УТВЕРЖДАЮ:
Генеральный директор
Акционерного общества
«Эволента»

_____ С.Н. Лизин
«___» _____ 2018 г.

РУКОВОДСТВО АДМИНИСТРАТОРА

Автоматизированная информационная система многофункциональных центров предоставления государственных и муниципальных услуг Ивановской области

На 41 листах

Иваново 2018

Содержание

СОКРАЩЕНИЯ

АИС	Автоматизированная информационная система
ЕЛК	Единый личный кабинет
ЕПГУ	Федеральная государственная информационная система «Единый портал государственных и муниципальных услуг (функций)»
МФЦ	Многофункциональный центр предоставления государственных и муниципальных услуг
Система	Информационная система для многофункционального центра оказания государственных и муниципальных услуг Ивановской области
СМЭВ	Федеральная государственная система «Система межведомственного электронного взаимодействия»

1. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ

Программный комплекс предназначен для автоматизации основных административно-управленческих процессов предоставления государственных и муниципальных услуг, а также оперативного доступа всех участников процессов предоставления государственных (муниципальных) услуг к соответствующей информации.

Основными целями использования Системы являются:

– повышение открытости и эффективное исполнение государственных функций и предоставления государственных и муниципальных услуг;

– обеспечение участия ведомственных информационных систем в межведомственном электронном взаимодействии в рамках электронного правительства области;

– создание единой системы приема, регистрации и выдачи необходимых документов организациям и гражданам при предоставлении государственных и муниципальных услуг органами власти в сети Интернет;

– обеспечение комфортных условий доступа к государственным и муниципальным услугам для населения и хозяйствующих субъектов при их минимальном участии в процессах предоставления услуг;

– повышение эффективности оказания государственных и муниципальных услуг физическим и юридическим лицам;

– минимизация использования бумажного документооборота.

Пользователями Системы являются сотрудники МФЦ, ответственные за предоставление государственных и муниципальных услуг, осуществление межведомственного взаимодействия.

Система обеспечивает автоматизацию процесса приема и регистрации, а также проверки комплектности и правильности заполнения как бумажных, так и электронных документов, подаваемых заявителями для получения государственных и муниципальных услуг. При этом заявление может подаваться, в том числе от нескольких лиц, а в качестве заявителей могут выступать как физические лица, так и юридические лица или индивидуальные предприниматели (в т.ч. крестьянские фермерские

хозяйства). В отдельных случаях заявление может подаваться не лично заявителем, а его уполномоченным представителем. Также в системе ведется учет консультирования заявителей по вопросам предоставления государственных и муниципальных услуг.

Система обеспечивает автоматизацию межведомственного информационного взаимодействия каждого объекта автоматизации с другими государственными органами и органами местного самоуправления, участвующими в предоставлении соответствующих государственных и муниципальных услуг. Автоматизация бумажного документооборота обеспечивается в части учета движения бумажных документов между органами и идентификации местоположения каждого документа (органа в котором находится документ). Автоматизация межведомственного электронного взаимодействия включает интеграцию с системой межведомственного электронного взаимодействия (СМЭВ).

Система обеспечивает возможность электронной подачи заявлений на предоставление государственных или муниципальных услуг посредством Единого портала государственных и муниципальных услуг, а также передачу статусов (событий), возникающих в процессе предоставления государственных и муниципальных услуг в ЕЛК на ЕПГУ.

2. ВЫПОЛНЕНИЕ ПРОГРАММЫ

2.1. Вход в программу

Откройте интернет-браузер Chrome 62 и выше.

Введите в адресной строке ссылку на программу, которую Вам должен сообщить администратор, и нажмите клавишу ввода.

В результате выполненных действий открывается страница входа в Систему (Рисунок 1 - Вход в Систему).

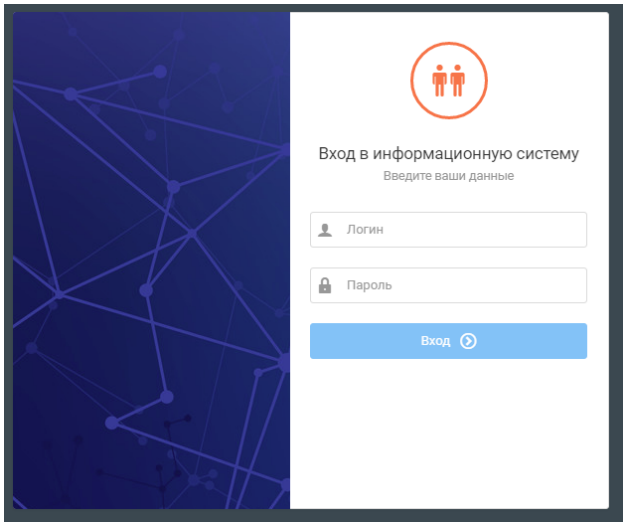

Рисунок 1 - «Вход в Систему»

После успешного входа в систему, требуется выбрать организацию, под которой будет осуществляться ведение дел в приложении «АИС» (Рисунок 2 – Выбор организации, под которой будет осуществляться ведение дел в приложении «АИС»).

