**МИНприложенияТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«КАЛУЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**им. К. Э. ЦИОЛКОВСКОГО»**

Физико-технологический институт

Кафедра информатики и информационных технологий

Направление подготовки 09.03.02 Информационные системы и технологии

*ДИПЛОМНАЯ РАБОТА*

по дисциплине «Проектирование информационных систем»

на тему: Разработка генератора электронного документооборота «EDOCript»

студента 4 курса очной формы обучения

**Андреева Павла Олеговича**

научный руководитель

Виноградская М.Ю.

**г. Калуга, 2020 г.**

Оглавление

[Введение 3](#_Toc40118166)

[Глава 1. Основы проектирования и разработки приложения 5](#_Toc40118167)

[1.1 Основные понятия 5](#_Toc40118168)

[1.2 Назначения и область применения разработки 9](#_Toc40118169)

[1.3 Анализ существующих разработок 12](#_Toc40118170)

[Глава 2. Проектирование генератора электронного документооборота «EDOCript» 12](#_Toc40118171)

[2.1 Выбор методов и средств проектирования приложения, установление требований к проектируемому приложению 12](#_Toc40118172)

[2.2 Проектирование модели и архитектуры построения приложения 18](#_Toc40118173)

[2.3 Проектирование модели данных и базы данных приложения 21](#_Toc40118174)

[2.4 Проектирование пользовательского интерфейса 27](#_Toc40118175)

[Глава 3. Программная реализация генератора электронного документооборота «EDOCript» 30](#_Toc40118176)

[3.1 Обоснование выбора состава программных средств реализации 30](#_Toc40118177)

[3.2 Описание пользовательского интерфейса и инструкций пользователю 35](#_Toc40118178)

[3.3 Руководство разработчика 47](#_Toc40118179)

[Заключение 48](#_Toc40118180)

[Список использованных источников 48](#_Toc40118181)

[ПРИЛОЖЕНИЕ А 48](#_Toc40118182)

# Введение

В современной организации системы электронного документооборота становятся обязательным элементом ИТ-инфраструктуры. С их помощью повышают эффективность деятельности коммерческие компании и промышленные предприятия, а в государственных учреждениях на базе технологий электронного документооборота решаются задачи внутреннего управления, межведомственного взаимодействия и взаимодействия с населением.

Изначально системы этого класса рассматривались лишь как инструмент автоматизации задач классического делопроизводства, но со временем стали охватывать все более широкий спектр задач. Сегодня разработчики системы электронного документооборота ориентируют свои продукты на работу не только с корреспонденцией и организационно-распорядительными документами, но и с различными внутренними документами (договорами, нормативной, справочной и проектной документацией, документами по кадровой деятельности и др.). системы электронного документооборота также используются для решения прикладных задач, в которых важной составляющей является работа с электронными документами: управление взаимодействием с клиентами, обработка обращений граждан, автоматизация работы сервисной службы, организация проектного документооборота и др. Фактически системой электронного документооборота называют любую информационную систему, обеспечивающую работу с электронными документами.

Так как современные компании, использующие такие системы, имеют очень большой поток информации, разработка подобных систем становится все сложнее и сложнее. В связи с этим, требуются системы тестов, для выявления ошибок в работе информационной системы.

Первые программные системы разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны. Тестирование таких продуктов проводилось строго формализовано с записью всех тестовых процедур, тестовых данных, полученных результатов. Тестирование выделялось в отдельный процесс, который начинался после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом.

Из года в год объемы информационных систем растут, также, как и необходимость тестировочных систем для них. Но в связи с масштабами информационных систем и их сложности реализации, также усложняются системы тестирования.

Для упрощения процесса тестирования требуется организация кейсов с данными различных тестов. Порой количество и наполнение таких кейсов имеет весьма сложную структуру для составления их руками человека.

Основными сложностями составления кейсов является: строгость формирования документа, соответствующего всем стандартам и ГОСТам, нормализующим оформление, а также наличие грамотно сформированной электронной цифровой подписи, которую подделать довольно сложно, ибо одним из важнейших направлений развития российского законодательства и правоприменительной практики в настоящее время является правовое регулирование отношений в области электронного документооборота и придания юридической силы электронным документам.

Принятие ФЗ «Об электронной цифровой подписи» узаконило использование электронной цифровой подписи как аналога собственноручно. Электронная цифровая подпись является необходимым атрибутом электронного документа, который используется в качестве универсального способа подтверждения факта волеизъявления лица. Она представляет собой результат криптографического преобразования информации в виде последовательности нескольких цифр и является средством защиты информации, обеспечивающим возможность контроля целостности и подтверждения подлинности электронных документов. Чтобы документ, «хранимый, обрабатываемый и передаваемый с помощью автоматизированных информационных и телекоммуникационных систем», приобрел юридическую силу, достаточно удостоверить его электронной подписью.

Поэтому, создаются электронные подписи в специальном программном обеспечении, разработанные исключительно для создания валидных подписей.

Для автоматизации процесса составления таких кейсов можно использовать различные генераторы данных, тем самым значительно упрощая работу тестировщика.

Целью дипломной работы является автоматизация деятельности тестировщика в виде реализации генератора пакетов данных эмулирующий процесс обмена электронными документами, с использованием электронной подписи, в системе электронного документооборота.

# Глава 1. Основы проектирования и разработки приложения

# Основные понятия

Электронная цифровая подпись (ЭЦП) - (англ. digital signature) - последовательность символов, полученная в результате криптографического преобразования исходной информации (криптографическая защита информации основана на функциональном преобразовании информации и используется с целью скрыть содержание информации), которая позволяет подтверждать целостность и неизменность этой информации, а также ее авторство.

Системы электронной цифровой подписи (далее - ЭЦП) являются разделом криптографии.

В настоящее время ЭЦП используется в основном для аутентификации автора (создателя) информации и для доказательства (проверки) того факта, что подписанное сообщение или данные не были модифицированы при передаче информации в компьютерных сетях.

Механизм электронной цифровой подписи (ЭЦП) возник как побочный эффект криптографии с открытым ключом. Поэтому, характерное для систем с открытым ключом разделение ключа на 2 части - секретную и несекретную, позволяет реализовать возможность проверки подлинности без возможности подписать другой документ.

XML (англ. eXtensible Markup Language) — расширяемый язык разметки. Рекомендован Консорциумом Всемирной паутины (W3C). Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Расширение XML — это конкретная грамматика, созданная на базе XML и представленная словарём тегов и их атрибутов, а также набором правил, определяющих какие атрибуты и элементы могут входить в состав других элементов. Сочетание простого формального синтаксиса, удобства для человека, расширяемости, а также базирование на кодировках Юникод для представления содержания документов привело к широкому использованию как, собственно, XML, так и множества производных специализированных языков на базе XML в самых разнообразных программных средствах.

XPath (XML Path Language) — язык запросов к элементам XML-документа. Разработан для организации доступа к частям документа XML в файлах трансформации XSLT и является стандартом консорциума W3C. XPath призван реализовать навигацию по DOM в XML. В XPath используется компактный синтаксис, отличный от принятого в XML. В 2007 году завершилась разработка версии 2.0, которая теперь является составной частью языка XQuery 1.0. В декабре 2009 года началась разработка версии 2.1, которая использует XQuery 1.1.

На данный момент самой популярной версией является XPath 1.0. Это связано с отсутствием поддержки XPath 2.0 со стороны открытых библиотек. В частности, речь идёт о LibXML, от которой зависит поддержка языка в браузерах, с одной стороны, и поддержка со стороны серверного интерпретатора, с другой.

XSLT (eXtensible Stylesheet Language Transformations) — язык преобразования XML-документов. Спецификация XSLT входит в состав XSL и является рекомендацией W3C.

При применении таблицы стилей XSLT, состоящей из набора шаблонов, к XML-документу (исходное дерево) образуется конечное дерево, которое может быть сериализовано в виде XML-документа, XHTML-документа (только для XSLT 2.0), HTML-документа или простого текстового файла. Правила выбора (и, отчасти, преобразования) данных из исходного дерева пишутся на языке запросов XPath.

XSLT имеет множество различных применений, в основном в области веб-программирования и генерации отчётов. Одной из задач, решаемых языком XSLT, является отделение данных от их представления, как часть общей парадигмы MVC (англ. Model-view-controller). Другой стандартной задачей является преобразование XML-документов из одной XML-схемы в другую.

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Flask — фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Относится к категории так называемых микрофреймворков — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

Jinja (произносится как дзиндзя) — это шаблонизатор для языка программирования Python. Он подобен шаблонизатору Django, но предоставляет Python-подобные выражения, обеспечивая исполнение шаблонов в песочнице. Это текстовый шаблонизатор, поэтому он может быть использован для создания любого вида разметки, а также исходного кода. Лицензирован под BSD лицензией.

Шаблонизатор Jinja позволяет настраивать теги, фильтры, тесты и глобальные переменные. Также, в отличие от шаблонизатора Django, Jinja позволяет конструктору шаблонов вызывать функции с аргументами на объектах.

SQLite — компактная встраиваемая СУБД. Исходный код библиотеки передан в общественное достояние. В 2005 году проект получил награду Google-O’Reilly Open Source Awards.

JavaScript — мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией стандарта ECMAScript (стандарт ECMA-262).

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java, но при этом лёгким для использования непрограммистами. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

jQuery — набор функций JavaScript, фокусирующийся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX. Разработка jQuery ведётся командой добровольцев на пожертвования.

# Назначения и область применения разработки

В современных реалиях тестировочные системы для крупных приложения, является жизненной необходимостью в процессе разработки и поддержки приложения. Потому грамотное составление кейсов для тестов, очень важно в работе тестировщика. Разрабатываемый генератор пакетов предназначается как раз, чтобы снизить элемент человеческого фактора и самое важное ускорить работу тестировщика системы, тем самым положительно влиять на процесс разработки проекта.

Разрабатываемый программный продукт будет использован в целях генерации пакетов, представляющих собой zip-архив, сформированный по стандартам системы докуметооборота.

Под аббревиатурой EDI понимают Electronic Data Interchange или Электронный Обмен Данными. Проще говоря, EDI– это отправка и получение информации с использованием компьютерных технологий. Благодаря тому, что применение технологий EDI в бизнесе удобно и практично, этот стандарт стал широко использоваться в различных отраслях экономики и социального обслуживания. Любые стандартные деловые документы, которыми, к примеру, одна FMCG компания обменивается с другой (такие как заказ на поставку, счёт-фактура, план отгрузок, запрос о наличии товара) могут быть переданы при помощи EDI, если обе стороны провели необходимую для этого подготовку. Стандарт EDI разработан в Американском национальном институте стандартов (ANSI). Наряду с EDI существуют и другие стандарты для электронного обмена данными. Например, EDIFACT широко используется в Европе и в автомобильной промышленности. HIPPA (закон об учете и безопасности медицинского страхования) разработан специально для соответствия деятельности учреждений здравоохранения законодательству. Транслятор EDI, который Вы выбираете, должен поддерживать все стандарты.

Также желательно выполнение стандартов CMS (Cryptographic Message Syntax) для подписанных сообщений.

