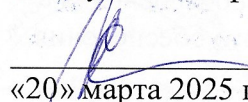


Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)

УТВЕРЖДАЮ

Заведующий кафедрой ИСПИ

 И.Е. Жигалов

«20» марта 2025 г.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ЛАБОРАТОРНЫМ РАБОТАМ
МЕЖДИСЦИПЛИНАРНОГО КУРСА

«ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ПОДДЕРЖКИ РАЗРАБОТКИ КОДА
ИНФОРМАЦИОННЫХ РЕСУРСОВ»

В РАМКАХ ПРОФЕССИОНАЛЬНОГО МОДУЛЯ

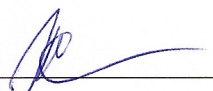
«ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ИНФОРМАЦИОННЫХ РЕСУРСОВ»

09.02.09 Веб-разработка
Разработчик веб приложений

Владимир, 2025

Методические указания к лабораторным работам междисциплинарного курса «Инструментальные средства поддержки разработки кода информационных ресурсов» разработал старший преподаватель кафедры ИСПИ Шамышев А.А.

Методические указания к лабораторным работам рассмотрены и одобрены на заседании УМК специальности 09.02.09 Веб-разработка протокол № 1 от «10» марта 2025 г.

Председатель УМК специальности  И.Е. Жигалов

Методические указания к лабораторным работам рассмотрены и одобрены на заседании кафедры ИСПИ протокол № 7а от «12» марта 2025 г.

Рецензент от работодателя:
руководитель группы обеспечения
качества программного обеспечения
ООО «БСЦ МСК»



 С.С. Смирнова

ОГЛАВЛЕНИЕ

Лабораторная работа №1	3
Лабораторная работа №2	9
Лабораторная работа №3	14
Лабораторная работа №4	20
Лабораторная работа №5	27
Лабораторная работа №6	36
Лабораторная работа №7	45
Лабораторная работа №8	53
Лабораторная работа №9	61
СПИСОК ЛИТЕРАТУРЫ.....	69

Лабораторная работа №1

«Установка и управление пакетами с использованием npm»

Цель выполнения работы:

- Ознакомиться с системой управления пакетами npm (Node Package Manager).
- Освоить основные команды npm для установки, обновления и удаления пакетов.
- Научиться инициализировать проекты, управлять зависимостями и публиковать собственные пакеты.
- Развить навыки работы с файлами package.json и понимание различий между dependencies и devDependencies.

Введение в npm

npm (Node Package Manager) — это система управления пакетами для платформы Node.js, которая позволяет разработчикам легко устанавливать, обновлять и управлять зависимостями проектов. npm предоставляет доступ к огромному количеству пакетов, которые могут быть использованы для ускорения процесса разработки.

Основные понятия

- **Пакет** — это модуль или библиотека, которую можно установить и использовать в проекте.
- **Репозиторий** — место хранения пакетов. Центральным репозиторием для npm является **npm Registry**.
- **package.json** — файл, содержащий информацию о проекте и его зависимостях.
- **dependencies** — зависимости, необходимые для работы приложения.
- **devDependencies** — зависимости, необходимые только для разработки и тестирования.

Основные команды npm

- *npm init* — инициализация нового проекта и создание файла package.json.
- *npm install <package>* — установка пакета и добавление его в зависимости проекта.

- `npm install <package> --save-dev` — установка пакета как зависимости для разработки.
- `npm update` — обновление установленных пакетов до последних версий.
- `npm uninstall <package>` — удаление пакета из проекта.
- `npm publish` — публикация собственного пакета в npm Registry.

Структура файла `package.json`

Файл `package.json` содержит метаданные о проекте, такие как имя, версия, описание, автор и список зависимостей. Пример структуры:

```
{
  "name": "my-project",
  "version": "1.0.0",
  "description": "Описание проекта",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "express": "^4.17.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.7"
  },
  "author": "Ваше Имя",
  "license": "MIT"
}
```

Различия между `dependencies` и `devDependencies`:

- **`dependencies`** — содержат пакеты, необходимые для запуска приложения в рабочей среде.
- **`devDependencies`** — содержат пакеты, необходимые только в процессе разработки, такие как инструменты для тестирования или автоперезагрузки сервера.

Порядок выполнения работы

Шаг 1: Установка Node.js и npm

1. Скачать установочный файл Node.js:

- Перейдите на официальный сайт nodejs.org.
- Выберите LTS версию для установки.

2. Установить Node.js:

- Запустите скачанный установочный файл.
- Следуйте инструкциям установщика, принимая лицензионное соглашение и выбирая стандартные параметры установки.

3. Проверить установку:

- Откройте терминал или командную строку.
- Выполните команды `node -v` и `npm -v` для проверки установленных версий Node.js и npm.

Шаг 2: Инициализация проекта

1. Создать директорию проекта:

- В терминале выполните команду `mkdir my-project` для создания новой папки.
- Перейдите в созданную директорию с помощью `cd my-project`.

2. Инициализировать package.json:

- Выполните команду `npm init` и следуйте интерактивным инструкциям.
- Альтернативно, используйте `npm init -y` для автоматической генерации `package.json` с настройками по умолчанию.

Шаг 3: Установка пакетов

1. Установить пакет как зависимость:

- Выполните команду `npm install express` для установки фреймворка Express.
- Проверьте, что пакет добавлен в раздел `dependencies` файла `package.json`.

2. Установить пакет как зависимость для разработки:

- Выполните команду `npm install nodemon --save-dev` для установки nodemon.
- Проверьте, что пакет добавлен в раздел `devDependencies`.

Шаг 4: Управление пакетами

1. Обновление пакетов:

- Выполните команду `npm update` для обновления всех установленных пакетов до последних версий, соответствующих указанным диапазонам версий.

2. Удаление пакета:

- Выполните команду `npm uninstall express` для удаления пакета Express из проекта.
- Убедитесь, что пакет удален из `dependencies` в `package.json`.

Шаг 5: Работа с файлами `package.json`

1. Редактирование скриптов:

- Откройте файл `package.json` в текстовом редакторе.
- Добавьте или измените скрипты в разделе `scripts`. Например:

```
"scripts": {  
  "start": "node index.js",  
  "dev": "nodemon index.js"  
}
```

2. Запуск скриптов

- Выполните команду: `npm run start` для запуска приложения
- Выполните команду `npm run dev` для запуска приложения nodemon

Задание к лабораторной работе:

1. Установка и управление пакетами

- Создайте новый проект, инициализируйте `package.json` с помощью `npm init -y`.
- Установите несколько пакетов, например, `lodash` как зависимость и `eslint` как зависимость для разработки.

- Добавьте скрипты для запуска приложения и для запуска ESLint в package.json.
 - Обновите установленные пакеты до последних версий.
 - Удалите один из установленных пакетов и убедитесь, что он удален из package.json.
2. Добавьте в package.json скрипт для автоматической проверки кода с помощью ESLint:

```
"scripts": {  
  "lint": "eslint ."  
}
```

3. Использование глобальных пакетов:
1. Установите пакет глобально, например, `npm install -g typescript`.
 2. Проверьте, что пакет доступен из любого места в системе.

Содержание отчета

Введение

- **Цель и задачи лабораторной работы:**
 - Краткое изложение целей выполнения работы и ожидаемых результатов.

Теоретическая часть

- **Описание изученных понятий и технологий:**
 - Введение в npm, основные команды и их назначение.
 - Структура файла package.json и различия между dependencies и devDependencies.

Практическая часть

- **Пошаговое описание выполненных действий:**
 - Установка Node.js и npm.
 - Инициализация проекта и настройка package.json.
 - Установка, обновление и удаление пакетов.
 - Создание и использование скриптов в package.json.
- **Примеры команд и их вывод:**

- Включите скриншоты терминала или копии вывода команд для подтверждения выполненных действий.

Результаты

- **Полученные результаты и их анализ:**
 - Описание установленных пакетов и настроенных скриптов
 - Скриншоты результата работы.
 - Сравнение версий пакетов до и после обновления.

Заключение

- **Выводы о проделанной работе:**
 - Оценка достигнутых целей.

Приложения (по необходимости)

- **Скриншоты, код, дополнительные материалы:**
 - Включите все необходимые визуальные материалы для иллюстрации выполненной работы.

Контрольные вопросы

1. Что такое npm и для чего он используется?
2. Опишите процесс инициализации нового проекта с помощью npm.
3. Какие команды используются для установки и удаления пакетов с помощью npm?
4. Что такое package.json и какие данные он содержит?
5. Каковы основные отличия между dependencies и devDependencies в package.json?
6. Как обновить все установленные пакеты до последних версий с помощью npm?
7. Как установить пакет глобально и какие преимущества это дает?
8. Что такое package-lock.json и какую роль он играет в управлении зависимостями?

Лабораторная работа №2

«Поиск и интеграция готовых модулей и библиотек в общедоступных репозиториях»

Цель выполнения работы:

- Ознакомиться с процессом поиска готовых модулей и библиотек в общедоступных репозиториях.
- Освоить методы оценки качества и совместимости найденных модулей с текущим проектом.
- Научиться интегрировать выбранные модули и библиотеки в проект, управлять зависимостями и конфигурациями.
- Развить навыки работы с различными инструментами и платформами для поиска и интеграции программных компонентов.

Повторное использование кода — это практика использования уже существующих модулей, библиотек или компонентов в новых проектах для ускорения разработки, повышения надежности и снижения затрат на создание новых решений с нуля. Повторное использование способствует стандартизации и повышению качества программного обеспечения.

Общедоступные репозитории — это онлайн-платформы, где разработчики могут размещать, искать и обмениваться модулями и библиотеками. Основные преимущества использования общедоступных репозиторий включают доступ к большому количеству проверенных решений, возможность сотрудничества с сообществом и ускорение разработки за счет использования готовых компонентов.

Основные общедоступные репозитории:

- npm (Node Package Manager): Репозиторий для пакетов JavaScript и Node.js.
- PyPI (Python Package Index): Репозиторий для пакетов Python.
- Maven Central: Репозиторий для Java-пакетов.
- RubyGems: Репозиторий для Ruby-пакетов.
- GitHub Packages: Универсальный репозиторий, поддерживающий различные языки и пакеты.
- NuGet: Репозиторий для .NET-пакетов.

Поиск модулей и библиотек включает использование поисковых функций на платформах репозитория, применение фильтров по категориям, популярности, обновлениям и рейтингу.

Критерии оценки качества модуля:

- Популярность и количество загрузок: Высокое количество загрузок и звезд на GitHub могут указывать на надежность и востребованность.
- Активность разработки: Регулярные обновления и активное участие сообщества свидетельствуют о поддержке и актуальности.
- Документация: Хорошая документация облегчает интеграцию и использование модуля.
- Совместимость: Проверка совместимости версии модуля с текущими версиями используемых технологий.
- Лицензия: Убедитесь, что лицензия модуля позволяет его использование в вашем проекте (например, MIT, Apache 2.0).

Интеграция модулей включает добавление выбранных библиотек в проект, настройку зависимостей и конфигураций, а также тестирование для обеспечения корректной работы.