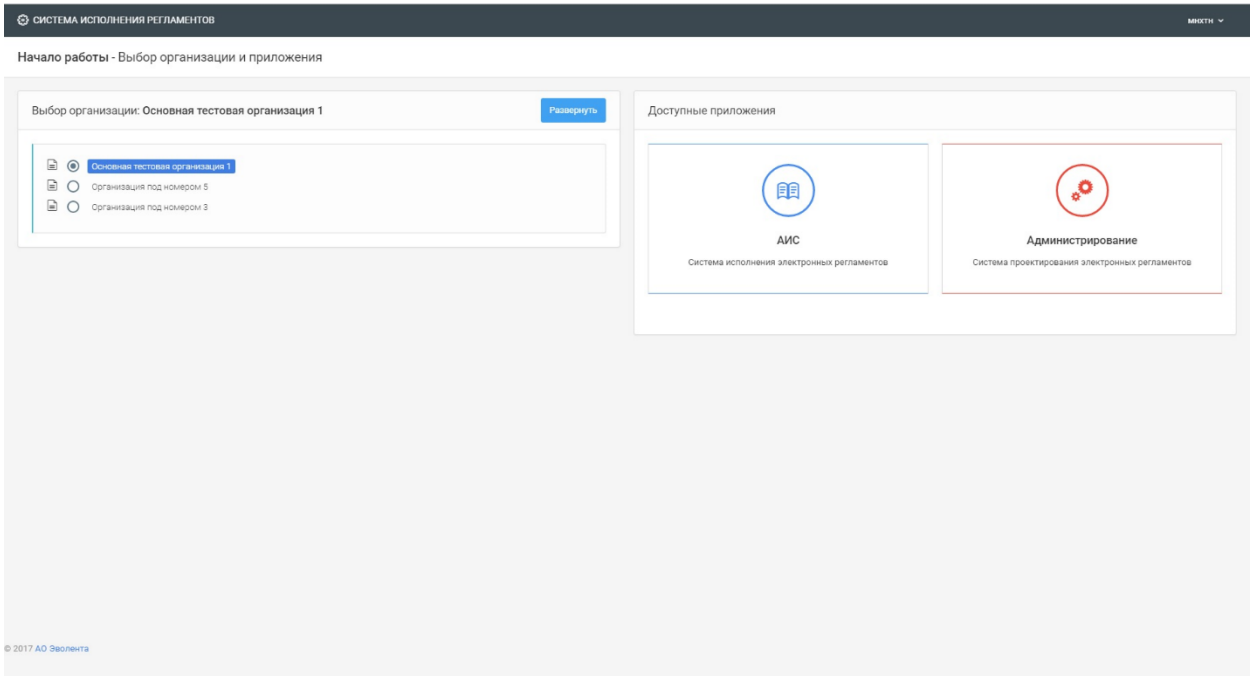

Рисунок 2 – Выбор организации, под которой будет осуществляться ведение дел в АИС

Из списка доступных приложений выбрать Администрирование. В результате осуществлён переход в пункт бокового меню «Пользователи» (Рисунок 3 - Вид. Раздел бокового меню «Пользователи»).

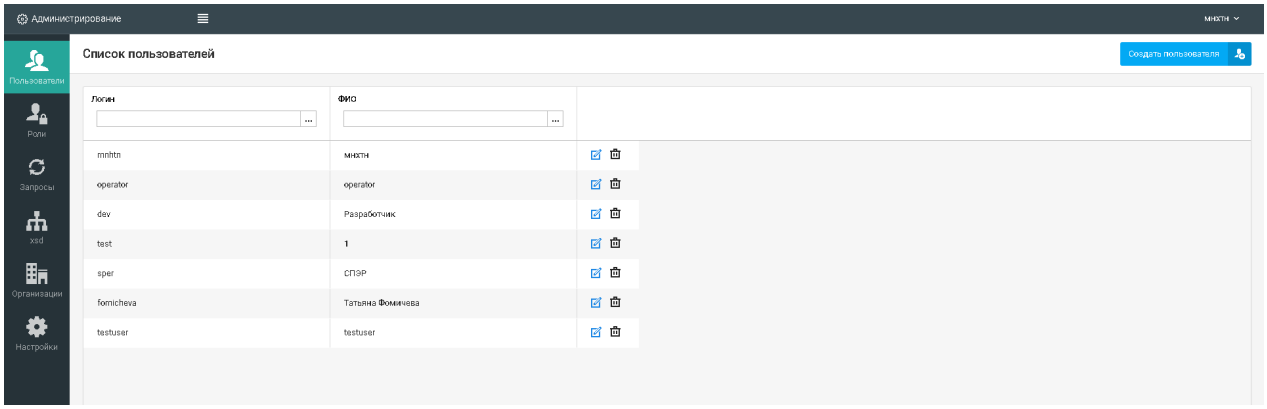

Рисунок 3 - Вид. Пункт бокового меню «Пользователи»

2.2. Описание элементов экранных форм

Основные элементы экранных форм приведены на Рисунке 4 - Основные элементы экранных форм:

– Боковое меню. Содержит набор вкладок.

– Панель перечня элементов. Содержит краткую информацию об элементах соответствующей вкладки (на рисунке указаны элементы вкладки «Пользователи»).

