**Федеральное государственное образовательное бюджетное учреждение высшего образования**

**«Финансовый университет при Правительстве Российской Федерации»**

**Колледж информатики и программирования**

Группа: 4ПКС-115

**Отчет по производственной практике**

**(преддипломной)**

**Руководитель практики от предприятия**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Руководитель практики от колледжа**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Студент**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Оценка** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.2019

2019

Содержание

[Введение 3](#_Toc8814188)

[1. Предпроектное исследование предметной области 6](#_Toc8814189)

[2 Среда и язык программирования 10](#_Toc8814190)

[3 Анализ технологий, используемых для реализации программы 11](#_Toc8814191)

[Заключение 26](#_Toc8814192)

[Список литературы 27](#_Toc8814193)

[Приложение A 30](#_Toc8814194)

# Введение

Преддипломная практика проходила на предприятии АО КБ «РУСНАРБАНК» и была направлена на изучение и сбор научно-практического материала по теме ВКР «Разработкаавтоматизированной система обработки финансовой отчётности с использованием библиотеки компьютерного зрения OpenCV для АО КБ Руснарбанк»

Целями преддипломной практики являлись структуризация материала ВКР по следующим разделам: предпроектное исследование предметной области, анализ технологий в программе, а также среды и языка программирования.

Задачами преддипломной практики являлись:

* Проведение анализа и обобщения научно-технической информации по теме ВКР;
* Подбор фактического материала по теме ВКР с учетом профессиональных модулей и его изучение;
* Подготовка плана пояснительной записки к ВКР;
* Подбор программных, аппаратных и/или инженерно-технических средств для реализации практической части ВКР;
* Описание входных параметров задачи дипломного проектирования в соответствии с техническим заданием;
* Оформление и сдача отчета по преддипломной практики.

На сегодняшний день компьютерное зрение все сильнее входит в нашу повседневную жизнь: теперь можно разблокировать телефон по Face ID, использовать еще больше масок и эффектов в Instagram, примерить виртуальные кроссовки у себя дома, автоматически нанести мейкап на фото, найти известную актрису по лицу вашей подруги, автоматически припарковать автомобиль, переопределить область работы робота-пылесоса и т.д.

В большинстве крупных предприятий алгоритмы компьютерного зрения уже используется относительно давно, причем пишутся программы для совершенно разных сфер: мониторинг соблюдения прямых линий посева для автоматизированной системы удобрения урожая на фермах, контроль качества отливки изделий на заводе металлоконструкций, автоматический учёт парковочных мест в торговых центрах. Компьютерное зрение плотно вошло коммерческое использование.

Также обособленно можно выделить абсолютно уникальную сферу применения компьютерного зрения – цифровизация и её автоматизация: оцифровка старых фото в приложении «Фотосканер» от Google, программа для сканирования докуменов с телефона CamScanner, перевод текста с картинки в реальном времени в Google-переводчике, автоопределение QR-кода в приложении камеры, сканирование номера карты в основных банковских приложениях и т.д.

При этом процесс перехода от аналоговых на цифровые носители для большинства небольших предприятий в странах СНГ уже давно завершился, но существует ряд сфер, где глобальная цифровизация проходит очень медленно в силу масштабности экосистемы или иных причин. Такими предприятиями являются:

* Большинство государственных учреждений;
* Учебные учреждения (детские сады, школы, колледжи, университеты);
* Крупные коммерческие банки;

Т.к. цифровизация проходит по этапам, то очень часто бывают ситуации, при которых используется комбинация аналоговых и цифровых носителей информации, например, необходимо распечатать документ на цветном принтере, поставить подпись и печать, отсканировать этот же документ и послать на почту, где другой человек из другого отдела введет данные из этого документа в базу данных.

Во избежание подобных ситуаций и внедряют системы автоматизирования документооборота, которые используют совокупность алгоритмов компьютерного зрения с оптическим распознаванием символов. Конкретная реализация необходима для распознавания данных бухгалтерской отчетности для клиентов АО КБ «Руснарбанк».

Таким образом, целью ВКР является разработка программы перевода данных файлов pdf в json за счёт оптического распознавания символов в целях архивации данных.

Задачами ВКР являются:

* Анализ требований и определение спецификации ПО;
* Проектирование программного обеспечения;
* Разработка пользовательских интерфейсов ПО;
* Анализ технологий, применяемых при разработке ПО;
* Тестирование и отладка ПО.

## 1. Предпроектное исследование предметной области

На текущий момент не существует каких-либо программ с открытым исходным кодов для распознавания финансовой отчетности в странах СНГ.

Все данные из физического вида переходят в цифровой четырьмя наиболее распространёнными способами:

* Предприятие арендует или покупает набор программных средств для распознавания финансовой отчетности;
* Предприятие пишет свою собственную систему распознавания для тех типов отчетностей, которые её необходимо обрабатывать;
* Сотрудники предприятия переносят данные из физического вида в электронный;
* Предприятие использует проприетарное ПО от ЦБ для распознавания некоторых типов отчетностей (только для коммерческих банков с лицензией);
* Гибридное использование способов выше.

Под понятием «предприятие» мы понимаем как банки, так и прочие организации.