Основные инструменты для управления зависимостями:

- npm: Для проектов на JavaScript и Node.js.
- pip: Для проектов на Python.
- Maven/Gradle: Для проектов на Java.
- Bundler: Для проектов на Ruby.
- NuGet: Для проектов на .NET.

Порядок выполнения работы:

Шаг 1: Определение требований проекта

1. Анализ проекта:

- Определите функциональные и нефункциональные требования проекта.
- Определите, какие модули или библиотеки могут потребоваться для реализации этих требований.

Шаг 2: Поиск готовых модулей и библиотек

1. Использование общедоступных репозитория:

- Перейдите на платформу репозитория [npm](#)
- 2. **Поиск по ключевым словам:**
 - Введите ключевые слова, связанные с необходимым функционалом (например, "authentication", "database ORM", "data visualization").
- 3. **Фильтрация результатов:**
 - Используйте фильтры по популярности, обновлениям, лицензии и другим критериям для выбора наиболее подходящих модулей.

Шаг 3: Оценка качества и совместимости модулей

1. **Анализ документации:**
 - Изучите документацию модуля для понимания его функционала и способов интеграции.
2. **Проверка лицензии:**
 - Убедитесь, что лицензия модуля позволяет его использование в вашем проекте.
3. **Анализ активности разработки:**
 - Проверьте последние обновления и активность сообщества разработчиков.
4. **Тестирование:**
 - Установите модуль в тестовом окружении и проверьте его работу.

Шаг 4: Интеграция выбранных модулей в проект

1. **Установка модуля:**
 - `npm install <package-name>`
2. **Настройка зависимостей:**
 - Обновите файл зависимостей `package.json` при необходимости.
3. **Конфигурация:**
 - Настройте параметры модуля в соответствии с требованиями проекта.
4. **Интеграция кода:**
 - Добавьте необходимые вызовы и импорты в код проекта для использования функций модуля.
5. **Тестирование интеграции:**
 - Проведите тестирование, чтобы убедиться в корректной работе интегрированного модуля.

Шаг 5: Документирование процесса интеграции

1. Запись выполненных шагов:

- Описать процесс поиска, оценки и интеграции модулей.

2. Отображение результатов:

- Включить скриншоты, примеры кода и выводы тестирования.

Задание к лабораторной работе:

1. Найдите и интегрируйте готовый модуль npm в ваш проект. Оцените его функциональность и совместимость с проектом. Приведите примеры использования выбранного модуля.

Содержание отчета:

Введение

- **Цель и задачи лабораторной работы:**

- Краткое изложение целей выполнения работы и ожидаемых результатов.

Теоретическая часть

- **Описание изученных понятий и технологий:**

- Введение в повторное использование кода, общедоступные репозитории, критерии оценки модулей и библиотек.

Практическая часть

- **Пошаговое описание выполненных действий:**

- Процесс поиска модулей, оценка их качества и совместимости, интеграция в проект.
- Процесс использования модулей в проекте.

- **Примеры команд и их вывод:**

- Включите скриншоты или текст из терминала, среды разработки для подтверждения выполненных действий.

Результаты

- **Полученные результаты и их анализ:**

- Описание интегрированных модулей, настроенных зависимостей и выполненных тестов.

Заключение

- **Выводы о проделанной работе:**
 - Оценка достигнутых целей, полезность использованных модулей и их влияние на проект.

Приложения (по необходимости)

- **Скриншоты, код, дополнительные материалы:**
 - Включите все необходимые визуальные материалы для иллюстрации выполненной работы (скриншоты установки, примеры кода, результаты тестов).

Контрольные вопросы:

1. Что такое общедоступный репозиторий и какие основные преимущества его использования?
2. Какие критерии следует учитывать при выборе модуля или библиотеки для интеграции в проект?
3. Опишите процесс интеграции выбранного модуля в проект.
4. Что такое `package.json` и какую роль он играет при работе с `npm`?
5. Какие шаги необходимо предпринять для проверки совместимости нового модуля с текущим проектом?
6. Почему важно проверять лицензию модуля перед его использованием в проекте?
7. Какие инструменты могут помочь в управлении зависимостями проекта?
8. Опишите, как можно обновить модуль до последней версии с помощью `npm`.
9. Как публикация собственного модуля в общедоступном репозитории может быть полезна для сообщества разработчиков?

Лабораторная работа №3

«Создание и публикация собственного пакета в общедоступном репозитории»

Цель выполнения работы:

- Научиться создавать собственные пакеты или модули для использования в проектах.
- Освоить процесс подготовки и настройки проекта для публикации в общедоступном репозитории npm.
- Изучить методы управления версиями пакетов и обеспечения их совместимости.
- Развить навыки документирования и тестирования собственных пакетов для повышения их качества и удобства использования другими разработчиками.

Пакет — это коллекция модулей или библиотек, которая предоставляет определённую функциональность и может быть использована в других проектах. Создание собственных пакетов позволяет разработчикам делиться своими решениями, повторно использовать код и способствовать развитию сообщества.

Основные понятия:

- **npm (Node Package Manager):** Система управления пакетами для платформы Node.js, которая позволяет публиковать и распространять пакеты.
- **package.json:** Файл конфигурации, содержащий метаданные о пакете, такие как название, версия, автор, зависимости и скрипты.
- **Лицензия:** Правовые условия использования, изменения и распространения пакета (например, MIT, Apache 2.0).

Подготовка к публикации пакета

Перед публикацией пакета необходимо:

- **Разработать функционал:** Написать код, который будет предоставлять полезную функциональность другим разработчикам.
- **Настроить package.json:** Указать необходимые метаданные, зависимости и скрипты.

- **Документировать пакет:** Создать README файл с описанием, инструкциями по установке и примерами использования.

Процесс публикации пакета:

1. **Создание учетной записи на npm:**
 - Перейдите на [страницу регистрации в npm](#) и создайте учетную запись.
2. **Вход в npm через терминал:**
 - Выполнить команду `npm login` и ввести свои учетные данные.
3. **Публикация пакета:**
 - Выполнить команду `npm publish` в корневой директории проекта.
 - Убедитесь, что пакет доступен в [npm Registry](#).

Порядок выполнения работы:

Шаг 1: Создание нового пакета

1. **Создать директорию для пакета:**
 - В терминале выполните команду `mkdir project-name`.
 - Перейдите в созданную директорию с помощью `cd project-name`.
2. **Инициализировать package.json:**
 - Выполните команду `npm init` и следуйте инструкциям.
 - Альтернативно, используйте `npm init -y` для автоматической генерации `package.json` с настройками по умолчанию.

Шаг 2: Разработка функционала пакета

1. **Создать основной файл пакета:**
 - Создайте файл `index.js`.
 - Напишите простой модуль, например, функцию для сложения двух чисел:

```
function add(a, b) {  
    return a + b;  
}  
  
module.exports = add;
```

Шаг 3: Документирование пакета

1. Создать файл README.md:

- Опишите назначение пакета, инструкции по установке и примеры использования, используя разметку Markdown.

Шаг 4: Настройка package.json для публикации

1. Проверить метаданные:

- Убедитесь, что поля name, version, description, main, scripts, keywords, author, license заполнены корректно.

2. Добавить ключевые слова:

- Это поможет другим разработчикам найти ваш пакет. Например:

```
"keywords": [  
  
  "addition",  
  
  "math",  
  
  "utility"  
]
```

Шаг 5: Публикация пакета на npm

1. Войти в npm через терминал:

- Выполните команду `npm login` и введите свои учетные данные.

2. Опубликовать пакет:

- Выполните команду `npm publish`.
- Убедитесь, что пакет успешно опубликован и доступен в [npm Registry](https://www.npmjs.com/).

3. Проверить опубликованный пакет:

- Перейдите на страницу вашего пакета на npm и убедитесь в наличии всех метаданных и документации.

Шаг 6: Управление версиями и обновления пакета

1. Внести изменения в пакет:

- Например, добавить новую функцию в `index.js`:

```
function subtract(a, b) {  
  return a - b;  
}  
  
module.exports = { add, subtract };
```

2. Изменить версию в package.json:

- Увеличьте номер версии согласно семантическому версионированию, например, с 1.0.0 на 1.1.0.

3. Опубликовать обновленный пакет:

- Выполните команду `npm publish`.

Задание к лабораторной работе:

3. Создание и публикация собственного пакета:

- Создайте пакет, который предоставляет функцию для умножения двух чисел.
- Документируйте пакет в README.md.
- Опубликуйте пакет на npm Registry.

2. Управление зависимостями:

- Добавьте в ваш пакет зависимость от внешней библиотеки (например, `lodash`).
- Обновите функционал пакета с использованием этой библиотеки.
- Опубликуйте обновленную версию пакета.

3. Документирование API пакета:

- Используйте инструмент JSDoc для генерации документации API вашего пакета.
- Добавьте примеры использования каждой функции в документацию.

Содержание отчета:

Введение

- **Цель и задачи лабораторной работы:**

- Описание целей создания и публикации собственного пакета.

Теоретическая часть

- **Описание изученных понятий и технологий**

Практическая часть

- **Пошаговое описание выполненных действий:**
 - Создание директории проекта, инициализация package.json.
 - Разработка функционала пакета.
 - Документирование пакета, настройка package.json для публикации.
 - Публикация пакета на npm, управление версиями.
- **Примеры команд и их вывод:**
 - Включите скриншоты терминала или среды разработки с выполненными командами и их результатами.

Результаты

- **Полученные результаты и их анализ:**
 - Описание опубликованного пакета, его функциональности.
 - Анализ процесса публикации и управления версиями.

Заключение

- **Выводы о проделанной работе:**
 - Оценка достигнутых целей.
 - Возможности дальнейшего развития пакета.

Приложения (по необходимости)

- **Скриншоты, код, дополнительные материалы:**
 - Скриншоты установленных пакетов, выполненных команд.
 - Примеры кода пакета.
 - Ссылки на опубликованный пакет на npm.

Контрольные вопросы

1. Что такое npm и для чего он используется?

2. Опишите процесс инициализации нового проекта с помощью `npm init`.
3. Что такое `package.json` и какие данные он содержит?
4. Объясните семантическое версионирование и его принципы.
5. Как опубликовать собственный пакет на `npm Registry`?
6. Какие шаги необходимо предпринять перед публикацией пакета?
7. Что такое лицензия пакета и почему она важна?
8. Как управлять зависимостями в проекте с помощью `npm`?
9. Какие команды используются для обновления и удаления пакетов с помощью `npm`?
10. Почему важно писать тесты для собственного пакета?

Лабораторная работа №4

«Генерация исходного кода с помощью нейронных сетей»

Цель выполнения работы:

- **Изучить** основы работы нейронных сетей, предназначенных для генерации исходного кода.
- **Освоить** использование свободно доступных нейронных сетей для автоматической генерации кода.
- **Развить** навыки интеграции и настройки нейронных сетей в процесс разработки программного обеспечения.
- **Оценить** качество и применимость автоматически сгенерированного кода в реальных проектах.

Введение в нейронные сети для генерации кода

Нейронные сети, такие как GPT, позволяют не только автоматизировать процесс написания кода, но и значительно сократить время на выполнение рутинных задач. Они могут генерировать шаблоны, проверять корректность алгоритмов, предлагать оптимизированные решения и адаптировать код под различные контексты. Применение таких технологий особенно полезно в условиях многозадачности и высокой интенсивности разработки. Важно отметить, что использование нейронных сетей не заменяет навыки программиста, а становится инструментом, дополняющим его работу.