Стандарт CMS описывает структуру криптографических сообщений, включающих в себя защищенные данные вместе со сведениями, необходимыми для их корректного открытия или использования. Например, в сообщении размещаются защищенные данные, информация об алгоритме хеширования и подписи, времени подписи, сертификате открытого ключа, цепочке сертификации и т.д. Некоторые из указанных атрибутов носят опциональный характер, но приложение может само определить необходимость их наличия. У каждого алгоритма есть набор параметров, который должен быть согласован на обеих сторонах: для ГОСТ 34.10-2001, помимо открытого ключа, это модуль p, коэффициенты эллиптической кривой a и b и порядок циклической подгруппы точек эллиптической кривой q. И все это нужно каким-то образом передать адресату сообщения.

RSA Laboratories в серии своих стандартов криптографии с открытом ключом (PKCS) предложила решение этой проблемы путем определения синтаксиса для защищенных сообщений в следующих стандартах:

PKCS #7 «Cryptographic Message Syntax Standard»,

PKCS #10 «Certificate Request Syntax Standard».

Развитием этих стандартов стал стандарт CMS. CMS кроме определенной заголовком статьи подписи поддерживает операции шифрования, хеширования и вычисления имитовставки, в том числе и по российским алгоритмам (RFC 4490), а также множественную инкапсуляцию. Последнее означает, что сообщение формата CMS может лежать внутри другого CMS сообщения.Всего CMS поддерживает шесть типов данных:

* просто данные (data),
* данные с электронной подписью (signed data),
* упакованные данные (enveloped data),
* хешированные данные (digested data),
* зашифрованные данные (encrypted data),
* данные с проверкой подлинности (authenticated data).

Электронный документооборот, как правило происходит в передаче документов, описанных на расширяемом языке разметки XML. Для формирования шаблонов таких документов используется XSD.

XSD — это язык описания структуры XML документа. Его также называют XML Schema. При использовании XML Schema XML парсер может проверить не только правильность синтаксиса XML документа, но также его структуру, модель содержания и типы данных.

Посмотреть XSD шаблоны для документов участвующих в разработке генератора можно на сайте Ассоциации «РОСЭУ».

Документооборот может осуществляться между тремя видами участников:

* Абонент – налогоплательщик (юридическое лицо или индивидуальный предприниматель) или его уполномоченный представитель
* Налоговый орган – Налоговый орган ФНС России
* Спецоператор – оператор электронного документооборота

У каждого участника документооборота должен быть GUID (Globally Unique Identifier), ИНН (Идентификационный номер налогоплательщика), а также если он является организацией, то ему должен быть присвоен КПП (Код постановки на учет).

# Анализ существующих разработок

# Глава 2. Проектирование генератора электронного документооборота «EDOCript»

# 2.1 Выбор методов и средств проектирования приложения, установление требований к проектируемому приложению

Проектирование приложения представляет сложный многоступенчатый вид деятельности, без научной организации которого немыслимо создание и использование современных сложных приложений, в том числе в образовании, предпринимательстве, менеджменте и других областях жизнедеятельности общества. Наряду с получением необходимых для этого теоретических знаний проектировщику приложения требуется обрести устойчивые практические навыки этого вида деятельности.

Главной особенностью проектирования является работа с еще не существующим объектом. В этом отличие проектирования от моделирования, где объект не может не существовать.

Проектирование приложения охватывает три основные области:

* Проектирование объектов данных, которые будут реализованы в базе данных;
* Проектирование программ, экранных форм, отчетов, которые будут обеспечивать выполнение запросов к данным;
* Учет конкретной среды или технологии, а именно: топологии сети, конфигурации аппаратных средств, используемой архитектуры (файл-сервер или клиент-сервер), параллельной обработки, распределенной обработки данных и т.п.

Проектирование приложения всегда начинается с определения цели проекта. В общем виде цель проекта можно определить как решение ряда взаимосвязанных задач, включающих в себя обеспечение на момент запуска системы и в течение всего времени ее эксплуатации:

* Требуемой функциональности системы и уровня ее адаптивности к изменяющимся условиям функционирования;
* Требуемой пропускной способности системы;
* Требуемого времени реакции системы на запрос;
* Безотказной работы системы;
* Необходимого уровня безопасности;
* Простоты эксплуатации и поддержки системы

Отдельно стоит заметить, что при разработке приложения стоит опираться на модели жизненного цикла приложения. Существуют несколько распространённых моделей: каскадная инкрементная и спиральная.

Каскадная модель характеризуется тем, что данная модель применяется при разработке информационных систем, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования.

Достоинства модели:

* На каждой стадии формируется законченный набор документации, программного и аппаратного обеспечения, отвечающий критериям полноты и согласованности;
* Выполняемые в четкой последовательности стадии позволяют уверенно планировать сроки выполнения работ и соответствующие ресурсы (денежные, материальные и людские).

Недостатки модели:

* Реальный процесс разработки информационной системы редко полностью укладывается в такую жесткую схему. Особенно это относится к разработке нетиповых и новаторских систем;
* Основана на точной формулировке исходных требований к информационной системе. Реально в начале проекта требования заказчика определены лишь частично;
* Основной недостаток – результаты разработки доступны заказчику только в конце проекта.

В случае неточного изложения требований или их изменения в течение длительного периода создания приложения заказчик получает систему, не удовлетворяющую его потребностям.

Инкрементная стратегия (англ. increment – увеличение, приращение) подразумевает разработку информационной системы с линейной последовательностью стадий, но в несколько инкрементов (версий), т. е. с запланированным улучшением продукта.

В начале работы над проектом определяются все основные требования к системе, после чего выполняется ее разработка в виде последовательности версий. При этом каждая версия является законченным и работоспособным продуктом. Первая версия реализует часть запланированных возможностей, следующая версия реализует дополнительные возможности и т. д., пока не будет получена полная система.

Данная модель жизненного цикла характерна при разработке сложных и комплексных систем, для которых имеется четкое видение (как со стороны заказчика, так и со стороны разработчика) того, что собой должен представлять конечный результат (приложение). Разработка версиями ведется в силу разного рода причин:

* Отсутствия у заказчика возможности сразу профинансировать весь дорогостоящий проект;
* Отсутствия у разработчика необходимых ресурсов для реализации сложного проекта в сжатые сроки;
* Требований поэтапного внедрения и освоения продукта конечными пользователями.

Внедрение всей системы сразу может вызвать у ее пользователей неприятие и только «затормозить» процесс перехода на новые технологии. Образно говоря, они могут просто «не переварить большой кусок, поэтому его надо измельчить и давать по частям».

Достоинства и недостатки этой стратегии такие же, как и у классической. Но в отличие от классической стратегии заказчик может раньше увидеть результаты. Уже по результатам разработки и внедрения первой версии он может незначительно изменить требования к разработке, отказаться от нее или предложить разработку более совершенного продукта с заключением нового договора.

Спиральная стратегия (эволюционная или итерационная модель, автор Барри Боэм, 1988 г.) подразумевает разработку в виде последовательности версий, но в начале проекта определены не все требования. Требования уточняются в результате разработки версий.

Данная модель жизненного цикла характерна при разработке новаторских (нетиповых) систем. В начале работы над проектом у заказчика и разработчика нет четкого видения итогового продукта (требования не могут быть четко определены) или стопроцентной уверенности в успешной реализации проекта (риски очень велики). В связи с этим принимается решение разработки системы по частям с возможностью изменения требований или отказа от ее дальнейшего развития.

Достоинства модели:

* Позволяет быстрее показать пользователям системы работоспособный продукт, тем самым, активизируя процесс уточнения и дополнения требований;
* Допускает изменение требований при разработке информационной системы, что характерно для большинства разработок, в том числе и типовых;
* Обеспечивает большую гибкость в управлении проектом;
* Позволяет получить более надежную и устойчивую систему. По мере развития системы ошибки и слабые места обнаруживаются и исправляются на каждой итерации;
* Позволяет совершенствовать процесс разработки – анализ, проводимый в каждой итерации, позволяет проводить оценку того, что должно быть изменено в организации разработки, и улучшить ее на следующей итерации;
* Уменьшаются риски заказчика. Заказчик может с минимальными для себя финансовыми потерями завершить развитие неперспективного проекта.

Недостатки модели:

* Увеличивается неопределенность у разработчика в перспективах развития проекта. Этот недостаток вытекает из предыдущего достоинства модели;
* Затруднены операции временного и ресурсного планирования всего проекта в целом.

Для решения этой проблемы необходимо ввести временные ограничения на каждую из стадий жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа выполнена. План составляется на основе статистических данных, полученных в предыдущих проектах и личного опыта разработчиков.

Из-за того, что наша разрабатываемая информационная система является простой и по своим масштабам довольно небольшой, было решено выбрать урезанную каскадную модель жизненного цикла данной приложения. Из изменений модели – совмещение этапов системного анализа и анализа требований, также совмещение этапов проектирования, кодирования и тестирования. На выходе получаем всего 3 этапа-блока, которые можно назвать как: Оценка требований и подбор инструментов для реализации, реализация и устранение критичных неполадок, развертывание системы и её поддержка.

Основными требованиями для приложения являются:

* Адекватная скорость работы
* Удобный понятный интерфейс
* Расширяемость системы
* Корректная работа с AstralToolBox
* Быстрая работа с исходными данными

Требования к функционалу приложения:

* Загрузка сертификатов пользователя для заполнения полей формы сайта
* Генерация пакетов документов по заданным параметрам в форме сайта
* Подпись сгенерированных документов электронной цифровой подписью полученной из сертификатов
* Получение архива со сгенерированными пакетами данных

Требования к системе пользователя:

* Наличие или возможность установки браузера «Google Chrome»
* Наличие монитора
* Наличие клавиатуры и мыши для навигации
* Минимальный объем ОЗУ 2Гб
* Наличие установленного ПО «Python 3.6.5» (и выше)
* Наличие установленного ПО «ViPNet CSP»
* Наличие установленного ПО «AstralToolBox»
* Наличие сертификатов предполагаемого участника документооборота для электронной цифровой подписи

# 2.2 Проектирование модели и архитектуры построения приложения

При написании любой программы, особенно, если для ее корректной работы требуется большое количество модулей, также стоит заранее проработать построение приложения.

В процессе разработки нередко приходится переделывать некоторые модули программы, и если модель приложение предусматривает такой расклад событий, то остальные модули сохранят свою работоспобность.

