создавать документацию и автоматизировать множество рабочих процессов.

Современные ИИ-инструменты могут выполнять следующие задачи:

- генерация программного кода
- анализ текстовой информации
- автоматическое создание документации
- помощь в разработке программных алгоритмов
- обработка больших объёмов данных
- генерация идей и решений для проектирования систем

Использование подобных систем позволяет значительно повысить производительность труда специалистов и ускорить процесс разработки программного обеспечения.

В настоящее время активно развиваются отечественные технологии искусственного интеллекта. Многие российские компании создают собственные интеллектуальные системы, которые могут использоваться в образовательной и профессиональной деятельности.

Среди наиболее известных российских нейросетевых инструментов можно выделить:

- GigaChat – интеллектуальная система компании Сбер, предназначенная для генерации текстов, ответов на вопросы и помощи в программировании.

- YandexGPT – нейросетевая модель компании Яндекс, способная генерировать тексты, анализировать информацию и помогать в создании программных решений.
- Kandinsky – модель искусственного интеллекта для генерации изображений по текстовому описанию.
- ruGPT – языковая модель, ориентированная на обработку текстов на русском языке.

Кроме отечественных решений существуют также открытые инструменты, которые могут использоваться для разработки программного обеспечения:

- CodeGeeX – модель генерации программного кода.
- TabbyML – открытый помощник для автодополнения и генерации кода.

Сравнительный анализ ИИ-инструментов

Поскольку различные интеллектуальные системы имеют разные архитектуры и цели применения, их возможности могут существенно отличаться. Поэтому при использовании таких инструментов важно проводить сравнительный анализ их работы.

Сравнительный анализ позволяет определить:

- качество генерируемых ответов
- точность выполнения поставленных задач
- скорость работы системы
- удобство интерфейса
- доступность и ограничения использования
- возможность интеграции с другими программами

Для анализа часто используются сравнительные таблицы, в которых фиксируются результаты выполнения одинаковых заданий различными системами.

Например, можно сравнить:

- генерацию программного кода
- объяснение алгоритмов
- создание текстовой документации
- анализ программных ошибок

Каждая система получает одинаковый запрос (промпт), после чего результаты сравниваются по определённым критериям.

Такой подход позволяет объективно оценить возможности различных ИИ-инструментов и выбрать наиболее подходящий инструмент для решения профессиональных задач.

Практическая часть

Задание 1. Выбрать не менее трёх интеллектуальных инструментов для анализа. При необходимости можно дополнительно использовать открытые инструменты.

Задание 2. Подготовить несколько одинаковых запросов (промптов) для всех выбранных систем. Примеры заданий:

- написать программу на Python для вычисления факториала числа
- объяснить алгоритм сортировки массива
- сгенерировать описание программы
- предложить тесты для функции

Каждый запрос необходимо отправить во все выбранные системы.

Задание 3. Зафиксировать полученные результаты и оформить их в сравнительной таблице.

Пример таблицы анализа генерации кода:

Таблица 1 – Анализ генерации

Критерий	GigaChat	YandexGPT	ruGPT
Качество кода			
Наличие комментариев			
Корректность программы			
Читаемость кода			

Задание 4. Создать вторую таблицу для анализа текстовых ответов. Пример критериев:

Таблица 2 – Анализ ответов

Критерий	GigaChat	YandexGPT	ruGPT
Полнота ответа			
Точность информации			
Понятность объяснения			
Структура текста			

Задание 5. Создать третью сравнительную таблицу, оценивающую удобство работы с инструментами.

Пример критериев:

Таблица 3 – Удобство работы

Критерий	GigaChat	YandexGPT	ruGPT
Простота использования			
Скорость получения ответа			
Удобство интерфейса			
Возможности генерации кода			

Задание 6. На основе полученных таблиц выполнить анализ результатов и сделать выводы:

- какая система лучше справляется с генерацией кода
- какая система лучше объясняет алгоритмы
- какой инструмент наиболее удобен в использовании
- в каких задачах лучше применять каждую систему

Задание 7. Подготовить итоговый отчёт, содержащий:

1) описание использованных инструментов
2) примеры использованных запросов
3) сравнительные таблицы
4) анализ полученных результатов
5) итоговые выводы.

Контрольные вопросы

1. Какие задачи могут выполнять системы искусственного интеллекта?
2. Какие российские нейросетевые системы существуют?
3. Для чего проводится сравнительный анализ ИИ-инструментов?
4. Какие критерии можно использовать при сравнении интеллектуальных систем?
5. Почему важно использовать одинаковые запросы при сравнении систем?
6. Какие преимущества дают ИИ-инструменты в профессиональной деятельности специалистов IT?

Лабораторная работа №4. Работа с Git и GitHub

Цель работы: изучить систему контроля версий Git, освоить основные команды работы с репозиториями, научиться управлять версиями программного кода и использовать платформы для совместной разработки программного обеспечения.

Теоретическая часть

При разработке программного обеспечения разработчики постоянно изменяют и дополняют код. Без специальных инструментов становится сложно отслеживать изменения, возвращаться к предыдущим версиям программы и работать над проектом нескольким разработчикам одновременно.

Для решения этих задач используются системы контроля версий (Version Control Systems). Они позволяют:

- отслеживать изменения файлов
- сохранять различные версии проекта
- возвращаться к предыдущим версиям
- работать над проектом нескольким разработчикам одновременно
- контролировать процесс разработки

Существует два основных типа систем контроля версий:

1) централизованные системы
2) распределённые системы

В централизованных системах существует один центральный сервер, на котором хранится основной репозиторий проекта. Разработчики подключаются к серверу и получают доступ к файлам.

В распределённых системах каждый разработчик имеет собственную копию репозитория. Это делает систему более надёжной и гибкой.

Одной из самых популярных распределённых систем контроля версий является Git.

Git – это распределённая система контроля версий, предназначенная для отслеживания изменений в файлах и организации совместной работы разработчиков. Она широко используется в разработке программного обеспечения, веб-разработке, анализе данных и других областях информационных технологий.

Основные возможности Git:

- хранение истории изменений проекта

- создание различных веток разработки
- объединение изменений
- восстановление предыдущих версий проекта
- совместная работа над программным кодом

Git работает с репозиториями. Репозиторий – это специальная папка, содержащая файлы проекта и информацию об их изменениях.

Основные команды Git

Для работы с Git используется командная строка или специальные графические интерфейсы. Основные команды Git включают:

- git init - создаёт новый локальный репозиторий.
- git clone – создаёт копию удалённого репозитория.
- git status – показывает текущее состояние файлов проекта.
- git add – добавляет файлы в область подготовки к коммиту.
- git commit – сохраняет изменения в репозитории.
- git log –показывает историю изменений проекта.
- git branch –создаёт и отображает ветки разработки.
- git merge – объединяет изменения из разных веток.
- git push – отправляет изменения в удалённый репозиторий.
- git pull – получает изменения из удалённого репозитория.