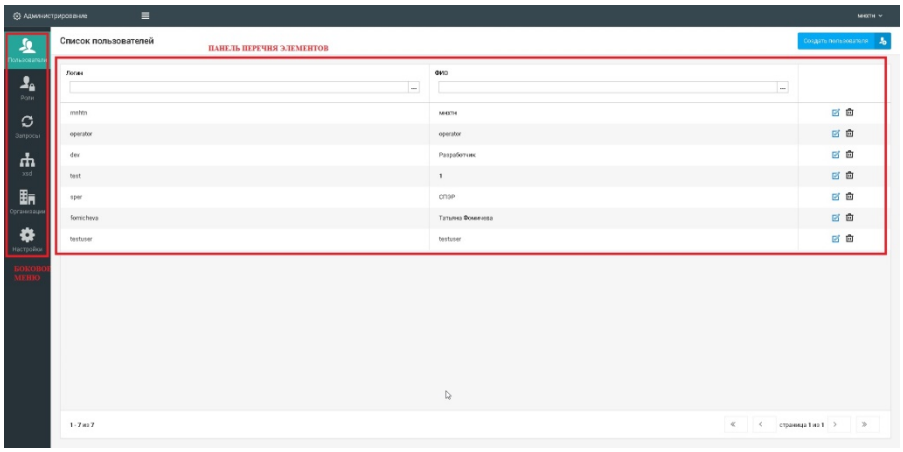

Рисунок 4 - Основные элементы экранных форм

Боковое меню содержит набор следующих вкладок:

1. Пользователи. На этой вкладке происходит создание, редактирование, удаление учётных записей пользователей Системы.

2. Роли. На этой вкладке происходит создание, редактирование, удаление ролей учётных записей пользователей Системы путём настройки соответствующего набора полномочий (Рисунок 5 - Вид. Пункт бокового меню «Роли»).

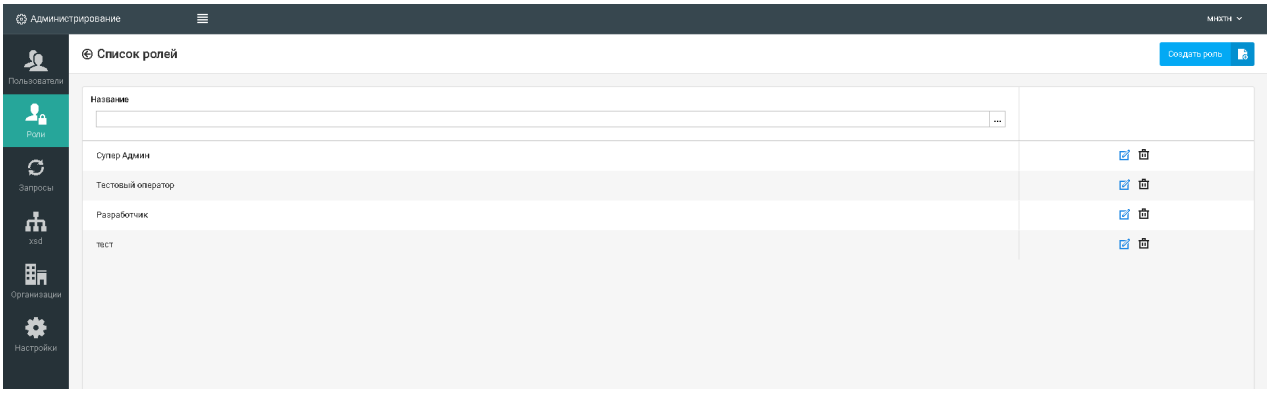

Рисунок 5 - Вид. Пункт бокового меню «Роли»

3. Запросы. На этой вкладке происходит работа по настройке межведомственных запросов, используемых в приложении «АИС» (Рисунок 6 - Вид. Пункт бокового меню «Запросы»).

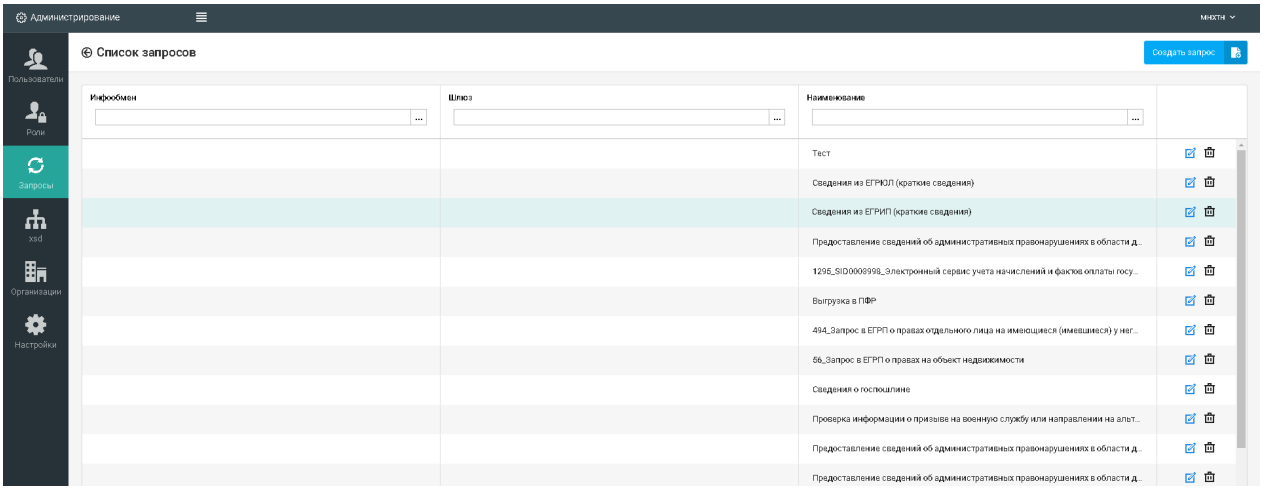

Рисунок 6 - Вид. Пункт бокового меню «Запросы»

4. XSD. На этой вкладке происходит работа с XSD-элементами интерактивных форм.

5. Организации. На этой вкладке происходит создание, редактирование, удаление организаций (Рисунок 7 - Вид. Пункт бокового меню «Организации»).