Под финансовой отчетностью понимаются следующие типы документов:

* Бухгалтерский баланс (ОКУД 0710001);
* Отчет о финансовых результатах (ОКУД 0710002);
* Отчет об изменениях капитала (ОКУД 0710003);
* Отчет о движении денежных средств (ОКУД 0710004);
* Приложение к бухгалтерскому балансу (ОКУД 0710005);
* Книга учёта доходов и расходов организаций и индивидуальных предпринимателей, применяющих упрощенную систему налогообложения;
* Извещение о вводе сведений, указанных в налоговой декларации (расчете) в электронной форме (КНД 1166007);
* Налоговая декларация по налогу, уплачиваемому в связи с применением упрощенной системы налогообложения (по форме КНД 1152017);
* Налоговая декларация по налогу на прибыль организаций (КНД 1151006);
* Налоговая декларация по налогу на прибыль иностранной организации (КНД 1151038);
* Бухгалтерская (финансовая) отчетность (КНД 0710099);
* Оборотно-сальдовая ведомость.

Конкретно, в случае рассмотрения банков, используются два наиболее популярных способа: использование проприетарного ПО от ЦБ или написание собственной закрытой системы.

При использовании ПО от ЦБ:

Плюсы:

* Нет необходимости поддержки продукта;
* Относительно незначительные затраты на обслуживание системы;
* Нет финансовых затрат на разработку;

Минусы:

* При глобальных изменениях в законодательстве оптимизация программы будет невозможна;
* Монолитность решения;
* Не всегда корректное распознавание документов;
* Распознавание ограниченного количества типов документов;
* Невозможность анализа исходного кода программы, а также не известны технологии на которых работает данное ПО, что вызывает ряд вопросов, связанных с безопасностью.

При использовании собственного ПО:

Плюсы:

* Разработка для задач конкретного банка;
* Возможность изменения и адаптации программы под какие-либо новые условия;
* Модульность решения: простая интеграция новых типов документов и задач;
* Понятность технологий по которым работает ПО т.к. есть доступ к исходному коду решения, отсутствие вопросов с безопасностью;

Минусы:

* Необходимо выделение ресурсов на разработку;
* Необходимо выделение ресурсов на поддержку;
* Также может быть не всегда корректное распознавание документов как и в случае с ПО от ЦБ;

Для относительно малых банков, где количество бухгалтерской отчетности невелико, рациональнее использовать ПО от ЦБ т.к. средства, вложенные в разработку собственного ПО не будут оправдывать результатов т.е. проще нанять несколько сотрудников для перепечатывания данных с физического вида в электронный в случае, если программа не может распознать данные, чем тратить эти средства на разработку собственного ПО.

Для средних и крупных банков, через которое каждый день проходит большое количество документов, логичнее использовать собственное программное решение с возможностью распознавания расширенных типов документов и меньшей погрешности распознавания.

В случае, если процент распознавания определенного типа документа выше у ПО от ЦБ, чем у внутреннего ПО, то можно просто продолжать использовать это ПО для этих типов документов, например из данных результатов распознавания таблицы ниже мы можем использовать разработанное ПО для типов документов 0710001 и 0710005, а ПО от ЦБ для типа 0710002 (таблица 1 ниже).

Таблица 1 – Сравнение разработанного ПО и ПО от ЦБ

|  |  |  |
| --- | --- | --- |
| ПО | Тип документа | Распознавание (%) |
| Разработанное | 0710001 | 93.1 % |
| От ЦБ | 0710001 | 69.4% |
| Разработанное | 0710002 | 82.8% |
| От ЦБ | 0710002 | 87.3% |
| Разработанное | 0710005 | 95.3% |
| От ЦБ | 0710005 | 74.6% |

Таким образом, для решения поставленных задач было составлено приложение А с содержимым ВКР.

## 2 Среда и язык программирования

В качестве языка программирования для разработки алгоритма предобразования файла pdf в json, связанного с компьютерным зрением, рассматривались два возможных варианта: С++ и Python.

При использовании C++ мы получаем следующие преимущества:

* Отличная производительность разрабатываемого кода. Ни для кого не секрет, что С++ является намного более производительным ЯП, нежели Python (да и сам Python написан на C). Стоит учитывать, что используемый модуль python-opencv является всего лишь обёрткой для кода C++, на котором и написана библиотека (т.е. существенной разницы в производительности внутренних методов библиотеки на разных ЯП нет), что нельзя утверждать о внешних методах;
* Возможность работы с системой на относительно низком уровне: более гибкие возможности менеджмента потоков выполнения, возможность использования ассемблерных вставок  и т.д.;

Все недостатки решения на C++ связаны с обслуживанием разрабатываемой программы: более трудозатратный процесс разработки, поддержки проекта, а также интеграции иных модулей и решений. Как следствие, необходимо большее финансирование разработки.

При использовании Python в качестве основного языка программирования, мы получаем следующие преимущества:

* Работа с большими данными. При использовании действительно большого количества данных, python может обрабатывать их на уровне C++ или даже быстрее;
* Меньшие затраты на обслуживание системы. За счёт простого синтаксиса возможна более быстрая переносимость и масштабируемость решения;
* Универсальность - python является языком сценариев и пригоден для разработки множества решений, начиная от управления портами GIPO, заканчивая рендерингом страниц web-сайтов.