Чтобы это работало желательно придерживаться семи основных принципов:

1. Эффективность системы – программы должна решать поставленные задачи с максимально корректно в минимальные сроки, при этом не забывая об надежности, производительности, безопасностью и растущей со временем нагрузкой.
2. Гибкость системы – со временем приходится менять некоторые требования или добавлять новые. Задача разработчика получить такую модель приложения, чтобы вносить эти изменения с минимальными затратами по переписыванию кода. Для этого архитектура построения программы должна предполагать возможное дальнейшее развитие проекта.
3. Расширяемость системы – в начальном этапе разработки имеет смысл использовать принцип добавлять только самый необходимый функционал, но при этом не забывать про возможность наращивания, того самого, функционала не самой первой необходимости. То есть это предполагает расширение программы преимущественно написанием дополнительного кода, а не переписыванием старого.
4. Масштабируемость процесса разработки – желательно иметь наличие возможности сокращения процесса разработки проекта, посредством добавления новых людей. Модель должна предполагать распараллеливание задач программы по модулям, которые в свою очередь можно распределить между сотрудниками.
5. Тестируемость – адекватная возможность тестирования проекта, позволяет вовремя отловить наличие ошибок, тем самым позволяя конкретно выполнить принципы, описанные в пунктах выше.  
   Для этого в процессе работы обычно используют GIT (Распределенная система управления версиями). В структуре проекта Git используются несколько репозиториев или веток, один из которых является истинно верным. Обычно главная ветка называется origin или master. А также множество веток для каждой поставленной задачи. Тем самым тестировщик может удобно и быстро загрузить нужную ветку с задачей и составить по ней отчет, которой также можно отобразить в Git.
6. Возможность повторного использования – предполагает, что есть вероятность миграции людей от одного проекта к другому и пришедшему новому человеку будет довольно сложно вникнуть в проект. Потому желательно использовать наименее экзотические методы и средства реализации, естественного не в ущерб адекватности.

Практически все вышеперечисленные задачи можно решить, придерживаясь постулатов ООП (Объектно ориентированного программирования).

Главное

Инкапсулируйте все, что может изменяться;

Уделяйте больше внимания интерфейсам, а не их реализациям;

Каждый класс в вашем приложении должен иметь только одно назначение;

Классы — это их поведение и функциональность.

Базовые принципы ООП

Абстракция — отделение концепции от ее экземпляра;

Полиморфизм — реализация задач одной и той же идеи разными способами;

Наследование — способность объекта или класса базироваться на другом объекте или классе. Это главный механизм для повторного использования кода. Наследственное отношение классов четко определяет их иерархию;

Инкапсуляция — размещение одного объекта или класса внутри другого для разграничения доступа к ним.

Используйте следующее вместе с наследованием

Делегация — перепоручение задачи от внешнего объекта внутреннему;

Композиция — включение объектом-контейнером объекта-содержимого и управление его поведением; последний не может существовать вне первого;

Агрегация — включение объектом-контейнером ссылки на объект-содержимое; при уничтожении первого последний продолжает существование.

Не повторяйся (Don’t repeat yourself — DRY)

Избегайте повторного написания кода, вынося в абстракции часто используемые задачи и данные. Каждая часть вашего кода или информации должна находиться в единственном числе в единственном доступном месте.

Принцип единственной обязанности

Для каждого класса должно быть определено единственное назначение. Все ресурсы, необходимые для его осуществления, должны быть инкапсулированы в этот класс и подчинены только этой задаче.

Принцип открытости/закрытости

Программные сущности должны быть открыты для расширения, но закрыты для изменений.

Принцип подстановки Барбары Лисков

Методы, использующие некий тип, должны иметь возможность использовать его подтипы, не зная об этом.

Принцип разделения интерфейсов

Предпочтительнее разделять интерфейсы на более мелкие тематические, чтобы реализующие их классы не были вынуждены определять методы, которые непосредственно в них не используются.

Принцип инверсии зависимостей

Система должна конструироваться на основе абстракций “сверху вниз”: не абстракции должны формироваться на основе деталей, а детали должны формироваться на основе абстракций.

В данном приложении была выбрана модульная архитектура. Ввиду ее соответствия всем нормам современных требований к написанию программ, имеющих сложную конструкцию.

Главным принципом модульной архитектуры является глобальная декомпозиция, которая исполняет важнейшую роль снижения сложности работы проекта.

Иерархическая декомпозиция предполагает собой детализацию высокоуровневых процессов проекта на более конкретные точные процессы, обеспечивая возможность представления работы процессов высокого уровня с помощью нижележащих процессов.

# 2.3 Проектирование модели данных и базы данных приложения

Эволюция Модели данных

Начальный этап

На начальном этапе первоначальные задачи моделирования данных могут выполняться совместно с разработкой опытных прототипов как часть задач "Выполнить архитектурный синтез". В проектах, в которых база данных уже существует, проектировщик базы данных может выполнить обратную разработку существующей базы данных, чтобы разработать начальную Физическую модель данных на основе структуры существующей базы данных. Дополнительная информация приведена в разделе. Элементы Физической модели данных при необходимости можно преобразовать в элементы Модели проектирования, чтобы поддержать задачи по разработке опытных прототипов.

Этап уточнения

Этап уточнения предназначен для исключения технического риска и создания устойчивой (контрольной версии) архитектуры системы. В крупномасштабных системах низкая производительность, обусловленная неудачно спроектированной Моделью данных, является основной архитектурной проблемой. По этой причине, и моделирование данных, и разработка архитектурного прототипа, позволяющие оценить будущую производительность базы данных, существенны при разработке устойчивой архитектуры. Поскольку архитектурно значимые варианты использования подробно разбираются и анализируются на каждой итерации, элементы Модели данных определяются на основе разработки макетов постоянных классов из вариантов использования. Кода макеты классов стабилизируются, проектировщик базы данных может время от времени преобразовывать их в таблицы Модели данных и определять соответствующие элементы модели хранения данных.

К моменту окончания этапа уточнения основные структуры базы данных (таблицы, предметные указатели и столбцы первичного и внешнего ключей) должны уже находиться на месте, чтобы поддерживать выполнение определенных архитектурно значимых сценариев для приложения. Кроме того, в базу данных должен быть загружен репрезентативный объем данных для поддержки тестирования архитектурной производительности. В зависимости от результатов тестирования производительности, может потребоваться скорректировать Модель данных с помощью приемов оптимизации, включая, но не ограничиваясь этим, денормализацию, оптимизацию атрибутов физической памяти или распределения, а также индексацию.

Этап построения

Основная реструктуризация Модели данных не должна происходить на этапе построения. Однако во время итераций этапа построения могут быть определены дополнительные таблицы и элементы хранения данных - на основе подробного проектирования набора вариантов использования и утвержденных запросов на изменение, предназначенных для данной итерации. При проектировании базы данных, на этапе построения основное внимание должно уделяться непрерывному отслеживанию производительности базы данных и оптимизации ее макета с помощью денормализации, индексации, создания панелей базы данных и применения других приемов.

Физическая модель данных - это артефакт проектирования, обслуживаемый проектировщиком базы данных на этапе построения. Обслуживание может выполняться путем внесения непосредственных обновлений в модель или путем считывания обновлений, внесенных непосредственно в базу данных, с помощью соответствующих инструментов.

Этап внедрения

Модель данных, как и Модель проектирования, обслуживается на этапе внедрения в ответ на утвержденные запросы на изменение. Проектировщик базы данных должен поддерживать синхронизацию Модели данных с базой данных по мере того, как приложение проходит через окончательный тест на приемлемость и развертывается в рабочую версию.

Общие рекомендации по разработке

Если коллектив разработчиков применяет современные средства визуального моделирования, которые могут преобразовывать классы в таблицы (и наоборот), или может выполнять прямую и обратную разработку баз данных, или и то, и другое, то он должен выработать рекомендации по управлению процессами преобразования и создания. Эти рекомендации необходимы прежде всего для больших проектов, когда коллектив работает параллельно над базой данных и над макетом приложения. Коллектив разработчиков должен определить моменты в процессе разработки приложения (цикл компоновки/выпуска), когда будет уместно выполнить преобразования классов в таблицы и прямую разработку базы данных. После создания первоначальной базы данных коллектив разработчиков должен создать для своих членов рекомендации по управлению синхронизацией Модели данных и базы данных, которым нужно будет следовать по мере эволюции макета и кода системы в процессе создания проекта.

Можно представить модели для рассматриваемой предметной области с использованием функционального и объектно-ориентированного подходов к проектированию. На основании этих моделей осуществляется построение соответствующих диаграмм с описанием бизнес-функций, входящих в состав моделируемых бизнес-процессов

На рисунке 1 ниже предcтавлена диаграмма IDEF3, отражающая представление рабочих процессов проектируемой информационной системы со стороны пользователя.

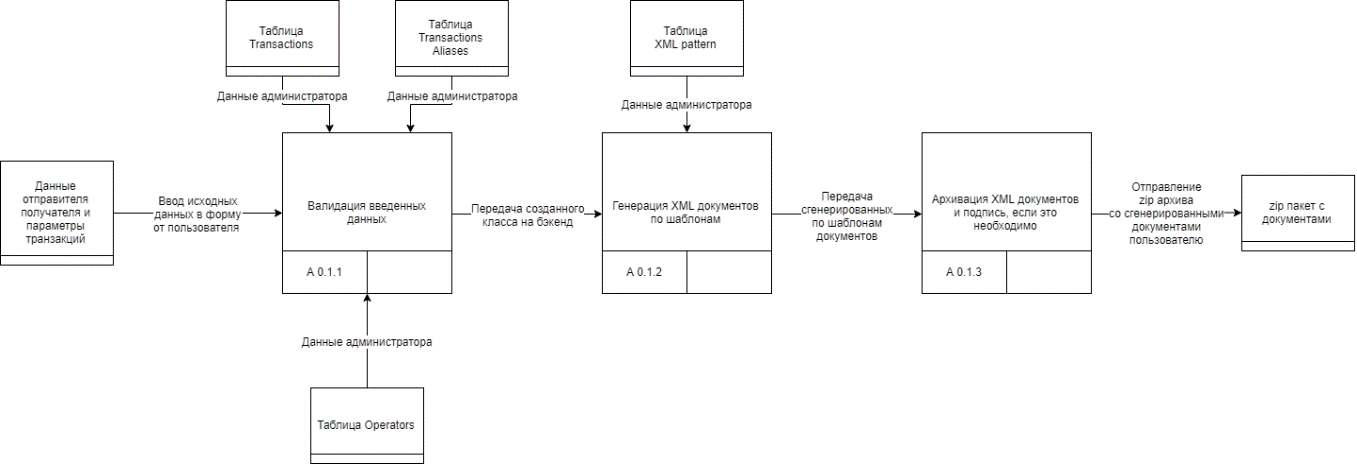


Рисунок 1. Диаграмма IDEF3

На данной диаграмме:

A 0.1.1. Валидация введенных данных: запрос к базе данных администратора – корректировки введенной информации на уровне фронтенда.

A 0.1.2. Генерация XML документов по шаблонам: обращение к базе данных администратора – выгрузка нужного шаблона из базы – корректировка шаблона с учетом введенных пользователем данных.

A 0.1.3. Архивация XML документов и подпись, если это необходимо: сбор пакетов документов по указанному пользователем методу – шифрование документа (при необходимости) – подпись документа (при необходимости).

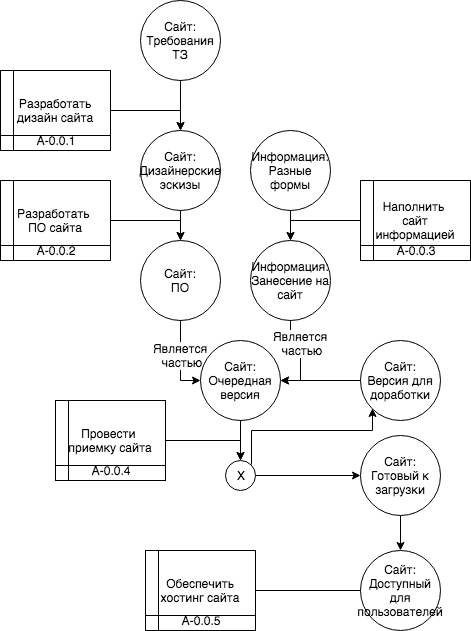
Диаграмма переходов состояний системы представлена на рисунке 2.  