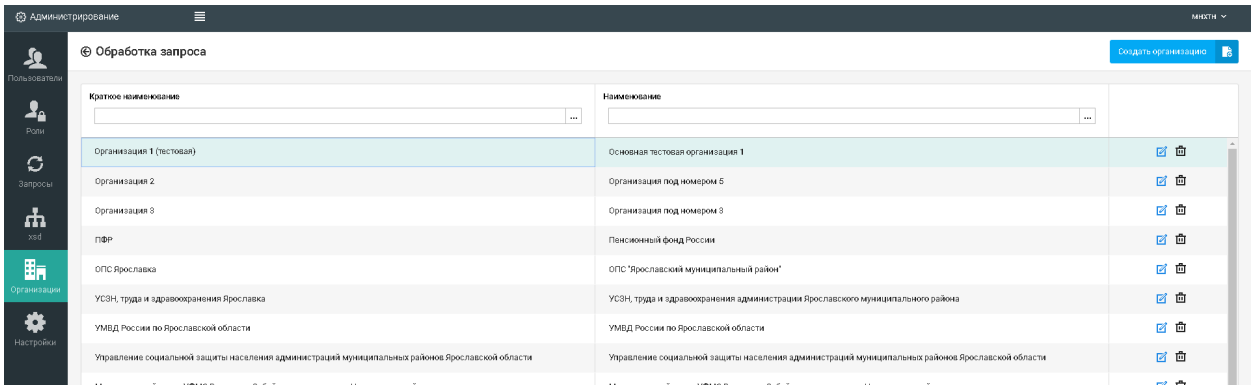

Рисунок 7 - Вид. Пункт бокового меню «Организации»

6. Настройки. На этой вкладке производятся базовые настройки Системы, параметры взаимодействия со СПЭР, управление обновлением данных из СПЭР (Рисунок 8 - Вид. Пункт бокового меню «Настройки»).

Рисунок 8 - Вид. Пункт бокового меню «Настройки»

2.3. Управление пользователями

2.3.1. Поиск пользователя

Для поиска нужного пользователя в списке следует воспользоваться поисковой строкой. Поиск можно осуществлять по логину и (или) ФИО. В результате список будет отфильтрован в соответствии с введенным значением. Для удобства Система разбивает длинные таблицы пользователей постранично. Для навигации между страницами служат элементы управления, расположенные внизу каждой таблицы (Рисунок 9 – Поиск пользователя и навигация по страницам таблиц со сведениями о пользователях Системы). Для перехода на нужную страницу можно использовать кнопки со стрелками.

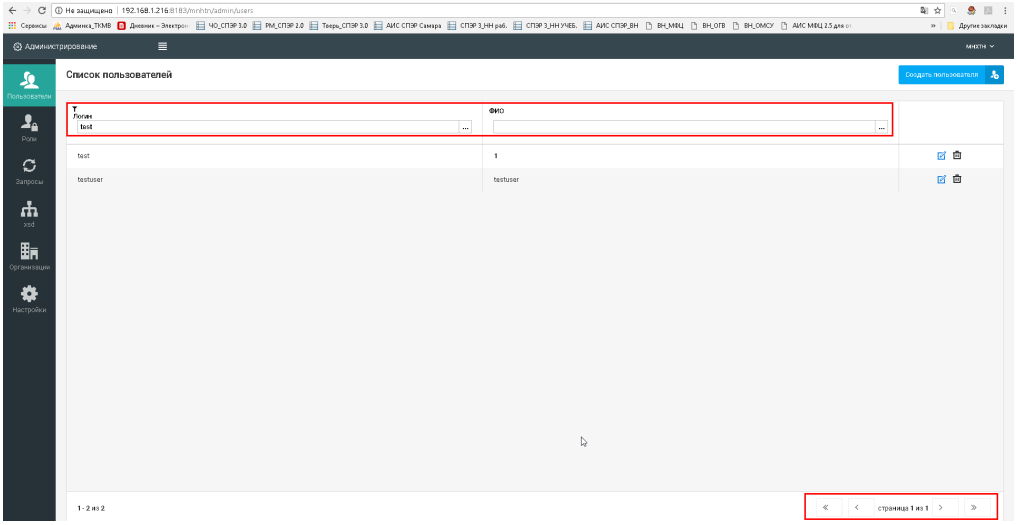

Рисунок 9 – Поиск пользователя и навигация по страницам таблиц со сведениями о пользователях Системы

2.3.2. Создание пользователя

Администратор Системы может зарегистрировать нового пользователя путем нажатия кнопки «Создать пользователя». При этом открывается форма «Регистрации нового пользователя» (Рисунок 10 - Форма «Регистрация нового пользователя»).