Нейронные сети представляют собой модели машинного обучения, способные анализировать большие объемы данных и выполнять сложные задачи, включая генерацию текста и кода. В контексте разработки программного обеспечения нейронные сети могут использоваться для автоматического написания кода, что ускоряет процесс разработки и снижает вероятность ошибок. Также, нейронные сети можно использовать для обучения незнакомой технологии и пояснения к уже существующему коду в проектах.

Основные концепции:

- **GPT (Generative Pre-trained Transformer):** Архитектура нейронных сетей, разработанная компанией OpenAI, предназначенная для генерации текста.
- **Языковая модель:** Тип нейронной сети, предназначенной для обработки и генерации текста на естественном языке. Языковая модель обучается на большом

объеме текстовых данных для предсказания следующего слова, понимания контекста и выполнения задач, связанных с текстом, таких как перевод, резюмирование и написание программного кода.

- **Обучение с учителем:** Процесс обучения нейронных сетей, на основе заранее размеченных данных.
- **Токенизация:** Процесс разбиения текста на меньшие единицы (токены) для обработки нейронной сетью.
- **Промпт:** Входной запрос или инструкция, которая передается нейронной сети для выполнения задачи. Промпт формирует контекст для генерации текста, кода или выполнения других функций, определяя качество и точность результата.

Свободно доступные нейронные сети в России

В России разработаны и доступны несколько моделей нейронных сетей для генерации кода:

- **GigaChat:** Разработанный компанией Сбер, GigaChat способен вести диалоги, генерировать тексты и изображения, а также писать программный код. Он поддерживает русский и английский языки, с основным фокусом на русскоязычных пользователей. Доступен бесплатно через веб-интерфейс на сайте giga.chat, а также в виде ботов в Telegram и «ВКонтакте».
- **YandexGPT:** Инструмент от компании «Яндекс», который предоставляет возможности для создания и редактирования кода. Подходит для генерации новых идей в программировании и анализа кода в контексте заданных параметров. Нейросеть учитывает контекст запроса, что позволяет улучшить качество генерируемых решений.
- **Nicebot:** Бесплатная нейросеть от российских разработчиков, умеющая отвечать на вопросы, писать тексты, решать математические задачи и составлять программные коды на Python, JavaScript, HTML, C++ и Java. Nicebot может анализировать код, устранять ошибки и улучшать его структуру. Доступен бесплатно на сайте nicebot.ru, в мобильном приложения и Telegram-боте.

Для генерации кода рекомендуется использовать **GigaChat** от Сбер, так как его интерфейс доступен бесплатно на сайте giga.chat и более удобен в использовании, чем его аналоги.

YandexGPT не имеет своего интерфейса для ввода промпта и подразумевает использование API YandexGPT.

Применение нейронных сетей в разработке кода

Использование нейронных сетей для генерации кода включает несколько этапов:

1. **Выбор модели:** Определение подходящей нейронной сети для задачи.
2. **Настройка окружения:** Установка необходимых библиотек и инструментов.
3. **Обучение модели:** Если необходимо, обучение модели на специфических данных.
4. **Генерация кода:** Ввод запроса в модель и получение сгенерированного кода.
5. **Анализ и доработка:** Проверка корректности и доработка сгенерированного кода.

При использовании нейронных сетей для генерации кода важно проверять качество и эффективность сгенерированного кода в рамках проекта и задачи.

Порядок выполнения работы:

Шаг 1: Подготовка окружения

1. **Установка необходимых инструментов:**
 - Убедитесь, что на вашем компьютере установлены **Node.js** и **npm**.
 - Если у вас еще не установлена среда разработки, то скачайте и установите **Visual Studio Code** с официального сайта code.visualstudio.com. Можете использовать любую другую среду разработки, по вашему усмотрению.

2. **Создание нового проекта:**

- Откройте терминал и выполните команду

```
mkdir project-name
```

```
cd project-name
```

Где, *project-name* название вашего проекта

- Инициализируйте package.json:

```
npm init -y
```

Шаг 2: Изучение интерфейса GigaChat

1. **Доступ к GigaChat:**

- Перейдите на сайт giga.chat.
- Зарегистрируйтесь или войдите в существующую учетную запись.

2. Знакомство с интерфейсом:

- Изучите основные функции интерфейса: ввод промптов, просмотр сгенерированного кода и качество ответов.

Шаг 3: Генерация кода с помощью GigaChat

1. Формулирование промпта:

- Определите задачу, которую необходимо автоматизировать.

Например:

Напиши функцию на JavaScript, которая сортирует массив чисел по возрастанию.

2. Ввод промпта в GigaChat:

- Введите сформулированный промпт в поле ввода на сайте.
- Запустите процесс генерации кода.

3. Получение и анализ кода:

- Просмотрите сгенерированный код.
- Скопируйте его для дальнейшей интеграции в проект.

Шаг 4: Интеграция сгенерированного кода в проект

1. Создание файла с кодом:

- В вашем **проекте** создайте файл `sort.js` и вставьте туда сгенерированный код.

```
function sortArray(arr) {  
  
    return arr.sort((a, b) => a - b);  
  
}  
  
module.exports = sortArray;
```

2. Использование функции в проекте:

- Создайте файл `index.js` и используйте функцию сортировки

```
const sortArray = require('./sort');
```



```
const numbers = [5, 3, 8, 1, 2];

const sortedNumbers = sortArray(numbers);

console.log('Оптимизированный массив:', sortedNumbers);
```

3. Запуск проекта:

- В терминале выполните команду: `node index.js`
- Убедитесь, что вывод соответствует ожиданиям.

4. Оптимизация кода (при необходимости):

- Если сгенерированный код требует улучшений, внесите необходимые изменения и повторно протестируйте его.

Задания к лабораторной работе:

1. Генерация функции для фильтрации массива:

- Используя GigaChat, сгенерируйте функцию на JavaScript, которая фильтрует массив чисел, оставляя только четные.
- Интегрируйте сгенерированную функцию в проект и протестируйте её работу.

2. Интеграция сгенерированного кода в существующий проект

- Интегрируйте с использованием GigaChat или другой нейросети получите код функции для подсчета суммы чисел в массиве. Вставьте эту функцию в уже существующий проект и протестируйте её работу.

```
const numbers = [1, 2, 3, 4, 5];

// Вставьте функцию для подсчёта суммы здесь

// Вызовите функцию и выведите результат

const sum = calculateSum(numbers);

console.log(`Сумма чисел: ${sum}`);
```

- Интегрируйте сгенерированную функцию в проект и протестируйте её работу.

3. Исправление ошибки в коде, найденной нейросетью

- В существующем коде намеренно допущена ошибка. Используйте нейросеть для нахождения и исправления этой ошибки.

```
const numbers = [1, 2, 3, 4, 5];
```

```
const filterEvenNumbers = (arr) => arr.map(num => num % 2 === 0);
```

```
const evenNumbers = filterEvenNumbers(numbers);
```

```
console.log(`Чётные числа: ${evenNumbers}`);
```

- Передайте этот код нейросети и попросите её исправить ошибку. Убедитесь, что исправленный код работает корректно и возвращает массив только с чётными числами.

Содержание отчета:

Введение

- **Цель и задачи лабораторной работы:**
 - Описание целей создания и использования нейронных сетей для генерации исходного кода на языке JavaScript.
 - Ожидаемые результаты и навыки, которые студент должен приобрести.

Теоретическая часть

- **Описание изученных понятий и технологий:**
 - Введение в нейронные сети для генерации кода.
 - Принципы работы GPT-архитектуры и языковых моделей.

Практическая часть

- **Пошаговое описание выполненных действий для выполнения заданий:**
- **Примеры кода и результаты тестирования:**
 - Включите скриншоты среды разработки или текст кода.

Результаты

- **Полученные результаты и их анализ:**
 - Описание интегрированных функций, их функциональности и корректности работы.
 - Оценка качества сгенерированного кода и его соответствие требованиям.
 - Скриншоты результатов работы.

Заключение

- **Выводы о проделанной работе:**
 - Оценка достигнутых целей лабораторной работы.
 - Возможности дальнейшего применения и развития навыков.

Приложения

- **Скриншоты, код, дополнительные материалы:**
 - Скриншоты установленных библиотек и выполнения скриптов.
 - Примеры сгенерированного кода и результаты его тестирования.
 - Ссылки на использованные ресурсы и модели.

Контрольные вопросы:

1. Что такое нейронные сети и как они используются для генерации исходного кода?
2. Опишите основные этапы процесса генерации кода с помощью нейронной сети.
3. Какие критерии следует учитывать при выборе модели нейронной сети для генерации кода?
4. Опишите процесс интеграции сгенерированного кода в существующий проект на JavaScript.
5. Какие шаги необходимо предпринять для проверки корректности сгенерированного кода?
6. Как можно улучшить качество и функциональность сгенерированного кода?
7. Чем промпт для генерации текста отличается от промпта для генерации кода?
8. Какие типичные ошибки могут допускаться нейронной сетью при генерации кода, и как их можно выявить?
9. Какие подходы можно использовать для повышения точности результатов нейронной сети?

Лабораторная работа №5

«Основы работы с системой контроля версий Git»

Цель выполнения работы:

- Познакомиться с основами системы контроля версий Git.
- Освоить основные команды Git для управления версиями кода.
- Научиться создавать и работать с репозиториями, ветками и слияниями.
- Развить навыки совместной работы над проектами с использованием Git.
- Понять принципы работы с удалёнными репозиториями на локальном уровне.

Введение в системы контроля версий

Система контроля версий (Version Control System, VCS) — это программный инструмент, предназначенный для отслеживания изменений в файлах и управления этими изменениями. Она позволяет разработчикам работать над проектами совместно, отслеживать историю изменений, возвращаться к предыдущим версиям и управлять различными версиями кода.

Преимущества использования VCS:

- **Отслеживание изменений:** Позволяет видеть, какие изменения были внесены, кем и когда.
- **Совместная работа:** Облегчает сотрудничество между разработчиками.
- **Восстановление:** Возможность отката к предыдущим версиям в случае ошибок.
- **Ветвление и слияние:** Позволяет работать над новыми функциями независимо от основной линии кода.

Основы системы Git:

Git — это распределённая система контроля версий, разработанная Линусом Торвальдсом в 2005 году для управления разработкой ядра Linux. В отличие от централизованных VCS, Git позволяет каждому разработчику иметь полную копию репозитория на своём компьютере, что обеспечивает высокую скорость работы и надёжность.

Ключевые особенности Git:

- **Распределённость:** Каждый разработчик имеет локальную копию всего репозитория.
- **Эффективность:** Быстрое выполнение операций благодаря оптимизированной структуре хранения данных.
- **Поддержка ветвления:** Простое создание и переключение между ветками.
- **Целостность данных:** Использование SHA-1 хешей для обеспечения целостности и неизменности данных.

