**РЕГЛАМЕНТ РАЗРАБОТКИ**

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

Москва, 2018 год

Сведения о документе

|  |  |
| --- | --- |
| Учреждение | ООО «Кодерлинк» |
| Название документа | Регламент разработки. |
| Автор | Слесаренко Артем Александрович (aslesarenko@coderlink.cc) |
| Создан | 15.08.2018 |
| Количество страниц | 25 |

История изменений

| **Редакция** | **Дата** | **Автор изменений** | **Описание** |
| --- | --- | --- | --- |
| 0.1 | 15.08.2018 | Слесаренко А.А | Первая версия структуры документа |

Содержание

[1. Общие положения 4](#_Toc522108358)

[1.1. Назначение и задачи документа 4](#_Toc522108359)

[1.2. Порядок внесения изменений 4](#_Toc522108360)

[1.3. Контроль исполнения требований документа 4](#_Toc522108361)

[2. Термины и сокращения 5](#_Toc522108362)

[2.1. Список определений 5](#_Toc522108363)

[2.2. Список сокращений 5](#_Toc522108364)

[3. Средства разработки 6](#_Toc522108365)

[3.1. Среда разработки 6](#_Toc522108366)

[3.2. Система контроля версий 6](#_Toc522108367)

[3.3. Система управления проектами 6](#_Toc522108368)

[3.4. Перечень прикладного ПО 6](#_Toc522108369)

[4. Работа с системой контроля версий 7](#_Toc522108370)

[4.1. Подготовка к работе c GIT win 7](#_Toc522108371)

[4.2. Подготовка к работе c GIT Linux 8](#_Toc522108372)

[4.3. Создание локального репозитория кода 9](#_Toc522108373)

[4.4. Принципы работы с репозиторием кода 9](#_Toc522108374)

[4.5. Хранение модели данных 10](#_Toc522108375)

[5. Работа с системой управления проектами 11](#_Toc522108376)

[6. Ведение проектной документации 12](#_Toc522108377)

[6.1. Соглашение по именованию объектов 12](#_Toc522108378)

[6.2. Соглашение о форматировании исходного кода 12](#_Toc522108379)

[7. Сборка и состав дистрибутивов 13](#_Toc522108380)

[7.1. Типы обновлений системы 13](#_Toc522108381)

[7.2. Общие принципы сборки 13](#_Toc522108382)

[7.2.1. Сборка: 13](#_Toc522108383)

[7.2.2. Состав дистрибутива 13](#_Toc522108384)

[7.2.3. Развертывание 14](#_Toc522108385)

[7.3. Формат файлов сборки 15](#_Toc522108386)

[7.4. Нумерация релизов 15](#_Toc522108387)

[8. Тестирование 16](#_Toc522108388)

[8.1. Внутреннее тестирование исполнителя 16](#_Toc522108389)

[8.2. Внутренне тестирование у субподрядчика 17](#_Toc522108390)

[8.3. Критерии качества приложения 17](#_Toc522108391)

[8.4. Функциональное тестирование 17](#_Toc522108392)

[Приложение А. Генерация ключей доступа 18](#_Toc522108393)

[Приложение Б. Добавления ключа в gitlab 22](#_Toc522108394)

[Приложение С. Оформление кода 24](#_Toc522108395)

[**A)** **Общие пожелания** 24](#_Toc522108396)

[**B)** **Оформление java** 24](#_Toc522108397)

[**C)** **Оформление Java Script** 24](#_Toc522108398)

[Приложение Д. Оформление документации 25](#_Toc522108399)

1. Общие положения
   1. Назначение и задачи документа

Настоящий Регламент устанавливает порядок и основные правила проведения разработки в рамках проектов компании ООО «Кодерлинк».

Задачами настоящего Регламента являются:

* определение общих организационных положений разработки на проекте;
* задания перечня использованных программных продуктов в процессе разработки;
* указание по сборке и составу дистрибутивов.
  1. Порядок внесения изменений

Настоящий регламент может быть пересмотрен в ходе работ по разработке Системы в связи с изменением требований Заказчика.

Предложения о пересмотре или внесении изменений в настоящий регламент выносятся на рассмотрение архитектору проекта. В случае внесения изменений настоящий регламент утверждается и доводится до сведения всех участников процесса разработки и внедрения Системы в порядке, установленном в ООО «Кодерлинк».

* 1. Контроль исполнения требований документа

Контроль исполнения требований настоящего Регламента осуществляется ООО «Кодерлинк».

1. Термины и сокращения
   1. Список определений

| **Термин** | **Определение** |
| --- | --- |
| **GIT** | [Система управления версиями](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8) |
| **Redmine** | Система оперативного управления проектами |

* 1. Список сокращений

| **Сокращение** | **Расшифровка** |
| --- | --- |
| **ПО** | Программное обеспечение |
| **IDE** | Integrated Development Environment |
| **КШД** | Корпоративная шина данных |

1. Средства разработки
   1. Среда разработки

Разработка подсистем проекта осуществляется с использованием интегрированной среды разработки Eclipse, где это возможно. IDE представляет собой комплекс средств для разработки, отладки и тестирования приложений Java и Java EE-приложений, предоставляет средства совместной работы в проекте и другой функционал.

* 1. Система контроля версий

В качестве средства контроля версий используется система GIT. В качестве утилит поддержки рекомендуется использовать также: Tortoise GIT, EGit, kdiff3, putty, msys git. Более подробное описание процесса настройки системы контроля версий дано в разделе 4.

* 1. Система управления проектами

В качестве средства управления проектами следует использовать систему Redmine, доступную по следующим ссылкам:

* внутренняя ссылка для входа – <http://redmine.chelny.taximaxim.ru>.

В данной системе заводятся задачи по аналитике, написанию документов, разработке, исправлению дефектов и тестированию.

Система предоставляет средства управления и хранения документов. Все проектные документы следует хранить и актуализировать через Redmine.