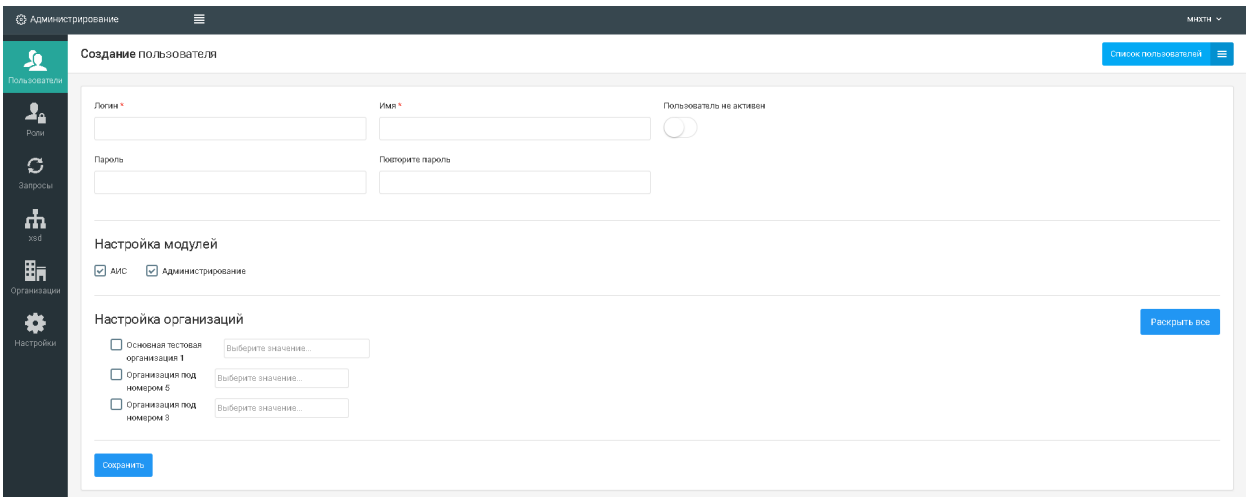

Рисунок 10 - Форма «Регистрация нового пользователя»

При создании пользователя следует заполнить следующие поля:

– Логин
– Пароль
– Повторите пароль
– Имя
– Пользователь не активен (по умолчанию переключатель выключен)
– Настройка доступа к модулям (по умолчанию доступны модули «АИС», «Администрирование»)
– Настройка организаций. Это обязательный для заполнения блок. К каждому пользователю должна быть привязана хотя бы одна организация, а также указана роль пользователя при работе в Системе под данной организацией. Для отображения всего списка организаций необходимо нажать на кнопку «Раскрыть все».

Если пользователю привязано несколько организаций, в Системе возможна работа под каждой из них. Для переключения между

добавленными организациями одного пользователя, необходимо нажать в правом верхнем углу на имя пользователя, после чего нажать на «Выбор приложения» (Рисунок 11 – Смена приложения, под которой работает пользователь Системы).

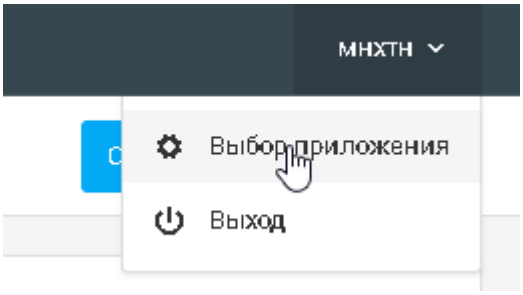

Рисунок 11 – Смена приложения, под которой работает пользователь Системы

В результате будет осуществлён переход на главную страницу, где следует сменить организацию и выбрать приложения для продолжения работы в Системе (Рисунок 12 - Переход между организациями пользователя).

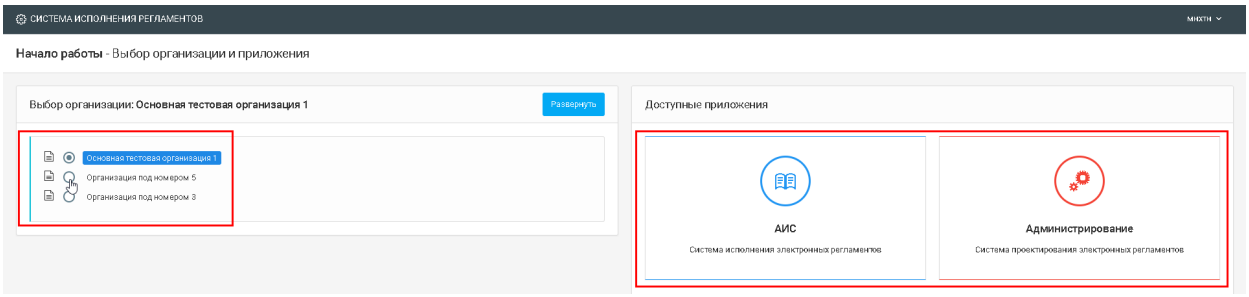

Рисунок 12 - Переход между организациями пользователя

2.3.3. Редактирование настроек пользователя

Администратор Системы может редактировать настройки существующих пользователей Системы путем нажатия кнопки «Редактировать» в строке таблицы пользователей.

При редактировании настроек пользователя (Рисунок 13 - Редактирование профиля пользователя) обязательными для заполнения

являются те же поля, что и при регистрации нового пользователя, кроме поля «Изменение пароля».