Недостатки:

* Низкая производительность при использовании встроенных методов. Является следствием за счёт особенностей архитектуры ЯП: интерпретируемость, все встроенные типы являются объектами, списки могут содержать объекты различного типа и т.д.

## 3 Анализ технологий, используемых для реализации программы

При любой разработке программы, связанной с компьютерным зрением, необходимо выбрать оптимальные технологии и алгоритмы для наиболее рациональных результатов. В любых решениях следует соблюдать идеальный баланс между следующими критериями:

* Финансовые вложения в разработку и работу системы;
* Время разработки системы;
* Качество и производительность распознавания системы.

На сегодняшний день существует большое количество технологий, которые возможно интегрировать в программы, связанные с компьютерным зрением. Во-первых, необходимо определиться c библиотекой компьютерного зрения.

Рассмотрим наиболее популярные решения на сегодняшний день:

* SimpleCV является open-source фреймворком для создания приложений, связанных с компьютерным зрением. C его помощью можно получить доступ к нескольким мощным библиотекам компьютерного зрения, таким как OpenCV – без необходимости учета многих нюансов: форматов файлов, цветовых пространств, управлении буферами, битовых глубин цвета и так далее. Относительно простой в освоении, написан на Python и для Python;
* BoofCV – open-source библиотека, написанная без какой-либо основы с нуля для систем машинного обучения реального времени. В качестве ключевых особенностей выступают низкоуровневые (а значит быстрые) алгоритмы: обработки изображений, AR-визуализации, определения объектов по каким-либо свойствам, захвата объектов и распознавания. BoofCV выпущен под лицензией Apache 2.0 для коммерческого и академического использования, написан на Java;
* FastCV – библиотека компьютерного зрения от копании Qualcomm, оптимизированная под мобильные устройства с процессорами Qualcomm и включает в себя наиболее часто используемые функции обработки данных для широкого спектра мобильных устройств;
* OpenCV – является наиболее популярной библиотекой алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым исходным кодом, имеет более 2500 оптимизированных алгоритмов, которые включают в себя полный набор как классических, так и современных алгоритмов компьютерного зрения и машинного обучения. Огромнейшая база поддержки сообществом разработчиков, реализация на C/C++;
* Accord.NET Framework – представляет собой платформу машинного обучения .NET с библиотеками обработки аудио и изображений, полностью написанных на C#. Хороший фреймворк для создания open-source и коммерческих приложений, связанных с компьютерным зрением, просчётом статистических данных, компьютерным анализом и т.д. Есть отличная поддержка сообщества разработчиков.

Когда определили список возможных библиотек, выявляем те, которые точно не будем использовать:

* Accord.NET Framework - фреймворк создан только для платформы .NET. Поддержки Python, что естественно, в нем нет и не будет;
* FastCV – очень сильно ограничен набор методов, оптимизация под мобильные решения в нашем случае совершенно не нужна;

Остаётся 3 наиболее подходящих решения:

* SimpleCV;
* BoofCV;
* OpenCV.

Разберем плюсы и минусы каждого из них.

Плюсы SimpleCV:

* SimpleCV использует OpenCV и другие библиотеки, такие как pygame и т.д. в качестве основных модулей;
* Низкий порог вхождения в разработку;
* Скорость разработки для относительно простых проектов по компьютерному зрению.

Минусы SimpleCV:

* Очень слабая поддержка сообщества разработчиков;
* Слабая производительность т.к. сама библиотека написана на Python.

Плюсы BoofCV:

* Высокая производительность на некоторых алгоритмах: в некоторых случаях не уступают реализации на C++ OpenCV;
* Хорошая поддержка сообщества разработчиков;
* Легкая интеграция с Android: можно подключить уже скомпилированные jar-файлы к проекту;

Минусы BoofCV:

* Под Python необходимо использовать обёртку pyboof из-за которой падает производительность решения при передаче данных;
* Высокий порог вхождения в разработку;

Плюсы OpenCV:

* Самая большая поддержка сообщества разработчиков среди альтернатив, активная разработка новых версий;
* Использование OpenCL и CUDA для параллельный вычислений с использованием процессора и видеокарты;
* Определенная поддержка со стороны команды разработчиков Intel;
* Лицензия BSD, позволяющая переписывать исходный код библиотеки под свои задачи и нужды;
* Использует процессорные инструкции MMX (SIMD) и SSE;
* Мой опыт использования библиотеки в предыдущих проектах.

Минусы OpenCV:

* Нерациональная трата ресурсов при интеграции в мобильные приложения;
* Средний порог вхождения в разработку;

С учётом всех положительных и отрицательных сторон, выбор был сделан в пользу OpenCV из-за идеального баланса между поддержкой сообщества, порога вхождения в разработку и относительно высокой производительностью.

Теперь необходимо определиться с библиотекой для оптического распознавания символов (Optical Character Recognition - OCR).