Основные понятия Git:

- **Репозиторий (Repository):** Хранилище проекта, включающее все файлы и историю изменений.
 - **Локальный репозиторий:** Репозиторий, находящийся на локальном компьютере.
 - **Удалённый репозиторий:** Репозиторий, размещённый на сервере.
- **Коммит (Commit):** Снимок состояния файлов в репозитории на определённый момент времени. Каждый коммит имеет уникальный идентификатор (SHA-1 хеш) и сообщение, описывающее изменения.
- **Ветка (Branch):** Отдельная линия разработки в репозитории. Основная ветка обычно называется `main` или `master`.
- **Слияние (Merge):** Процесс объединения изменений из одной ветки в другую.
- **Клон (Clone):** Создание копии удалённого репозитория на локальном компьютере.
- **Отслеживаемые и неотслеживаемые файлы:**
 - **Отслеживаемые файлы:** Файлы, добавленные в индекс Git и подверженные контролю версий.
 - **Неотслеживаемые файлы:** Файлы, которые ещё не добавлены в индекс.

Основные команды Git:

- `git init` — Инициализация нового локального репозитория.
- `git clone <ссылка>` — Клонирование удалённого репозитория.
- `git status` — Проверка состояния репозитория (изменения, готовность к коммиту).
- `git add <файл>` — Добавление файла в индекс (подготовка к коммиту).

- `git add .` — Добавление всех файлов в индекс.
- `git commit -m "сообщение"` — Создание коммита с описанием изменений.
- `git push` — Отправка локальных коммитов в удалённый репозиторий.
- `git pull` — Получение и интеграция изменений из удалённого репозитория.
- `git branch` — Просмотр, создание и удаление веток.
- `git checkout <ветка>` — Переключение на указанную ветку.
- `git merge <ветка>` — Слияние указанной ветки с текущей.
- `git log` — Просмотр истории коммитов.
- `git diff` — Просмотр различий между версиями файлов.
- `git remote` — Управление удалёнными репозиториям

Ветвление и слияние

Ветвление позволяет создавать независимые линии разработки, что полезно для работы над новыми функциями или исправлениями без влияния на основную ветку кода.

Основные команды для работы с ветками:

- `git branch <название ветки>` — Создание новой ветки.
- `git checkout <ветка>` — Переключение на ветку.
- `git merge <ветка>` — Слияние ветки с текущей.
- `git branch -d <ветка>` — Удаление ветки.

Конфликты при слиянии возникают, когда одни и те же строки в одном и том же файле были изменены в двух ветках. В таком случае Git попросит пользователя вручную разрешить конфликт.

Файл `.gitignore` используется для указания файлов и директорий, которые Git должен игнорировать. Это полезно для исключения временных файлов, зависимостей и других ненужных для контроля версий элементов.

Пример содержимого `.gitignore`:

```
node_modules/
*.log
.env
```

Порядок выполнения работы (шаги)

Шаг 1: Установка Git

1. Скачать Git:

- Перейдите на официальный сайт git-scm.com/downloads.
- Выберите установочный файл для вашей операционной системы (Windows, macOS, Linux).

2. Установить Git:

- Запустите скачанный установочный файл.
- Следуйте инструкциям установщика, принимая лицензионное соглашение и выбирая стандартные параметры установки.

3. Проверить установку:

- Откройте терминал или командную строку.
- Выполните команду `git --version` для проверки установленной версии Git.

Шаг 2: Настройка Git

1. Настроить имя пользователя и email:

- В терминале выполните команды:

```
git config --global user.name "Ваше Имя"  
git config --global user.email "ваш.email@example.com"
```

- Эти настройки будут использоваться для всех ваших коммитов.

2. Проверить настройки:

- В терминале выполните команду:

```
git config --list
```

- Убедитесь, что имя пользователя и email настроены правильно.

Шаг 3: Создание локального репозитория

1. Создать директорию проекта:

- В терминале выполните команды:

```
mkdir my-git-project  
cd my-git-project
```

2. Инициализировать Git репозиторий:

- В терминале выполните команду: `git init`
- В текущей директории будет создана скрытая папка `.git`, которая содержит все необходимые файлы для управления версиями.

3. Добавить файл проекта:

- Создайте файл `index.js` с простым содержимым: `console.log("Hello, Git!");`

4. Добавить файл в индекс Git:

- Выполните команду: `git add index.js`

5. Создать первый коммит:

- Выполните команду:
`git commit -m "Initial commit: Add index.js "`

Шаг 4: Ветвление и слияние

1. Создать новую ветку для разработки:

- Выполните команду: `git branch feature-branch`

2. Переключиться на новую ветку:

- Выполните команду: `git checkout feature-branch`

3. Внести изменения в проект:

- Откройте файл `index.js` и измените сообщение:
`console.log("Hello, Git Branching!");`

4. Добавить и закоммитить изменения:

- Выполните команды
`git add index.js`
`git commit -m "Update index.js "`

5. Переключиться обратно на основную ветку:

- Выполните команду: `git checkout main`

6. Слить изменения из ветки `feature-branch` в `main`:

- Выполните команду: `git merge feature-branch`

7. Удалить ветку `feature-branch`:

- Выполните команду: `git branch -d feature-branch`

Шаг 5: Разрешение конфликтов при слиянии

1. Создать конфликтную ситуацию:

- Переключитесь на новую ветку: `git checkout -b conflict-branch`
- Внесите изменение в файл `index.js`: `console.log("Hello from conflict branch!");`
- Закоммитите изменения:

```
git add index.js
```

```
git commit -m "Update index.js from conflict branch"
```

- Переключитесь на основную ветку: `git checkout main`
- Внесите другое изменение в файл `index.js`:
`console.log("Hello from main branch!");`

- Закоммитите изменения

```
git add index.js
```

```
git commit -m "Update index.js from main branch"
```

2. Попытаться слить ветку `conflict-branch` в `main`:

- Выполните команду: `git merge conflict-branch`
- Git обнаружит конфликт и попросит его разрешить.

3. Разрешить конфликт:

- Откройте файл `index.js` и вручную разрешите конфликт, выбрав нужные изменения. Пример разрешённого файла:

```
console.log("Hello from main branch!");
```

```
console.log("Hello from conflict branch!");
```

- После разрешения конфликта добавьте файл в индекс и завершите слияние:

```
git add index.js
```

```
git commit -m "Merge conflict-branch into main, resolve conflict in index.js"
```

Шаг 6: Просмотр истории коммитов

1. Просмотреть историю коммитов:

- Выполните команду: `git log`
- Используйте флаг `--oneline` для компактного отображения: `git log --oneline`

2. Просмотреть различия между коммитами:

- Выполните команду: `git diff <коммит1> <коммит2>`

Шаг 7: Использование `.gitignore`

1. Создать файл `.gitignore`:

- В корневой директории проекта создайте файл `.gitignore`

2. Добавить правила игнорирования:

- В `.gitignore` директорию `node_modules` и другие ненужные файлы:

```
node_modules/
```

```
*.log
```

```
.env
```

3. Проверить, что файлы игнорируются:

- Выполните команду `git status` и убедитесь, что игнорируемые файлы не отображаются как изменённые или новые.

Задания к лабораторной работе:

1. Создать и работа с ветками:

- Создайте новую ветку `feature-new-feature` в вашем проекте.
- Внесите изменения в файл `index.js`, добавив новую функцию.

```
function greet(name) {  
  return `Hello, ${name}!`;  
}
```

```
console.log(greet("World"));
```

- Закоммитите изменения и слейте ветку `feature-new-feature` с основной веткой `main`.

2. Использование `.gitignore`:

- Создайте файл `.gitignore` в корневой директории проекта.
- Добавьте в него директорию `node_modules` и другие ненужные файлы
- Проверьте, что эти файлы больше не отслеживаются `git`

3. Разрешение конфликтов:

- Создайте две ветки, внесите противоречивые изменения в один и тот же файл.
- Попытайтесь слить их и разрешите конфликт, выбрав нужные изменения.

Содержание отчета

Введение

- **Цель и задачи лабораторной работы:**
 - Описание целей изучения основ Git, работы с ветками, слиянием и управлением изменениями в проекте.

Теоретическая часть

- **Описание изученных понятий и технологий:**
 - Введение в системы контроля версий и особенности Git.
 - Основные понятия Git: репозиторий, коммит, ветка, слияние.
 - Принципы работы с ветками и разрешения конфликтов.

Практическая часть

- **Пошаговое описание выполненных действий:**
 - Установка и настройка Git.
 - Создание локального репозитория и начальный коммит.
 - Работа с ветками: создание, переключение, слияние.
 - Разрешение конфликтов при слиянии веток.
 - Использование `.gitignore` для игнорирования ненужных файлов.
- **Примеры команд и их вывод:**
 - Включите скриншоты терминала с выполненными командами и их результатами.
 - Примеры вывода команды `git log`, создание веток и разрешение конфликтов.

Результаты

- **Полученные результаты и их анализ:**
 - Описание созданных веток, выполненных слияний и разрешённых конфликтов.

Заключение

- **Выводы о проделанной работе:**
 - Оценка достигнутых целей лабораторной работы.

Приложения (по необходимости)

- **Скриншоты, код, дополнительные материалы:**
 - Скриншоты выполненных команд и их выводов.
 - Примеры изменённых файлов и разрешённых конфликтов.

Контрольные вопросы

1. Что такое система контроля версий и какие преимущества она предоставляет разработчикам?
2. Опишите основные различия между локальным и удалённым репозиториями в Git.
3. Как создать новую ветку в Git и переключиться на неё?
4. Что происходит при выполнении команды `git merge` и как разрешить конфликт при слиянии веток?
5. Для чего используется файл `.gitignore` и как его правильно настроить?
6. Как создать и использовать теги в Git для обозначения версий проекта?
7. Объясните процесс клонирования репозитория и синхронизации изменений между локальным и удалённым репозиториями.
8. Что такое коммит в Git и какие элементы он включает?
9. Как проверить историю коммитов в Git и какие команды для этого используются?
10. Почему важно регулярно делать коммиты и как это влияет на совместную работу над проектом?

Лабораторная работа №6

«Работа с удалёнными репозиториями и платформами хостинга кода»

1. Цель выполнения работы

- Освоить концепции и принципы работы с удалёнными репозиториями.
- Изучить использование платформ хостинга кода, таких как GitHub, для совместной разработки проектов.
- Научиться эффективно синхронизировать локальные и удалённые репозитории.
- Развить навыки управления доступом и сотрудничества с другими разработчиками через удалённые репозитории.
- Понять процессы создания пулл-реквестов и управления ветками на удалённых платформах.

Удалённый репозиторий — это версия вашего проекта, которая хранится на сервере в интернете или внутри корпоративной сети. Он позволяет разработчикам совместно работать над проектом, обмениваться изменениями и управлять различными версиями кода.

Преимущества использования удалённых репозиторий:

- **Совместная работа:** Позволяет нескольким разработчикам одновременно работать над одним проектом.
- **Резервное копирование:** Обеспечивает сохранность кода в случае сбоя локальной машины.
- **Доступность:** Позволяет получать доступ к проекту из любой точки мира.
- **Управление версиями:** Облегчает отслеживание изменений и возврат к предыдущим версиям при необходимости.

Платформы хостинга кода предоставляют сервисы для хранения удалённых репозиторий и предлагают дополнительные инструменты для совместной работы.