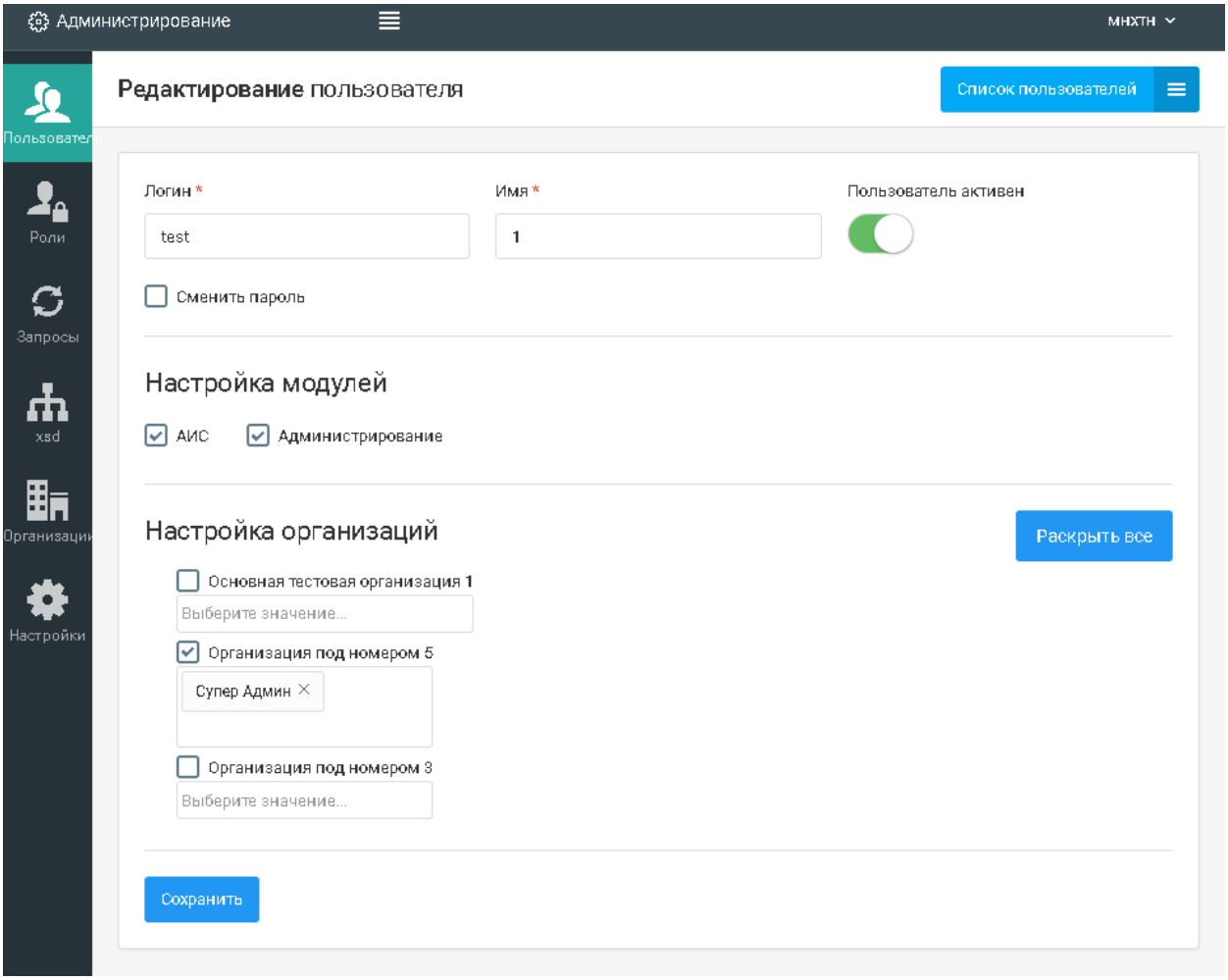

Рисунок 13 - Редактирование профиля пользователя

В случае если вы хотите изменять пароль, то следует нажать на чекбокс «Сменить пароль». В результате отобразятся скрытые поля «Пароль», «Повторить пароль» (Рисунок 14 - Редактирование профиля пользователя со сменой пароля).