На момент написания работы существуют следующие, наиболее востребованные решения:

* Tesseract OCR – был разработан компанией HP как OCR с открытым исходным кодом под лицензией Apache 2.0, далее разработка была поддержана Google. Является одной из лучших систем OCR, используемых во всем мире;
* Google Drive OCR/Google Cloud Vision OCR – это простой в использовании сервис, предлагаемый Google в рамках сервисов Drive и Cloud. На стороне Drive существует возможность конвертировать различные типы файлов в текстовые данные, а Cloud предоставляет API с простой и краткой характеристикой: быстро, качественно, очень дорого;
* ABBYY FineReader Engine – один из лучших OCR-движков, разработанный компанией ABBYY для предоставления услуг в сфере распознавания данных, все время внедряются новые функции и возможности оптического распознавания символов. Алгоритмы компьютерного зрения, лежащие в основе работы движка являются закрытыми т.к. это коммерческое программное обеспечение без открытого исходного кода. Существует два основных способа взаимодействия с движком: с помощью SDK или с помощью клиента в браузере;
* CIB OCR Engine – движок оптической системы распознавания, разработанный немецкой компанией CIB software GmbH и активно используется ей же для поддержки работы комплекса программ doXiview, doXisafe App и doXisafe Web для поддержки распознавания штрих-кодов и текста. Движок поддерживает различные типы вывода обработанной информации по типу HTML, PDF, TXT и может быть интегрирован в какие-либо сторонние приложения;
* Asprise OCR SDK – SDK, созданное для распознавания текста и штрих-кодов. Предлагает высокопроизводительный API с простой интеграцией в приложения. Существует огромное сочетание входных и выходных форматов файлов, например PNG в DOCX, TIFF в XML, PDF в XLSX и т.д. Есть отдельное решение для сканированных документов - Asprise Image Scanning SDK;
* OCR.space – веб-сервис, разрабатываемый компанией a9t9 software GmbH, основная идея – простой инструмент распознавания символов для сообщества. Предоставляет простой и понятный API для распознавания текста с форматов PDF, PNG или JPG в TXT;
* GOCR – программа для оптической обработки и распознавания символов с открытым исходным кодом, лицензирована под GPL 2.0. Может быть использована с различными компонентами и легко портируется на другие архитектуры и ОС. Приемлемое качество распознавания со всеми основными форматами файлов;
* GNU Ocrad – оптическая система распознавания символов с открытым исходным кодом. Включает в себя анализ макета изображения для выявления колонок или отдельных блоков текста в файле, читает изображения в растровом, черно-белом или цветном форматах и создаёт текст в 8-битном или UTF-8 форматах. Может использоваться как отдельное приложение в терминале UNIX-подобных ОС или в качестве интегрируемого модуля в backend-решение.

С точки зрения пользователя, наборы инструментов OCR и лежащие в их основе алгоритмы должны:

* Анализировать макет и контекст, где именно распознаётся текст, определять тип элементов (таблица, текст, штрих-код, изображение);
* Поддерживать разные входные типы документов (PDF, JPG, PNG);
* Поддерживать большое количество различных типов алфавитов (русский, вьетнамский, арабский, японский);
* Поддерживать экспорт результатов распознавания в разных форматах (HTML, TXT, CSV);

С точки зрения разработчика, в OCR-решениях необходимо лишь два параметра:

* Быстродействие системы;
* Качество распознавания системы.

Также желателен:

* Базовый автопрепроцессинг изображения: устранение шума, выправление угла поворота, повышение контрастности, обратные аффинные преобразования;
* Открытый исходный код для минимизации финансовых затрат на обслуживание разработанной системы;

Для большей наглядности сведём все данные о библиотеках оптического распознавания в единую таблицу и выберем из неё решение, подходящее под наши критерии разработки системы (таблица 2 ниже).

Таблица 2 – Сравнение библиотек оптического распознавания

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Имя | Лицензия | Онлайн | Язык реализации | SDK | Поддержка русского языка | Шрифты |
| Tesseract | Apache | Нет | C++, C | Да | Да | Любой печатный шрифт |
| Google Cloud Vision | Проприетарное ПО, частично под Apache | Да | ? | Да | Да | Любой шрифт |
| ABBYY FineReader | Проприетарное ПО | Да | C/C++ | Да | Да | ? |
| CIB OCR | Бесплатное ПО (freeware) | Да | C++, Java, Python, Objective-C | Да | Да | Любой печатный шрифт |
| Asprise OCR SDK | Проприетарное ПО | Да | Java, C#,VB.NET, C/C++/Delphi | Да | Да | ? |
| OCR.space | GPL | Да | C# | Да | Да | Любой печатный шрифт |
| GOCR | GPL | Да | C | ? | Да | ? |

Сначала необходимо выбирать из open-source и freeware продуктов с поддержкой русского языка т.к. требуется минимизация расходов на обслуживание системы, получаем таблицу 3 ниже:

Таблица 3 – Сравнение библиотек оптического распознавания c поддержкой русского языка

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Имя | Лицензия | Онлайн | Язык реализации | SDK | Поддержка русского языка | Шрифты |
| Tesseract | Apache | Нет | C++, C | Да | Да | Любой печатный шрифт |
| CIB OCR | Бесплатное ПО (freeware) | Да | C++, Java, Python, Objective-C | Да | Да | Любой печатный шрифт |
| OCR.space | GPL | Да | C# | Да | Да | Любой печатный шрифт |
| GOCR | GPL | Да | C | ? | Да | ? |

Теперь желательно, чтоб решение было автономным без обращения к API стороннего сервиса, получаем единственное поле в таблице 4

Таблица 4 – Сравнение библиотек автономного оптического распознавания c поддержкой русского языка

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Имя | Лицензия | Онлайн | Язык реализации | SDK | Поддержка русского языка | Шрифты |
| Tesseract | Apache | Нет | C++, C | Да | Да | Любой печатный шрифт |

Следовательно, наш выбор останавливается на автономном Tesseract от Google с лицензией Apache, под Python реализован в качестве модуля pytesseract.