* 1. Перечень прикладного ПО

Версии программных продуктов, которые должны быть использованы при разработке системы, перечислен в Таблице 2:

Таблица 1 Список прикладного и базового ПО

| Продукт | Версия | Описание |
| --- | --- | --- |
| CentOS | 7 | Операционная система |
| Java SDK | 1.8 | Java Development Kit |
| Apache HTTP сервер |  | HTTP-сервер для IBM Watson Explorer |

1. Работа с системой контроля версий

На стендах ООО «CoderLink» развернут сервер GIT, для работы с ним необходимо установить и настроить указанные ниже программы.

* 1. Подготовка к работе c GIT win
* Для работы c GIT необходимо установить Git for Windows (<http://msysgit.github.io/>):
* выбрать опции при установке « Git Bash here»;
* на другой вкладке «Use Git from the Windows Command Prompt»;
* далее «Checkout as-is, commit Unix-style line endings».
* Для работы с репозиторием GIT можно установить TortoiseGit из проекта <http://code.google.com/p/tortoisegit/>.
* После установки TortoiseGit необходимо в настройках TortoiseGit → Settings → Git → проставить опции:
* для windows платформ: AutoCRLF и SafeCRLF в false;
* для unix плафторм: AutoCRLF = false, SafeCRLF = true;
* после установки для осуществления комтов необходимо указать имя, фамилия, email разработчика.
* Необходимо установить putty (<http://www.chiark.greenend.org.uk/~sgtatham/putty/>). Следует установить полный пакет со всеми программами, понадобятся plink, puttygen.
* С помощью puttygen необходимо сгенерировать приватный и публичный ключ. (см. Приложение А).
* Публичный ключ необходимо добавить в gitlab, как указана в приложении
* Запустить pageant.exe. В области уведомления Windows (систем трей) выбрать pageant нажать правую кнопкой на иконку и в меню нажать add key.
* Утилита pagent должна быть запущена для автоматической аутентификации по ключу. При перезагрузке необходимо подключать ключ повторно.
* В настройках Eclipse необходимо прописать UTF-8 и line delimeter.

Меню настроек Eclipse представлено на Рис. 1.

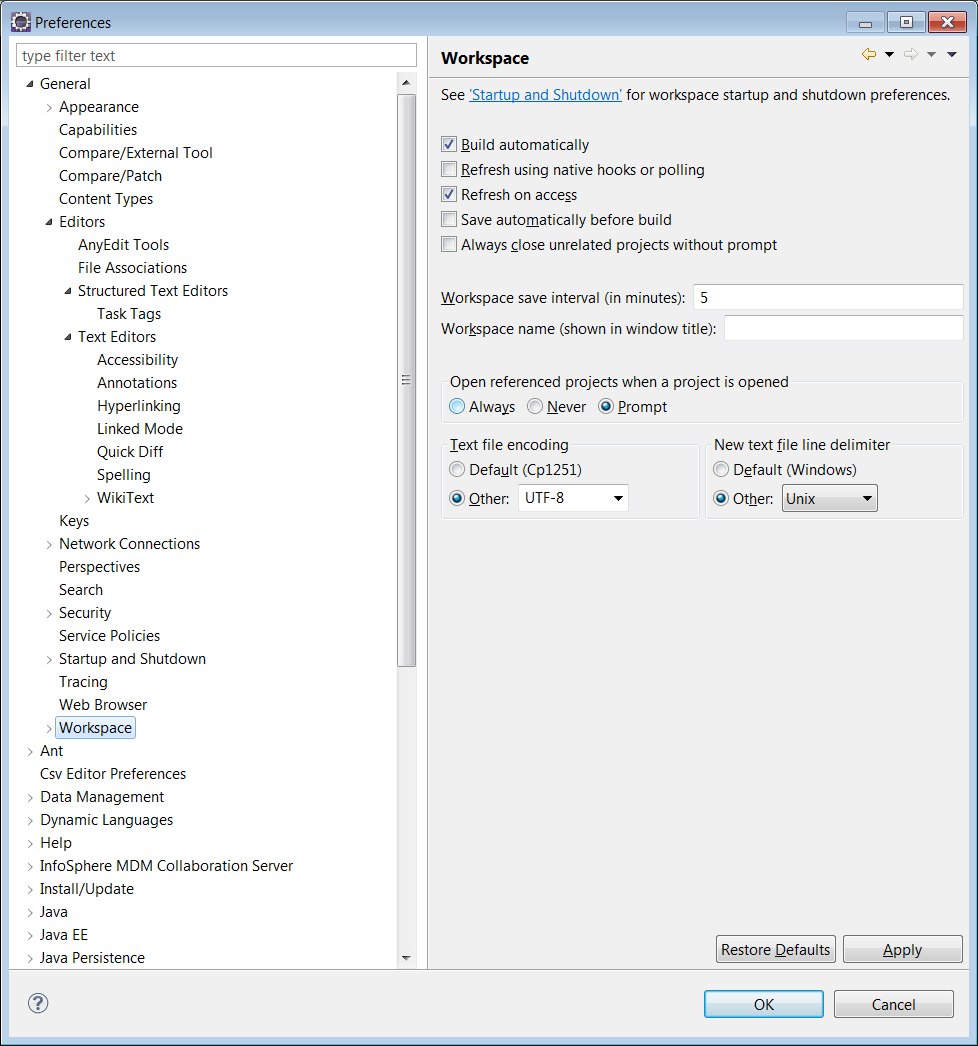


Рис. . Настройки Eclipse

* 1. Подготовка к работе c GIT Linux

Пример создания ключа и подключения для Linux систем. Пример описан от root пользователя.

1. Создаем ключ

**ssh-keygen -t rsa**

При создании ключа необходимо указать пароль, так как многие системы будут его спрашивать.

**Система созданы два файла ключей:**

/root/.ssh/id\_rsa.pub

/root/.ssh/id\_rsa

Публичный ключ (pub) необходимо прислать мне.