Рисунок 2. Сеть переходов состояний системы

База данных в текущей версии сайта основана на SQL. В тестовом сайте база данных может быть реализована на yaml файлах, где в каждом отдельном файле храниться некоторая информация. Этот способ может быть оправдан при небольшой нагрузке на сайт.

SQLite

SQLite — это компактная встраиваемая система управления базами данных. Исходный код является открытым и все что касается этой СУБД очень хорошо документировано.

MySQL

Основным достоинством является огромная популярность и довольно большое количество документации. Используя MySQL, можно создать огромный проект, требует некоторых знаний для реализации. К минусам можно отнести не слишком высокую скорость выполнения запросов.

Redis

Это довольно молодая технология, позволяющая существенно сократить время на обработку запроса за счет хранения элементов БД в ОЗУ. Идеально для небольших проектов. Технология довольно легка для освоения, есть достаточное количество документации. Данная технология в конечном итоге идеальна для кеширования и совместного использования с PostgreSQL.

Исходя из всего вышеперечисленного было решено использовать SQLite т.к. это технология очень проста для освоения и работы и идеальна для небольших проектов. Все это вписывается в требования для разрабатываемой информационной системы: Ориентирована на достижение максимальной производительности на атомарных операциях, написана на Си, интерфейсы доступа созданы для большинства основных языков программирования, Хранит базу данных в оперативной памяти, снабжена механизмами снимков и журналирования для обеспечения постоянного хранения. Также поддерживает транзакции и пакетную обработку команд.

Даталогическая модель БД приведена на Рисунке 5.



Рисунок 5. Даталогическая модель БД

На рисунке 6 представлена инфологическая модель БД.



Рисунок 6. Инфологическая модель БД

На рисунке 7 представлена диаграмма потоков данных

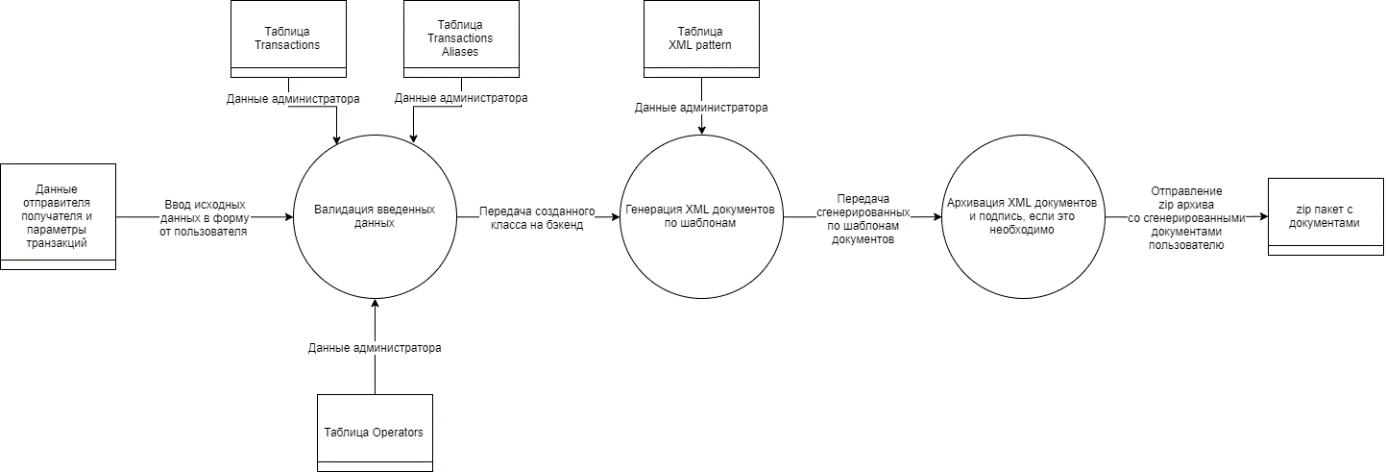


Рисунок 7. Диаграмма потоков данных

В этой модели:

Поток 1 Валидация введенных данных: запрос к базе данных администратора – корректировки введенной информации на уровне фронтенда.

Поток 2 Генерация XML документов по шаблонам: обращение к базе данных администратора – выгрузка нужного шаблона из базы – корректировка шаблона с учетом введенных пользователем данных.

Поток 3 Архивация XML документов и подпись, если это необходимо: сбор пакетов документов по указанному пользователем методу – шифрование документа (при необходимости) – подпись документа (при необходимости).

# 2.4 Проектирование пользовательского интерфейса

Интерфейс пользователя (UI) — это часть программы, которая находится на виду у пользователя и призвана обеспечивать отображение данных, управление или диалог с пользователем. При разработке программного обеспечения программисты не должны оставлять дизайн интерфейса пользователя "на потом", считая, что в программе только код должен быть краеугольным камнем. Время от времени у пользователей возникает чувство раздражённости, например, даже из-за неудачно подобранных шрифтов, неудобного расположения элементов управления, отображения данных. Для удобства пользователей должно быть затрачено n-ое количество часов, так как это серьёзная работа. Это следует учитывать при озвучивании сроков разработки программы.

Удобство пользовательского интерфейса входит в такое понятие как "Эргономика", которое включается в процессы разработки и тестирования программного продукта как часть системы качества. Разработка пользовательского интерфейса (ПИ) должна вестись совместно с дизайном программного продукта в целом.

Программа разрабатывается для обеспечения работы пользователя, т.е. для того, чтобы он, с помощью компьютерной программы, быстрее, надёжнее и качественнее решал свои производственные задачи.

С точки зрения удобства ПИ в программе важно создать такой пользовательский интерфейс, который сделает работу эффективной и производительной, а также обеспечит удовлетворенность пользователя от работы с программой.

Разработчику необходимо тщательно продумать и осознать сценарий взаимодействия программы с пользователем, приведя его к оптимальной системе выполнения задач, и реализовать ПИ в соответствии с этой системой.

Для того, чтобы разобраться в технологии решения задач пользователя, разработчику необходимо выяснить следующие моменты:

* Какая информация необходима пользователю для решения задачи?
* Какую информацию пользователь может игнорировать (не учитывать)?
* Совместно с пользователем разделить всю информацию на сигнальную, отображаемую, редактируемую, поисковую и результирующую.
* Какие решения пользователю необходимо принимать в процессе работы с программой?
* Может ли пользователь совершать несколько различных действий (решать несколько задач) одновременно?
* Какие типовые операции использует пользователь при решении задачи?

Что произойдет, если пользователь будет действовать не по предписанному Вами алгоритму, пропуская те или иные шаги или обходя их?

Дизайн ПИ должен обеспечивать минимизацию усилий пользователя при выполнении работы и приводить к:

* Сокращению длительности операций чтения, редактирования и поиска информации,
* Уменьшению времени навигации и выбора команды,
* Повышению общей продуктивности пользователя, заключающейся в объеме обработанных данных за определенный период времени.
* Увеличению длительности устойчивой работы пользователя и др.

# Глава 3. Программная реализация генератора электронного документооборота «EDOCript»

# 3.1 Обоснование выбора состава программных средств реализации

Основным языком программирования проекта был выбран Python, ввиду его огромной мощности и распространенности как в решении бытовых задач, так и написании сложных вещей наподобие нейросетей.

Python представляет собой универсальный мультипарадигменный скриптовый язык.

Универсальность основывается на том, что у данного языка не имеется определенной привязки к какой-либо области разработок. Он отлично подходит для реализации веб-приложений, десктопных решений и множества других вещей.

Мультипарадигменность заключается в минимальном ограничении разработчика во многих аспектах. Любая концепция, будь это функциональных подход или объектно-ориентированный, имеет возможности реализации, тем самым позволяет реализовать самые сложные проекты разной направленности.

Скриптовость выражается в выполнении системой различных кратких описаний действии, именуемых сценариями, то есть по сути является гибридом системного и командного языка.

Главные преимущества Python заключаются в:

* Максимально низкий порог вхождения, тем самым позволяет человеку с минимальными знаниями теории начать простые разработки за очень короткий срок.
* Язык очень грамотно спроектирован. Он хорошо отлажен, в процессе разработки принимает в себя новые приемы программирования различных типов и конфессий.
* Предельно читабельный синтаксис позволяет легко вникать в уже написанный код.
* Легкость работы привела к рождению огромного количества сторонних библиотек, максимально разной направленности.
* Невероятная легкость в процессе развертывания на почти любой платформе начиная Windows заканчивая Arduino.

Но лучше всего этот язык характеризует его философия «The Zen of Python»:

* Красивое лучше, чем уродливое.
* Явное лучше, чем неявное.
* Простое лучше, чем сложное.
* Сложное лучше, чем запутанное.
* Плоское лучше, чем вложенное.
* Разреженное лучше, чем плотное.
* Читаемость имеет значение.
* Особые случаи не настолько особые, чтобы нарушать правила.
* При этом практичность важнее безупречности.
* Ошибки никогда не должны замалчиваться.
* Если они не замалчиваются явно.
* Встретив двусмысленность, отбрось искушение угадать.
* Должен существовать один и, желательно, только один очевидный способ сделать это.
* Хотя он поначалу может быть и не очевиден, если вы не голландец.
* Сейчас лучше, чем никогда.
* Хотя никогда зачастую лучше, чем прямо сейчас.
* Если реализацию сложно объяснить — идея плоха.
* Если реализацию легко объяснить — идея, возможно, хороша.
* Пространства имён — отличная штука! Будем делать их больше!

Для разработки генератора пакетов электронного документооборота необходимо использование одного из нескольких инструментов, позволяющих с помощью языка Python создать сайт, поскольку этот язык является наиболее освоенным у исполнителей.

Рассмотрим несколько таких инструментов:

Django.

Django является очень распространенным веб-фреймворком. Основными достоинствами данной платформы являются:

* Очень подробная документация, позволяющая достаточно быстро разобраться в своей работе.
* ORM (Object-relational mapper), или система проецирования реляционных данных в объекты. Практически во всех случаях не требуется использование SQL запросов, тем самым часто обеспечивает защиту от SQL-injection уязвимости.
* Автоматически создаваемая панель администратора сайта. Сильно ускоряет процесс разработки в плане отсутствия надобности прописывания прав администратора, при сохранении всех норм безопасности.
* MTV (Model-Template-View), предполагает разделение моделей баз данных, шаблонов веб-странницы и логику этих странниц на разные модули, что очень помогает организовать работу.

Flask;

Flask – это минималистичный инструмент, предоставляемый по принципу «Только нужное». Идеален для небольших проектов, отлично документирован, организован по принципу MTV, однако для любого более крупного проекта потребует большого объема самостоятельных доработок. Этот инструмент идеален для проектирования систем подобных нашей.