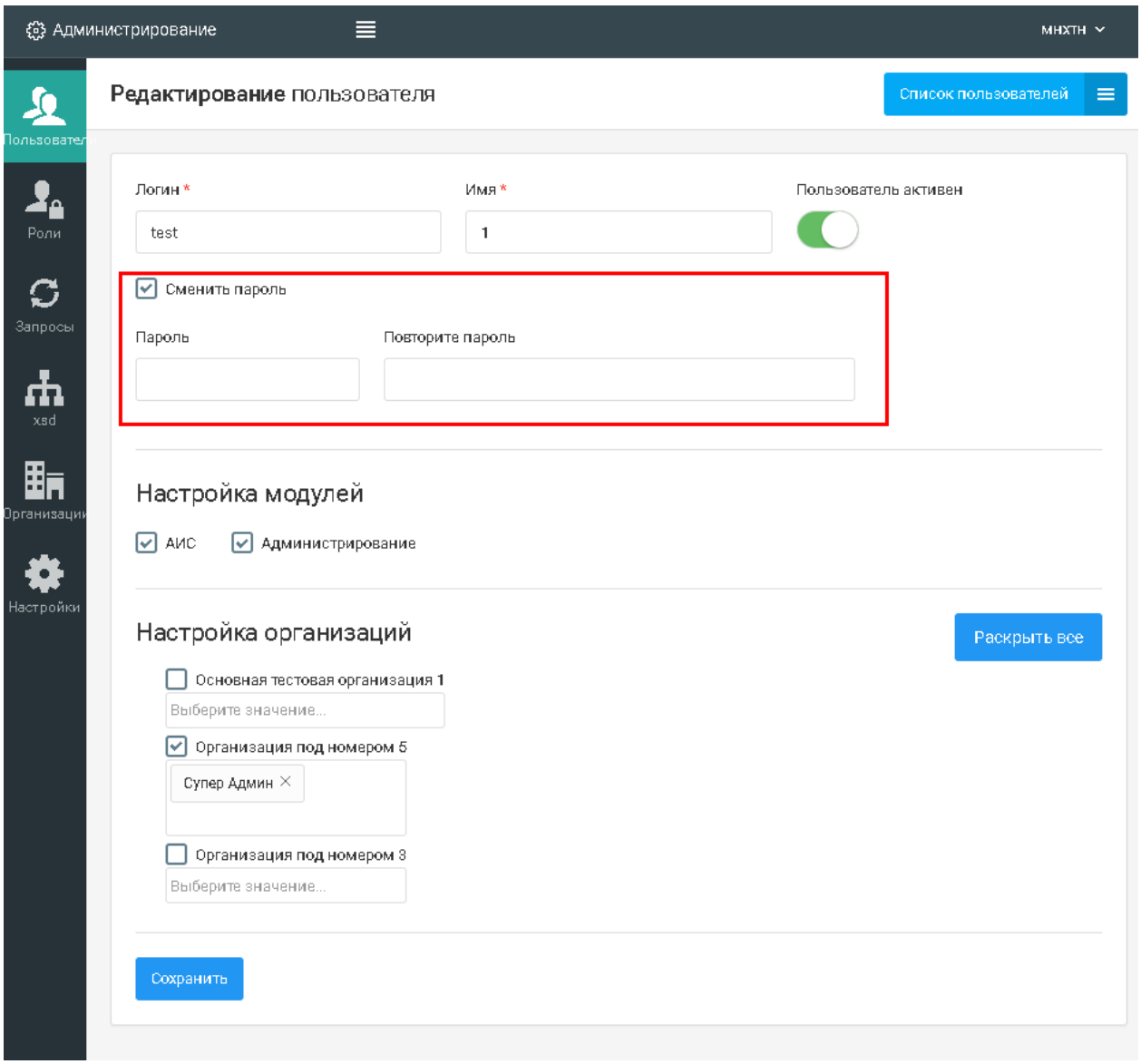

Рисунок 14 - Редактирование профиля пользователя со сменой пароля

2.3.4. Удаление пользователя

Существует два способа для исключения активности пользователя:

– удаление учётной записи путём нажатия на пиктограмму  в соответствующей строке списка пользователей;

– деактивация учётной записи путём перевода переключателя

в профиле пользователя.

В случае удаления сведения об учётной записи будут удалены из списка пользователей, в случае деактивации учётной записи она будет

присутствовать в списке пользователей и в любой момент профиль пользователя со всеми ранее произведёнными настройками можно будет вернуть в работу.

Приложение

Инструкция по развертыванию ИС МФЦ.

1. Обзор системы.

В Системе используется концепция микросервисов. Концепция микросервисов предполагает, что весь функционал приложения поделен на части и каждая часть выделена в отдельный сервис. Благодаря этому появляется возможность «горизонтального» масштабирования системы и динамического обновления компонентов без остановки работы всего приложения.

Для упрощения управления десятками микросервисов в архитектуру системы введены некоторые управляющие сервисы и сервис мониторинга нагрузки:

- сервис конфигурирования
- сервис оркестровки
- сервис маршрутизации
- сервис мониторинга

Сервис конфигурирования предназначен для централизованного хранения и раздачи конфигурационных файлов всем остальным сервисам системы.

Сервис оркестровки предназначен для предоставления возможности сервисам найти друг друга. Изначально, при старте, каждый сервис знает только расположение сервера конфигурации. От него он получает свою текущую конфигурацию и адрес сервиса оркестровки. Подключаясь к сервису оркестровки, каждый сервис регистрируется в нем, передавая информацию о своем расположении (хост и порт по которому его можно найти, а также свое имя) и получает список уже зарегистрированных сервисов. Благодаря этому, микросервисам нет необходимости знать где и как конкретно в сетевой инфраструктуре располагаются другие микросервисы, они могут взаимодействовать друг с другом зная только имена требуемых им сервисов – данные же об их сетевом расположении динамически запрашиваются от сервиса оркестровки. Каждый микросервис раз в 30 секунд обновляет свою информацию о других микросервисах от сервиса оркестровки и подтверждает, что он жив и может быть вызван другими сервисами.

Сервис маршрутизации представляет из себя точку входа приложения – по сути - прокси, которая маршрутизирует внешние запросы к конкретным микросервисам, реализующим тот или иной функционал системы.

Сервис мониторинга предназначен для динамического мониторинга нагрузки на конкретный микросервис. Каждый микросервис предоставляет поток динамической метрики, к которому может быть подключен сервис мониторинга, который визуализирует нагрузочную информацию, получаемую от сервиса.