Наиболее популярные из них:

- **GitHub:** Одна из самых популярных платформ для хостинга Git-репозиторий, предоставляющая инструменты для управления проектами, отслеживания ошибок и обсуждения кода.

- **GitLab:** Альтернатива GitHub с дополнительными возможностями, такими как встроенный CI/CD (непрерывная интеграция и доставка).
- **Bitbucket:** Платформа от Atlassian, интегрированная с инструментами разработки, такими как Jira и Trello.

Основные функции платформ хостинга кода:

- **Пулл-реквесты (Pull Requests):** Механизм для предложения и обсуждения изменений перед их слиянием в основную ветку.
- **Трекеры задач:** Инструменты для отслеживания багов и планирования новых функций.
- **Веб-интерфейсы:** Удобные интерфейсы для просмотра кода, истории коммитов и управления проектом.
- **Интеграции:** Возможность интеграции с различными инструментами разработки и автоматизации.

Основные концепции Git при работе с удалёнными репозиториями:

- **Remote:** Удалённый репозиторий, с которым связан ваш локальный репозиторий. По умолчанию основной удалённый репозиторий называется `origin`.
- **Fetch:** Получение обновлений из удалённого репозитория без их автоматического слияния.
- **Pull:** Получение и автоматическое слияние обновлений из удалённого репозитория.
- **Push:** Отправка локальных коммитов в удалённый репозиторий.
- **Fork:** Копирование чужого репозитория на свой аккаунт для внесения изменений без влияния на оригинальный проект.
- **Clone:** Создание полной копии удалённого репозитория на локальном компьютере.

Основные команды Git для работы с удалёнными репозиториями:

- `git remote add <имя> <ссылка>` — Добавление нового удалённого репозитория.
- `git remote -v` — Просмотр списка подключённых удалённых репозиторияев.
- `git fetch <имя>` — Получение обновлений из указанного удалённого репозитория.

- `git pull <имя> <ветка>` — Получение и слияние обновлений из указанной ветки удалённого репозитория.
- `git push <имя> <ветка>` — Отправка изменений в указанную ветку удалённого репозитория.
- `git push -u <имя> <ветка>` — Отправка изменений и установка отслеживания для указанной ветки.
- `git clone <ссылка>` — Клонирование удалённого репозитория на локальный компьютер.

Работа с пулл-реквестами:

Пулл-реквест — это предложение внести изменения из одной ветки в другую, обычно из ветки с новыми функциями в основную ветку (`main` или `master`). Пулл-реквесты позволяют другим разработчикам просмотреть, обсудить и протестировать изменения перед их слиянием.

Этапы создания пулл-реквеста:

1. **Создание ветки:** Создайте новую ветку для вашей функции или исправления.
2. **Внесение изменений и коммит:** Внесите необходимые изменения и зафиксируйте их.
3. **Отправка ветки в удалённый репозиторий:** Используйте `git push` для отправки ветки.
4. **Создание пулл-реквеста:** Перейдите на платформу хостинга кода и создайте пулл-реквест, указав исходную и целевую ветки.
5. **Обзор и обсуждение:** Другие разработчики могут просмотреть ваш пулл-реквест, оставить комментарии и предложить изменения.
6. **Слияние:** После одобрения пулл-реквеста внесённые изменения могут быть слиты в целевую ветку.

Управление доступом и ролями

На платформах хостинга кода можно управлять доступом к репозиториям, назначая различные роли участникам проекта:

- **Владелец (Owner):** Полный контроль над репозиторием, включая управление доступом.

- **Администратор (Administrator):** Возможность управлять настройками репозитория и доступом пользователей.
- **Коллаборатор (Collaborator):** Возможность вносить изменения в репозиторий.
- **Читатель (Reader):** Только чтение содержимого репозитория без возможности внесения изменений.

Порядок выполнения работы

Шаг 1: Создание удалённого репозитория

1. **Выбор платформы хостинга кода:**
 - Для данной лабораторной работы будет использоваться **GitHub**. Перейдите на сайт github.com и войдите в свой аккаунт. Если у вас его нет, то зарегистрируйтесь.
2. **Создание нового репозитория:**
 - Нажмите кнопку "**New**" на странице репозитория.
 - Введите название репозитория, например, `my-git-project`.
 - Добавьте описание проекта (опционально).
 - Выберите видимость репозитория: **Public** (публичный) или **Private** (приватный).
 - Нажмите кнопку "**Create repository**".

Шаг 2: Клонирование удалённого репозитория на локальный компьютер

1. **Получение URL репозитория:**
 - На странице вашего нового репозитория нажмите кнопку "**Code**" и скопируйте HTTPS-ссылку, например: `https://github.com/ваш_логин/my-git-project.git`
 - Перейдите в директорию проекта: `cd my-git-project`

Шаг 3: Внесение изменений и отправка их в удалённый репозиторий

1. **Создание файла `index.js`:**
 - Создайте файл `index.js` с содержимым: `console.log("Hello, GitHub!")`
2. **Добавление файла в индекс и коммит:**
 - Выполните команды:


```
git add index.js
git commit -m "Add index.js with Hello, GitHub! message"
```


3. Отправка изменений в удалённый репозиторий:

- Выполните команду: `git push origin main`
- Введите свои учетные данные GitHub при запросе (если используется HTTPS).

4. Проверка изменений на GitHub:

- Перейдите на страницу вашего репозитория на GitHub и убедитесь, что файл `index.js` добавлен.

Шаг 4: Создание и работа с ветками

1. Создание новой ветки `feature-add-function`:

- Выполните команду: `git branch feature-add-function`

2. Переключение на новую ветку:

- Выполните команду: `git checkout feature-add-function`

3. Внесение изменений в `index.js`:

- Откройте файл `index.js` и добавьте функцию:

```
function greet(name) {  
  return `Hello, ${name}!`;  
}  
console.log(greet("World"));
```

4. Добавление изменений и коммит:

- Выполните команды:

```
git add index.js
```

```
git commit -m "Add greet function to index.js"
```

5. Отправка ветки в удалённый репозиторий:

- Выполните команду: `git push origin feature-add-function`

6. Создание пулл-реквеста:

- Перейдите на страницу репозитория на GitHub.
- Вы увидите уведомление о недавно отправленной ветке с предложением создать пулл-реквест. Нажмите **"Compare & pull request"**.
- Добавьте описание изменений и нажмите **"Create pull request"**.

7. Обзор и слияние пулл-реквеста:

- После проверки изменений нажмите кнопку **"Merge pull request"** и затем **"Confirm merge"**.
- Удалите ветку `feature-add-function`, нажав кнопку **"Delete branch"** на странице пулл-реквеста.

Шаг 5: Разрешение конфликтов при слиянии

1. Создание конфликтной ситуации:

- Создайте новую ветку `feature-conflict`: `git checkout -b feature-conflict`
- Внесите изменение в `index.js`: `console.log("Hello from conflict branch!");`
- Добавьте и закоммитьте изменения:
`git add index.js`
`git commit -m "Update index.js from conflict branch"`
- Переключитесь на основную ветку: `git checkout main`
- Внесите другое изменение в `index.js`: `console.log("Hello from main branch!");`
- Добавьте и закоммитьте изменения:
`git add index.js`
`git commit -m "Update index.js from main branch"`

2. Попытка слияния ветки `feature-conflict` в `main`:

- Выполните команду: `git merge feature-conflict`
- Git обнаружит конфликт и сообщит о нём.

3. Разрешение конфликта:

- Откройте файл `index.js` в редакторе.
- Вы увидите конфликтные изменения, обозначенные специальными маркерами:

```
<<<<<<< HEAD
console.log("Hello from main branch!");
=====
console.log("Hello from conflict branch!");
>>>>>>> feature-conflict
```

- Разрешите конфликт, выбрав нужные строки или объединив их:
`console.log("Hello from main branch!");`
`console.log("Hello from conflict branch!");`
- Сохраните файл

4. Добавление разрешённого файла в индекс и завершение слияния:

- Выполните команды:
`git add index.js`
`git commit -m "Merge feature-conflict into main, resolve conflict in index.js"`

5. Отправка изменений в удалённый репозиторий:

- Выполните команду: `git push origin main`

Шаг 6: Просмотр истории коммитов

3. Просмотреть историю коммитов:

- Выполните команду: `git log`
- Используйте флаг `--oneline` для компактного отображения: `git log --oneline`

4. Просмотреть различия между коммитами:

- Определите SHA-1 хеши двух коммитов, которые хотите сравнить
- Выполните команду: `git diff <коммит1> <коммит2>`

Задания к лабораторной работе:

4. Создание и работа с пулл-реквестами:

- Создайте новую ветку `feature-add-utility`.
- Внесите изменения в файл `index.js`, добавив новую утилитную функцию:

```
function multiply(a, b) {  
  return a * b;  
}
```

```
console.log(multiply(3, 4));
```

- Закоммитите изменения и отправьте ветку в удалённый репозиторий.
- Создайте пулл-реквест для слияния ветки `feature-add-utility` в `main`.
- После одобрения слияния удалите ветку.

5. Настройка `.gitignore` для различных окружений:

- Добавьте в `.gitignore` файлы и директории, специфичные для различных сред разработки (например, конфигурационные файлы IDE, временные файлы).
- Убедитесь, что эти файлы не отслеживаются Git.

6. Восстановление удалённых коммитов:

- Создайте коммит, затем удалите его с помощью `git reset`.
- Восстановите удалённый коммит с помощью команды `git reflog` и `git reset`.

Содержание отчёта:

Введение

- **Цель и задачи лабораторной работы:**

- Описание целей изучения работы с удалёнными репозиториями и платформами хостинга кода.

Теоретическая часть

- **Описание изученных понятий и технологий:**
 - Введение в удалённые репозитории и их преимущества.
 - Обзор платформ хостинга кода, таких как GitHub.
 - Основные концепции и команды Git для работы с удалёнными репозиториями.
 - Принципы создания и управления пулл-реквестами.
 - Управление доступом и ролями на платформах хостинга кода.

Практическая часть

- **Пошаговое описание выполненных действий:**
 - Создание удалённого репозитория на GitHub.
 - Клонирование репозитория на локальный компьютер.
 - Внесение изменений, коммит и отправка изменений в удалённый репозиторий.
 - Создание и слияние веток через пулл-реквесты.
 - Разрешение конфликтов при слиянии веток.

Примеры команд и их вывод:

- Включите в отчет скриншоты терминала с выполненными командами и их результатами.
- Примеры содержимого файлов после внесённых изменений.
- Снимки экрана с пулл-реквестами и их слиянием на GitHub.

Результаты

- **Полученные результаты и их анализ:**
 - Описание созданных и слитых веток, выполненных пулл-реквестов.
 - Демонстрация работы с тегами и их использования для релизов.
 - Анализ процесса разрешения конфликтов и его влияния на проект.
- Прикрепите ссылку на итоговый гит-репозиторий

Заключение

- **Выводы о проделанной работе:**
 - Оценка достигнутых целей лабораторной работы.
 - Возможности дальнейшего применения и развития навыков работы с Git и платформами хостинга кода.