Наконец необходимо выбрать технологию виртуализации для обеспечения модульности всего решения. На 2019 год наибольшим спросом пользуются две: Docker и CoreOS Rocket (rkt). В целом технологии очень похожи между собой, но есть небольшие различия (таблица 5 ниже)

Таблица 5 – Сравнение Docker и RKT

|  |  |  |
| --- | --- | --- |
| Функция | Rkt | Docker |
| Запуск образов Docker | Да | Да |
| Подпись образа | Проверка подписей по умолчанию | Клиентоориентированная; проверка подписи в демоне Docker |
| Права доступа | Получение, подтверждение, проверка подписи на непривилегированного пользователя | Все операции выполняются демоном Docker как суперпользователем |
| Съемная изоляция | Несколько сред изоляции stage1. От chroot до cgroups и KVM | Изоляция только с точки зрения параметров демона Docker для сетевого моста или полного привилегированного режима |
| Создание образа | Инструмент для сборки контейнеров на основе сценариев оболочки, использующей похожие инструменты Unix | Сборка определена в Dockerfile, собрана демоном Docker (как суперпользователь) |
| Распределение контейнеров | Контейнерные образы - это набор файлов в архиве, обслуживаемый обычным HTTPS. DNS-обнаружение пользовательских пространств имён и подписей | Docker-реестр. Ограниченное пространство имен по умолчанию (docker.com) |

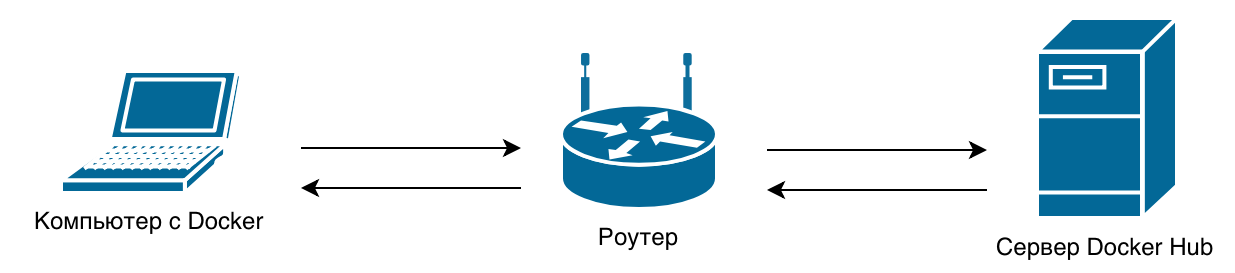
Из-за этих различий и формируются следующие преимущества и недостатки данных систем:

Преимущества Docker:

* Существует docker hub – публичный репозиторий, откуда каждый может скачать или загрузить своё изображение docker image;
* Больше методов в REST API и SDK по сравнению с CoreOS;
* Большая распространенность в сообществе разработчиков;

Недостатки Docker:

* Теоретически злоумышленник может загрузить свой собственный docker image с backshell’ом на docker hub и получить доступ к каким-либо вашим данным. На практике же вероятность крайне мала т.к. требуется серьёзная обфускация исходного кода shell’а и использование пользовательского репозитория, что также не очень частое явление;
* Теоретически возможна MITM (man in the middle) атака в процессе загрузки контейнеров на локальный компьютер (рисунки 1-2 ниже)

Рисунок 1 – Обычный роутинг пакетов без применения MITM-атаки

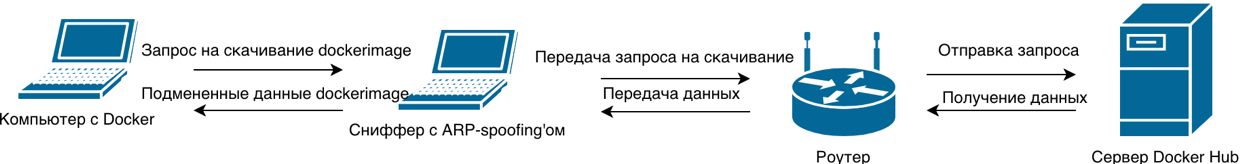


Рисунок 2 – Модифицированный роутинг пакетов c MITM-атакой

* Демон docker (docker daemon) запускается по умолчанию под суперпользователем root и создает новые контейнеры как подпроцессы демона (рисунок 3). По данной архитектуре возможен вариант, что из-за плохой изоляции контейнера и уязвимости в docker, злоумышленник может получить root-доступ ко всему серверу под пользователем docker. Для предотвращения подобных случаев используются твики для улучшения безопасности наподобие AppArmor или SELinux.