Платформы для хранения репозиториев

Для удобной совместной работы разработчиков используются онлайн-платформы для хранения репозиториев.

Одной из самых известных платформ является GitHub. Она предоставляет возможности:

- хранения программных проектов
- совместной разработки
- управления версиями
- отслеживания ошибок
- управления задачами проекта

Кроме GitHub существуют и другие платформы, например:

GitLab – платформа для управления проектами и совместной разработки.

Gitea – лёгкая система управления репозиториями с открытым исходным кодом.

Использование подобных платформ позволяет организовать удобную совместную разработку программного обеспечения.

Основные элементы репозитория

Репозиторий Git обычно содержит:

- исходный код программы
- документацию проекта
- конфигурационные файлы
- историю изменений
- ветки разработки

Каждое изменение фиксируется в виде коммита.

Коммит – это сохранённое состояние проекта с описанием выполненных изменений.

Коммиты позволяют отслеживать, кто и когда внёс изменения в проект.

Ветки разработки

Одной из важных возможностей Git является использование веток (branches). Ветка – это отдельная линия разработки проекта.

Основные преимущества использования веток:

- разработка новых функций без изменения основной версии программы
- тестирование новых возможностей
- работа нескольких разработчиков над разными частями проекта
- безопасное экспериментирование с кодом

После завершения работы ветки могут объединяться с основной веткой проекта.

Практическая часть

Задание 1. Установить систему Git на компьютер и проверить корректность установки с помощью команды:

git --version

Задание 2. Создать новую папку проекта и инициализировать в ней Git-репозиторий.

Задание 3. Создать несколько файлов проекта, например:

program.py

README.md

notes.txt

Добавить их в репозиторий и выполнить первый коммит.

Задание 4. Создать файл README.md с кратким описанием проекта. Включить в него:

- название проекта
- описание программы
- инструкцию по запуску

Задание 5. Просмотреть историю коммитов с помощью команды git log.

Задание 6. Создать новую ветку разработки, например feature.

Добавить в проект новые изменения и выполнить коммит.

Задание 7. Переключиться обратно на основную ветку и объединить изменения с помощью команды merge.

Задание 8. Создать аккаунт на платформе GitHub или GitLab и создать удалённый репозиторий.

Задание 9. Связать локальный репозиторий с удалённым и отправить проект на сервер.

Задание 10. Внести изменения в проект, выполнить новый коммит и отправить обновления в удалённый репозиторий.

Задание 11. Создать новую ветку проекта и добавить в неё новую функцию программы.

Задание 12. Создать краткий отчёт, содержащий:

- список выполненных команд Git
- скриншоты основных этапов работы
- описание структуры репозитория
- ссылку на созданный удалённый репозиторий.

Контрольные вопросы

1. Что такое система контроля версий?
2. Какие преимущества даёт использование Git при разработке программ?
3. Что такое репозиторий?
4. Для чего используются коммиты?
5. Что такое ветки разработки и зачем они нужны?
6. Какие платформы используются для хранения Git-репозиториев?
7. Какие основные команды Git используются при работе с проектом?

Лабораторная работа №5. Документирование проекта с использованием Markdown

Цель работы: изучить основы языка разметки Markdown, научиться создавать структурированную документацию программного проекта, а также освоить оформление текстовой документации для размещения в системах контроля версий и репозиториях программного обеспечения.

Теоретическая часть

Документация является важной частью любого программного проекта. Она помогает разработчикам, пользователям и администраторам понять назначение программы, её функциональные возможности, структуру проекта и способы использования.

Документация позволяет:

- описывать назначение программного продукта
- объяснять принцип работы программы
- облегчать поддержку и развитие проекта
- помогать новым разработчикам быстро разобраться в коде
- предоставлять пользователям инструкции по использованию программы

В современных программных проектах документация часто хранится вместе с исходным кодом в репозитории системы контроля версий.

Одним из самых распространённых форматов документации является язык разметки Markdown.

Markdown – это простой язык разметки, предназначенный для форматирования текстовых документов. Он используется для создания читаемой и структурированной документации.

Markdown широко применяется в системах управления проектами, репозиториях программного кода, блогах и системах совместной разработки.

Основные преимущества Markdown:

- простота синтаксиса
- удобство чтения исходного текста
- поддержка на большинстве платформ
- возможность быстрого создания документации
- интеграция с системами контроля версий

Markdown активно используется на платформах разработки программного обеспечения, таких как GitHub и GitLab.

Основные элементы Markdown

1. **Заголовки**

Для создания заголовков используются символы решётки (#). Количество символов определяет уровень заголовка.

Пример:

#Заголовок первого уровня

Заголовок второго уровня

Заголовок третьего уровня

2. **Форматирование текста**

Markdown позволяет выделять текст различными способами.

Курсив:

текст

Жирный текст:

текст

3. **Списки**

Нумерованные списки:

1. *Первый пункт*
2. *Второй пункт*
3. *Третий пункт*

Маркированные списки:

- *Первый пункт*
- *Второй пункт*
- *Третий пункт*

4. **Ссылки**

Для создания ссылок используется следующий синтаксис:

[текст ссылки](адрес)

Например:

[GitHub](https://github.com)

5. **Изображения**

Markdown также поддерживает вставку изображений.

![описание изображения](адрес изображения)

6. **Код и блоки кода**

Для оформления кода используются специальные блоки.

Пример однострочного кода:

`print("Hello world")`

Пример блока кода:

```python
print("Hello world")
```

7. Таблицы

Markdown позволяет создавать таблицы для представления структурированной информации.

Пример:

Язык	Назначение
Python	программирование
HTML	создание веб-страниц

Использование Markdown в проектах

В программных проектах Markdown чаще всего используется для создания файла README.md. Этот файл содержит основную информацию о проекте:

- название проекта
- описание программы
- список функций
- инструкцию по установке
- примеры использования
- описание структуры проекта

Файл README.md отображается на главной странице репозитория и помогает пользователям быстро понять назначение проекта.

Практическая часть

Задание 1. Создать новый файл README.md в папке проекта.

Задание 2. Создать структуру документации проекта с использованием Markdown. Документ должен содержать следующие разделы:

- название проекта
- краткое описание программы
- цели и задачи проекта
- используемые технологии
- инструкция по установке
- инструкция по запуску программы

Задание 3. Добавить в документ:

- заголовки разных уровней
- маркированные и нумерованные списки
- примеры кода
- ссылки на используемые ресурсы

Задание 4. Создать таблицу, описывающую основные файлы проекта.

Пример структуры таблицы:

Таблица 4 – Основные файлы проекта

Файл	Назначение
main.py	основной файл программы
utils.py	вспомогательные функции
README.md	документация проекта

Задание 5. Добавить в документацию изображение или схему структуры проекта.

Задание 6. Разместить созданный файл README.md в репозитории проекта и проверить корректность отображения документации.

