Отчет на тему 1

На первом этапе мы создали личный репозиторий, добавив файл уREADME.md с описанием проекта , Файл .gitignore для языка программирования, Папку src/ с простым скриптом. Сделали коммиты с осмысленными сообщениями, таким образом изучив некоторые основные функции GitHub.

На втором этапе мы изучили возможности репозитория, т.е. открыв доступный репозиторий на GitHub:

1. Изучили историю коммитов : в проект были внесены обновления README.md, дополнительные слияния pull request, удаление повторяющихся шагов и точек, незначительные исправления орфографических и грамматических ошибок в описаниях в data.json;
2. Проанализировали структуру проекта: присутствуют папки такие как .github, contributiong.md , README.md, data.json. Да, имеются файлы README.md, .gitignore, а также документация такая как сценарий и рабочие процессы.

Мини-ревью на основе данных репозитория:

Проект соответствует правилам чистого кода а именно: код максимально понятен, логика проекта очевидна из структуры и названия проекта, проект подразумевает единый стиль оформления, имена краткие, но в тоже время достаточно информативные. Информация не дублируется которая уже очевидна, и поясняет сложные алгоритмы, структура заранее продумана, а также следит, чтобы каждый метод решил только 1 задачу.

Качество коммитов влияет на понимание кода через упорядочевание истории изменений. Сообщения коммитов должны быть ясными, краткими и информативными. Они должны описывать какие изменения были внесены и почему. Плохие сообщения затрудняют отслеживание истории измений и их анализ.

Регулярное коммитирование помогает поддерживать атомарность изменений. Каждый коммит должен решать одну задачу или исправлять одну проблему делая изменения изолированными. Также частое коммитирование упрощает код-ревью – при проверке когда проще анализировать каждый коммит когда в нём содержится только одно логическое изменение.

В проекте для код ревью использовались следующие инструменты:

GitHub Pull Request для отслеживания изменений и обсуждений.

Автоматические инструменты для проверки кода на соответствие стилистическим стандартам и выявления потенциальных ошибок и уязвимостей

Эффективность этих инструментов оценивалась по следующим критериям:

1. Выявление ошибок
2. Оптимизация производительности
3. Соблюдение стандартов
4. Обмен знаниями

Некоторые проблемы, которые можно заметить в истории и структуре коммитов:

* Неправильно скомпонованные изменения. В одном коммите сочетаются исправление ошибки, новые фичи, правки документации и тестов, что делает историю сложной для понимания.
* Незавершённые изменения. Более поздний коммит отменяет что-то из более раннего частичного коммита, что усложняет установление связей между изменениями.
* Ошибки в сообщении коммита. Некорректное описание изменений может привести к неправильному пониманию истории.
* Ненужные файлы в истории. Например, логи, файлы IDE или временный вывод при тестировании проекта, которые остались в истории, хотя в текущей версии проекта их нет.

Отчет на тему 2

1. Цели ревью для приведённого фрагмента кода:

проверить, соответствует ли код идиомам и шаблонам языка;

убедиться, что в коде используются возможности языка и стандартные библиотеки;

оценить, делает ли код то, что должен, и не содержит ли очевидных логических ошибок.

2. Задачи для ревьюера:

проверить, соответствует ли код стандарту PEP-8;

убедиться, что соблюдаются все соглашения языка и проекта;

проверить, что идентификаторам присвоены значимые и соответствующие стандартам имена;

оценить, свободен ли код от обфускации и ненужной сложности.

3. Объекты проверки:

использование функций языка и стандартных библиотек;

читаемость кода;

соответствие стандартам оформления;

тесты безопасности;

пригодность кода к обслуживанию.

Чек лист

1 **Работоспособность кода** - Проверить, что модуль выполняет свои функции корректно, логика реализована правильно, ошибок при запуске нет.

2 **Понятность кода** - Оценить читаемость: ясные названия методов и переменных, структура кода, наличие комментариев и документации.

3 **Соответствие стилю оформления** Соблюдение стандартов оформления (PEP-8 для Python, Google Java Style для Java): правильные отступы, имена, форматирование.

4 **Избыточность кода** - Обнаружить повторяющиеся участки, которые можно вынести в отдельные функции или методы для повышения повторного использования.

