ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА ПРОВЕРКИ ДОКУМЕНТОВ, УДОСТОВЕРЯЮЩИХ ЛИЧНОСТЬ

1. Предназначение

«Интеллектуальная система проверки документов, удостоверяющих личность» предназначена для автоматического распознавания паспорта гражданина России удостоверяющего личность, практически в любых условиях. Система автоматически распознает сканы, фотографии в независимости от ракурса съемки, избытка, неравномерности или недостатка освещения, перекосов или поворотов.

2. Обоснование использования технологического стека

2.1 Git

Для многих работа с контролем версий превратилась в простое копирование файлов в локальный каталог либо добавление в текущий файл даты изменения. Но подход «просто забэкапить этот файл» весьма опасен и может быть фатальным. Цена такой ошибки – не один час, а может мыть даже и денб потраченного времени на восстановление.

Самый известный способ избежать подобных проблем в случае работы команды программистов, работающих над проектом – на этапе разработки использовать полноценную систему контроля версий, а именно Git.

Выбор системы контроля версии основывался на доступности всех участников разработки проекта. Данная доступность к репозиторию должна быть организована 24/7 в связи с этим была выбрана распределенная система контроля GIT. Централизованные и локальные системы хранения не позволяют всем участникам работать над одним проектом в каком угодно месте и в удобное время.

Git представляет собой одну из разновидностей VCS (Version Control System) программ для работы с постоянно изменяющейся информацией. Такое программное обеспечение может хранить множество версий одного и того же файла (документа) и возвращаться к более раннему состоянию (версии). Git не зависит от центрального сервера, где хранятся файлы, он сохраняет данные в локальном репозитории. После окончания работы программист выгружает код в общий репозиторий, где ответственным за проект устраняются возможные конфликты в программном коде в случае их возникновения. Для большей стабильности и ускорения синхронизации разных версий проекта локальный репозиторий хранят онлайн в специальных сервисах: Github, Gitlab, Bitbucket.

С приходом VCS работать стало гораздо проще:

- можно откатывать изменения, если смысла в их внедрении нет;
- можно быстро и безболезненно восстанавливать поврежденные файлы;
- можно определить, кто из команды писал определенный блок кода;

- можно следить за процессом, даже если в один и тот же момент над одним модулем работает несколько разработчиков или даже команд по всему миру.

Причины работать с Git

1. Легко масштабировать работу по проекту, легко включать в процесс нужных сотрудников. При грамотном ведении репозитория в кратчайшие сроки можно развернуть дополнительную площадку или удалить лишнего разработчика из проекта.
2. Проект легко передавать из команды в команду, от разработчика к разработчику: просто отправьте ссылку на репозиторий.
3. Ничего не теряется. Все версии файлов сохраняются, и ситуаций, при которой может «потеряться» безвозвратно та или иная разработка, не бывает.
4. Можно «распараллелить» работу между разработчиками и целыми командами: распределите нагрузки между сотрудниками – так вы значительно повысите скорость выполнения проектных работ.
5. Можно отслеживать, как идет работа, логирует ее, и в любой момент можно увидеть, кто и когда внес изменения.
6. Подводя итог: Git – это система работы команды программистов, при которой все они могут вносить изменения одновременно, не опасаясь за работоспособность проекта. Git необходим, если над сайтом работают сразу несколько команд разработчиков, и сам сайт при этом весьма сложен в архитектуре и состоит из большого числа файлов.

2.2 GitLab

В качестве инструмента для хранения и управления репозиторием Git использовался GitLab. Приведем ключевые отличия от ближайшего конкурента GitHub.

Вот некоторые из ключевых отличий между GitHub и GitLab:

1. **Встроенная непрерывная интеграция.** GitLab известен своей бесплатной встроенной непрерывной интеграцией, которую GitHub не предлагает. Вместо этого GitHub предлагает сторонние интеграции, которые обеспечивают непрерывную интеграцию.
2. **Аутентификация.** В GitLab разрешение предоставляется на основе ролей людей, в то время как в GitHub разработчики могут предоставлять доступ на чтение или запись к определенным репозиториям.
3. **Импорт/экспорт данных.** GitLab предлагает гораздо более подробную документацию о том, как импортировать/экспортировать данные от внешних поставщиков, в то время как документация GitHub не такая подробная. GitLab способен импортировать проекты и проблемы из большего количества источников, чем GitHub. При этом GitHub предлагает инструмент под названием GitHub Importer для ввода данных. Что касается экспорта, GitLab предлагает комплексное решение для экспорта вики, репозиториев проектов, загрузки проектов, веб-хуков и сервисов, а также проблем. С другой стороны, GitHub немного более ограничен с точки зрения возможностей экспорта.
4. **Платформа развертывания.** GitHub не поставляется со встроенной платформой развертывания и требует сторонней интеграции с внешним приложением для развертывания приложений. С другой стороны, GitLab использует Kubernetes для беспроблемного развертывания.