Bootstrap;

Bootstrap – это одна из популярных frontend библиотек. Её разработчиком является компания Twitter. Поскольку работать с css довольно сложно, а без css сделать красивый сайт не удастся, то необходимо использовать Bootstrap в нашем проекте. Данный фреймворк характеризуется рядом прекрасных достоинств:

* Адаптивность – позволяет корректно отображать сайты на экранах различных устройств и диагоналей, при этом оставаясь читабельным и адекватным в плане использования.
* Кросс-браузерность – в связи с различием работы некоторых движков популярных веб-браузеров, не всегда одинаково описанный элемент в коде будет отрисован одинаково во всех браузерах. Этот нюанс полностью учитывался при разработке данного фреймворка.
* Легкость освоения – выражена в огромном количестве статей по поводу работы этой системы стилей. А также не перегруженностью и понятностью написанных классов, используемых в работе.

Естественно, Bootstrap не лишен недостатков.

Например:

* Шаблонность – очень часто сайты использующие макеты классов Bootstrap выглядят довольно похоже друг на друга. Решением этой проблемы может быть отказ от готовых решений, что повлечет собой весьма большие затраты по времени освоения понятий UX (User experience), и применить их в проекте. Или банальном переписывании кусков кода с импортированными классами Bootstrap.
* Старые браузеры – могут не очень корректно отображать наполнение странницы, ввиду банального устаревания методов работы с css.

И поскольку для нашего приложения не требуется наличие уникального внешнего вида, а тестировщики обязаны работать с самым современным оборудованием, то это решение будет идеальным для наших задач.

Так как основной задачей систем электронного документооборота является обеспечение валидности информации, содержащейся в передаваемых документах, автоматически приходит в голову вопрос обеспечения безопасности документов. Для таких целей как правило обязательно использование электронной цифровой подписи.

В связи с отсутствием профессиональных знаний в области шифрования было решено использовать стороннее программное обеспечение, позволяющее проводить операции с сертификатами условных пользователей смоделированного документооборота.

ViPNet CSP 4.

Является российским криптопровайдером, сертифицированным ФСБ России как средство защиты информации и электронной подписи.

Позволяет создавать ключи электронной подписи, формировать и проверять валидность подписи по ГОСТ Р 34.10-2001, ГОСТ Р 34.10-2012.

Хэширует данные по ГОСТ Р 34.11-94 и ГОСТ Р 34.11-2012.

Шифрует и обеспечивает имитозащиту данных по ГОСТ 28147-89

Базовая комплектация программы предоставляет класс защищенности КС1, утвержденный приказом ФСБ РФ от 27 декабря 2011г. № 796 «Об утверждении Требований к средствам электронной подписи и Требований к средствам удостоверяющего центра»

Существует бесплатная версия данного ПО, тем самым делая данный криптопровайдер идеальным для решения поставленных задач при реализации нашего программного продукта.

Для подписи электронных документов с соблюдением всех современных нормативов было также решено использовать сторонний продукт с уже реализованной требуемой функцией.

Выбор пал на AstralToolBox. Программное обеспечение, предназначенное для того, чтобы дать пользователям интернет обозревателей выполнять функции операционной системы, недоступные самим браузерам. Например, MS Crypto Api.

Приложение реализовано по принципу локального веб-сервера, работающий по протоколу WebSocket, принимающий запросы пользователя из браузера. Сервер реализован для платформы Net Framework.

Приложение предоставляет интерфейс для выполнение основных криптографических функций:

* Генерация ключевой пары
* Генерация запроса на сертификат
* Установка сертификата в контейнер
* Проверка подписи
* Подписание данных
* Шифрование данных
* Расшифрование данных
* Получение списка сертификатов с хранилища пользователя раздела «Личные»

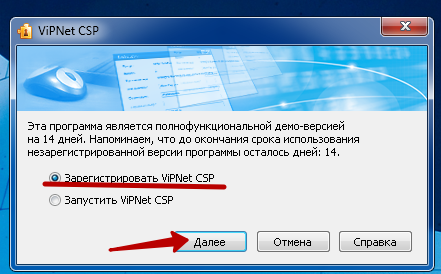
Для всего этого реализовано весьма удобное для работы API.

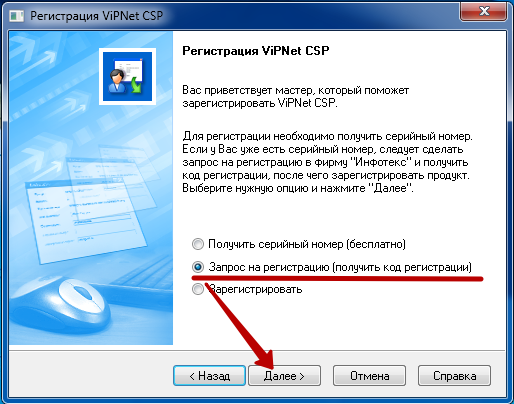
Но самое важное для нас то, что AstralToolBox полностью совместим с работой ViPNet CSP 4, что сильно упрощает процесс работы с документами и интеграцию автоматизации в код проекта.

# 3.2 Описание пользовательского интерфейса и инструкций пользователю

Для начала работы пользователя требуется предварительная подготовка. Во-первых, нужно установить требуемое программное обеспечение.

Регистрация программы. Открываем VipNet CSP.



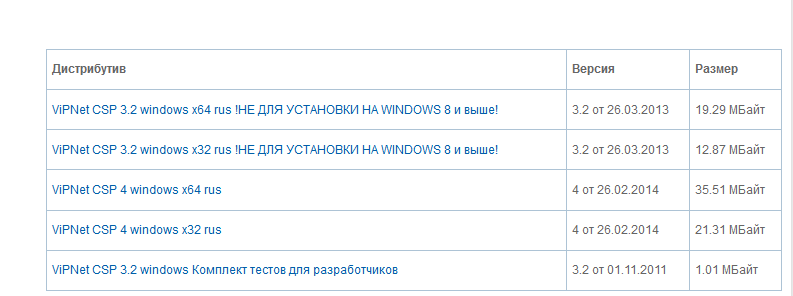


Способ запроса на регистрацию – Через Интернет (online).

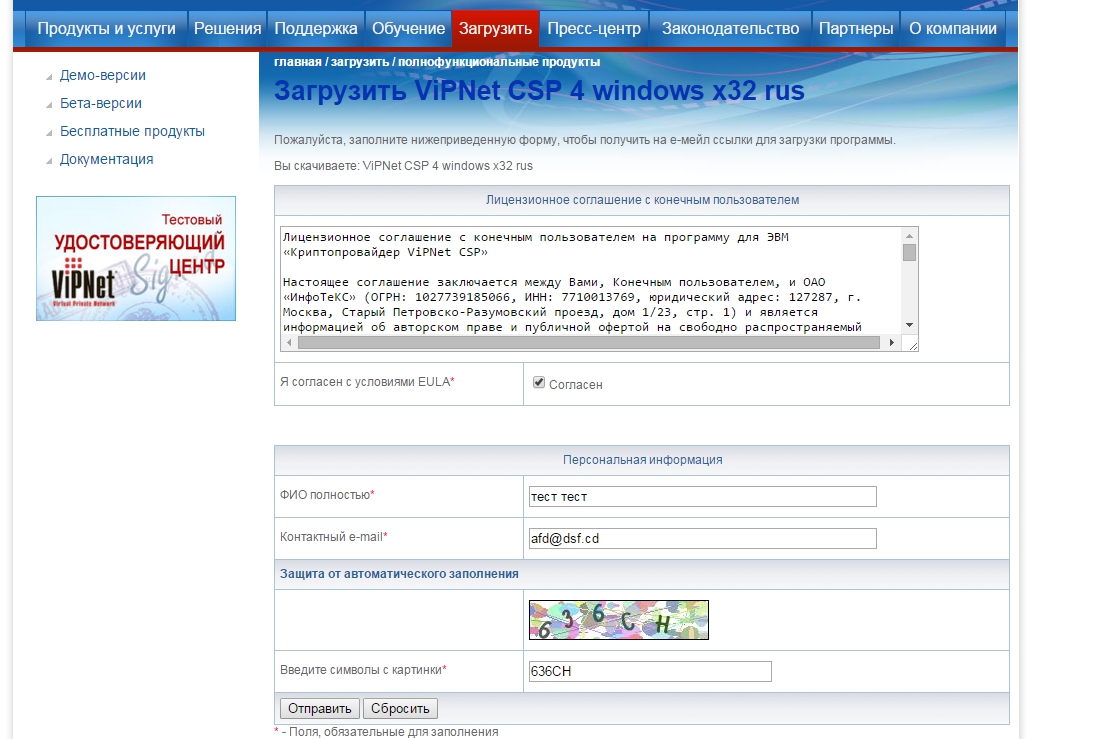
Далее необходимо зайти на сайт Infotecs

http://www.infotecs.ru/downloads/product\_full.php?id\_product=2096

Выбираем нужный дистрибутив.

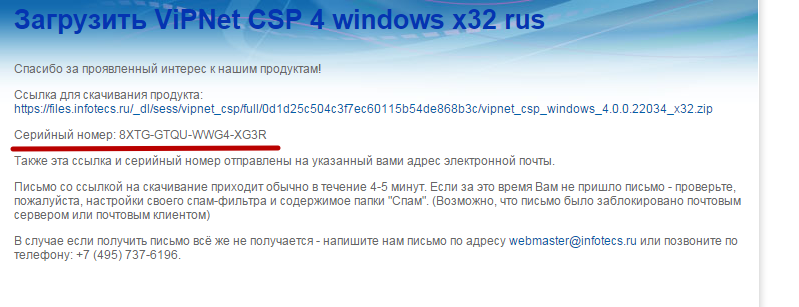


Далее:

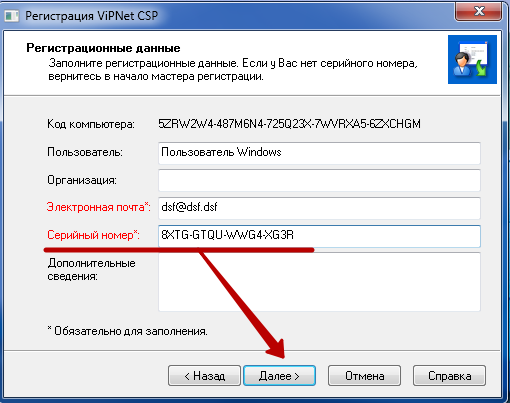


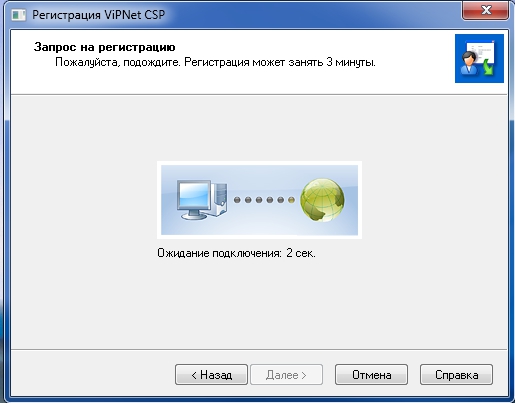
В персональной информации заполняем.