5 **Обработка исключений** - Проверить, что исключения используются по назначению, есть обработка ошибок, логирование или информирование пользователя.

6 **Безопасность** - Проверить валидацию входных данных: типы, диапазоны, формат, отсутствие уязвимостей (например, SQL-инъекций, XSS).

7 **Документация** - Наличие комментариев, docstring для функций, описание параметров и возвращаемых значений.

8 **Тестирование** - Есть ли юнит-тесты, покрывающие основные сценарии, граничные случаи, ошибки.

9 **Наличие unit-тестов для всех методов** - Проверить, что все публичные и важные методы протестированы.

10 **Проверка уязвимостей** - Анализ потенциальных уязвимостей, таких как SQL-инъекции, уязвимости безопасности, связанные с вводом данных.

**План этапов ревью (в формате таблицы)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Этап** | **Действия** | **Срок** | **Участники** |
| Подготовка | Выбор кода, назначение ревьюера, ознакомление с модулем | 45 мин | Tech Lead, Ревьюер |
| Анализ | Проверка кода по чек-листу, выявление проблем | 2 часа | Ревьюер |
| Обсуждение | Обсуждение выявленных вопросов, уточнение деталей | 30 мин | Автор, Ревьюер |
| Исправление | Внесение правок по результатам ревью | 1-2 дня | Автор |
| Повторная проверка | Анализ исправленного кода | 1 час | Ревьюер |
| Финальное утверждение | Подписание итогового варианта | 30 мин | Tech Lead, Автор |

* Модуль Test содержит около 200 строк кода.
* Время на анализ: 2 часа.
* Время на подготовку и обсуждение: 1 час.
* Итоговая длительность всего процесса — примерно 4-5 часов.

Выводы (что было сложным в планировании? Как улучшить процесс?)

**Что было сложным в планировании:**

* **Определение критериев оценки:** Выбор общих и ясных пунктов контрольного списка, чтобы он был достаточно детализированным, но не избыточным.
* **Равновесие времени:** Верно рассчитать временные промежутки для каждого шага, чтобы не было как слишком сжатых, так и излишне растянутых периодов.
* **Гарантия всесторонности инспектирования:** Принять во внимание все вероятные стороны защиты, испытания и оформления, чтобы не упустить существенные детали.
* **Согласование стадий:** Обеспечить ясное осознание ролей и обязанностей участников, особенно при повторяющихся циклах.

**Как улучшить процесс:**

* **Автоматизация инспектирования:** Внедрить средства статического анализа кода (например, pylint, flake8, SonarQube), чтобы уменьшить бремя на проверяющих и увеличить точность.
* **Применение контрольных списков и образцов:** Заблаговременно подготовленные образцы дают возможность ускорить предварительную фазу и обеспечить всесторонность инспектирования.
* **Систематические разборы полетов:** После каждой проверки анализировать, что получилось хорошо, а что можно скорректировать, чтобы оптимизировать сроки и качество.
* **Инструктаж и обмен знаниями:** Организовывать внутренние занятия по передовым приемам проверки, чтобы увеличить продуктивность и уменьшить предвзятость.
* **Точное планирование времени:** Отводить конкретные временные отрезки для каждого шага, чтобы избежать задержек и обеспечить вовремя завершение.

Отчет на тему 3

**Таблица сравнения офисных пакетов**

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **Microsoft Office 365** | **Google Workspace** | **LibreOffice** |
| **Совместимость форматов (DOCX → ODT)** | Хорошая, но возможны небольшие потери форматирования | Умеренная, конвертация через экспорт/импорт | Отличная, нативный формат ODT, хорошая поддержка DOCX |
| **Стоимость лицензии** | Платная (подписка от $6.99/мес) | Платная (подписка от $6/мес) | Бесплатная (открытое ПО) |
| **Возможности совместного редактирования** | Редактирование в реальном времени (через OneDrive/Teams) | Лучшая поддержка (Google Docs) | Ограниченная (через облачные платформы, например NextCloud) |
| **Поддержка скриптов/автоматизации** | VBA, Office Scripts, Power Automate | Google Apps Script (JavaScript-подобный) | LibreOffice Basic, Python, API |
| **Мобильная функциональность** | Полнофункциональные приложения (iOS, Android) | Отличные мобильные приложения (онлайн-офис) | Ограниченные возможности (мобильные просмотрщики) |
| **Офлайн-работа** | Да ( Полная поддержка) | Только с включённым офлайн-режимом(ограничения) | Полностью офлайн |
| **Интеграция с облаком** | OneDrive, SharePoint | Google Drive | Сторонние решения (NextCloud, OwnCloud) |
| **Безопасность и конфиденциальность** | Корпоративный уровень(Но данные у Microsoft) | Зависит от политики Google | Локальное хранение, высокая приватность |