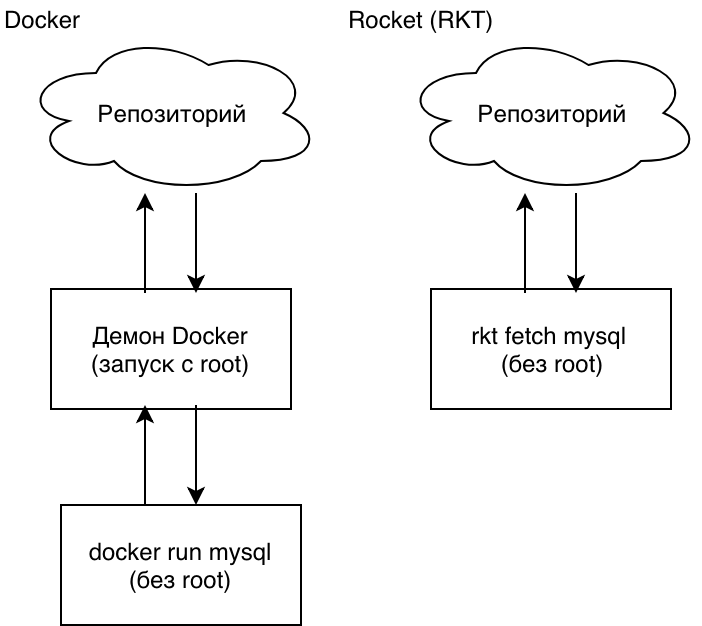


Рисунок 3 – Сравнение архитектуры Docker и Rocket

Преимущества RKT:

* Проверка подписей контейнеров реализована по умолчанию, как следствие, невозможно использовать MITM атаку для подмены изображений контейнера;
* Нет доступа к суперпользователю. Если изоляция контейнера нарушена, то получить доступ к root гораздо тяжелее;
* Возможность использования команды rkt torrent pull для скачивания rkt-контейнеров с помощью torrent, нет необходимости регистрироваться для того, чтоб поделиться контейнером с кем-либо ещё;
* Все контейнеры размещены на Github;

Недостатки RKT:

* Относительно меньшая вовлеченность сообщества разработчиков;
* Субъективно более тяжелая настройка под DevOps с использованием Kubernetes;

Если во всех предыдущих сравнениях все было достаточно очевидно, то здесь нет однозначного ответа на то, что именно лучше использовать. Я выбрал Docker из-за того, что у меня уже был относительный опыт разработки и развертки масштабируемых решений на нём.

Далее необходимо выбрать фреймворк, на базе которого будет производиться разработка web-клиента на PHP для текущего решения.

Фреймворки очень сильно ускоряют процесс разработки ПО, помогают писать и структурировать код, делать его лёгким и хорошо масштабируемым. Таким образом, нельзя отрицать тот факт, что фреймворк PHP является единственным способом чтобы успешно реализовать развитие проекта. Давайте рассмотрим самые популярные из них:

* Laravel – с 2015 года признан как лучший фреймворк для разработки web-приложений на PHP, является достаточно новым (создан в 2011 году), но он уже имеет внушительную экосистему и поддержку сообщества разработчиков. Также расширение имеет отличную документацию и множество функций, целью которых является обеспечение быстрого процесса разработки. У Laravel существует свой собственный механизм шаблонов Blade с простым и понятным синтаксисом, который делает его отличным дополнением для выполнения общих, таких как: аутентификация, сессии, анализ очереди ,RESTful маршрутизации и т.п. Laravel имеет встроенную среду программирования Homestead, которая базируется на Vagrant (https://github.com/laravel/homestead);
* Symfony – этот фреймворк, состоящий из набора нескольких компонентов, также поддерживает паттерны MVC (Model, View, Controller) как и в случае с Laravel. Предлагает быструю разработку и управление в web, помогает выполнять легко все рутинные задачи по web-программированию. Одним из главных его преимуществ является поддержка нескольких баз данных и горячая миграция между MySQL, PostgreSQL, SQLite или любой другой PDO-совместимой БД за счёт использования ORM Doctrine. Symfony состоит из PHP-библиотек, которые могут быть повторно использованы и развёрнуты для различных задач: создание форм, объектов и шаблонов конфигурации и аутентификации, а также многие другие задачи. Любой компонент может быть доустановлен с помощью пакетного менеджера composer’а, в качестве шаблонизатора выступает twig;
* Falcon – низкоуровневый фреймворк, считается самым быстрым из данного списка т.к. его исходный код написан на ЯП C/C++, что позволяет достичь высокой оптимизации и производительности. Все функциональные возможности находятся в PHP-классах, которые могут быть использованы для совершенно разносторонних задач. Есть ряд особенных функций: многоцелевая автозагрузка, собственная система безопасности, управление ресурсами, переводы, кеширование и т.д. Документация по Falcon достаточно обширная, субъективно более высокий порог вхождения в разработку;
* CodeIgniter – универсальный фреймворк, который широко используется при каких-либо конфликтах версий PHP и одинаково работает на всех платформах, не полностью основан на модели MVC, хорошо подходит для новичков. Идея решения состояла в том, чтоб сделать простую структуру с весом до 2мб с возможностью добавления каких-либо плагинов, если необходимость в более сложной функциональности будет присутствовать. Важная особенность: в задачах, связанных с базами данных, CodeIgniter справляется быстрее всех по сравнению с другими фреймворками;