Получаем нужный нам серийный номер.

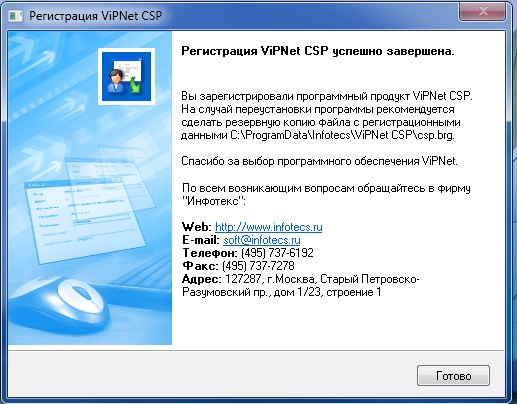


Вписываем его в регистрационные данные.

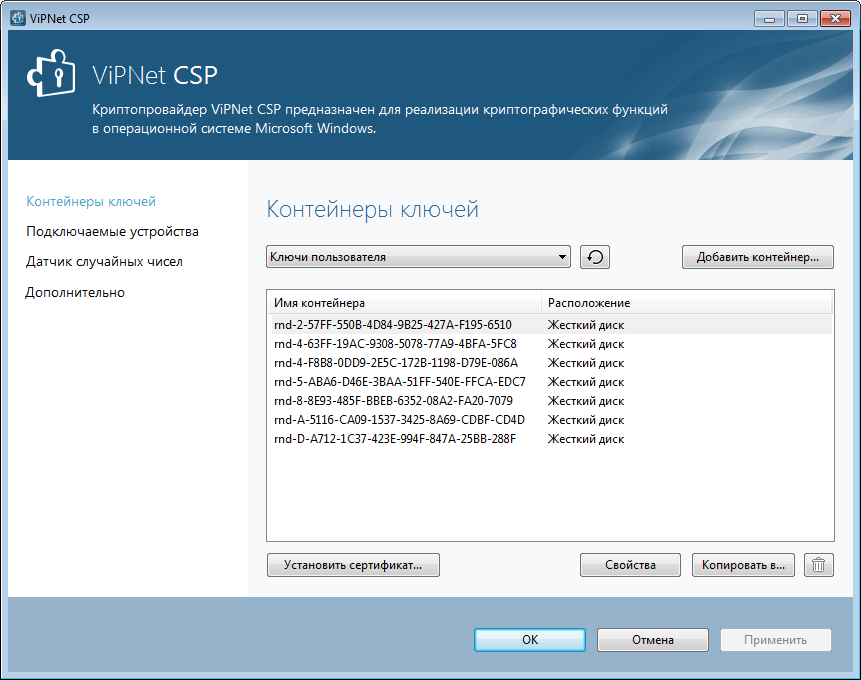




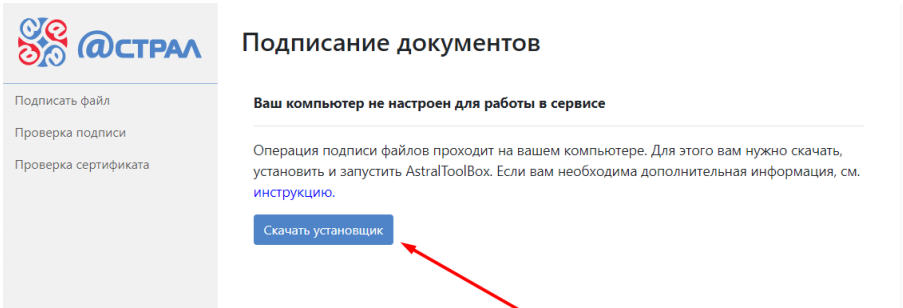
Обычно не больше 10 секунд. Если дольше значит что-то пошло не так.



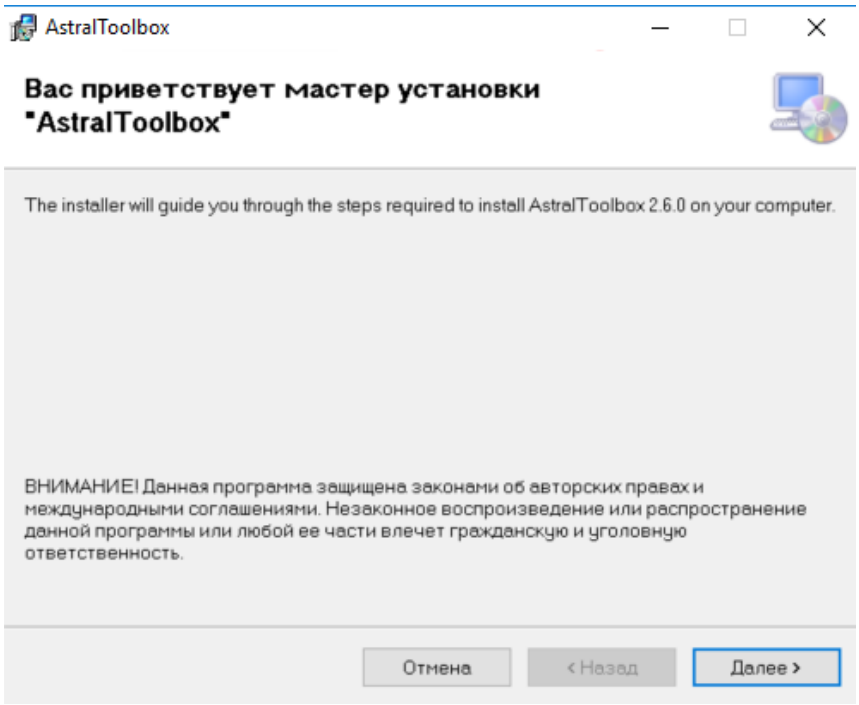
После завершению установки программного продукта ViPNet CSP 4, следует добавить в контейнер сертификаты предполагаемых пользователей моделируемого процесса электронного документооборота.



Теперь требуется установить AstralToolBox. Перейдите на странницу «http://crypto.keydisk.ru/». После загрузки установщика.

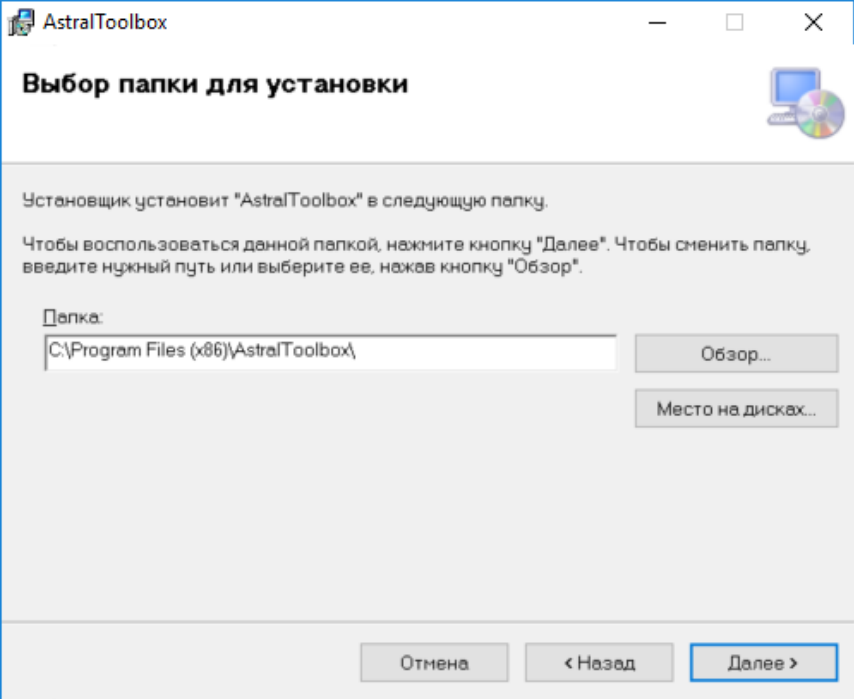


Запустите установочный файл. В открывшемся окне нажмите Далее.

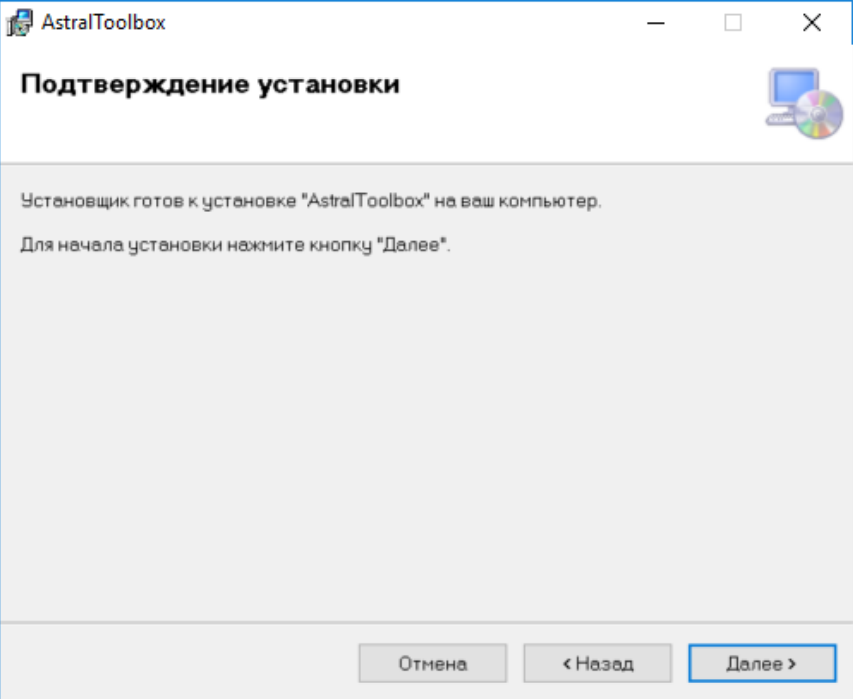


Перед вами экран выбора папки, в которую будет установлена программа. По умолчанию это «C:\Program Files(x86)\AstralToolbox». Если необходимо, укажите другую папку или выберите ее, нажав кнопку Обзор.

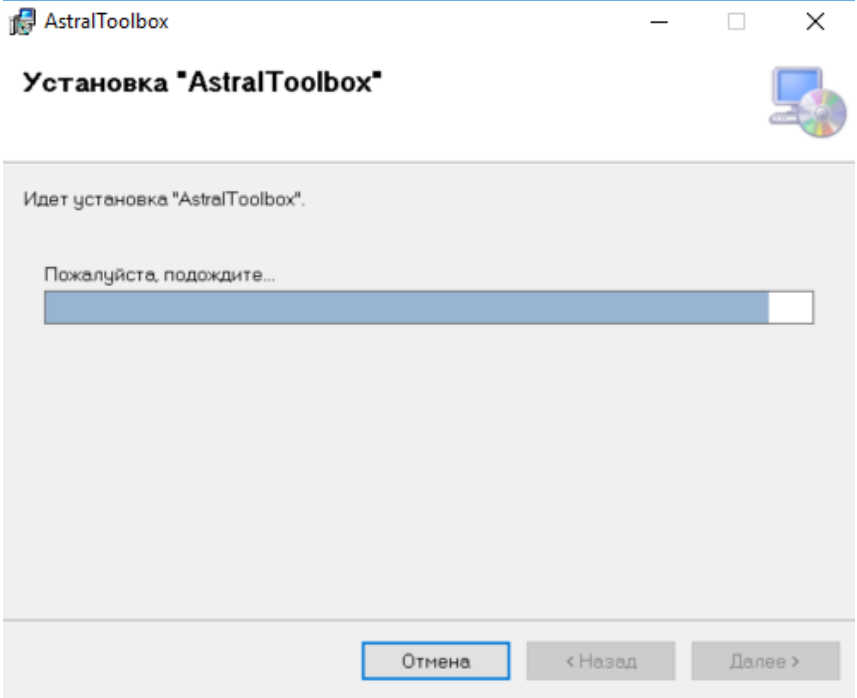
Нажмите Далее для прохождения.