1. Запускаем агент

**eval `ssh-agent`**

В таком варианте ssh-agent, будет выполнит его в текущей оболочке.

1. Загрузка ключа к агенту

Перед добавлением ключа, необходимо проверить, что уровень доступа ключа rw------- (0600), иначе система будет ругаться

Пример: **ssh-add /opt/id\_rsa.new1**

1. Проверка загруженных ключей

**ssh-add -l**

Примерный ответ системы:

**[root@localhost opt]# ssh-add -l**

**2048 SHA256:ZD3P9WF70wb3a7ZOuWO73AWE9w4m3qzclr/BaKAcIFE /root/.ssh/id\_rsa (RSA)**

1. Удаление ключа

Пример: **ssh-add -d id\_rsa.new2**

1. Загрузка репозитория

**git clone ssh://git@nsi-rm-novatek.csmteam.ru:8123/Novatek**

1. При возникновении проблем

При возникновении проблемы с ключом, необходимо проверить работу его для варианта в Windows. Для этого необходимо скорректировать ключ linux при помощи команды puttygen.exe и использовать, как описано в варианте для windows.

1. Полезные ссылки:

<https://sergeyvasin.net/2017/03/30/git-ssh/>

<http://dreamcatcher.ru/2009/11/30/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F-%D0%BD%D0%B0-ssh-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%B5/>

http://www.calculate-linux.org/main/ru/gitolite

* 1. Создание локального репозитория кода

Для начала работы с проектом необходимо:

1. В папке, в которой предполагается размещать все рабочие проекты, нажать правую кнопку мыши и выбрать TortoiseGit→Clone, далее ввести адрес центрального репозитория. URL подключения к GIT серверам:

* внутренний: https://git.technology45.ru/coderlink/filecloud/filecloud\_client.

1. Убрать отметку напротив поля «Load Putty Key» или выбрать путь до приватного ключа.
2. В результате выполнения команды на локальном диске будет создана копия репозитория кода.
   1. Принципы работы с репозиторием кода

Выделяются следующие основные принципы работы с репозитарием кода:

1. Все новые ресурсы рабочей директории разработчика (папки, файлы) следует индексировать для того, чтобы данные изменения стали видимы для GIT. Команда индексации – Add index.
2. Обновление локального репозитория осуществляется по команде Pull.
3. Фиксация изменений в локальном репозитории GIT осуществляется по команде Commit.
4. Передача изменений на удаленный репозиторий GIT осуществляется по команде Push.
5. Команда push не отрабатывает в том случае, если локальный репозиторий не синхронизирован с удаленным репозиторием, следует повторно выполнить команду Pull и разрешить все существующие конфликты.
6. Разработка субподрядчиков по системе должна вестись в едином репозитории кода, который предоставляет компания.
7. Работа с GIT должна осуществляться под своей персонифицированной учетной записью.
8. Выкладывание изменений на удаленный сервер GIT (команда push) следует проводить после завершения задачи по разработке или по исправлению дефекта.
9. Commit изменений на удаленный сервер GIT для незавершенных задач допустим только в исключительных случаях.
10. Недопустим одновременный commit изменений на удаленный сервер GIT по нескольким задачам на разработку.
11. Каждая фиксация изменений в GIT должна сопровождаться комментарием с указанием номера задачи в Redmine.
12. Все commit в GIT без указания реальной задачи из Redmine отклоняются.
13. Фиксация релиза в GIT осуществляется созданием отдельной ветки релиза – Branch, как правило, после передачи релиза на стенд Заказчика.
14. Разработка новой функциональности по релизу, который не передан на стенд Заказчика, должна вестись в ветке Master.
15. Исправление ошибок по релизу, для которого создан branch, должно проводиться в ветке данного релиза и копироваться в Master ветку.
16. Commit не должен содержать файлы, не относящиеся к изменениям. Если IDE, OS, плагин к софту, использующемуся при разработке, создают технические файлы, необходимо добавить их в gitignore либо не добавлять к commit/удалять перед commit.
17. Все файлы проектов GIT следует хранить в кодировке UTF-8.
18. Все файлы проектов GIT следует хранить без символа возврата корретки для переноса строк (Unix style).
    1. Хранение модели данных

Рабочее пространство после создания локальной копии репозитория кода должно состоять из следующих проектов (Таблица 2).

Таблица 2. Проекты рабочего пространства

| Название проекта | Тип проекта | Описание |
| --- | --- | --- |
|  |  |  |

1. Работа с системой управления проектами

Для управления всеми проектными активностями работа по проекту осуществляется в системе Redmine. При работе с Redmine следует руководствоваться следующими принципами:

* Redmine должна использоваться всеми по проекту для ведения и актуализации списка задач и дефектов, а также для ведения проектной документации.
* Работа в системе Redmine должна проводиться под своей персонифицированной учётной записью.
* При работе с Redmine следует придерживаться принципа распределения задач по релизам. При невозможности указания номера релиза следует заводить задачу по форме «Х.Х» с последующей уточнением.
* Следует использовать следующие типы задач:
* ошибка – задача на исправление дефекта;
* новая функциональность – задача на разработку.
* Задачи на разработку нового функционала могут быть объединены в группы под одной задачей.
* Ошибки системы следует привязывать к соответствующей задаче функционалу.
* Желательно указание ссылки на документ, загруженный в redmine c указанием с пункта.
* Следует поддерживать статус каждой задачи, заведенной в Redmine, в актуальном состоянии.
* Необходимо проставлять трудозатраты на задачу.
* Завершение разработки должно оканчиваться статусом задачи «Решена», при этом должен быть указан процент выполнения – 100%.
* Перед сборкой релизов необходимо проводить актуализацию задач, заведенных в Redmine, в связи с тем, что отчетные документы в виде списка реализованной функциональности и исправленных дефектов будут формироваться через Redmine.