Задание 7. Подготовить итоговую документацию проекта, включающую:

- описание программы
- структуру проекта
- примеры использования
- таблицы и списки
- фрагменты программного кода

Контрольные вопросы

1. Что такое Markdown?
2. Какие преимущества имеет Markdown при создании документации?
3. Какие основные элементы разметки используются в Markdown?
4. Для чего используется файл README.md в программных проектах?
5. Как оформить таблицу в Markdown?
6. Какие элементы документации должны присутствовать в программном проекте?

Лабораторная работа №6. Развертывание приложения на облачном сервере

Цель работы: изучить основы облачных технологий, освоить принципы развертывания программного приложения на удалённом сервере и научиться использовать облачные платформы для размещения и запуска программных проектов.

Теоретическая часть

Облачные технологии – это модель предоставления вычислительных ресурсов через интернет. Пользователь получает доступ к серверам, системам хранения данных, базам данных и программным приложениям без необходимости приобретения и обслуживания собственного оборудования.

Облачные сервисы широко применяются в современной информационной инфраструктуре и позволяют быстро развёртывать программные решения, обеспечивать доступность сервисов и масштабировать вычислительные ресурсы.

Основные преимущества облачных технологий:

- удалённый доступ к ресурсам через интернет
- высокая масштабируемость
- снижение затрат на оборудование
- высокая надёжность и отказоустойчивость
- возможность быстрого развёртывания приложений

Модели облачных сервисов

Существует несколько основных моделей предоставления облачных сервисов.

IaaS (Infrastructure as a Service) – инфраструктура как услуга. Пользователь получает виртуальный сервер и самостоятельно устанавливает операционную систему и программное обеспечение.

PaaS (Platform as a Service) – платформа как услуга. Пользователь получает готовую платформу для размещения приложения.

SaaS (Software as a Service) – программное обеспечение как услуга.

Пользователь работает с готовым программным продуктом через интернет.

Облачные платформы

Для размещения приложений используются специальные облачные платформы. Они предоставляют виртуальные серверы,

системы хранения данных и инструменты управления инфраструктурой.

Среди облачных сервисов можно выделить:

- Yandex Cloud – российская облачная платформа для размещения приложений и сервисов.
- VK Cloud – облачная платформа для хранения данных и развертывания приложений.
- Selectel Cloud – сервис облачной инфраструктуры и виртуальных серверов.

Эти платформы позволяют создавать виртуальные машины, настраивать серверную инфраструктуру и размещать программные приложения.

Виртуальный сервер

Виртуальный сервер (VPS – Virtual Private Server) – это программная имитация физического сервера, работающая на базе облачной инфраструктуры.

Пользователь получает полный доступ к серверу и может:

- устанавливать операционную систему
- размещать веб-приложения
- настраивать базы данных
- управлять программным обеспечением

Для управления сервером обычно используется протокол SSH (Secure Shell), который позволяет подключаться к удалённому компьютеру через интернет.

Развертывание приложения

Процесс развертывания приложения включает несколько этапов:

1. создание виртуального сервера
2. установка необходимого программного обеспечения
3. загрузка файлов проекта
4. настройка окружения
5. запуск приложения

Веб-приложения часто разворачиваются с использованием веб-серверов, таких как:

- Nginx – высокопроизводительный веб-сервер.
- Apache HTTP Server – один из наиболее популярных веб-серверов.

Для передачи файлов на сервер могут использоваться инструменты:

- SCP
- SFTP
- Git-репозитории

Использование системы контроля версий значительно упрощает процесс развертывания приложения на сервере.

Автоматизация развертывания

В современных проектах часто используется автоматическое развертывание приложений. Для этого применяются инструменты CI/CD (Continuous Integration / Continuous Deployment), которые позволяют автоматически обновлять приложение при изменении кода.

Подобный подход используется при профессиональной разработке программного обеспечения и позволяет значительно ускорить процесс внедрения новых версий программ.

Практическая часть

Задание 1. Выбрать облачную платформу для работы (например Yandex Cloud, VK Cloud или Selectel Cloud).

Задание 2. Создать аккаунт на выбранной платформе и изучить интерфейс управления облачными ресурсами.

Задание 3. Создать виртуальный сервер (VPS) со следующими параметрами:

- операционная система Linux (например Ubuntu)
- базовая конфигурация сервера
- доступ через SSH

Задание 4. Подключиться к созданному серверу с использованием SSH.

Задание 5. Установить необходимое программное обеспечение для запуска приложения. Например:

- Python
- Git
- веб-сервер

Задание 6. Загрузить программный проект на сервер с помощью Git-репозитория или передачи файлов.

Задание 7. Настроить запуск приложения на сервере.

Задание 8. Проверить работу приложения через веб-браузер или командную строку.

Задание 9. При необходимости внести изменения в проект и обновить приложение на сервере.

Задание 10. Подготовить отчёт о выполнении работы, включающий:

- описание выбранной облачной платформы
- этапы создания виртуального сервера
- список установленных программ
- описание процесса развертывания приложения
- скриншоты основных этапов работы

Контрольные вопросы

1. Что такое облачные технологии?
2. Какие модели облачных сервисов существуют?
3. Что такое виртуальный сервер?
4. Какие облачные платформы используются для размещения приложений?
5. Какие этапы включает процесс развертывания приложения?
6. Для чего используется протокол SSH?
7. Какие преимущества даёт использование облачных серверов?

Лабораторная работа №7. Автоматизация развертывания и настройка облачной инфраструктуры

Цель работы: изучить методы автоматизации развертывания программных систем, освоить принципы управления облачной инфраструктурой и научиться использовать современные инструменты автоматизации для настройки серверной среды и развертывания приложений.

Теоретическая часть

В современных программных проектах важную роль играет автоматизация процессов разработки, тестирования и развертывания программного обеспечения. Автоматизация позволяет значительно ускорить внедрение новых версий программных продуктов, уменьшить количество ошибок и повысить стабильность системы.

Развертывание приложения вручную может включать множество операций:

- установку программного обеспечения
- настройку серверной среды
- загрузку файлов проекта
- настройку баз данных
- запуск и тестирование приложения

Если выполнять эти действия вручную, существует риск ошибок и несогласованности настроек. Автоматизация позволяет описать весь процесс развертывания в виде сценариев, которые могут выполняться автоматически.

Инфраструктура как код

Одним из современных подходов к управлению серверной инфраструктурой является концепция «инфраструктура как код».

Infrastructure as Code – это метод управления и настройки инфраструктуры с использованием программного кода и автоматизированных сценариев.

В этом подходе конфигурация серверов, сети и сервисов описывается в виде файлов конфигурации. Эти файлы могут храниться в системе контроля версий и использоваться для автоматического создания и настройки серверной среды.

Основные преимущества данного подхода:

- повторяемость развертывания
- уменьшение количества ошибок
- быстрое создание новой инфраструктуры

- возможность автоматического масштабирования
- удобство управления конфигурациями

Инструменты автоматизации

Для автоматизации управления серверами используются специальные программные инструменты.

Одним из наиболее популярных является **Ansible**. Это система автоматизации конфигурации и управления серверами.