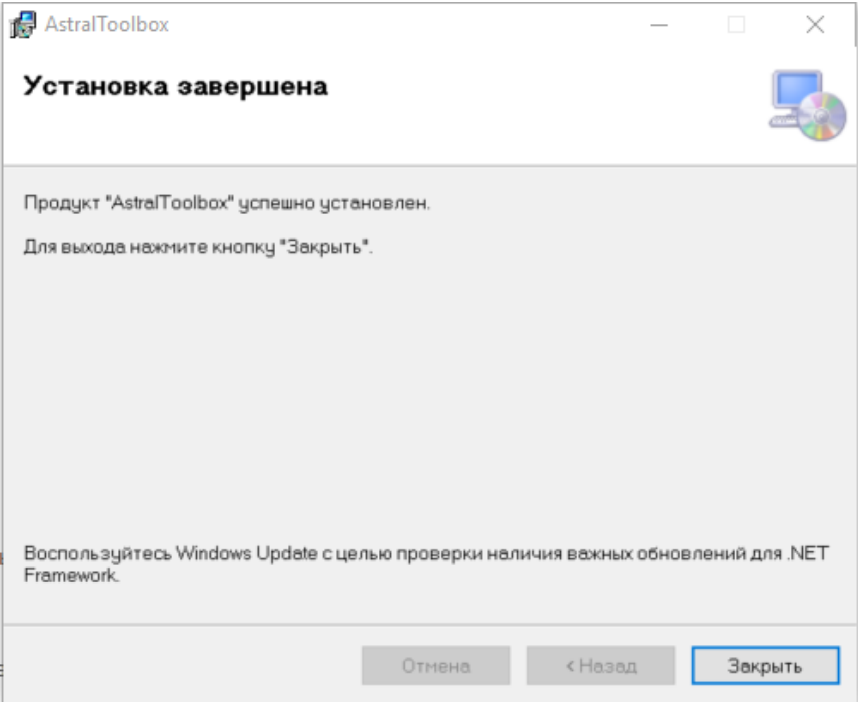
Установщик готов к установке. Нажмите Далее для установки.



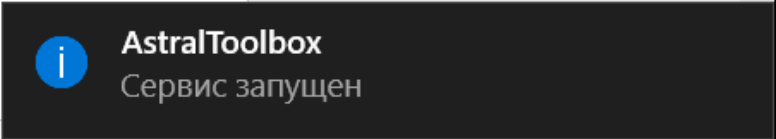
Дождитесь установки программы.



Программа успешно установлена. Нажмите Закрыть.



После установки AstralToolbox должен был автоматически запуститься на вашем компьютере.

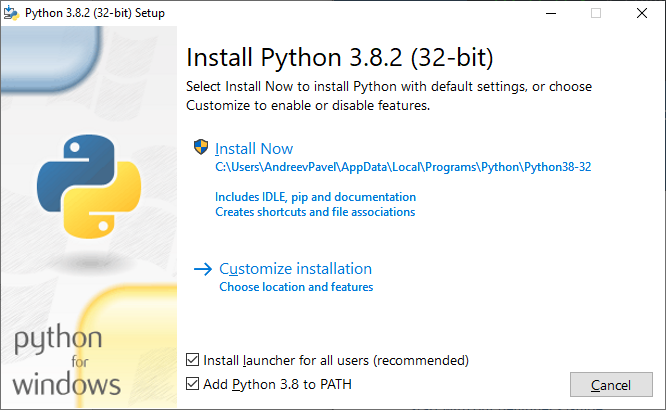


Если этого не произошло, заходим в папку, которую вы выбрали при установке программы. И запускаем AstralToolbox.exe

Далее устанавливаем Python. Переходим на сайт «https://www.python.org/» и сохраняем инсталлятор версии 3.6.5 и выше.

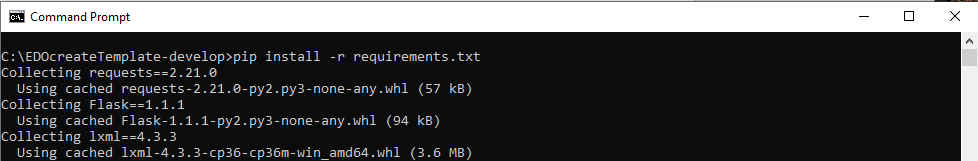


Во время установки обязательно ставим галочку на пункте «Add Python to PATH» или «Добавить путь в глобальные переменные».

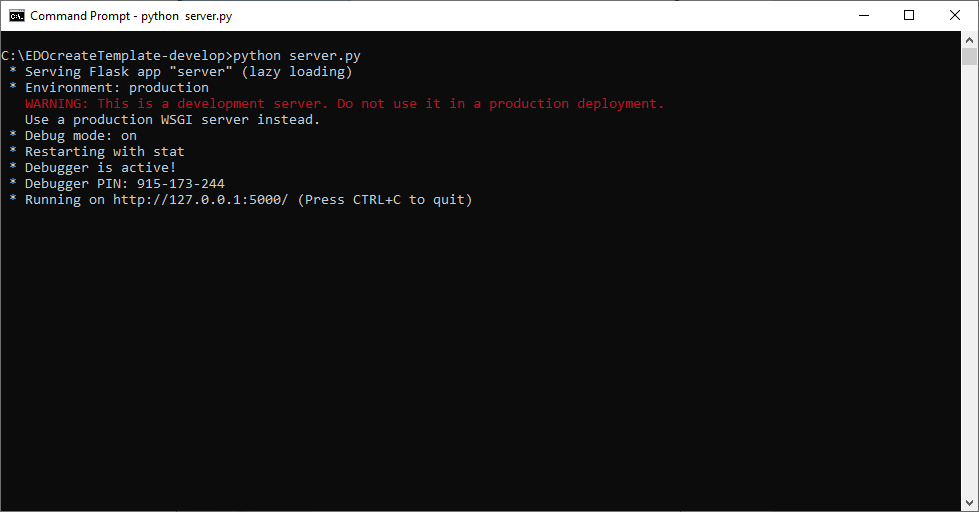


По окончанию установки можно приступать к установке генератора.

Перейдем в директорию проекта в командной строке Windows. Далее установим нужные библиотеки для Python посредством команды «pip install -r requirements.txt».



После всех приготовлений можно запускать сервер приложения «python server.py».



Если все прошло успешно, то сайт приложения будет доступен на странице, по адресу «http://127.0.0.1:5000/».

Общий алгоритм работы генератора для пользователей такой системы реализован и показан на рисунке 8.



Рисунок 8. Алгоритм сайта

При открытии страницы пользователю открывается форма для заполнения параметров генерации документов (рисунок 9). После этой процедуры, если данные введены достаточно корректно, пользователь получает zip пакет со сгенерированными документами.

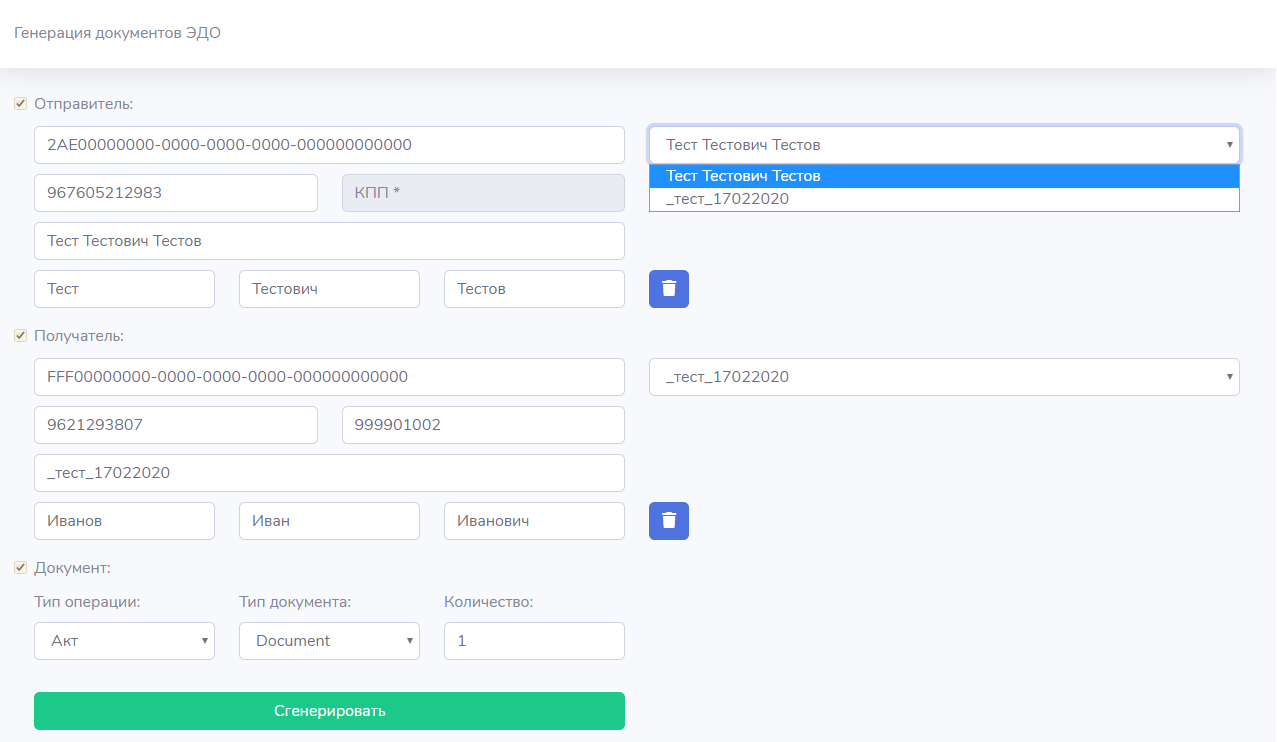
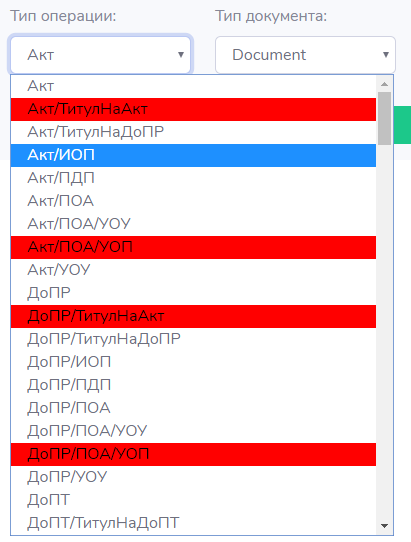


Рисунок 9. Форма заполнения параметров генерации

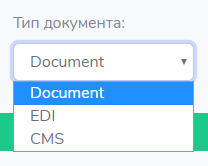
На страннице приложения присутствует форма для заполнения ключевых данных о двух абонентах, между которыми должен быть смоделирован электронный документооборот. Справа от основных полей находятся выпадающие списки с уже установленными сертификатами, для электронной цифровой подписи генерируемых документов. При выборе сертификата остальные поля автоматически заполняются в соответствии с данными из сертификата. Также поля можно редактировать по своему усмотрению, в угоду тестировщика.

