Технический долг. Устранение технического долга

Введение

Проект online-shop представляет собой open-source платформу на Python, аналог onliner, amazon, 5-element и других, предназначенную для возможности покупки различных товаров в интеренете и просмотра рынка. Основная цель проекта — предоставить сообществу инструмент, который можно быстро настроить и использовать для онлайн-шопинга. Тем не менее, как и в любом проекте, в процессе разработки накапливается технический долг. Это может включать устаревшие зависимости, недостаток документации, сложности в архитектуре или отсутствие автоматизированных тестов, что может замедлять развитие проекта и затруднять участие новых контрибьюторов. Устранение этих проблем станет важным шагом для обеспечения долгосрочной поддержки и масштабируемости платформы.

1. Непонятный / нечитабельный код

- Описание: В коде отсутствуют комментарии и пояснения, что делает его сложным для понимания, особенно для новых разработчиков или контрибьюторов. Это может привести к ошибкам при внесении изменений или расширении функционала. Кроме того, отсутствие структурированных docstrings и описаний функций усложняет поддержку и документирование проекта.

- Пример: В модуле auth.py отсутствуют комментарии, объясняющие сложные логические блоки или настройки. Например, неясно, зачем используются определенные параметры JWToken, а также как они влияют на работу аутентификации. Это может затруднить понимание кода и его отладку.

2. Дублирующийся код

- Описание: Фрагменты кода, которые повторяются в нескольких местах проекта. Это приводит к увеличению объема кодовой базы, усложняет поддержку и повышает вероятность ошибок при внесении изменений. Если логика, описанная в дублирующемся коде, требует изменений, то придется вносить правки во всех местах, где она используется, что увеличивает риск упустить что-то важное.

- Пример: В проекте обнаружено не было.

3. Отсутствие автоматизации (тестов, сборки, развёртывания)

- Описание: В проекте отсутствуют автоматические тесты, что увеличивает риск появления ошибок при внесении изменений. Также нет автоматизированных процессов сборки и развёртывания.

- Пример: Каждый новый билд собирается вручную, что занимает время и может привести к ошибкам.

4. Запутанная архитектура и ненужные сложные зависимости

- Описание: Запутанная архитектура и избыточные зависимости — это распространенная проблема в проектах, которые развиваются без четкого плана или архитектурного видения. Это приводит к тому, что код становится сложным для понимания, поддержки и масштабирования. Модули и компоненты могут быть тесно связаны между собой, что затрудняет их замену или модификацию. Кроме того, использование ненужных или избыточных библиотек увеличивает сложность проекта, замедляет его работу и усложняет процесс разработки.

- Пример: В проекте могут использоваться библиотеки, которые не несут реальной пользы, но увеличивают сложность и размер проекта. В нашем проекте используется тяжелая ORM (Object-Relational Mapping) `SQLAlchemy` для простых запросов к базе данных.

5. Медленные / неэффективные средства

- Описание: Использование устаревших или неэффективных библиотек и инструментов может значительно замедлить работу приложения, увеличить потребление ресурсов (память, процессор) и усложнить поддержку проекта. Это особенно критично для высоконагруженных систем, где производительность играет ключевую роль. Устаревшие библиотеки могут также содержать уязвимости или не поддерживать современные стандарты, что создает риски для безопасности и стабильности проекта.

- Пример: В проекте для работы с HTTP-запросами используется `requests` вместо асинхронной библиотеки `aiohttp`, что замедляет обработку запросов в асинхронных приложениях.

6. Незакоммиченный код / долгоживущие ветки

- Описание: Незакоммиченный код и долгоживущие ветки — это распространенные проблемы в проектах, где разработка ведется без четкого процесса управления версиями. Незакоммиченный код может быть потерян, если что-то пойдет не так (например, сбой системы или ошибка разработчика). Долгоживущие ветки, которые существуют в репозитории неделями или месяцами, усложняют процесс слияния (merge) и увеличивают риск конфликтов. Это также замедляет интеграцию новых функций и исправлений в основную ветку.

- Пример: В проекте обнаружено не было.

7. Отсутствие / несоответствие технической документации

- Описание: Документация либо отсутствует, либо не соответствует текущему состоянию проекта.

- Пример: В проекте обнаружено не было, так как используется автоматический генератор документации `Swagger`.

8. Отсутствие тестовой среды

- Описание: Нет отдельной среды для тестирования новых функций перед их внедрением в основную ветку.
- Пример: Все тестирование проводится вручную на локальных машинах разработчиков, что увеличивает риск пропуска ошибок.

9. Длинные циклы интеграции / отсутствие непрерывной интеграции

- Описание: Отсутствие настроенной системы CI/CD приводит к длительным циклам интеграции и возможным ошибкам при ручном деплое.
- Пример: Каждое обновление требует ручного тестирования и деплоя, что занимает значительное время.

План мероприятий по устранению технического долга

1. Улучшение читаемости кода

- Действия : Добавить комментарии и docstrings к ключевым функциям и классам, особенно в модулях аутентификации (`auth.py`), работы с вопросами и ответами.
- Оценка времени: ~6 часов.

2. Внедрение автоматического тестирования

- Действия: Написать юнит-тесты для критически важных компонентов: аутентификации, API вопросов и ответов, системы рейтинга пользователей. Настроить автоматическое выполнение тестов при коммитах.
- Оценка времени: ~15 часов.

3. Обновление используемых библиотек

- Действия: Провести аудит зависимостей и обновить их до актуальных версий, избавиться от устаревших и ненужных библиотек. Например, заменить `requests` на `aiohttp` в асинхронных местах.
- Оценка времени: ~5 часов.

4. Создание тестовой среды

- Действия: Настроить тестовую среду, идентичную продакшн-окружению, для безопасного тестирования новых функций.
- Оценка времени: ~8 часов.

5. Внедрение системы непрерывной интеграции/развёртывания (CI/CD)

- Действия: Настроить автоматизированные процессы сборки, тестирования и развертывания с помощью GitHub Actions, GitLab CI или Jenkins.
- Оценка времени: ~10 часов.

Оценка плана мероприятий

Общий объем технического долга составляет около 44 часов.

Сравнение объёма долга и недоимплементированных фич

Для точного анализа следует оценить время, необходимое для реализации новых функций, таких как:

- Улучшение механизма поиска по вопросам.
- Реализация системы меток и категорий.

Оценочное время реализации новых функций составляет 40-50 часов, что сопоставимо с объемом технического долга.

Выводы и рекомендации

Устранение технического долга является ключевым фактором для долгосрочного развития online-shop. Следует интегрировать задачи по его устранению в текущий план разработки, чтобы постепенно улучшать качество кода и не откладывать новые функции на неопределенный срок.