Ansible позволяет выполнять следующие задачи:

- автоматическая установка программного обеспечения
- настройка операционной системы
- развертывание приложений
- управление группами серверов
- обновление конфигураций

Конфигурация в Ansible описывается с помощью специальных файлов – playbook. Playbook представляет собой набор инструкций, которые определяют действия, выполняемые на сервере.

Пример задач, которые может выполнять playbook:

- установка пакетов
- создание пользователей
- копирование файлов
- запуск служб
- перезапуск серверов

Другим распространённым инструментом является:

Terraform – инструмент для автоматического создания облачной инфраструктуры.

Terraform позволяет автоматически:

- создавать виртуальные машины
- настраивать сетевую инфраструктуру
- управлять ресурсами облачной платформы
- создавать базы данных и хранилища

Этот инструмент широко используется для управления инфраструктурой в облачных платформах.

Контейнеризация приложений

Для упрощения развертывания приложений также используется контейнеризация.

Контейнер – это изолированная среда, содержащая приложение и все необходимые зависимости для его работы.

Одним из самых популярных инструментов контейнеризации является Docker.

Контейнеризация позволяет:

- обеспечить одинаковую среду выполнения
- упростить переносимость приложений
- ускорить процесс развертывания
- повысить стабильность системы

Docker позволяет упаковать приложение вместе со всеми зависимостями в контейнер, который может запускаться на любом сервере.

Автоматизация развертывания приложений

В современных проектах автоматизация развертывания тесно связана с процессами непрерывной интеграции и непрерывной доставки.

Continuous Integration – практика регулярного объединения изменений в коде и автоматической проверки проекта.

Continuous Deployment – процесс автоматического развертывания новых версий программного обеспечения.

Использование автоматизации позволяет:

- сократить время вывода продукта на рынок
- обеспечить стабильность программных систем
- уменьшить вероятность ошибок при развертывании
- автоматически обновлять приложения

Практическая часть

Задание 1. Изучить интерфейс выбранной облачной платформы (Yandex Cloud, VK Cloud или Selectel Cloud) и определить основные элементы управления инфраструктурой.

Задание 2. Создать виртуальный сервер в облачной платформе со следующими параметрами:

- операционная система Linux (например Ubuntu)
- доступ через SSH
- базовая конфигурация сервера

Задание 3. Подключиться к серверу через SSH и выполнить базовую настройку системы:

- обновить список пакетов
- установить необходимые системные утилиты
- создать нового пользователя

Задание 4. Установить систему управления конфигурациями Ansible на локальный компьютер или сервер.

Задание 5. Создать первый playbook Ansible, выполняющий следующие действия:

- установка программного пакета
- создание директории проекта
- копирование файла конфигурации
- вывод сообщения о завершении выполнения сценария

Задание 6. Запустить созданный playbook и проверить выполнение всех операций.

Задание 7. Создать конфигурацию автоматического развертывания приложения, включающую:

- установку среды выполнения
- копирование файлов проекта
- запуск приложения

Задание 8. Изучить основы работы с Docker. Создать файл конфигурации Dockerfile для простого приложения.

Задание 9. Собрать контейнер приложения и запустить его на сервере.

Задание 10. Подготовить схему инфраструктуры проекта, включающую:

- облачный сервер
- систему автоматизации
- контейнер приложения
- систему контроля версий

Задание 11. Подготовить сравнительную таблицу инструментов автоматизации инфраструктуры.

Пример структуры таблицы:

Таблица 5 – Инструменты автоматизации

Инструмент	**Назначение**	**Основные возможности**
Ansible	управление конфигурациями	автоматизация настройки серверов
Terraform	управление инфраструктурой	создание облачных ресурсов
Docker	контейнеризация	запуск приложений в изолированной среде

Задание 12. Подготовить отчёт о выполнении лабораторной работы, включающий:

- описание используемой облачной платформы

- описание процесса настройки сервера
- пример playbook Ansible
- пример Dockerfile
- схему инфраструктуры проекта
- скриншоты выполненных операций

Контрольные вопросы

1. Что понимается под автоматизацией развертывания программного обеспечения?
2. Что такое Infrastructure as Code?
3. Какие задачи решает система Ansible?
4. Для чего используется Terraform?
5. Что такое контейнеризация?
6. Какие преимущества дает использование Docker?
7. Что такое непрерывная интеграция и непрерывное развертывание?
8. Какие преимущества даёт автоматизация управления инфраструктурой?

Лабораторная работа №8. Настройка инструментов разработки и автоматизация рабочих процессов

Цель работы: изучить современные инструменты разработки программного обеспечения, научиться настраивать рабочую среду программиста, а также освоить методы автоматизации типовых рабочих процессов при разработке программных проектов.

Теоретическая часть

Разработка программного обеспечения требует использования специализированных инструментов, которые помогают программистам писать, тестировать и поддерживать программный код. Совокупность таких инструментов образует рабочую среду разработчика.

Одним из ключевых элементов среды разработки является интегрированная среда разработки (IDE).

Visual Studio Code – один из наиболее популярных редакторов кода, поддерживающий большое количество языков программирования и расширений.

Интегрированные среды разработки предоставляют следующие возможности:

- редактирование программного кода
- подсветка синтаксиса
- автодополнение кода
- отладка программ
- интеграция с системами контроля версий запуск и тестирование программ

Помимо IDE разработчики используют различные вспомогательные инструменты, позволяющие автоматизировать часть рабочих задач.

Система контроля версий

Одним из важнейших инструментов современной разработки является система контроля версий.

Git – распределённая система управления версиями программного кода.

Git позволяет:

- сохранять историю изменений проекта
- работать над проектом нескольким разработчикам одновременно
- отслеживать изменения файлов

- возвращаться к предыдущим версиям проекта
- управлять ветками разработки

Хранение проектов обычно осуществляется в удалённых репозиториях, например на платформе GitHub или GitLab.

Автоматизация рабочих процессов

В процессе разработки программисты регулярно выполняют повторяющиеся операции:

- сборка проекта
- установка зависимостей
- запуск тестов
- проверка кода
- форматирование файлов
- развертывание приложения

Автоматизация этих операций позволяет значительно ускорить разработку и уменьшить вероятность ошибок.

Для автоматизации используются специальные инструменты сборки и управления задачами.

Одним из распространённых инструментов является

Make – инструмент автоматизации сборки программного обеспечения.

Make позволяет описывать последовательность выполнения задач с помощью специального файла конфигурации (Makefile).

Например, можно автоматизировать:

- компиляцию программы
- запуск тестов
- очистку временных файлов
- сборку проекта

Форматирование и проверка кода

Для повышения качества программного кода используются специальные инструменты анализа и форматирования.

Автоматические средства позволяют:

- обнаруживать ошибки
- проверять стиль написания кода
- находить потенциальные проблемы
- форматировать код по заданным правилам

Подобные инструменты особенно полезны при командной разработке.