Приложения (по необходимости)

- **Скриншоты, код, дополнительные материалы:**
 - Скриншоты выполненных команд и их выводов.
 - Примеры изменённых файлов и разрешённых конфликтов.
 - Ссылки на созданные пулл-реквесты и теги на GitHub.
 - Дополнительные материалы, такие как графики истории коммитов или диаграммы ветвления.

Контрольные вопросы:

1. Что такое удалённый репозиторий и каковы его основные преимущества?
2. Опишите процесс создания и клонирования удалённого репозитория на GitHub.
3. Что такое пулл-реквест и для чего он используется?
4. Как разрешить конфликт при слиянии веток в Git?
5. Для чего используется файл `.gitignore` и как его правильно настроить?
6. Что такое тег в Git и как его использовать для обозначения версий проекта?
7. Как добавить новый удалённый репозиторий в существующий локальный репозиторий?
8. Объясните разницу между командами `git fetch` и `git pull`.
9. Какие роли можно назначить участникам проекта на GitHub и как они влияют на доступ к репозиторию?
10. Почему важно регулярно отправлять изменения в удалённый репозиторий и как это влияет на совместную работу над проектом?

Лабораторная работа №7

«Планирование проекта с использованием системы управления проектами»

Цель выполнения работы:

- Освоить основные принципы и методы управления проектами.
- Изучить функциональные возможности современных систем управления проектами, таких как Jira, Trello и Asana.
- Научиться планировать проект, определять задачи, устанавливать сроки и распределять ресурсы с использованием выбранной системы.
- Развить навыки мониторинга прогресса проекта и управления изменениями.

Система управления проектами (Project Management System, PMS) — это программный инструмент, предназначенный для планирования, организации и управления ресурсами для достижения конкретных целей проекта. Такие системы помогают командам эффективно сотрудничать, отслеживать прогресс и управлять временем и бюджетом.

Преимущества использования PMS:

- **Централизованное управление:** Все аспекты проекта находятся в одном месте.
- **Улучшенная коммуникация:** Облегчает обмен информацией между участниками проекта.
- **Прозрачность:** Позволяет видеть статус задач и общий прогресс проекта.
- **Эффективное распределение ресурсов:** Помогает оптимально использовать доступные ресурсы.
- **Управление рисками:** Позволяет выявлять и минимизировать потенциальные риски

Проектное управление включает в себя применение знаний, навыков, инструментов и методов для выполнения проектных требований. Основные процессы проектного управления включают:

1. **Инициация:** Определение целей проекта, его обоснование и назначение менеджера проекта.

2. **Планирование:** Разработка детального плана проекта, включая задачи, сроки, ресурсы и бюджет.
3. **Исполнение:** Реализация плана проекта, распределение задач и управление командой.
4. **Мониторинг и контроль:** Отслеживание прогресса, управление изменениями и обеспечение качества.
5. **Завершение:** Финализация всех активностей, оценка результатов и закрытие проекта.

Существует множество систем управления проектами, каждая из которых обладает уникальными функциями и подходит для различных типов проектов. Рассмотрим три популярных инструмента:

- **Jira:**
 - Разработана компанией Atlassian.
 - Изначально предназначена для управления разработкой программного обеспечения.
 - Поддерживает методологии Agile, включая Scrum и Kanban.
 - Включает инструменты для отслеживания задач, багов и прогресса спринтов.
- **Trello:**
 - Также разработана Atlassian.
 - Использует визуальную систему досок и карточек.
 - Подходит для небольших команд и простых проектов.
 - Легко настраивается под различные нужды через плагины и интеграции.
- **Asana:**
 - Разработана компанией Asana, Inc.
 - Предоставляет расширенные возможности для планирования и отслеживания задач.
 - Поддерживает различные виды представления задач: списки, доски, календари.
 - Включает инструменты для управления зависимостями и отчетности.

Основные функциональные возможности Jira:

Jira — мощная система управления проектами, ориентированная на команды разработки программного обеспечения. Основные функции включают:

- **Доски Scrum и Kanban:** Позволяют организовать рабочий процесс в соответствии с выбранной методологией.
- **Управление задачами и эпиками:** Создание иерархии задач для эффективного планирования.
- **Отслеживание багов:** Инструменты для регистрации и управления дефектами.
- **Спринты:** Планирование и управление итерациями работы.
- **Отчеты и аналитика:** Встроенные инструменты для анализа производительности и прогресса.
- **Интеграции:** Возможность интеграции с другими инструментами Atlassian и сторонними сервисами.

Создание и настройка проекта в Jira

Процесс создания проекта в Jira включает:

1. Выбор типа проекта:

- **Scrum:** Подходит для проектов, использующих спринты и методологию Scrum, где задачи разбиваются на итерации (спринты).
- **Kanban:** Подходит для проектов с непрерывным потоком работы без фиксированных итераций, где задачи обрабатываются в порядке приоритета.
- **Bug Tracking:** Специализированный тип проекта, ориентированный на управление багами, их приоритизацию и устранение.

2. Настройка доски:

- Определение колонок и статусов задач (например, "To Do", "In Progress", "Done").
- Настройка ограничений на количество задач в колонках (WIP лимиты) для улучшения управления потоком работы.

3. Создание задач:

- **Эпики:** Крупные компоненты или ключевые направления работы, которые делятся на более мелкие задачи (истории и задачи). Эпики обычно охватывают длительный период и объединяют схожие задачи для достижения общей цели.

- **Задачи:** Конкретные шаги или работы, необходимые для выполнения историй пользователей или других аспектов проекта.
- Установка приоритетов (например, "Высокий", "Средний", "Низкий") и сроков выполнения для каждой задачи.

Управление задачами, спринтами и бэклогом

- **Бэклог (Backlog):** Список всех задач и требований проекта. Задачи из бэклога выбираются для включения в спринты.
- **Спринт (Sprint):** Фиксированный период времени (обычно 2-4 недели), в течение которого команда выполняет выбранные задачи из бэклога.
- **Управление задачами:** Назначение ответственных, установка приоритетов, отслеживание прогресса и статусов.

Отчеты и аналитика в системах управления проектами

Системы управления проектами предоставляют разнообразные отчеты для анализа производительности и эффективности команды:

- **Burn-down Chart:** Отображает оставшееся количество работы в спринте.
- **Burn-up Chart:** Показывает прогресс выполнения задач.
- **Velocity Chart:** Отображает количество выполненных задач за спринт.
- **Cumulative Flow Diagram:** Показывает распределение задач по статусам во времени.
- **Отчеты по времени выполнения:** Анализ времени, затраченного на выполнение задач.

Порядок выполнения работы:

Шаг 1: Регистрация и вход в систему управления проектами

1. **Выбор платформы:**
 - Для данной лабораторной работы будет использоваться **Jira**. Перейдите на сайт jira.atlassian.com и зарегистрируйтесь или войдите в существующую учетную запись.
2. **Выбор тарифного плана:**
 - Выберите бесплатный (Free) тарифный план

Шаг 2: Создание нового проекта в Jira

1. Создание проекта:

- Нажмите кнопку **"Create project"**.
- Выберите тип проекта (например, **Scrum**).
- Введите название проекта, например, `My Project`.
- Нажмите **"Create"**.

2. Настройка проекта:

- Определите ключ проекта (например, `MPM`).
- Настройте доску Scrum: добавьте колонки, установите WIP лимиты при необходимости.

Шаг 3: Планирование проекта

1. Создание эпиков:

- В разделе **"Backlog"** нажмите **"Create Epic"**.
- Введите название эпика, например, `User Authentication Module`.

2. Создание задач и историй пользователей:

- В разделе **"Backlog"** нажмите **"Create Issue"**.
- Выберите тип задачи (Story, Task, Bug).
- Введите описание задачи, например, `Implement login functionality`.
- Назначьте приоритет и эпик, к которому относится задача.

3. Организация бэклога:

- Распределите задачи по приоритетам.
- Оцените задачи с помощью story points или часов.

Шаг 4: Создание и управление спринтами

1. Создание спринта:

- В разделе **"Backlog"** нажмите **"Create Sprint"**.
- Назовите спринт, например, `Sprint 1`.

2. Планирование спринта:

- Перетащите задачи из бэклога в спринт.
- Установите цели спринта.

3. Начало спринта:

- Нажмите "**Start Sprint**".
- Установите продолжительность спринта и дату окончания.

Шаг 5: Выполнение задач и мониторинг прогресса

1. Работа над задачами:

- Перемещайте задачи по доске от To Do к In Progress и далее к Done.
- Обновляйте статусы задач по мере выполнения.

2. Мониторинг прогресса:

- Используйте **Burn-down Chart** и **Velocity Chart** для отслеживания прогресса спринта.
- Анализируйте производительность команды и корректируйте план при необходимости.

Шаг 6: Завершение спринта и ретроспектива

1. Завершение спринта:

- По окончании спринта нажмите "**Complete Sprint**".
- Просмотрите выполненные и невыполненные задачи.

2. Ретроспектива:

- Проанализируйте, что было сделано хорошо, а что можно улучшить.
- Запланируйте изменения для следующего спринта.

Шаг 7: Создание отчетов и аналитика

1. Просмотр отчетов:

- Перейдите в раздел "**Reports**".
- Выберите нужный отчет, например, **Burn-down Chart**.

2. Анализ отчетов:

- Изучите показатели производительности.
- Используйте полученные данные для улучшения процессов управления проектом.

Задания к лабораторной работе:

7. Планирование и управление спринтом:

- Создайте новый проект в Jira.
- Определите минимум два эпика для проекта.
- Создайте не менее пяти задач и распределите их по эпикам.
- Организуйте первый спринт, включив в него выбранные задачи.
- Выполните задачи спринта, перемещая их по доске и обновляя статусы.
- Завершите спринт и проанализируйте результаты.

Содержание отчета:

Введение

- **Цель и задачи лабораторной работы:**
 - Описание целей изучения систем управления проектами, таких как Jira.
 - Ожидаемые результаты и навыки, которые студент должен приобрести.

Теоретическая часть

- **Описание изученных понятий и технологий:**
 - Введение в системы управления проектами и их важность.
 - Основные принципы проектного управления.
 - Обзор и сравнение различных инструментов управления проектами.
 - Подробное описание функциональных возможностей выбранной системы (например, Jira).
 - Принципы планирования спринтов, управления задачами и мониторинга прогресса.

Практическая часть

- **Пошаговое описание выполненных действий:**
 - Регистрация и настройка аккаунта в системе управления проектами.
 - Создание нового проекта и его настройка.
 - Планирование проекта: создание эпиков, задач и историй пользователей.
 - Организация и управление спринтами.

Результаты

- **Полученные результаты и их анализ:**
 - Описание созданных и управляемых задач и спринтов.

- Анализ эффективности использования системы управления проектами.
- Оценка производительности команды и достигнутых целей проекта.

Заключение

- **Выводы о проделанной работе:**
 - Оценка достигнутых целей лабораторной работы.
 - Впечатления от использования системы управления проектами.
 - Возможности дальнейшего применения и развития навыков управления проектами.

Приложения (по необходимости)

- **Скриншоты, код, дополнительные материалы:**
 - Скриншоты выполненных шагов в системе управления проектами.
 - Примеры настроек проекта, задач и спринтов.
 - Дополнительные материалы, такие как графики прогресса или диаграммы Ганта.