1. Ведение проектной документации

Все проектные документы следует хранить и актуализировать через Redmine.

Следует придерживаться следующего принципа распределения документов по папкам:

Таблица 3 Директории хранилища документов Redmine

|  |  |
| --- | --- |
| Директория | Назначение |
| Проектная документация | Документы по проектам |
| <Название проекта> | Краткое название проекта с указанием бизнес-заказчика |
| <Номер релиза> | Номер релиза системы |
| Пользовательская документация | Документы по проектам |
| Руководство пользователя | Документы с руководствами пользователей по проекту. |
| Руководство эксперта | Документы с руководствами экспертов по проекту. |
| Общая документация | Общая документация для всех проектов. Родительская папка. |
| Схемы | Схемы сообщений и описание маппингов полей |
| Описание инфраструктуры | Описание инфраструктуры |
| Организационная | Организационные документы: регламенты, контакты, расписание релизов и т.п. |
| Приёмо-сдаточная | Проектные отчетные документы: Руководства пользователя, администратора, установки системы и т.п. |
| Разработка | Документы по разработке системы, обучающие материалы по разработке |

Редактирование документов следует проводить с предварительным блокированием документа, нажав на кнопку E:\tmp\31.07\lock.png

Загрузка документов после редактирования осуществляется по кнопке «Add Files» в нужной директории хранилища. При редактировании документа не следует изменять его название. При изменении наименования система автоматически создаст новую версию документа, при этом все старые версии документа останутся доступными для работы.

Для получения предыдущих версий документа необходимо нажать на кнопку E:\tmp\31.07\filedetails.png, выбрать необходимую редакцию и скачать документ по ссылке E:\tmp\31.07\download.png.

* 1. Соглашение по именованию объектов

Для программных объектов следует использовать правила именования объектов, изложенные в **Ошибка! Источник ссылки не найден.**.

**Правило 1** - с заглавной буквы.

Таблица 4 Правила инменования

| №  п/п | Тип объекта | Язык | Пробелы | Правило именования | Примеры |
| --- | --- | --- | --- | --- | --- |
|  | Название таблиц БД | Eng | Нет | Правило 1  Правило 2  Не рекомендуется использовать более 20 символов | Справочник Банки |

* 1. Соглашение о форматировании исходного кода

Для унификации процесса разработки следует придерживаться единых стандартов по форматированию программного кода. За основу правил форматирования исходного кода взяты следующие рекомендации:

* «Java Code Convensions» от Sun Microsystems, 2007 – для написания кода на java.

1. Сборка и состав дистрибутивов
   1. Типы обновлений системы

Выделяется 2 типа сборки по отношению к предыдущим обновлениям:

* Кумулятивная сборка, начиная с релиза Х – содержит все обновления системы, которые проводились с релиза Х включительно. Данный тип сборки применяется в случае, если на целевую среду установки не ставились промежуточные обновления, начиная с релиза Х. Кумулятивная сборка строиться на основании комитов GIT .
* Некумулятивная сборка – содержит обновление системы по отношению к самому последнему по времени выпущенному релизу.

Как правило, кумулятивный тип сборки применяется для установки на продуктивный стенд, некумулятивный – на тестовый стенд.

Кроме того, применяется деление сборок системы по совокупному составу:

* Релиз - сборка системы, включающая все типы функциональных модулей системы, изменение которых потребовалось в течение заданного продолжительного цикла разработки. Релиз фиксирует результат очередного отчетного этапа разработки ПО.
* Полный релиз - сборка системы, включающая все типы функциональных модулей, входящих в её состав, предназначенная для установки на «чистую» программно-аппаратную платформу, что подразумевает отсутствие установленных компонентов системы на целевом стенде назначения.
* Обновление – промежуточная сборка системы или части её функциональных модулей. Как правило, предназначена для исправления ошибок ПО.

При сборке системы следует руководствоваться указанными типами обновлений.

* 1. Общие принципы сборки
     1. Сборка:
* Сборка дистрибутивов должна выполняться в автоматическом режиме из исходных кодов проекта, полученных из системы GIT.
* Утилита сборки должна принимать на вход дату/метку/ревизию, относительно которой будут получены исходные файлы из системы GIT для сборки релиза.
* Сборка дистрибутива должна быть максимально автоматизирована.
* Сборка должна содержать метку версии дистрибутива в каждом артефакте развертывания (приложение java – manifest.mf).
* Сборка должна содержать описание своего типа (кумулятивная по отношению к релизу Х или некумулятивная).
  + 1. Состав дистрибутива
* Результатом сборки приложений должен быть дистрибутив, состоящий из:
* архив модели данных;
* архив контента;
* библиотеки jar, war и ear, модули развертывания;
* скрипты развертывания;
* файлы конфигураций и свойств;
* исходные коды приложений;
* документация к релизу.
* Комплект документов к релизу должен содержать руководство по установке релиза, руководство администратора, Release notes.
* Все дистрибутивы (обновления системы) должны быть структурированы в едином виде в соответствии с *Таблица 5*:
* Дистрибутив должен передавать в виде zip архива (с копированием на сервер, установка должна начинаться с открытия архива).
* Release notes дистрибутива должен формироваться на основе списка задач Redmine.
* Release notes должен содержать отдельную графу со списком измененных в составе релиза бизнес-данных указанием идентификаторов объектов и номеров задач Redmine, по причине которых потребовались изменения в данных.
* В Redmine следует поддерживать актуальную версию Release notes (Документы/ Общая документация/Разработка).