В пункте формы «Документ» в подпункте «Тип операции» можно выбрать тип операции или цепочку генерируемых документов.



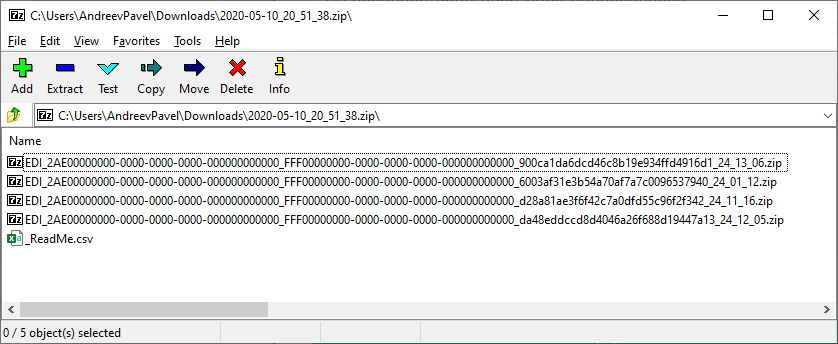
Красным выделены транзакции, которые устарели, и выбрать их нельзя.

В пункте «Тип документа» можно выбрать тип пакета генерируемого пакета для соответствующей системы.



После указания количества желаемых документов и нажатия на кнопку «Сгенерировать» сервер отправит пользователю архив со сгенерированными пакетами

Структура полученного архива состоит из zip пакетов, содержащих в себе сам сгенерированный документ и прилагающуюся подпись к нему. Также в архиве лежит csv таблица с основными данными о полученных пакетах документов.



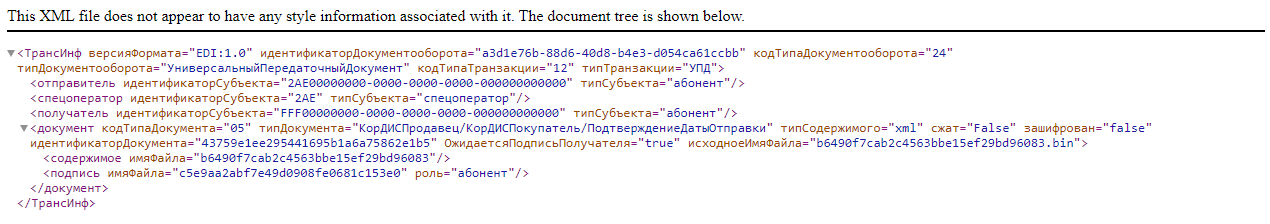
На этом функционал приложения заканчивается.

# 3.3 Руководство разработчика

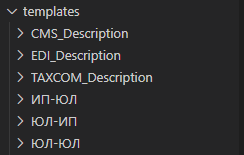
Для дальнейшего развития генератора пакетов стоит разъяснить, какие знания и действия пригодятся человеку, который продолжит развитие данного проекта или руководство разработчика.

Начнем с основ. Для начала разработчику требуются знания Python и реализация парадигм ООП (Объектно-ориентированного программирования), чтобы вникнуть в архитектуру приложения.

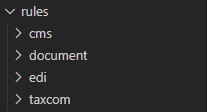
Также разработчику требуется иметь понимание электронного документооборота. Ведь взять любой шаблон для транзакции не получится. В данном генераторе учитываются три варианта построения шаблона, в зависимости от введенных данных из формы в интерфейсе сайта. Связано это с тем, что отправители и получатели могут быть как физическими лицами, так и юридическими, из-за чего у юридических лиц появляется пункт «КПП» его нужно подставлять в генерируемый документ, с учетом всех вытекающих из этого дополнительных тегов в самом документе, какие-то надо убрать, какие-то добавить. Шаблоны документов разделены на три вида, с учетом всего вышеперечисленного, на «ИП-ЮЛ», «ЮЛ-ИП» и «ЮЛ-ЮЛ». Однако этого недостаточно для создания пакета документа. Например, для создания пакета по стандарту EDI в пакете требуется иметь файл описание содержимого пакета, ведь некоторые файлы могут быть зашифрованы и лишены отличительных особенностей.



Для таких небольших xml файлов тоже выделены шаблоны, они рассортированы по папкам «cms», «edi», «document», «taxcom».

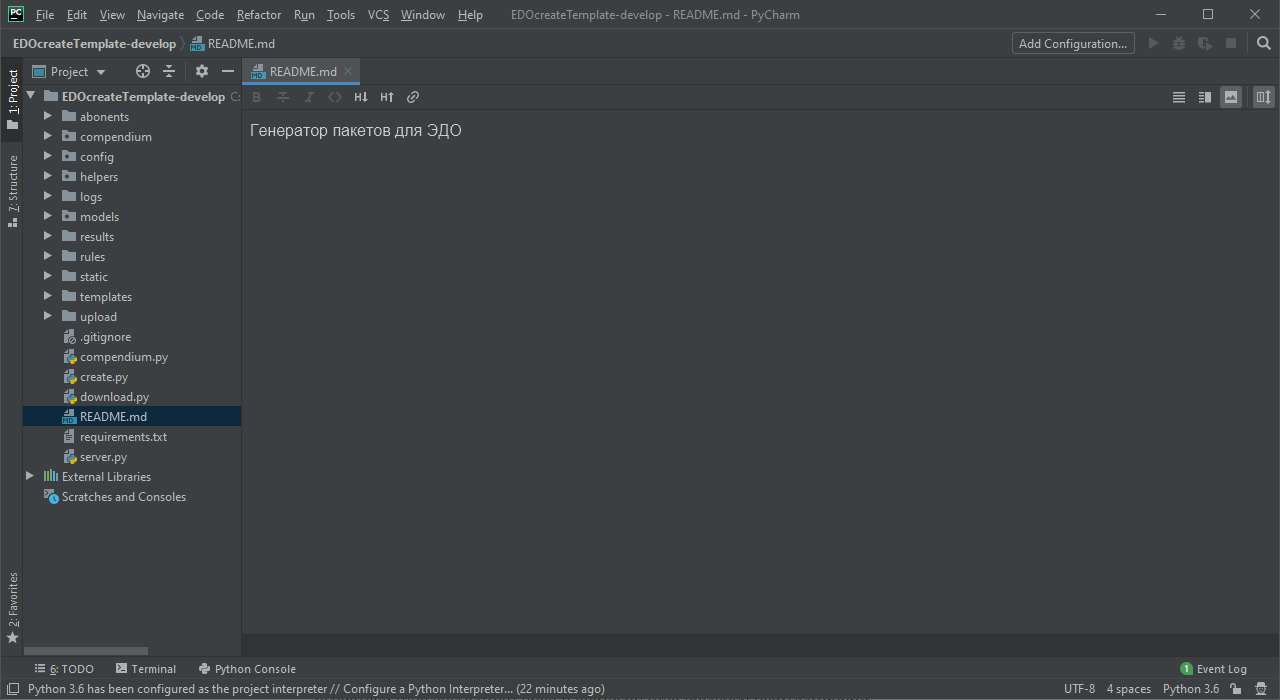


Чтобы заполнять полученные шаблоны, требуется прописать правила. Правила описаны на языке запросов к элементам XML-документа XPath. Хранятся все они в папке «rules» и отсортированы по моделям оформления пакетов документооборота. Все это разбиение сделано с учетом дальнейшего развития приложения и усложнения системы.

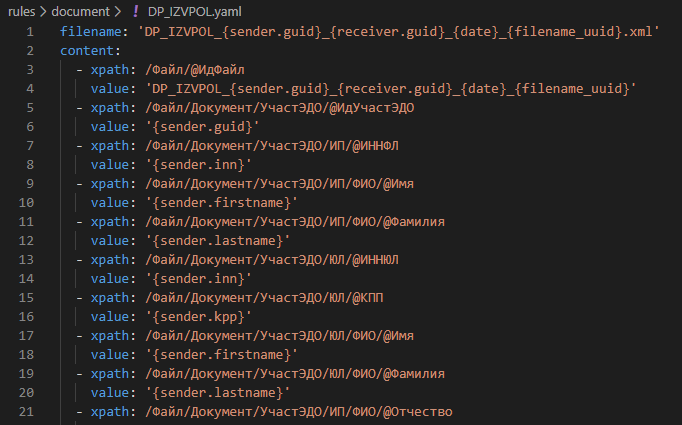


Заниматься разработкой стоит в IDE (Integrated Development Environment). В пример хочу привести разработку от компании JetBrains для разработки на языке программирования Python – «PyCharm».

PyCharm имеет очень весомое достоинство в виде мощной интеграции с современными фреймворками для работы в сфере разработок программного обеспечения. Также очень удобный редактор позволяет крайне быстро проводить навигацию по коду проекта, что очень чувствуется в больших, многоуровневых проектах. Для систем подобных нашей крайне необходим частое тестирование, в этом помогает встроенный отладчик среды разработки. Утилита поддерживает все свежие версии Flask, Django, IronPython, Jython, Cython, PyPy wxPython, PyQt, PyGTK и многие другие инструменты. В PyCharm можно, а главное удобно проводить Unit тестирование. Чтобы не засорять компьютер лишними данными библиотек, и исключить фактор влияния операционной системы на работы разрабатываемого приложения, есть возможность использовать виртуальные среды для функционирования приложения. А также очень важна полностью реализованная работа с Git системами.



Для добавления или изменения генерируемых шаблонов следует пройти в директорию «templates», далее добавить или изменить нужный шаблон документа. Хранятся они в формате XML. После чего проверить, есть ли нужный набор правил заполнения данного документа в директории «rules» и добавить, если таковой отсутствует.



# Заключение

В рамках выполнения выпускной квалификационной работы, достигнута поставленная цель и выполнены все предполагаемые требования задачи. А именно, разработано приложение генератор «EDOCript» для генерации пакетов с эмулированного электронного документооборота между двумя абонентами с учетом всех нормативов и стандартов, для упрощения работы тестировщика работающего в системах ЭДО.

Выполнены следующие задачи:

* Проанализирована предметная область
* Принято решение по выбору программной среды
* Разработаны технические требования к системе
* Разработан интерфейс пользователя
* Успешно разработано приложение генератор для генерации документооборота для систем тестирования ЭДО.

# Список использованных источников

# ПРИЛОЖЕНИЕ А