Контрольные вопросы:

1. Что такое система управления проектами и какие преимущества она предоставляет командам разработчиков?
2. Опишите основные этапы проектного управления и их роль в успешной реализации проекта.
3. Какие функциональные возможности предоставляет Jira для планирования и управления проектами?
4. Что такое спринт в методологии Scrum и как его правильно планировать?
5. Как создавать и управлять эпиками, задачами и историями пользователей в Jira?
6. Что такое бэклог и как его эффективно организовать?
7. Как использовать отчеты в Jira для мониторинга прогресса проекта?
8. Какие роли и права доступа можно назначить участникам проекта на платформе Jira?
9. Почему важно регулярно проводить ретроспективы после завершения спринтов?

Лабораторная работа №8

«Распределение задач и управление сроками выполнения проекта»

Цель выполнения работы

- **Освоить** принципы и методы распределения задач в проекте.
- **Изучить** техники управления сроками выполнения проекта.
- **Научиться** использовать современные инструменты и методологии для эффективного планирования и контроля выполнения задач.
- **Развить** навыки приоритизации задач и оптимального распределения ресурсов.
- **Понять** важность мониторинга прогресса и управления изменениями в проекте.

Введение в управление проектами

Управление проектами — это процесс планирования, организации и контроля ресурсов для достижения конкретных целей, и задач в рамках заданных сроков и бюджета.

Эффективное управление проектами позволяет повысить продуктивность команды, минимизировать риски и обеспечить качественное выполнение задач.

Ключевые компоненты управления проектами:

- **Цели проекта:** Четко определенные результаты, которых необходимо достичь.
- **Задачи:** Конкретные действия, направленные на достижение целей.
- **Ресурсы:** Время, люди, финансы и материалы, необходимые для выполнения задач.
- **Сроки:** Временные рамки, в которые должны быть выполнены задачи.
- **Риски:** Потенциальные проблемы, которые могут повлиять на выполнение проекта.

Распределение задач — это процесс назначения конкретных задач отдельным членам команды или группам для их выполнения. Эффективное распределение задач обеспечивает оптимальное использование ресурсов, повышает мотивацию сотрудников и ускоряет процесс выполнения проекта.

Основные принципы распределения задач:

- **Четкость:** Задачи должны быть четко определены и понятны.
- **Соответствие компетенциям:** Задачи должны соответствовать навыкам и опыту сотрудников.
- **Баланс нагрузки:** Равномерное распределение задач для предотвращения перегрузки отдельных участников.
- **Приоритизация:** Назначение задач в порядке их важности и срочности.
- **Ответственность:** Определение ответственных лиц за выполнение каждой задачи.

Существует несколько методов распределения задач, каждый из которых имеет свои преимущества и подходит для различных типов проектов:

- **Метод Парето (80/20):** Фокус на 20% задач, которые приносят 80% результатов.
- **Метод Канбан:** Визуализация задач на доске, управление потоком работы и ограничение незавершенных задач.
- **Метод Scrum:** Разделение работы на спринты, регулярные встречи для планирования и обзора прогресса.
- **Методология Agile:** Гибкий подход к управлению задачами с акцентом на итеративное развитие и адаптацию к изменениям.

Управление сроками включает в себя планирование и контроль времени, необходимого для выполнения задач проекта. Эффективное управление сроками позволяет соблюдать график проекта, своевременно реагировать на задержки и оптимизировать процессы.

Основные этапы управления сроками:

- **Определение задач и их длительности:** Разбиение проекта на конкретные задачи и оценка времени, необходимого для их выполнения.
- **Создание графика проекта:** Разработка временной шкалы с указанием сроков выполнения задач.
- **Назначение ответственных:** Определение лиц, ответственных за выполнение каждой задачи.
- **Мониторинг прогресса:** Отслеживание выполнения задач и сравнение с планом.
- **Управление изменениями:** Корректировка графика в случае возникновения непредвиденных обстоятельств.

Инструменты и методы управления сроками

Для эффективного управления сроками и распределения задач используются различные инструменты и методы:

- **Диаграмма Ганта:** Визуальное представление графика проекта, показывающее сроки выполнения задач и их взаимосвязи.
- **Технология критического пути (Critical Path Method, CPM):** Метод определения наиболее важных задач, влияющих на сроки проекта.
- **Технология оценки и пересмотра программ (Program Evaluation and Review Technique, PERT):** Метод оценки времени выполнения задач с учетом неопределенностей.
- **Системы управления проектами:** Инструменты, такие как Jira, Microsoft Project, Trello, Asana и т.п., которые помогают планировать, отслеживать и управлять задачами и сроками.

В практической работе важно учитывать следующие аспекты:

- **Коммуникация:** Регулярное общение с командой для уточнения статусов задач и решения возникающих проблем.
- **Гибкость:** Способность адаптироваться к изменениям и корректировать план в случае необходимости.
- **Мотивация:** Создание условий для мотивации команды, признание достижений и поддержка инициатив.
- **Анализ рисков:** Выявление потенциальных рисков и разработка стратегий их минимизации.
- **Отчетность:** Регулярное предоставление отчетов о прогрессе проекта заинтересованным сторонам.

Диаграмма Ганта — это визуальный инструмент для планирования и отслеживания прогресса выполнения задач проекта. Она представляет собой горизонтальные полосы, каждая из которых соответствует отдельной задаче, с указанием начала и окончания выполнения.

Преимущества диаграмм Ганта:

- **Визуализация временных рамок:** Позволяет видеть, когда начинается и заканчивается каждая задача.
- **Отслеживание прогресса:** Легко определить, какие задачи выполнены, находятся в процессе или задерживаются.
- **Управление зависимостями:** Отображает взаимосвязи между задачами.

Программные инструменты для создания диаграмм Ганта:

- **Microsoft Excel:** Подходит для создания простых диаграмм.
- **GanttPRO:** Онлайн-инструмент с расширенными возможностями.
- **Trello с плагинами:** Визуальные доски с возможностью добавления диаграмм Ганта через плагины.
- **Другие специализированные приложения:** Например, TeamGantt или ClickUp.

Установка сроков выполнения задач в Jira позволяет планировать время, необходимое для завершения каждой задачи, и отслеживать соблюдение графика проекта.

Основные шаги для установки сроков в Jira:

1. **Создание задачи:** Ввести название и описание задачи.
2. **Назначение исполнителя:** Назначить себя или другого участника проекта ответственным за выполнение задачи.
3. **Установка сроков:** Определить дату начала и окончания выполнения задачи.
4. **Добавление меток и приоритетов:** Помочь в организации и приоритизации задач.
5. **Отслеживание прогресса:** Обновлять статус задачи по мере ее выполнения.

Порядок выполнения работы:

Шаг 1: Планирование проекта

1. **Определение цели проекта:**
 - Четко сформулируйте цель проекта. Например, разработка простого веб-приложения, создание учебного проекта или выполнение исследовательской работы.
2. **Разбиение проекта на задачи:**
 - Разделите проект на несколько основных задач и подзадач. Например:
 - Исследование требований.
 - Проектирование архитектуры.

- Разработка функционала.
- Тестирование.
- Документация.

Шаг 2: Создание диаграммы Ганта

1. Выбор инструмента:

- Выберите программное обеспечение для создания диаграммы Ганта. Для простоты можно использовать Microsoft Excel или онлайн-инструменты, такие как GanttPRO.

2. Ввод данных:

- Введите список задач с указанием дат начала и окончания выполнения.
- Установите зависимости между задачами, если они имеются.

3. Визуализация:

- Создайте диаграмму, отображающую временные рамки выполнения каждой задачи.
- Обновляйте диаграмму по мере выполнения задач.

Шаг 3: Настройка проекта в Jira

- **Регистрация и вход:**
 - Перейдите на jira.atlassian.com и зарегистрируйтесь или войдите в существующий аккаунт.
- **Создание нового проекта:**
 - Нажмите "**Create project**".
 - Выберите шаблон проекта, например, "**Basic Kanban**" или "**Scrum**".
 - Введите название проекта, например, Мой учебный проект.
 - Нажмите "**Create**".

Шаг 4: Создание и распределение задач в Jira

1. Создание задач:

- Внутри проекта нажмите "**Create issue**".
- Выберите тип задачи (например, **Task** или **Story**).
- Введите название задачи и описание.
- Назначьте задачу себе в качестве исполнителя.
- Установите срок выполнения, указав **Due Date**.

2. Приоритизация задач:

- Установите приоритет задачи (например, **High, Medium, Low**) в зависимости от ее важности.

3. Добавление меток:

- Добавьте метки (labels) для лучшей организации задач, например, исследование, разработка, тестирование.

Шаг 5: Управление сроками выполнения в Jira

1. Установка сроков:

- Для каждой задачи установите **Due Date** — дату, к которой задача должна быть выполнена.
- Это можно сделать при создании задачи или позже, редактируя ее свойства.

2. Отслеживание прогресса:

- Регулярно обновляйте статус задач (например, **To Do, In Progress, Done**) по мере их выполнения.
- Используйте доску Kanban или Scrum для визуализации прогресса.

3. Использование отчетов:

- Перейдите в раздел "**Reports**" и используйте доступные отчеты для мониторинга сроков и прогресса выполнения задач.

Задания к лабораторной работе:

8. Создание и управление диаграммой Ганта:

- Разработайте диаграмму Ганта для выбранного индивидуального проекта.
- Определите основные задачи и установите для них сроки выполнения.
- Обновляйте диаграмму по мере выполнения задач.

9. Установка сроков выполнения задач в Jira:

- Создайте задачи для вашего проекта в Jira.
- Назначьте себе исполнителей и установите сроки выполнения для каждой задачи.
- Используйте доску Kanban или Scrum для отслеживания прогресса.

Содержание отчёта:

Введение

- **Цель и задачи лабораторной работы:**
 - Описание целей изучения методов распределения задач и управления сроками.
 - Определение ожидаемых результатов и навыков, которые студент должен приобрести.

Теоретическая часть

- **Описание изученных понятий и технологий:**
 - Введение в управление проектами и его значимость.
 - Принципы распределения задач и управления сроками.
 - Обзор инструментов для создания диаграмм Ганта.
 - Введение в систему управления проектами Jira.
 - Методы установки и отслеживания сроков выполнения задач в Jira.

Практическая часть

- **Пошаговое описание выполненных действий:**
 - Планирование проекта: определение целей и задач.
 - Создание диаграммы Ганта с указанием сроков выполнения задач.
 - Настройка проекта в Jira: создание задач, назначение исполнителей и установка сроков.
 - Управление задачами и мониторинг прогресса выполнения.
 - Корректировка плана при необходимости.

Примеры интерфейсов и графиков:

- Скриншоты созданной диаграммы Ганта.
- Скриншоты задач в Jira с установленными сроками.

Результаты

- **Полученные результаты и их анализ:**
 - Описание выполненных задач и достигнутых целей проекта.
 - Анализ эффективности распределения задач и соблюдения сроков.
 - Оценка использования ресурсов и времени.

Заключение

- **Выводы о проделанной работе:**
 - Оценка достигнутых целей лабораторной работы.
 - Впечатления от использования диаграмм Ганта и системы Jira для управления проектом.