Пакетные менеджеры

При разработке программного обеспечения часто используются сторонние библиотеки. Для управления такими зависимостями применяются пакетные менеджеры.

Пакетный менеджер позволяет:

- устанавливать библиотеки
- обновлять зависимости
- удалять пакеты
- контролировать версии библиотек

Использование пакетных менеджеров значительно упрощает управление программными проектами.

Скрипты автоматизации

Для автоматизации задач разработчики часто создают специальные скрипты.

Скрипты могут выполнять различные действия:

- запуск программы
- проверку кода
- сборку проекта
- запуск тестов
- подготовку среды разработки

Такие скрипты могут быть написаны на различных языках программирования или использовать возможности командной строки операционной системы.

Настройка рабочей среды

Правильная настройка среды разработки позволяет повысить эффективность работы программиста. Она может включать:

- установку редактора кода
- установку необходимых расширений
- настройку системы контроля версий
- установку инструментов автоматизации
- настройку форматирования и проверки кода

Хорошо настроенная рабочая среда значительно ускоряет процесс разработки и повышает качество программного продукта.

Практическая часть

Задание 1. Установить редактор кода Visual Studio Code или другую среду разработки.

Задание 2. Настроить рабочую среду разработчика:

- установить необходимые расширения для выбранного языка программирования
- включить подсветку синтаксиса
- настроить автодополнение кода
- настроить автоматическое форматирование файлов

Задание 3. Установить систему контроля версий Git и настроить её для работы:

- создать локальный репозиторий
- добавить файлы проекта
- выполнить первый коммит

Задание 4. Создать структуру программного проекта, включающую:

- каталог исходного кода
- каталог документации
- каталог тестов
- файл README.md

Задание 5. Создать скрипт автоматизации для выполнения следующих операций:

- запуск программы
- запуск тестов
- очистка временных файлов

Задание 6. Создать файл автоматизации сборки проекта (например Makefile), который выполняет:

- сборку проекта
- запуск тестов
- очистку временных файлов

Задание 7. Настроить инструмент форматирования кода и выполнить автоматическое форматирование файлов проекта.

Задание 8. Создать таблицу, описывающую используемые инструменты разработки.

Пример структуры таблицы:

Таблица 6 – Инструменты разработки

Инструмент	**Назначение**	**Основные функции**
Visual Studio Code	редактор кода	редактирование и отладка программ
Git	контроль версий	управление изменениями проекта
Make	автоматизация сборки	выполнение задач сборки

Задание 9. Подготовить отчёт о выполнении лабораторной работы, включающий:

- описание используемых инструментов разработки
- описание настроек среды разработки
- пример скрипта автоматизации
- пример файла автоматизации сборки
- скриншоты выполненных операций

Контрольные вопросы

1. Что такое интегрированная среда разработки?
2. Какие функции выполняет система контроля версий?
3. Для чего используется Git?
4. Какие задачи можно автоматизировать в процессе разработки программ?
5. Что такое инструмент сборки программного обеспечения?
6. Для чего используется Makefile?
7. Какие преимущества дает автоматизация рабочих процессов?
8. Какие инструменты используются для проверки качества программного кода?

Лабораторная работа №9. Тестирование API и работа с DevTools

Цель работы: изучить основы тестирования API, научиться анализировать сетевые запросы веб-приложений и освоить использование инструментов разработчика браузера для диагностики и отладки программных систем.

Теоретическая часть

Современные веб-приложения состоят из двух основных частей: клиентской и серверной.

Клиентская часть работает в браузере пользователя и отвечает за интерфейс приложения. Серверная часть обрабатывает запросы пользователей, выполняет вычисления и взаимодействует с базами данных.

Обмен данными между клиентом и сервером осуществляется через специальные программные интерфейсы.

API

API – это программный интерфейс, позволяющий различным программам взаимодействовать друг с другом.

API определяет:

- способы передачи данных
- форматы запросов
- форматы ответов
- доступные функции сервера

В веб-разработке чаще всего используется архитектурный стиль REST.

REST API использует стандартные HTTP-запросы для взаимодействия клиента и сервера.

HTTP-методы

Для выполнения операций с данными используются различные методы протокола HTTP.

Основные методы:

- GET – получение данных
- POST – создание новых данных
- PUT – изменение существующих данных
- DELETE – удаление данных

Каждый запрос отправляется по определённому адресу (endpoint), а сервер возвращает ответ в виде данных.

Формат передачи данных

В современных API чаще всего используется формат JSON.

JSON представляет данные в виде структурированных объектов, которые легко обрабатываются программами.

Пример ответа сервера:

```
{
    "id": 1,
    "name": "User",
    "email": user@example.com
}
```

Тестирование API

Тестирование API позволяет проверить корректность работы серверной части приложения.

Во время тестирования проверяется:

- правильность обработки запросов
- корректность возвращаемых данных
- скорость ответа сервера
- обработка ошибок

Для тестирования API используются специальные инструменты.

Одним из наиболее распространённых является **Postman** – инструмент для отправки HTTP-запросов и анализа ответов сервера.

С помощью Postman можно:

- отправлять различные типы запросов
- передавать параметры и данные
- проверять ответы сервера
- создавать автоматические тесты

Инструменты разработчика браузера

Современные браузеры предоставляют встроенные инструменты для анализа работы веб-приложений.

Chrome DevTools – это набор инструментов разработчика, встроенный в браузер.

DevTools позволяет:

- анализировать структуру веб-страницы
- исследовать сетевые запросы
- отслеживать ошибки JavaScript
- измерять производительность приложения
- редактировать HTML и CSS в режиме реального времени

Одной из наиболее полезных вкладок является вкладка Network.

Вкладка Network позволяет:

- просматривать все сетевые запросы

- анализировать параметры запросов
- проверять ответы сервера
- изучать время загрузки ресурсов

Это позволяет разработчику понимать, как именно работает веб-приложение и какие данные передаются между клиентом и сервером.

Отладка веб-приложений

Во время разработки веб-приложений могут возникать различные ошибки:

- неверные параметры запроса
- ошибки сервера
- неправильный формат данных
- проблемы с авторизацией

Использование инструментов DevTools и систем тестирования API позволяет быстро находить и исправлять подобные проблемы.

Практическая часть

Задание 1. Открыть инструменты разработчика браузера.

Для этого можно использовать браузер Google Chrome или другой современный браузер.

Открыть DevTools с помощью клавиши F12.

Задание 2. Изучить основные вкладки DevTools:

- Elements
- Console
- Network
- Sources
- Application

Кратко описать назначение каждой вкладки.

Задание 3. Перейти на любой веб-сайт и открыть вкладку Network.

Обновить страницу и проанализировать сетевые запросы.

Необходимо определить:

- тип запроса
- адрес запроса
- код ответа сервера
- размер переданных данных
- время выполнения запроса

Задание 4. Выбрать один из запросов и изучить:

- заголовки запроса (Headers)
- переданные параметры

- ответ сервера (Response)

Задание 5. Установить программу Postman.