Таблица 5 Состав обновления системы

| №  п/п | Директория | Список файлов | Обязательность\* | Описание |
| --- | --- | --- | --- | --- |
|  | Корневая папка:  ./ | Release notes.xlsx | Да | Описание изменений системы. Файл должен вестись кумулятивно: содержать историю изменений |
| Инструкция по развертыванию.docx | Да | Последовательность установки и настройки релиза, а также действия по откату |
|  | 01\_CONF | - | Да | Конфигурационные файлы релиза |
|  | 01\_CONF/mdmcs/<host> | bin/  etc/  jars/ | Нет | Конфигурационные файлы, файлы свойств, архивы расширений, которые были изменены в рамках релиза |
|  | 01\_CONF/release | test.cfg  prod.cfg | Да | Конфигурационные файлы, файлы свойств, которые необходимы при инсталляции релиза |
|  | 02\_BIN | \*.sh | Да | Скрипты инсталляции релиза |
|  | 02\_BIN/db | \*.sql  \*.sh | Нет | Cкрипты и sql-файлы, необходимые при инсталляции изменений для СУБД |
|  | 02\_BIN/was | \*.sh | Нет | Скрипты was и bash-скрипты изменения конфигурации сервера приложений |
|  | 03\_MODEL | Data\*.zip  Model.zip | Да | Архивы с моделью данных и контентом |
|  | 04\_APP | \*.ear  \*.war | Нет | Приложения для инсталляции на сервер приложений |
|  | 04\_SRC | \*.zip | Да | Исходные коды |
|  | 05\_DOCS | Руководство администратора  Руководство пользователя | Да | Документация к релизу |
|  | 06\_UTILS | \*.jar | Нет | Утилиты сборки |

\*Примечание – В случае отсутствия необходимости включать часть папок в релиз, не следует добавлять пустые директории в сборку и изменять нумерацию остальных директорий.

* + 1. Развертывание
* Дистрибутив должен быть универсальным в части целевой среды развертывания.
* Конфигурация среды развертывания должна задаваться через файлы свойств.
* При установке дистрибутива должен быть предоставлен выбор по целевой среде развертывания: тестовый стенд, продуктивный стенд.
* Установка дистрибутивов должна быть максимально автоматизирована.
* Все автоматические шаги установки релиза должны отражаться в подробном журнале выполняемых действий.
* Все неавтоматизированные шаги установки релиза должны быть описаны в руководстве по установке релиза.
* Для каждого неавтоматизированного шага установки релиза должна быть указана пользовательская роль или учетная запись (согласно руководству администратора и руководству пользователя), под которой должен быть выполнен шаг установки.
* Руководство по установке релиза должно содержать перечень версий Системы, на который возможен накат новой сборки.
* Журналы импорта модели данных не должны содержать ошибок.
  1. Формат файлов сборки

При сборке обновления должны быть выполнены правила по формату файлов релиза:

* Sql-скрипты должны быть сохранены в кодировке UTF-8.
* Bash-скрипты должны содержать переносы строк без символов возврата каретки;
* Все пути установки файловой системы в скриптах и конфигурационных параметрах должны быть прописаны в соответствии с сервером установки. В случае если релиз предполагает установку на несколько серверов, то скрипты должны быть выполнены в нескольких вариантах или должны быть настроены через properties-файлы.
* Sql-скрипты должны быть созданы с учетом запуска под административным пользователем СУБД из консоли.
* При копировании текстовых файлов на linux-машины должна быть учтена необходимость бинарного копирования для предотвращения потери кодировки файлов.
* В случае необходимости должны быть учтены и прописаны команды выдачи прав на исполнение скриптов установки и права записи и создания файлов журналов.
  1. Нумерация релизов

Формат номеров релизов поддерживается в виде \*\*.\*\*\*.\*\*, а именно: <номер релиза>.<номер обновления>.<номер патча>. На текущий момент номер обновления заполняется тремя нулями. Номер релиза должен быть прописан в каждом объекте развертывания релиза (см. п. 7.2).

Следует поддерживать восходящую последовательность нумерации обновлений. Выпуск обновлений с одинаковым номером недопустим. Номер релиза фиксируется в случае отправки релиза вне зависимости от того, был ли он установлен. В случае если релиз был собран, но не передан, то не следует пропускать данный номер релиза при сборке следующего обновления.

1. Тестирование

Для проекта версия приложения (релиз) считается готовой, если выполнено следующие:

1. Отсутствуют ошибки при развертывании обновления Системы.
2. Отсутствуют ошибки высокого и среднего приоритета при проведении функционального, регрессионного (тестирование направленное на обнаружение ошибок в уже протестированных места) или экспресс-тестирования.
3. Разработана и является актуальной документация по Системе: Руководство администратора, Руководство пользователя.

Тестирование релиза является обязательным этапом разработки системы. При этом необходимо проведение тестирования функциональности системы и тестирование процесса развертывания релиза. В частном случае следует проводить нагрузочное тестирование модулей.

Распределение ответственности за этапы проведения тестирования приведена в

Таблица *6*:

Таблица 6 Этапы тестирования Системы

|  |  |  |
| --- | --- | --- |
| Этап | Исполнитель | Примечание |
| Внутренне тестирование у субподрядчика | Субподрядчик | Функциональное тестирование  Тестирование развертывания  Экспресс-тестирование (по необходимости). |
| Внутренне тестирование | Исполнитель | Функциональное тестирование  Тестирование развертывания  Экспресс-тестирование (по необходимости). |
| Передачи релиза на тестовый контур заказчика | Заказчики | Функциональное тестирование  Интеграционное тестирование  Регресс |
| Передачи релиза на продуктивный контур заказчика | Заказчики | - |

* 1. Внутреннее тестирование исполнителя

Внутренне тестирование проводит Компания-разработчик Системы. Обязательными этапами внутреннего тестирования являются:

* тестирование новой функциональности;
* тестирование установки релиза.

По результатам этапа проекта, в котором были проведены значительные изменения функциональности Системы, в рамках внутреннего тестирования может потребоваться проведение экспресс-тестирования Системы. Решение о необходимости проведения экспресс-тестирования должно приниматься менеджерами проекта.

Функциональное тестирование как часть внутреннего тестирования должно проводиться по списку задач реализованных в релизе Системы. Список задач должен быть взят из Redmine.