Приложения (по необходимости)

- **Скриншоты, графики, дополнительные материалы:**
 - Скриншоты диаграммы Ганта.
 - Скриншоты задач в Jira с установленными сроками.
 - Дополнительные материалы, такие как графики прогресса или диаграммы приоритизации.

Контрольные вопросы

1. Что такое управление проектами и какие ключевые компоненты включает этот процесс?
2. Опишите основные принципы эффективного распределения задач в индивидуальном проекте.
3. Что такое диаграмма Ганта и как она помогает в планировании проекта?
4. Какие программные инструменты можно использовать для создания диаграмм Ганта?
5. Что такое система управления проектами Jira и для чего она используется?
6. Как установить сроки выполнения задач в Jira?
7. Какие преимущества предоставляет использование диаграмм Ганта в управлении проектами?
8. Почему важно регулярно мониторить прогресс выполнения задач и как это влияет на успех проекта?
9. Какие шаги необходимо предпринять при возникновении задержек в выполнении задач проекта?

Лабораторная работа №9

«Выбор методологии разработки и настройка инструментов управления проектом»

Цель выполнения работы

- Освоить различные методологии разработки программного обеспечения.
- Изучить критерии выбора подходящей методологии для конкретного проекта.
- Научиться настраивать инструменты управления проектами в соответствии с выбранной методологией.
- Развить навыки планирования и организации работы над индивидуальным проектом.
- Понять важность адаптации методологии и инструментов под особенности проекта.

Введение в методологии разработки

Методология разработки — это совокупность принципов, практик и процессов, направленных на эффективное создание программного обеспечения. Выбор правильной методологии влияет на качество, сроки и стоимость проекта.

Основные методологии разработки:

- **Водопад (Waterfall):**
 - Последовательная модель разработки.
 - Каждая фаза проекта (требования, дизайн, реализация, тестирование, внедрение) должна быть завершена перед переходом к следующей.
 - Подходит для проектов с четко определенными требованиями и малой вероятностью изменений.
- **Гибкая методология (Agile):**
 - Итеративный и инкрементальный подход.
 - Разработка происходит в коротких циклах (спринтах), с постоянным взаимодействием с заказчиком.
 - Позволяет быстро адаптироваться к изменениям требований.
- **Scrum:**

- Фреймворк внутри Agile.
 - Рабочий процесс разделен на спринты (обычно 2-4 недели).
 - Включает роли: владелец продукта, Scrum-мастер и команда разработки.
 - Регулярные встречи: планирование спринта, ежедневные стендапы, обзор спринта и ретроспектива.
- **Kanban:**
 - Визуальное управление рабочим процессом.
 - Использует доски с колонками для отображения стадий выполнения задач.
 - Акцент на непрерывное улучшение и управление потоком работы.
- **Lean:**
 - Ориентирован на минимизацию потерь и оптимизацию процессов.
 - Подходит для проектов, требующих высокой эффективности и гибкости.

Критерии выбора методологии

Выбор методологии зависит от ряда факторов:

- **Характеристики проекта:**
 - Размер и сложность.
 - Требования к гибкости и изменяемости.
- **Командные ресурсы:**
 - Опыт и навыки участников.
 - Размер команды.
- **Взаимодействие с заказчиком:**
 - Необходимость постоянной обратной связи.
 - Готовность заказчика к активному участию.
- **Сроки и бюджет:**
 - Ограничения по времени.
 - Финансовые рамки проекта.
- **Риски и неопределенности:**
 - Вероятность изменений в требованиях.
 - Технические риски.

Настройка инструментов управления проектами

Настройка инструментов управления проектами включает в себя создание проекта, определение структур задач, установка сроков и распределение ресурсов. Важно адаптировать инструмент под выбранную методологию для повышения эффективности работы.

Основные шаги настройки:

1. Создание проекта:

- Выбор шаблона (Scrum, Kanban, и т.д.).
- Введение основных параметров проекта (название, описание).

2. Определение структуры задач:

- Создание эпиков и историй пользователей.
- Разбиение задач на подзадачи.

3. Установка сроков и приоритетов:

- Определение дат начала и окончания задач.
- Назначение приоритетов для задач.

4. Распределение задач:

- Назначение ответственных лиц.
- Определение зависимостей между задачами.

5. Настройка отчетности и мониторинга:

- Выбор необходимых отчетов.
- Настройка уведомлений и напоминаний.

Примеры методологий и инструментов:

• Scrum с использованием Jira:

- Создание спринтов.
- Управление задачами и эпиками.
- Использование отчетов Burn-down и Velocity.

• Kanban с использованием Trello:

- Организация доски с колонками: To Do, In Progress, Done.
- Управление ограничениями на количество задач в колонках.
- Визуализация потока работы.

Порядок выполнения работы

Шаг 1: Выбор методологии разработки

1. Анализ проекта:

- Определите цели и задачи вашего индивидуального проекта.
- Оцените объем работ и предполагаемые сроки выполнения.
- Определите, насколько вероятны изменения в требованиях.

2. Выбор подходящей методологии:

- На основе анализа выберите методологию, которая лучше всего подходит для вашего проекта (например, Scrum для итеративной разработки или Kanban для непрерывного потока задач).

3. Обоснование выбора:

- Запишите причины выбора конкретной методологии, учитывая характеристики вашего проекта.

Шаг 2: Настройка инструмента управления проектом

1. Выбор инструмента:

- Решите, какой инструмент управления проектом вы будете использовать (Jira, Trello, Asana).

2. Регистрация и создание проекта:

- Зарегистрируйтесь в выбранном инструменте (если еще не зарегистрированы).
- Создайте новый проект, выбрав соответствующий шаблон методологии (Scrum, Kanban и т.д.).

3. Настройка доски:

- Определите колонки, соответствующие стадиям выполнения задач (например, To Do, In Progress, Done).
- Установите приоритеты и ограничения на количество задач в колонках (для Kanban).

Шаг 3: Планирование и распределение задач

1. Создание задач:

- Разбейте проект на основные задачи и подзадачи.
- Создайте карточки задач в выбранном инструменте, указав название и описание.

2. Установка сроков:

- Для каждой задачи установите дату начала и окончания.

- Убедитесь, что сроки реалистичны и соответствуют общему графику проекта.

3. Приоритизация задач:

- Определите приоритеты для задач (высокий, средний, низкий).
- Распределите задачи по приоритетам на доске.

Шаг 4: Мониторинг и управление проектом

1. Отслеживание прогресса:

- Регулярно обновляйте статусы задач по мере их выполнения.
- Используйте визуальные инструменты (диаграммы, доски) для контроля прогресса.

2. Корректировка плана:

- При возникновении задержек или изменений в требованиях корректируйте график и распределение задач.

3. Использование отчетов:

- Просматривайте доступные отчеты для анализа прогресса и эффективности выполнения задач.
- На основе отчетов принимайте решения о дальнейших шагах проекта.

Шаг 5: Завершение проекта

1. Финальная проверка:

- Убедитесь, что все задачи выполнены и цели проекта достигнуты.
- Проведите итоговую оценку качества выполненной работы.

2. Документирование результатов:

- Подготовьте итоговый отчет о проекте, включающий описание выполненных задач, достигнутых целей и полученных результатов.
- Оцените эффективность выбранной методологии и использования инструмента управления проектом.

Задания к лабораторной работе:

10. Выбор методологии и настройка инструмента управления проектом

- Определите цель вашего индивидуального проекта.
- Выберите подходящую методологию разработки (Scrum, Kanban, Waterfall).
- Зарегистрируйтесь и создайте проект в выбранном инструменте управления проектами (Jira, Trello, Asana).
- Настройте доску проекта в соответствии с выбранной методологией.
- Разбейте проект на задачи и установите для них сроки выполнения.
- Организуйте задачи на доске, установите приоритеты и отслеживайте прогресс.

11. Анализ эффективности выбранной методологии:

- По завершении проекта оцените, насколько выбранная методология соответствовала вашим потребностям.
- Опишите сильные и слабые стороны методологии в контексте вашего проекта.
- Предложите возможные улучшения для будущих проектов.

Содержание отчёта

Введение

- **Цель и задачи лабораторной работы:**
 - Описание целей изучения методологий разработки и настройки инструментов управления проектами.
 - Определение ожидаемых результатов и навыков, которые студент должен приобрести.

Теоретическая часть

- **Описание изученных понятий и технологий:**
 - Введение в различные методологии разработки.
 - Критерии выбора подходящей методологии для индивидуального проекта.
 - Обзор инструментов управления проектами и их функциональных возможностей.
 - Принципы настройки инструментов управления проектами в соответствии с выбранной методологией.

Практическая часть

- **Пошаговое описание выполненных действий:**
 - Выбор методологии разработки для проекта.
 - Настройка инструмента управления проектом (создание проекта, настройка доски).
 - Планирование проекта: создание и распределение задач, установка сроков.
 - Мониторинг прогресса выполнения задач и управление изменениями.
- **Примеры интерфейсов и графиков:**
 - Скриншоты настроенной доски проекта в выбранном инструменте.
 - Примеры созданных задач с установленными сроками.

Результаты

- **Полученные результаты и их анализ:**
 - Описание выполненных задач и достигнутых целей проекта.
 - Анализ эффективности распределения задач и соблюдения сроков.
 - Оценка использования ресурсов и времени.
 - Оценка выбранной методологии и настроенного инструмента управления проектом.

Заключение

- **Выводы о проделанной работе:**
 - Оценка достигнутых целей лабораторной работы.

Приложения (по необходимости)

- **Скриншоты, графики, дополнительные материалы:**
 - Скриншоты задач и доски проекта в инструменте управления проектом.
 - Дополнительные материалы, такие как графики прогресса или анализ приоритизации задач.

Контрольные вопросы

1. Что такое методология разработки и почему выбор подходящей методологии важен для успешного выполнения проекта?
2. Опишите основные характеристики методологий Waterfall и Agile. В чем их ключевые отличия?

3. Какие критерии следует учитывать при выборе методологии разработки для индивидуального проекта?
4. Что такое диаграмма Ганта и как она помогает в планировании проекта?
5. Как настроить доску проекта в Jira в соответствии с методологией Scrum?
6. Что такое спринт и как его правильно планировать в системе управления проектами?
7. Какие преимущества предоставляет использование инструментов управления проектами, таких как Jira, Trello или Asana?
8. Какие шаги необходимо предпринять при возникновении задержек в выполнении задач проекта?

СПИСОК ЛИТЕРАТУРЫ

1. Альтман, Е. А. Система контроля версий GIT : учебно-методическое пособие / Е. А. Альтман, А. В. Александров, Т. В. Васеева. — Омск : ОмГУПС. ЭБС «Лань» : [сайт]. — URL: <https://e.lanbook.com/book/190155> (дата обращения: 10.03.2025).
2. Рошин, П. Г. Командная разработка программного обеспечения с помощью системы контроля версий Git: Конспект лекций : учебное пособие / П. Г. Рошин. — М : НИЯУ МИФИ. ЭБС «Лань» : [сайт]. — URL: <https://e.lanbook.com/book/355550> (дата обращения: 10.03.2025).