Создать тестовый HTTP-запрос к открытому API.

Пример тестового API:

https://jsonplaceholder.typicode.com

Задание 6. Отправить следующие типы запросов:

- GET
- POST
- PUT
- DELETE

Зафиксировать полученные ответы сервера.

Задание 7. Создать таблицу результатов тестирования API.

Пример структуры таблицы:

Таблица 7 – Результаты тестирования

Метод	**Адрес запроса**	**Код ответа**	**Результат**
GET	/posts	200	успешный ответ
POST	/posts	201	создан новый объект

Задание 8. С помощью DevTools проанализировать сетевые запросы веб-приложения, которое использует API.

Задание 9. Найти пример запроса, содержащего JSON-ответ, и описать его структуру.

Задание 10. Подготовить отчёт о выполнении лабораторной работы, включающий:

- описание API
- описание HTTP-методов
- результаты тестирования API
- таблицу выполненных запросов
- скриншоты DevTools
- описание структуры JSON-ответа

Контрольные вопросы

1. Что такое API?
2. Для чего используется REST API?
3. Какие HTTP-методы используются при работе с API?
4. Что такое JSON?
5. Для чего используется инструмент Postman?
6. Какие возможности предоставляет Chrome DevTools?
7. Для чего используется вкладка Network?
8. Какие параметры можно анализировать в сетевых запросах?

Лабораторная работа №10. Организация задач и автоматизация рутинных процессов

Цель работы: изучить методы организации задач в программных проектах, освоить инструменты управления задачами и научиться автоматизировать рутинные процессы разработки с использованием современных цифровых сервисов.

Теоретическая часть

Разработка программного обеспечения представляет собой сложный процесс, включающий большое количество задач. Для эффективной работы необходимо правильно организовать процесс разработки и распределить задачи между участниками проекта.

Организация задач позволяет:

- планировать этапы разработки
- контролировать выполнение работ
- распределять обязанности между участниками проекта
- отслеживать прогресс выполнения проекта
- повышать эффективность командной работы

В современных проектах для управления задачами используются специальные системы управления проектами.

Системы управления задачами

Системы управления задачами позволяют создавать задачи, назначать исполнителей, устанавливать сроки выполнения и отслеживать прогресс проекта.

Одним из наиболее популярных подходов к организации задач является методология Kanban.

Метод Kanban основан на визуализации рабочего процесса с помощью специальной доски задач.

Классическая Kanban-доска состоит из нескольких колонок:

- Задачи (To Do)
- В работе (In Progress)
- Завершено (Done)

Каждая задача представлена карточкой, которая перемещается между колонками по мере выполнения.

Инструменты управления проектами

Для управления задачами используются различные программные сервисы.

Например:

Trello – инструмент для визуального управления задачами с использованием Kanban-досок.

YouTrack – система управления задачами и отслеживания ошибок, часто используемая в разработке программного обеспечения.

GitLab – платформа разработки программного обеспечения, включающая систему управления задачами и репозитории кода.

Такие системы позволяют команде разработчиков эффективно координировать свою работу.

Рутинные процессы разработки

В процессе разработки программисты выполняют множество повторяющихся операций:

- сборка проекта
- запуск тестов
- обновление зависимостей
- проверка кода
- развертывание приложения
- обновление документации

Если эти действия выполнять вручную, они могут занимать много времени и приводить к ошибкам.

Автоматизация процессов

Автоматизация позволяет значительно упростить выполнение повторяющихся задач.

В программной разработке автоматизация реализуется с помощью скриптов, инструментов сборки и систем непрерывной интеграции.

Одним из распространённых инструментов автоматизации является

GitHub Actions – система автоматизации процессов разработки.

Она позволяет автоматически выполнять действия при изменении кода в репозитории.

Например:

- запуск тестов при каждом изменении кода
- сборка проекта
- проверка качества кода
- автоматическое развертывание приложения

Аналогичные возможности предоставляет система автоматизации GitLab CI/CD.

Эти инструменты позволяют создавать сценарии автоматизации, которые выполняются на сервере при наступлении определённых событий.

Планирование задач

Грамотное планирование задач является важной частью управления проектами.

Задачи должны быть:

- чётко сформулированы
- разделены на небольшие этапы
- обладать конкретными сроками выполнения
- иметь назначенного исполнителя

Хорошо организованная система задач помогает команде эффективно работать и достигать поставленных целей.

Практическая часть

Задание 1. Изучить один из инструментов управления задачами:

- Trello
- YouTrack
- система задач в GitLab.

Задание 2. Создать новый проект в выбранной системе управления задачами.

Задание 3. Создать Kanban-доску проекта со следующими колонками:

- Новые задачи
- В работе
- Тестирование
- Завершено

Задание 4. Создать не менее 8–10 задач для программного проекта.

Примеры задач:

1. создание структуры проекта
2. разработка основной логики программы
3. создание интерфейса пользователя
4. написание тестов
5. подготовка документации

Задание 5. Назначить приоритет задач и установить сроки выполнения.

Задание 6. Переместить задачи между колонками Kanban-доски, имитируя процесс выполнения проекта.

Задание 7. Создать таблицу классификации задач проекта.

Пример структуры таблицы:

Таблица 8 – Классификации задач проекта

Задача	Тип задачи	Приоритет	Статус
Создание интерфейса	разработка	высокий	в работе
Написание тестов	тестирование	средний	запланировано

Задание 8. Создать простой скрипт автоматизации для проекта.

Например:

- запуск программы
- запуск тестов
- проверка структуры проекта

Задание 9. Изучить возможности автоматизации в системе контроля версий. Создать пример сценария автоматизации, который выполняет:

- проверку кода
- запуск тестов
- создание отчёта о выполнении

Задание 10. Подготовить отчёт о выполнении лабораторной работы, включающий:

- описание используемой системы управления задачами
- структуру Kanban-доски
- список созданных задач
- таблицу классификации задач
- пример скрипта автоматизации
- скриншоты системы управления задачами

Контрольные вопросы

1. Для чего используется организация задач в программных проектах?
2. Что представляет собой метод Kanban?
3. Какие системы управления задачами используются в разработке программного обеспечения?
4. Какие процессы можно автоматизировать в программных проектах?
5. Для чего используются системы CI/CD?
6. Какие преимущества даёт автоматизация рутинных процессов?
7. Какие элементы должна содержать задача в системе управления проектом?
8. Какие инструменты используются для автоматизации разработки?

Лабораторная работа №11. Настройка безопасного проекта и управление секретами

Цель работы: изучить основные принципы обеспечения безопасности программных проектов, освоить методы защиты конфиденциальных данных и научиться правильно управлять секретами в программных системах.

Теоретическая часть

В процессе разработки программного обеспечения разработчики работают с различными конфиденциальными данными. К таким данным могут относиться:

- пароли
- ключи доступа
- токены авторизации
- ключи API
- учётные данные для подключения к базам данных

Неправильное хранение таких данных может привести к серьёзным проблемам безопасности.