Все ошибки внутреннего тестирования релиза должны быть зафиксированы в протоколе в соответствии с шаблоном. Шаблон протокола размещен в Redmine (Документы / Общая документация / Разработка). Протокол внутреннего тестирования является обязательным документам поставки релиза.

Внутренне тестирование должно проводиться на специально выделенном стенде, повторяющем конфигурацию стенда Заказчика. Выполнение условия схожести конфигураций стенда внутреннего тестирования и стенда установки обновления на стенды Заказчика в значительной степени определяет успех выпуска новой версии Системы, в связи с чем должны соблюдаться следующие правила:

* В случае выпуска НЕПОЛНОГО релиза на стенде внутреннего тестирования должна быть развернута версия Системы, совпадающая с версией системы на стенды Заказчика.
* Настройки конфигурации Системы должны быть максимально приближены к настройкам целевого стенда установки.
* Стенд внутреннего тестирования должен полностью соответствовать по версиям ПО и аппаратной платформе стенду продуктива Заказчика.
  1. Внутренне тестирование у субподрядчика

Внутренне тестирование у субподрядчика проводит Компания субподрядчик Системы на своих серверах. Параметры совпадает с пунктом 0.

* 1. Критерии качества приложения

При выпуске релиза версия разрабатываемого приложения проверяется в соответствии с критериями качества. Критерий качества определяет готовность релиза.

Критерии:

1. Разработан весь запланированный функционал в соответствии с планом релиза( если план релиза написан и согласован).
2. Полнота и непротиворечивость исполнения программы в соответствии с документами на систему и задачами из Redmine.
3. Стабильность работы функционала.
4. Соответствие разработки с внутренними стандартами, описанными в данном документе.
   1. Функциональное тестирование

Функциональное тестирование обновления Системы предназначено для повторной проверки нового функционала, включенного в обновление, и проверки исправлений ошибок, зафиксированных в предыдущих релизах Системы. Функциональное тестирование является обязательным этапом в сборке любого типа обновления Системы (релиз, полный релиз, обновление).

Функциональное тестирование входит в этап внутреннего тестирования и в процесс переноса Системы на стенды Заказчика. Функциональное тестирование как часть внутреннего тестирования проводит исполнитель и субподрядчик в части разработанного функционала.

Приложение А. Генерация ключей доступа

Для генерации ключей доступа необходимо выполнить следующие действия:

1. Установить программу puttygen (входит в состав putty и WinSCP).
2. После завершения установки запустить программу puttygen, нажав на кнопку «Generate» в соответствии с Рис. 2.

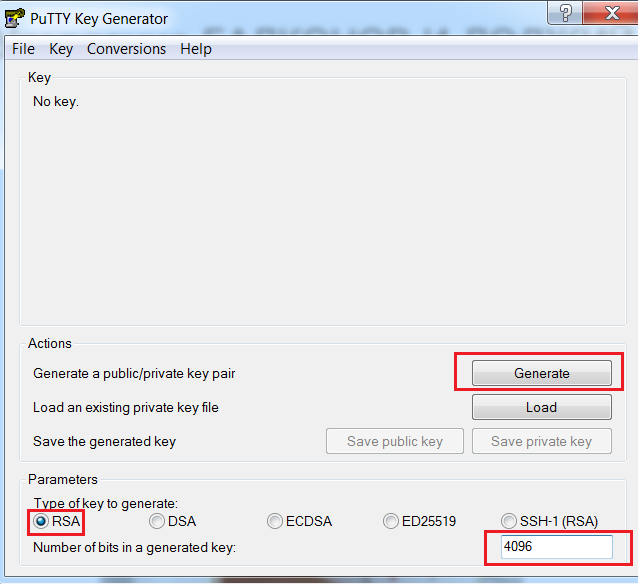


Рис. . Запуск программы

1. Двигать мышью над пустым полем окна программы, пока полоса прогресса вверху не дойдёт до конца.
2. Для идентификации изменить комментарий ключа на имя разработчика?. Не использовать в наименовании символы кириллического алфавита (Рис. 3).

  
Рис. . Изменение комментария ключа

1. Придумать и ввести пароль ключа в поля «Key passphrase» и «Confirm passphrase» (можно не вводить) как показано на Рис. 4.



Рис. . Ввод пароля ключа

1. Сохранить приватную часть ключа. С целью уменьшения вероятности доступа к ключу третьих лиц ни в коем случае не сохраняйте данную часть ключа на рабочий стол или в папку «Мои документы». Для сохранения нажмите на кнопку «Save private key» (Рис. 5).

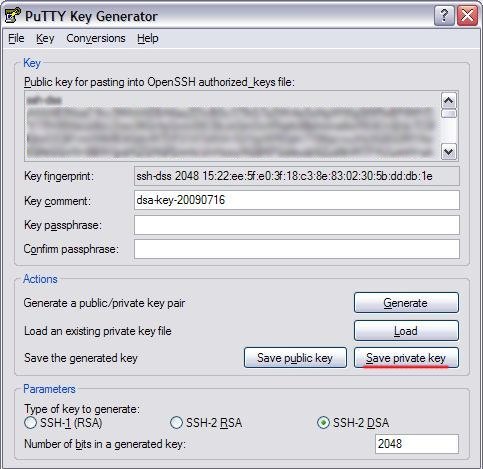
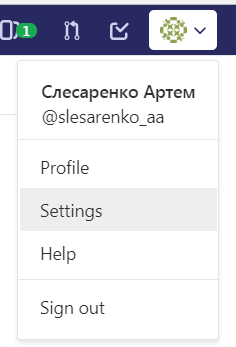