**Вывод:**

**1. Microsoft Office 365**

**Лучше всего подходит для:**

* **Корпоративных пользователей, которым нужна полная совместимость с DOCX и профессиональные инструменты (Excel, PowerPoint, Outlook).**
* **Командной работы (особенно в связке с Teams и OneDrive).**
* **Автоматизации процессов (VBA, Power Automate, интеграция с другими сервисами Microsoft).**
* **Офлайн-работы с полнофункциональными десктопными приложениями.**

**2. Google Workspace**

**Лучше всего подходит для:**

* **Онлайн-коллаборации** (лучший инструмент для совместного редактирования в реальном времени).
* **Малого бизнеса и стартапов** (дешевле Office 365, проще настройка).
* **Образования** (удобство для студентов и преподавателей).
* **Мобильной работы** (отличные приложения для Android/iOS).

#### ****3. LibreOffice****

**Лучше всего подходит для:**

* **Частных пользователей**, которым не нужны облачные функции.
* **Бюджетных решений** (полностью бесплатен).
* **Работы с ODT-форматами** (лучшая поддержка среди трёх).
* **Конфиденциальности** (все данные локально, нет облачной зависимости).

**Таблица сравнения браузеров**

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **Google Chrome** | **Mozilla Firefox** | **Microsoft Edge** |
| **Потребление RAM (10 вкладок)** | Высокое (~1.5–2.5 ГБ) | Среднее (~1–1.8 ГБ) | Оптимизированное (~1–2 ГБ) |
| **Энергоэффективность (разряд батареи за 10 мин)** | Средний (быстрый разряд) | Лучший (медленнее разряжает) | Лучший (режим экономии заряда в Edge) |
| **Скорость запуска (сек)** | Быстро (~1–3 сек) | Средне (~2–4 сек) | Очень быстро (~1–2 сек) |
| **Наличие расширений** | Самая большая библиотека (Chrome Web Store) | Много, но меньше, чем у Chrome (Firefox Add-ons) | Совместим с Chrome-расширениями + свои |

**Таблица критериев средств просмотра видео**

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | KMPlayer | MX Player | VLC Media Player |
| **Поддержка скриптов/автоматизации** | Умеренная (макросы, плагины) | Ограниченная (только базовые функции) | Лучшая (Lua-скрипты, API, CLI) |
| **Мобильная функциональность** | Хорошая (Android, iOS) | Отличная (оптимизация под Android) | Универсальная (все платформы) |
| **Субтитры** | Гибкие настройки, поддержка ASS/SSA | Автопоиск, синхронизация | Лучшая поддержка (включая встроенные) |
| **Редкие кодеки** | Хорошая поддержка | Зависит от аппаратного декодирования | Лучшая (встроенные кодеки, даже экзотические) |
| **Авто-коррекция** | Баланс цвета/звука | Улучшение видео (Dolby Audio) | Ручная коррекция (нет автоматики) |

Вывод:

1. **Если важна интеграция и экосистема** → Выбирайте продукты Microsoft (Office 365, Edge) или Google (Workspace, Chrome).
2. **Если нужна бесплатность и открытость** → LibreOffice, Firefox, VLC.
3. **Для мобильных устройств** → Google Workspace (документы), MX Player (видео), Firefox или Chrome (браузеры).
4. **Для профессионалов** → Office 365 (документы), Chrome/Edge (браузеры), VLC (медиа).

Каждый софт имеет свои сильные стороны, поэтому лучшая стратегия — комбинировать их под конкретные нужды.