Одна из распространённых ошибок начинающих разработчиков размещение секретных данных непосредственно в исходном коде проекта. Если такой код будет опубликован в репозитории, злоумышленники могут получить доступ к конфиденциальной информации.

Поэтому в современных программных проектах используются специальные методы управления секретами.

Секреты и конфиденциальные данные

В информационной безопасности под секретами понимаются данные, доступ к которым должен быть ограничен.

К таким данным относятся:

- пароли пользователей
- ключи шифрования
- токены доступа
- ключи API
- сертификаты безопасности

Правильное управление секретами является важной частью защиты информационных систем.

Переменные окружения

Одним из распространённых способов хранения конфиденциальных данных являются переменные окружения.

Environment Variable – это переменная операционной системы, которая может хранить различные параметры конфигурации программы.

Использование переменных окружения позволяет:

- отделить конфиденциальные данные от исходного кода
- упростить настройку программного окружения
- обеспечить гибкость конфигурации приложения

Например, данные для подключения к базе данных могут храниться в переменных окружения, а не в коде программы.

Файлы конфигурации

Для хранения настроек приложения часто используются специальные файлы конфигурации.

Одним из популярных форматов является JSON.

Также широко используются файлы формата YAML.

В конфигурационных файлах могут храниться параметры приложения:

- адреса серверов
- порты подключения
- настройки логирования
- параметры запуска программы

Однако секретные данные не рекомендуется хранить в открытых конфигурационных файлах.

Файл .env

Во многих проектах используется специальный файл конфигурации – .env.

Файл .env содержит переменные окружения, используемые приложением. Такой файл обычно не включается в репозиторий проекта.

Для предотвращения его публикации используется файл **Gitignore**.

Файл .gitignore определяет список файлов и каталогов, которые не должны попадать в репозиторий.

Например, в .gitignore могут быть указаны:

- файлы конфигурации
- файлы логов
- временные файлы
- локальные настройки среды разработки

Это помогает защитить конфиденциальную информацию.

Токены доступа

Для доступа к внешним сервисам часто используются специальные ключи – токены доступа.

API Token – это уникальный ключ, используемый для авторизации приложения при обращении к внешнему сервису.

Токены могут использоваться для:

- доступа к API сервисов
- интеграции с облачными платформами
- подключения к системам автоматизации

Надёжное хранение токенов является важной задачей при разработке программных систем.

Безопасность репозиториев

При работе с системами контроля версий необходимо соблюдать правила безопасности.

Основные рекомендации:

- не хранить пароли в коде
- не публиковать секретные ключи
- использовать файл .gitignore
- ограничивать доступ к репозиториям регулярно менять ключи доступа

Соблюдение этих правил позволяет значительно повысить безопасность программного проекта.

Практическая часть

Задание 1. Создать тестовый программный проект или использовать ранее созданный проект.

Задание 2. Создать файл конфигурации .env для хранения параметров приложения.

Добавить в файл следующие переменные:

- API_KEY
- DATABASE_PASSWORD
- APP_PORT

Задание 3. Настроить приложение для чтения параметров из переменных окружения.

Задание 4. Создать файл .gitignore и добавить в него следующие элементы:

- .env
- лог-файлы
- временные файлы проекта

Задание 5. Проверить, что файл .env не добавляется в репозиторий системы контроля версий.

Задание 6. Создать пример конфигурационного файла проекта в формате JSON или YAML.

Файл должен содержать:

- настройки приложения
- параметры подключения
- настройки логирования

Задание 7. Создать таблицу классификации конфиденциальных данных проекта.

Пример структуры таблицы:

Таблица 9 – Конфиденциальные данные проекта

Тип данных	Пример	Способ хранения
пароль	пароль базы данных	переменная окружения
API-ключ	ключ внешнего сервиса	файл .env
токен доступа	токен авторизации	защищённое хранилище

Задание 8. Проанализировать возможные риски безопасности проекта и описать способы их предотвращения.

Задание 9. Подготовить схему безопасной конфигурации проекта, включающую:

- репозиторий проекта
- файл .gitignore
- переменные окружения
- конфигурационные файлы

Задание 10. Подготовить отчёт о выполнении лабораторной работы, включающий:

- описание методов хранения секретов
- пример файла .env
- пример файла .gitignore
- пример конфигурационного файла
- таблицу конфиденциальных данных
- описание мер безопасности проекта

Контрольные вопросы

1. Что понимается под секретами в программных проектах?
2. Почему нельзя хранить пароли в исходном коде программы?
3. Что такое переменные окружения?
4. Для чего используется файл .env?
5. Какую функцию выполняет файл .gitignore?
6. Что такое API-токен?

7. Какие риски безопасности могут возникнуть при публикации проекта?
8. Какие методы используются для защиты конфиденциальных данных?

Лабораторная работа №12. Обеспечение безопасности проектов

Цель работы: изучить основные принципы обеспечения информационной безопасности программных проектов, познакомиться с распространёнными угрозами безопасности программных систем и освоить методы защиты программных приложений и инфраструктуры.

Теоретическая часть

Современные программные системы обрабатывают большое количество данных, включая персональную информацию пользователей, финансовые данные и конфиденциальные сведения организаций. Поэтому обеспечение безопасности программных проектов является одной из важнейших задач при разработке программного обеспечения.

Информационная безопасность программного проекта направлена на защиту программной системы от различных угроз, которые могут привести к утечке данных, нарушению работы системы или несанкционированному доступу.

Основные цели информационной безопасности можно описать с помощью классической модели CIA Triad.

Эта модель включает три ключевых принципа:

- конфиденциальность – защита данных от несанкционированного доступа
- целостность – защита данных от несанкционированного изменения
- доступность – обеспечение доступности системы для пользователей

Любая система информационной безопасности должна учитывать все три компонента.

Основные угрозы безопасности программных систем

Программные системы могут подвергаться различным видам атак. Злоумышленники могут использовать уязвимости в программном коде, неправильную конфигурацию серверов или ошибки в архитектуре системы.

К основным угрозам безопасности относятся:

- несанкционированный доступ к системе
- утечка конфиденциальных данных
- подмена данных
- нарушение работы сервера

- атаки на веб-приложения

Одним из наиболее известных источников информации о уязвимостях веб-приложений является OWASP.

Организация OWASP публикует список наиболее распространённых уязвимостей веб-приложений.

Наиболее известным является рейтинг OWASP Top 10.

В этот список входят наиболее распространённые типы уязвимостей программных систем.

Уязвимости веб-приложений

Существует множество типов уязвимостей, которые могут встречаться в программных системах.

SQL Injection – это тип атаки, при котором злоумышленник вставляет вредоносный SQL-код в запрос к базе данных.

Если приложение неправильно обрабатывает пользовательский ввод, злоумышленник может получить доступ к базе данных или изменить её содержимое.

Cross-Site Scripting – это атака, при которой вредоносный скрипт внедряется в веб-страницу и выполняется в браузере пользователя.