Рис. . Сохранение приватной части ключа

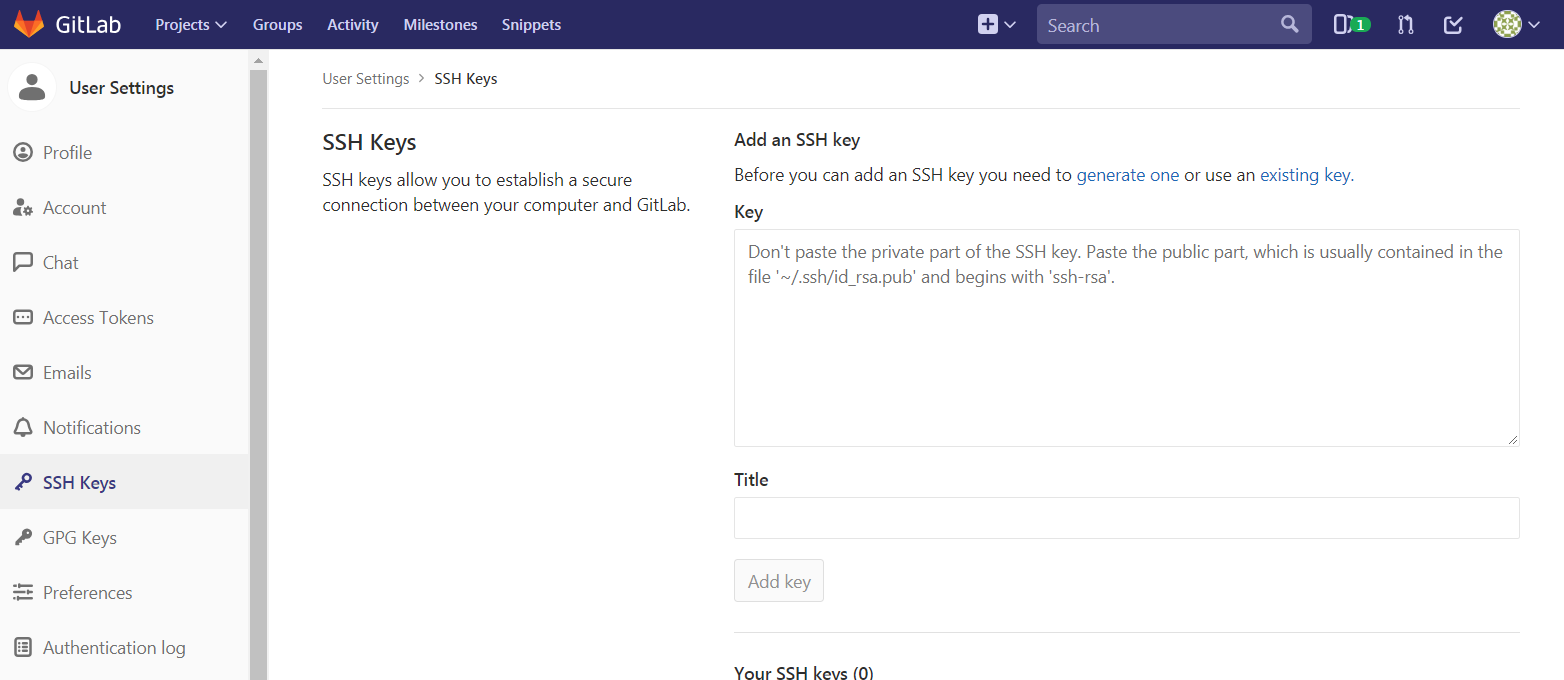
1. После сохранения приватной части ключа необходимо сохранить публичную часть ключа. Данная часть в обязательном порядке сохраняется в текстовом формате (txt), а не в формате редактора (doc, docx, odt, и т.п.).Публичная часть ключа может быть доступна третьим лицам. Для сохранения данной части необходимо выделить всё текстовое поле «Public key for passing...», затем нажать сочетание клавиш Ctrl+C или в контекстном меню над данным полем выбрать функцию «Копировать». Далее необходимо открыть текстовый редактор (желательно Notepad) и выбрать в меню функцию «Редактировать/вставить». После вставки следует убедиться, что вставленный текст записан в одну длинную строку. Данный файл возможно сохранить в любое удобное место на компьютере.

Приложение Б. Добавления ключа в gitlab

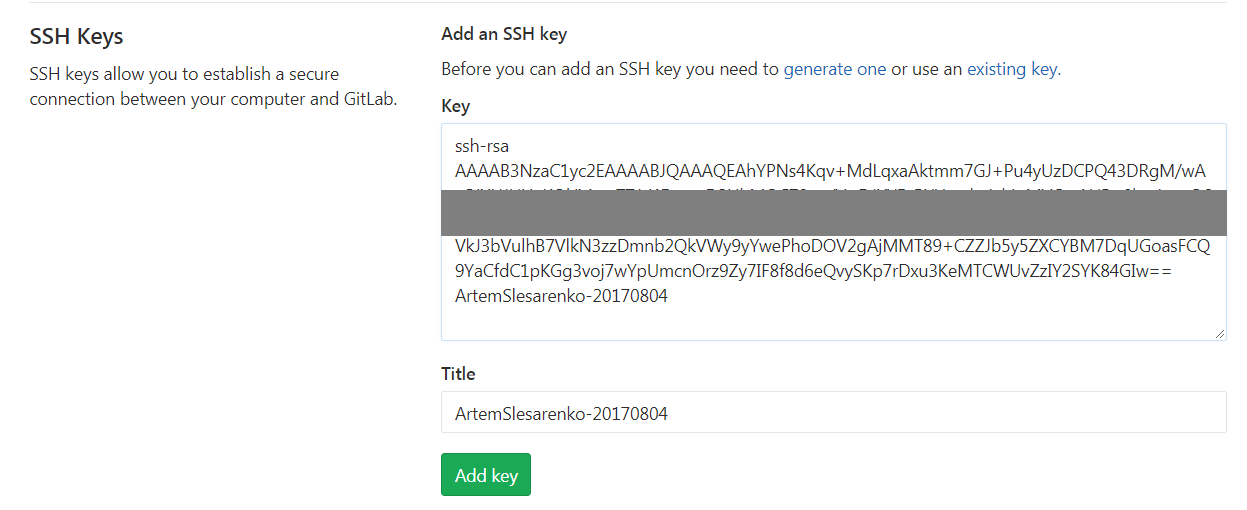
* 1. Необходимо авторизоваться в gitlab.
  2. Вверху-справа выберите нажмите на иконку.
  3. В выпадающем меню нажмите «Settings»