5. **Частные репозитории.** GitLab предлагает бесплатные частные репозитории для проектов с открытым исходным кодом, а GitHub - нет.
6. **Отслеживание комментариев.** GitHub способен предоставить полную историю обновлений комментариев - GitLab не поддерживает это.
7. **Экспорт файла CSV.** GitLab способен экспортировать файлы CSV с проблемами на адреса электронной почты уведомлений по умолчанию в виде вложений.
8. **Конфиденциальные вопросы.** Модуль GitLab Confidential Issues создает конфиденциальные проблемы, которые видны только участникам проекта с уровнем доступа Reporter или выше.
9. **Графики выгрузки.** В отличие от GitHub, GitLab предлагает Burndown Charts как часть этапов, которые позволяют разработчикам отслеживать прогресс во время спринтов или при работе над новыми версиями программного обеспечения.
10. **Циклическая аналитика.** GitLab предоставляет панель мониторинга для анализа времени, планирования и мониторинга.
11. **Ежемесячные новые функции.** GitLab известен своими ежемесячными обновлениями новых функций и улучшений, которые неукоснительно делаются 22-го числа каждого месяца.

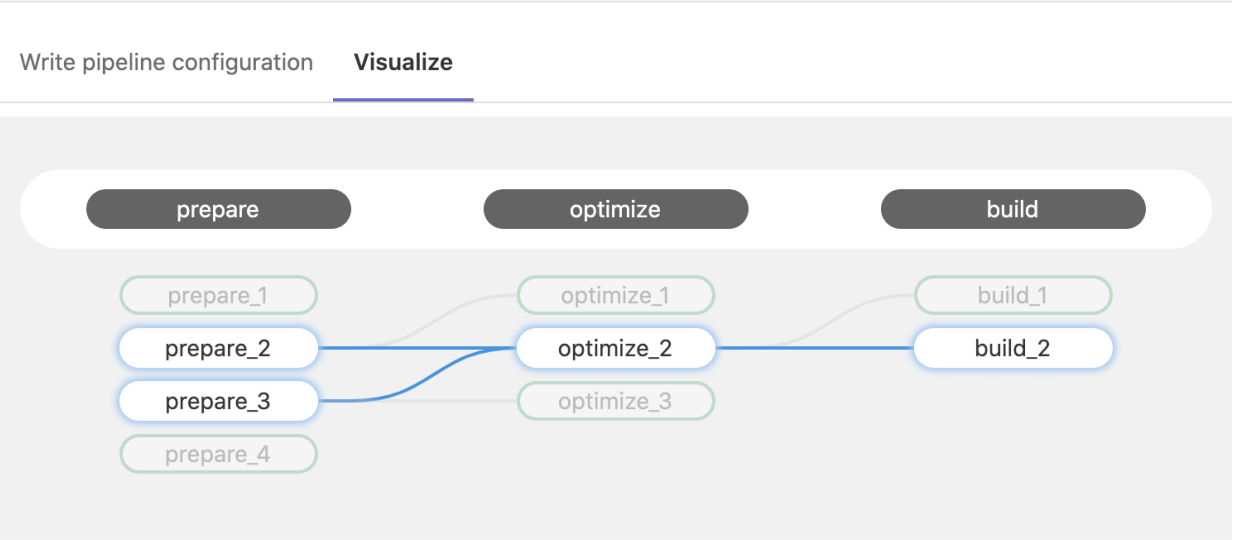
Помимо тех, что мы только что указали, основное различие между GitHub и GitLab заключается в философии каждой платформы. GitHub больше ориентирован на высокую доступность и производительность инфраструктуры, в то время как GitLab больше ориентирован на включение как можно большего количества функций в надежную и хорошо интегрированную платформу для полного и централизованного процесса DevOps. Данный фактор являлся ключевым при выборе сервиса для хранения репозитория.

GitLab CI / CD - это встроенный в GitLab инструмент для разработки программного обеспечения с использованием непрерывных методологий. Непрерывная интеграция работает, отправляя небольшие фрагменты кода в базу кода вашего приложения, размещенную в репозитории Git, и при каждом нажатии запускает конвейер сценариев для создания, тестирования и проверки изменений кода перед их объединением в основную ветвь.

Непрерывная доставка и развертывание состоят из следующего этапа CI, развертывания вашего приложения в производственной среде при каждом нажатии на ветвь репозитория по умолчанию.

Эти методологии позволяют обнаруживать ошибки на ранних этапах цикла разработки, гарантируя, что весь код, развернутый в производственной среде, соответствует стандартам кода, установленным для вашего приложения.

GitLab CI / CD настраивается файлом, который .gitlab-ci.ymlнаходится в корне репозитория. Этот файл создает конвейер , который запускается для изменений кода в репозитории. Конвейеры состоят из одного или нескольких этапов, которые выполняются по порядку, и каждый может содержать одно или несколько заданий, выполняемых параллельно. Этот сценарий выполняются агентом GitLab Runner .
При создании .gitlab-ci.ymlвы можете использовать визуализацию конфигурации CI / CD, чтобы облегчить ваш опыт написания



Этапы CI/CD

Написание кода. Каждый из разработчиков пишет код своего модуля, проводит ручное тестирование, а затем соединяет результат работы с текущей версией проекта в основной ветке. Для контроля версий используется система Git, либо аналогичные решения. Когда участники команды опубликуют код своих модулей в основной ветке, начнется следующий этап.