Как и в случае выбора технологии виртуализации, однозначного выбора нет, но мой выбор пал на Symfony по ряду следующих причин:

* Легкость миграции за счёт использования Doctrine ORM т.к. в случае каких-либо проблем с MySQL под Docker можно легко перенести структуру БД на тот же PostgreSQL;
* Огромная база плагинов и модулей с простой установкой при помощи composer;
* Модель MVC. Отдельно можно настроить routes в файле .yaml, назначить контроллеры обработки, которые вызывают наследованные шаблоны twig;
* Большое сообщество разработчиков;
* Хорошая совместимость между разными версиями;

В завершение необходим выбор web-сервера для интерпретатора php для нашего Symfony-клиента. Выбор достаточно малый: используем либо Apache, либо Nginx.

Apache является свободным HTTP-сервером с открытым исходным кодом для UNIX-подобных ОС и Windows. Был смоделирован как безопасный, эффективный и хорошо масштабируемый сервер, в котором отлично реализованы службы HTTP с учётом их базовых стандартов. До 2000-х годов являлся стандартом веб-сервера на Linux.

Nginx также является свободным высокопроизводительным HTTP-сервером, а также включает в себя почтовые протоколы POP3 и IMAP. Также, как и Apache запускается на разных UNIX-подобных ОС и архитектурах, начиная от обычных i386 и x86-64, заканчивая мобильными ARM, мейнфреймами IBM System Z и серверами Apple на PowerPC до из перехода на Intel.

Был выбран именно nginx по следующим причинам:

* Легковесность. Nginx является относительно лёгким web-сервером т.к. по сравнению с Apache не имеет достаточно многогранного спектра функций для приложений, поэтому в целом использование памяти и других ресурсов более эффективно. Как следствие, решение подходит для систем с ограниченными физическими ресурсами;
* Применение асинхронной модели, управляемой событиями. Nginx построен по модели параллелизма, как противоположность потокоориентированной архитектуре Apache (по типу создания процесса на каждое соединение и отдельного потока на каждую модель соединения). По модели Engine X есть демон/мастер-процесс, который создает подпроцессы по типу worker’ов и helper’ов. Worker’ы требуются для приёма HTTP-соединений, а также записи и чтения данных на диске. Helper’ы нужны для синхронизации операций кэшированных данных;
* Встроенный прокси по умолчанию. Существует отличная возможность настройки и использования встроенного прокси: он получает HTTP-запрос от клиента и передаёт его далее с использованием каких-либо сконфигурированных протоколов. Также есть встроенная возможность замены header’ов от клиентов и настройка буфера ответа от прокси-сервера;
* Хорошая производительность для статического контента. Все статические файлы (по типу HTML/CSS/JS), хранящиеся на диске сервера, могут быть закешированы. За счёт этого время ответа сервера для клиента (скрипт, web-браузер) будет намного меньше;

В итоге был выбран следующий стек технологий:

* Библиотека компьютерного зрения OpenCV для препроцессинга изображения и определения границ каждой ячейки таблицы на стороне backend worker’а;
* Офлайн библиотека оптического распознавания символов Google Tesseract для непосредственного перевода преобразованной картинки в строку;
* Docker в качестве ПО для менеджмента и псевдовиртуализации контейнеров с возможностью масштабирования архитектуры;
* Фреймворк Symfony 4 для использования на frontend для PHP-клиента;
* Nginx в качестве веб-сервера для интерпретатора php-клиента.

# Заключение

Во время прохождения производственной преддипломной практики было успешно проведено предпроектное исследование предметной области для выпускной квалификационной работы.

Выполнены следующие работы:

* Проведен анализ и обобщение научно-технической информации по теме ВКР;
* Проведен анализ актуальности работы и рассмотрены существующие аналоги;
* Подобран фактический материал по теме ВКР с учетом профессиональных модулей и изучение его;
* Подготовлен план пояснительной записки к ВКР;
* Подобран набор программных средств и языков программирования для реализации практической части ВКР;
* Проведён анализ и выбор технологий, используемых в решении.