* 1. Откроется окно настроек пользователя.
  2. В левой панели навигации выберите «SSH Key»

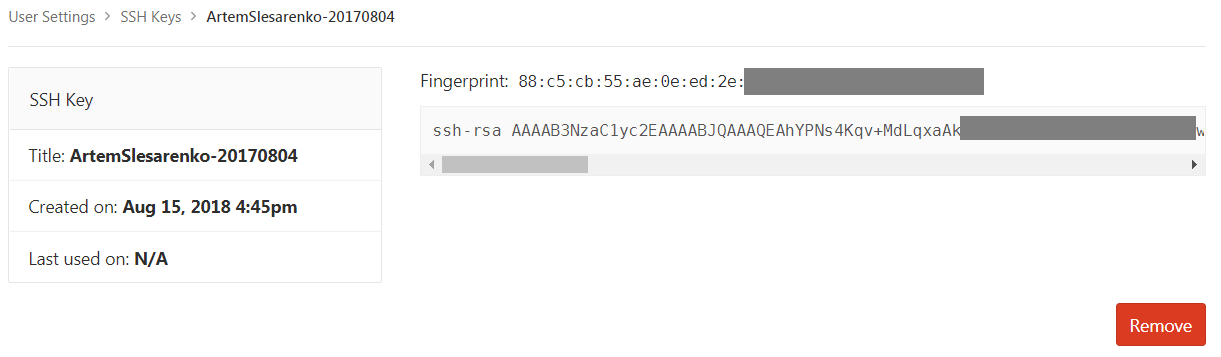


* 1. В поле «Key» скопируйте открытый ключ, который был сгенерирован puttygen(в Приложении А).



* 1. Нажмите на кнопку «Add key». Система должна добавить ваш ключ.

Если система добавит ключ, то на экране будет отображена форма как на картинке ниже.



При возникновении проблем, следуйте рекомендациям системы.

Если добавить ключ в gitlab и Pagant, то для скачивания и комита изменения не нужно бедует водить логин и пароль.

Приложение С. Оформление исходного кода приложений

Пожелания будут дорабатываться, можете присылать свои пожелания.

1. **Общие пожелания**
2. По функционалу.

* Бизнес логику выделять в отдельные классы.
* В функциях желательно обрабатывать на null входные значения.

1. Оформление комментариев

* Комментарии в коде писать на русском языке. Потом попросят создать и выложить документацию.
* Нужно писать комментарии у классов и методов – зачем он и что делают. Немного, но для понимания проще.

1. Картинки

* Картинки храните отдельно от кода, в отдельной папке.
* Сейчас можно использовать любые подходящие картинки. После разработки основной концепции функционала будем привлекать дизайнера для доработки дизайна и картинок.

1. Все проекты будут локализованы на несколько языков. Сейчас закладываемся на русский и английский. Текст нужно выносить отдельно от кода.
2. **Оформление java**
3. Желательно придерживаться единого формата оформления кода для java это- Java Code Conventions <https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>
4. **Оформление Java Script**
5. Нужно стремиться к уменьшению java script кода, что бы формы грузились быстрее <https://habr.com/company/ruvds/blog/419369/>
6. Для того чтобы веб-проекты работали быстро, нужно загружать лишь JavaScript-код, необходимый для текущей страницы.
7. Так, например, рассчитывайте на то, что размер минифицированного и сжатого JS-кода, необходимого для страниц, предназначенных для мобильных устройств, не должен превышать [170 Кб](https://infrequently.org/2017/10/can-you-afford-it-real-world-web-performance-budgets/).
8. Научитесь проводить [аудит](https://nolanlawson.com/2018/03/20/smaller-lodash-bundles-with-webpack-and-babel/) JavaScript-бандлов (сборок) и сокращать их размер.
9. Использовать разделение кода ([code splitting](https://survivejs.com/webpack/building/code-splitting/) <https://survivejs.com/webpack/building/code-splitting/>). [Angular](https://angular.io/guide/lazy-loading-ngmodules) <https://angular.io/guide/lazy-loading-ngmodules>.
10. Оформление кода <https://habr.com/company/ruvds/blog/418631/>

* Используйте понятные имена переменных и функций.
* Пишите короткие функции, решающие одну задачу.
* Пишите комментарии (Документируйте код).
* Не повторять код.

1. Подумайте об использований правил Сэнди Метц:

* Классы не должны быть длиннее 100 строк кода.
* Методы и функции не должны быть длиннее 5 строк кода.
* Методам следует передавать не более 4 параметров.
* Контроллеры могут инициализировать лишь один объект.

Приложение Д. Оформление документации

Для оформления документов желательно использовать одну из разработанных документов. Сейчас за основу можно использовать Функциональные требования переводчика, альбом форм по ИС файлового хранения или ТЗ.

1. Использовать стили оформления документов. Стили из одного документа можно экспортировать в базовый шаблон(normal) и потом импортировать в текущий документ с целью их использования.
2. У больших рисунков не нужно делать обтекание текстом. Вставляйте отдельно рисунок и отдельно текст.
3. Маленькие риски (например: изображение кнопок) можно вставлять в текст с обтеканием.
4. Если на рисунке с краев много пустого места, то это нужно обрезать. Что бы обрезать нажмите на рисунок двойным кликом левой кнопки. В меню сверху есть кнопка «Обрезка».
5. Разбивайте текст на пронумерованные разделы (заголовки, списки и т.д.), так людям проще воспринимать информацию и потом общаться в разговоре.