Сборка. Система контроля версий запускает автоматическую сборку и тестирование проекта. Триггеры для начала сборки настраиваются командой индивидуально — фиксация изменений в основной ветке проекта, сборка по расписанию, по запросу и т.д. Для автоматизации сборки используется Jenkins, либо аналогичный продукт.

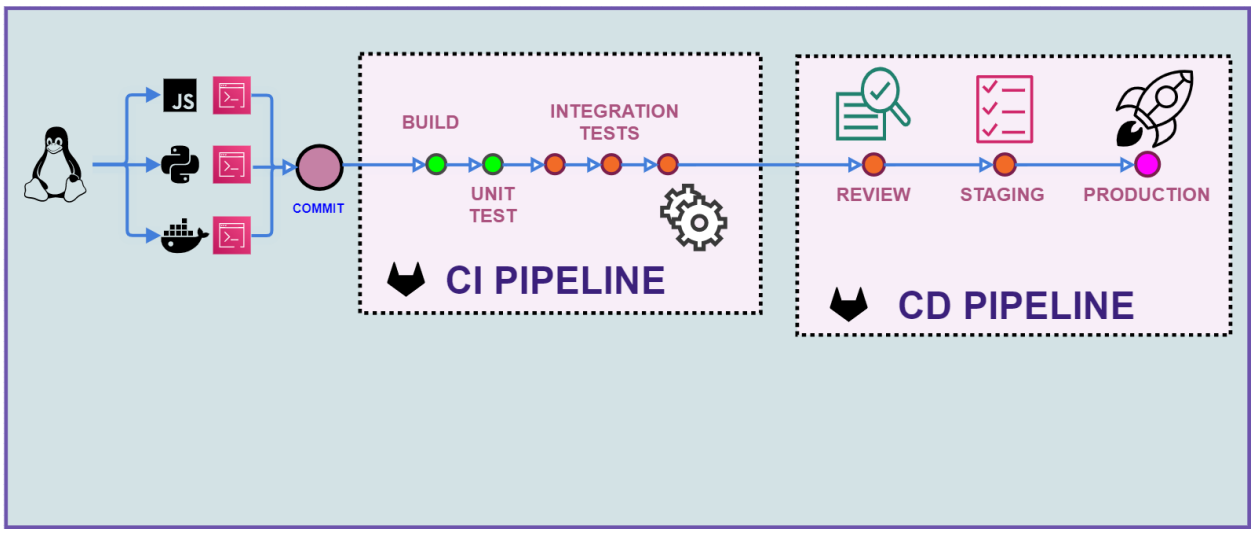
Ручное тестирование. Когда CI система успешно проверила работоспособность тестовой версии, то код отправляется тестировщикам для ручного обследования. При этом тестовая сборка получает номер кандидата для дальнейшего релиза продукта (например, v.1.0.0-1).

Релиз. По итогам ручного тестирования сборка получает исправления, а итоговый номер версии кандидата повышается (например, после первого исправления версия v.1.0.0-1 становится v.1.0.0-2). После этого выпускается версия кода для клиента (например, v.1.0.0) и начинается следующий этап цикла.

Развертывание. На этом этапе рабочая версия продукта для клиентов автоматически публикуется на production серверах разработчика. После этого клиент может взаимодействовать с программой и ознакомиться с ее функционалом как непосредственно через готовый интерфейс, так и через облачные сервисы.

Поддержка и мониторинг. Конечные пользователи начинают работать с продуктом. Команда разработки поддерживает его и анализирует пользовательский опыт.

Планирование. На основе пользовательского опыта формируются запросы на новый функционал для продукта, готовится план доработок. После этого цикл замыкается и переходит в начальную стадию — написание кода. Далее начинается новая итерация CI/CD разработки.



При обсуждении реализации кода были предложены и написаны автоматизированные CI/CD скрипты создания и тестирования нашего приложения для дальнейшего развертывания приложения в производственной среде. В случае если что-то пойдет не так мы сможем откатить свои изменения исправить ошибки и баги.

Первоначальные тесты выполнялись с использованием tox и pytest

Поскольку Tox является универсальным инструментом командной строки для управления и тестирования virtualenv, его можно использовать для:

- проверки правильности установки вашего пакета с разными версиями Python и интерпретаторами
- запуска тестов в каждой из сред, настройка инструмента тестирования по выбору
- выступает в качестве внешнего интерфейса для серверов непрерывной интеграции, значительно сокращая количество шаблонов и объединяя тестирование на основе CI и оболочки

В свою очередь Pytest инструмент, автоматически находит написанные тесты, запускает их и пишет отчеты с результатом. Он имеет широкую библиотеку, которую можно использовать в тестах, чтобы тестирование производилось более эффективно. Он может быть расширен путем написания собственных плагинов или установки сторонних.