Список литературы

1. MODX Documentation – [Электронный ресурс]. – Режим доступа: https://docs.modx.com/;
2. Защищаем MODX Revolution – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/335148/;
3. Ресурсы в MODx Revolution, MODx API – [Электронный ресурс]. – Режим доступа: http://www.webapplex.ru/resursyi-v-modx-revolution,-modx-api;
4. Как установить чат в ModX CMS – [Электронный ресурс]. – Режим доступа: https://chatra.io/ru/help/cms/modx/;
5. Плюсы и минусы CMS MODX – [Электронный ресурс]. – Режим доступа: https://contextonline.ru/blog/plyusy-minusy-cms-modx/;
6. Установка MODX Revolution – [Электронный ресурс]. – Режим доступа: https://modxinfo.ru/modx-setup;
7. Как перенести сайт на ModX на другой хостинг – [Электронный ресурс]. – Режим доступа: https://www.reg.ru/support/hosting-i-servery/perenos\_saitov/kak-perenesti-sayt-na-modx-na-drugoy-khosting.
8. [ScanDoc] предобработка сканов – [Электронный ресурс]. – Режим доступа: https://habr.com/company/tinkoff/blog/278073/;
9. OpenCV шаг за шагом. Обработка изображения - морфологические преобразования 2 – [Электронный ресурс]. – Режим доступа: http://robocraft.ru/blog/computervision/327.html;
10. Алгоритмы выделения контуров изображений – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/114452/;
11. Базовые алгоритмы нахождения кратчайших путей во взвешенных графах – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/119158/
12. Введение в OpenCV применительно к распознаванию линий дорожной разметки – [Электронный ресурс]. – Режим доступа: https://habr.com/company/newprolab/blog/328422/
13. Декораторы в Python – [Электронный ресурс]. – Режим доступа: https://pythonworld.ru/osnovy/dekoratory.html
14. Инструкция по работе с TensorFlow Object Detection API – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/nixsolutions/blog/422353/
15. Docker-compose.yml for mysql and phpmyadmin – [Электронный ресурс]. – Режим доступа: https://gotechnies.com/docker-compose-yml-mysql-phpmyadmin/
16. Get started with Docker Compose – [Электронный ресурс]. – Режим доступа: https://docs.docker.com/compose/gettingstarted/
17. How to install Docker and deploy a LAMP Stack – [Электронный ресурс]. – Режим доступа: https://www.linode.com/docs/applications/containers/how-to-install-docker-and-deploy-a-lamp-stack/
18. Использование NoSQL Redis в качестве основного хранилища – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/178525/
19. Docker + Flask. Simple Tutorial– [Электронный ресурс]. – Режим доступа: https://medium.com/@mtngt/docker-flask-a-simple-tutorial-bbcb2f4110b5
20. Транзакции, блокировки и многопользовательский доступ к данным. – [Электронный ресурс]. – Режим доступа: http://www.mstu.edu.ru/study/materials/zelenkov/ch\_4\_9.html
21. Основы правильного проектирования баз данных в веб-разработке – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/115777/
22. Using Transactions in Python Programs – [Электронный ресурс]. – Режим доступа: https://www.oreilly.com/library/view/mysql-cookbook-2nd/059652708X/ch15s08.html
23. Use Commit and Rollback to Manage MySQL Transactions in Python – [Электронный ресурс]. – Режим доступа: https://pynative.com/python-mysql-transaction-management-using-commit-rollback/
24. Транзакции в mysql и SELECT FOR UPDATE – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/238119/
25. Блокировки и уровни изоляции транзакций InnoDB в MySQL – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/238513/
26. GraphQL. Введение. – [Электронный ресурс]. – Режим доступа: https://habr.com/post/326986/
27. Проектирование Web API в 7 шагов – [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/geekfamily/blog/256495/

# Приложение A

[ВВЕДЕНИЕ 4](#_Toc8486271)

[1 ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ 6](#_Toc8486272)

[1.1 Предпроектное исследование предметной области 6](#_Toc8486273)

[1.2 Среда и язык программирования 10](#_Toc8486274)

[1.3 Анализ технологий, используемых для реализации программы 10](#_Toc8486275)

[1.4 Анализ требований и определение спецификации ПО 25](#_Toc8486276)

[2 СПЕЦИАЛЬНЫЙ РАЗДЕЛ 25](#_Toc8486277)

[2.1 Проектирование программного обеспечения 25](#_Toc8486278)

[2.1.1 Проектирование общей архитектуры решения в Docker и БД 26](#_Toc8486279)

[2.1.2 Проектирование Flask REST API 32](#_Toc8486280)

[2.1.3 Проектирование worker’ов: алгоритмы распознавания таблиц, особенности миграции решения в контейнеры Docker 40](#_Toc8486281)

[2.1.4 Проектирование web-клиента с использованием фреймворка Symfony 4 78](#_Toc8486282)

[2.2 Разработка пользовательских интерфейсов ПО 84](#_Toc8486283)

[2.2.1 Пользовательский интерфейс web-клиента 84](#_Toc8486284)

[2.2.2 Пользовательский интерфейс контейнера RQ Dashboard 93](#_Toc8486285)

[3 ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ 96](#_Toc8486286)

[3.1 Технологии, применяемые при разработке ПО 96](#_Toc8486287)

[3.1.1 Общая архитектура 96](#_Toc8486288)

[3.1.2 Worker’ы 96](#_Toc8486289)

[3.1.3 Web-клиент 98](#_Toc8486290)

[3.2 Тестирование и отладка программного обеспечения 99](#_Toc8486291)

[4 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ ПРОГРАММЫ 100](#_Toc8486292)

[4.1 Руководство программиста 100](#_Toc8486293)

[4.1.1 Общая информация о системе 100](#_Toc8486294)

[4.1.2 Работа с Flask API 103](#_Toc8486295)

[4.1.3 Работа с Symfony API 111](#_Toc8486296)

[4.1.4 Модули и методы worker’ов 116](#_Toc8486297)

[4.2 Руководство пользователя 118](#_Toc8486298)

[ЗАКЛЮЧЕНИЕ 122](#_Toc8486299)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 123](#_Toc8486300)

[ПРИЛОЖЕНИЕ А 126](#_Toc8486301)

[ПРИЛОЖЕНИЕ Б 126](#_Toc8486302)