Такие атаки могут использоваться для кражи сессионных данных или выполнения вредоносных действий.

Атаки подбора паролей

Злоумышленники могут пытаться получить доступ к системе методом перебора паролей.

Brute Force Attack – это метод подбора паролей путём перебора большого количества возможных комбинаций.

Для защиты от таких атак используются ограничения числа попыток входа и механизмы блокировки учетных записей.

Шифрование данных

Одним из важнейших способов защиты информации является шифрование.

Encryption – это процесс преобразования данных в форму, недоступную для чтения без специального ключа.

Шифрование используется для защиты:

- передаваемых данных
- конфиденциальной информации
- паролей пользователей
- ключей доступа

Для защиты соединения между клиентом и сервером используется протокол HTTPS.

HTTPS обеспечивает защищённую передачу данных между браузером пользователя и веб-сервером.

Аутентификация и авторизация

В программных системах необходимо различать два важных процесса:

- Authentication – процесс проверки личности пользователя.
- Authorization – процесс определения прав доступа пользователя.

После успешной аутентификации система определяет, какие действия пользователь может выполнять.

Для повышения безопасности могут использоваться дополнительные механизмы защиты, например Two-Factor Authentication.

Двухфакторная аутентификация требует подтверждения личности пользователя с использованием двух различных факторов.

Безопасность программного проекта включает не только защиту кода, но и защиту серверной инфраструктуры.

Необходимо учитывать следующие аспекты:

- безопасная настройка серверов
- обновление программного обеспечения
- ограничение доступа к серверу
- использование защищённых соединений
- контроль прав доступа

Неправильная конфигурация серверной инфраструктуры может привести к серьёзным уязвимостям системы.

Безопасность разработки

В процессе разработки программного обеспечения необходимо соблюдать принципы безопасного программирования.

Основные рекомендации:

- проверять пользовательский ввод
- использовать подготовленные SQL-запросы
- ограничивать доступ к конфиденциальным данным
- использовать безопасные методы аутентификации
- регулярно обновлять используемые библиотеки

В крупных проектах применяются методы безопасной разработки, такие как Secure Software Development Lifecycle.

Этот подход предполагает включение механизмов безопасности на всех этапах разработки программного обеспечения.

Практическая часть

Задание 1. Изучить основные угрозы безопасности программных систем и кратко описать не менее пяти видов уязвимостей.

Задание 2. Создать таблицу классификации угроз безопасности программного проекта.

Пример структуры таблицы:

Таблица 10 – Угрозы безопасности проекта

Тип угрозы	**Описание**	**Возможные последствия**	**Методы защиты**
SQL Injection	внедрение SQL-кода	доступ к базе данных	проверка ввода
XSS	внедрение скриптов	кража данных пользователя	фильтрация данных

Задание 3. Проанализировать архитектуру программного проекта и определить потенциальные точки угроз безопасности.

Задание 4. Разработать список мер защиты программного проекта.

Пример мер безопасности:

- проверка входных данных
- ограничение доступа пользователей
- использование шифрования
- защита API
- использование безопасной аутентификации

Задание 5. Создать схему системы безопасности программного проекта, включающую:

- клиентское приложение
- сервер приложения
- базу данных
- систему аутентификации
- систему защиты данных

Задание 6. Изучить механизм работы HTTPS и описать процесс защищённого соединения между клиентом и сервером.

Задание 7. Разработать рекомендации по обеспечению безопасности программного проекта.

Задание 8. Создать сравнительную таблицу методов защиты данных.

Пример структуры таблицы:

Таблица 11 – Методы защиты данных

Метод защиты	Назначение	Преимущества	Ограничения
шифрование	защита данных	высокая безопасность	требует вычислительных ресурсов
аутентификация	проверка пользователя	контроль доступа	необходимость управления учетными данными

Задание 9. Подготовить отчёт о выполнении лабораторной работы, включающий:

- описание угроз безопасности
- таблицу уязвимостей
- схему безопасности проекта
- описание методов защиты
- рекомендации по обеспечению безопасности

Контрольные вопросы

1. Что такое информационная безопасность программных проектов?
2. Какие цели включает модель CIA?
3. Какие угрозы могут возникать в программных системах?
4. Что такое SQL-инъекция?
5. Что представляет собой XSS-атака?
6. Для чего используется шифрование данных?
7. Чем отличается аутентификация от авторизации?
8. Какие методы используются для защиты программных проектов?
9. Что такое HTTPS?
10. Какие этапы включает безопасная разработка программного обеспечения?

Список литературы

1. Зубова, Е. Д. Информационные технологии в профессиональной деятельности : учебное пособие для СПО / Е. Д. Зубова. – 3-е изд., стер. – Санкт-Петербург : Лань, 2024. – 212 с. – ISBN 978-5-507-47558- 2. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/388985 (дата обращения: 20.03.2026). – Режим доступа: для авториз. пользователей.
2. Петлина, Е. М. Информационные технологии в профессиональной деятельности : учебное пособие для СПО / Е. М. Петлина, А. В. Горбачев. – 2-е изд. – Саратов : Профобразование, 2024. – 111 с. – ISBN 978-5-4488-2183-7. – Текст : электронный // Электронный ресурс цифровой образовательной среды СПО PROFобразование : [сайт]. – URL: https://profspo.ru/books/142224 (дата обращения: 20.03.2026). – Режим доступа: для авторизир. пользователей
3. Синаторов, С. В. Информационные технологии в профессиональной деятельности : учебное пособие / С. В. Синаторов, О. В. Пикулик. – Москва : ИНФРА–М, 2025. – 277 с. – (Среднее профессиональное образование). – DOI 10.12737/1092991. – ISBN 978-5-16-016278-2. – Текст : электронный. – URL: https://znanium.ru/catalog/product/2168881 (дата обращения: 20.03.2026). – Режим доступа: по подписке.
4. Евстафьев, В. А. Искуственный интеллект и нейросети: практика применения в рекламе : учебное пособие / В. А. Евстафьев, М. А. Тюков. – Москва : Издательско–торговая корпорация «Дашков и К°», 2023. – 426 с. – ISBN 978–5–394–05703–8. – Текст : электронный. – URL: https://znanium.ru/catalog/product/2133542 (дата обращения: 20.03.2026). – Режим доступа: по подписке.
5. Баланов, А. Н. Защита информационных систем. Кибербезопасность : учебное пособие для СПО / А. Н. Баланов. – 2-е изд., стер. – Санкт-Петербург : Лань, 2025. – 84 с. – ISBN 978-5-507-53004-5. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/464183 (дата обращения: 26.11.2025). – Режим доступа: для авториз. пользователей.

6. Белов, М. А. Облачные сервисы в корпоративном управлении : учебное пособие / М. А. Белов, С. В. Потемкина, А. А. Миловидова. – Дубна : Государственный университет «Дубна», 2025. – 68 с. – ISBN 978-5-89847-728-8. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/497414 (дата обращения: 20.03.2026). – Режим доступа: для авториз. пользователей.