Непосредственно приложение СИЭР состоит из следующих компонентов и сервисов:

- **«статика»**, клиентские компоненты для отображения в браузере пользователей
- **сервис ядра системы**, реализует обработку основных вызовов пользователей для работы с сущностями (поиск/создание/обновление/удаление сущностей)
- **сервис бизнес-логики**, реализует API основной логики работы учетной системы

– **сервис шлюза файлохранилища**, реализует API для работы с файлами, загружаемыми или создаваемыми при работе системы (загрузка файлов в хранилище, получение их из хранилища, удаление)

– **сервис нумерации**, предоставляет другим сервисам возможность получать уникальные автоинкрементные идентификаторы для своих внутренних нужд. Является синглетом и не масштабируется горизонтально (запрещается поднимать более одного инстанса этого сервиса, чтобы не нарушить автоинкрементные последовательности)

– **сервис рендеринга форм**, взаимодействует с рендером, предоставляет API для генерации печатных форм другим микросервисам

– **рендер форм**, реализует логику рендеринга печатных форм

– **СМЭВ-клиент**, предоставляет API для отправки межведомственных запросов, реализует логику взаимодействия с внешними интеграционными компонентами (шина интеграции, ESB) и удаленными сервисами

– **сервис взаимодействия со СПЭР-3**, предоставляет API для взаимодействия со СПЭР версии 3, реализует логику загрузки и обновления регламентов, печатных форм, обновления метаданных и прочей метаинформации, предоставляемой СПЭР-3

– **сервис архивации,** предоставляет API для асинхронной архивации данных дел (асинхронного перекладывания приложенных файлов к делу из основного хранилища в архивное)

2. Установка и настройка MongoDB

2.1. Установка MongoDB 3.6. на Centos 7.

2.1.1. Создаем файл .repo для репозитория монги:

```
sudo vi /etc/yum.repos.d/mongodb-org.repo
```

2.1.2. Заполняем его содержимое (скрипт для версии 3.6., если на момент разворачивания будет доступна более новая стабильная версия, в скрипте в наименовании и путях нужно сменить номер 3.6 на актуальный):

```
[mongodb-org-3.6]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-
org/3.6/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-3.6.asc
```

2.1.3. Сохраняем содержимое файла и выходим из vi :

```
<Esc>
:wq
<Enter>
```

2.1.4. Проверяем, что репозиторий доступен:

```
yum repolist
```

В списке репозиториев должен фигурировать репозиторий MongoDB Repository.

2.1.5. Устанавливаем MongoDb:

```
sudo yum install mongodb-org
```

2.1.6. Добавляем в автозапуск

```
sudo systemctl enable mongod
```

2.1.7. Запускаем:

```
sudo systemctl start mongod
```

2.2. Установка MongoDB на Ubuntu Server 18.04

Интегрируем публичный ключ репозитория

```
sudo apt-get update

sudo apt install -y mongodb
```

2.3. Первичная инициализация (создание базы данных и включение аутентификации).

Сразу после установки MongoDB сконфигурирована на работу по localhost с рутовым подключением без какой-либо авторизации (не запрашивается даже имя пользователя), что нас совершенно не устраивает. Необходимо провести реконфигурацию инстанса с включением аутентификации и разрешением сетевого взаимодействия.

2.3.1. Подключаеся к монге консольным клиентом:

```
mongo
```

Вывод консоли при успешном подключении (начальная часть):

```
MongoDB shell version v3.6.4
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.4
Welcome to the MongoDB shell.
…
```

2.3.2. Переключаемся на служебную базу данных:

```
use admin
```

Вывод консоли при успешной операции:

```
switched to db admin
```

2.3.3. Создаем пользователя-администратора «root»:

```
db.createUser(
  {
    user: "root",
    pwd: "pa$$w0rd",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

Вывод консоли при успешном создании:

```
Successfully added user: {
        "user" : "root",
        "roles" : [
                {
                        "role" : "userAdminAnyDatabase",
                        "db" : "admin"
                }
        ]
}
```

2.3.4. Проверяем, что пользователь создан:

show users

Вывод консоли при успешной проверке:

```
{
        "_id" : "admin.root",
        "user" : "root",
        "db" : "admin",
        "roles" : [
                {
                        "role" : "userAdminAnyDatabase",
                        "db" : "admin"
                }
        ]
}
```

2.3.5. Отключаемся от консоли базы данных:

```
exit
```

Вывод консоли при успешной операции:

```
bye
```

2.3.6. Останавливаем демон для дальнейшего включения авторизации:

```
systemctl stop mongod
```

2.3.7. Открываем конфиг MongoDB для редактирования:

```
vi /etc/mongod.conf
```

2.3.8. Изменяем секцию # network interfaces :

меняем `bindIp: 127.0.0.1` на `bindIp: 0.0.0.0` для включения возможности подключения с других серверов

2.3.9. Раскомментируем секцию #`security` и добавляем туда признак включения авторизации:

```
security:
  authorization: enabled
```

2.3.10. Сохраняем изменения и выходим из vi:

```
<Esc>
:wq
<Enter>
```

2.3.11. Запускаем демон MongoDB:

```
systemctl start mongod
```

2.3.12. Подключаемся консольным клиентом:

```
mongo
```

Вывод консоли при успешном подключении (начальная часть):

```
MongoDB shell version v3.6.4
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.4
```

2.3.13. Переключаемся на служебную базу данных:

```
use admin
```

Вывод консоли при успешной операции:

```
switched to db admin
```

2.3.14. Авторизуемся рутом:

```
db.auth("root","pa$$w0rd")
```

Вывод консоли при успешной авторизации:

```
1
```

2.3.15. Создаем БД приложения:

```
use sier
```

Вывод консоли при успешном переключении на БД sier:

```
switched to db sier
```

2.3.16. Создаем пользователя для подключения к базе данных:

```
db.createUser(
   {
     user: "sier",
     pwd: "sier123",
     roles: [ {role: "readWrite", db: "sier"} ]
   }
)
```

Вывод консоли при успешном добавлении пользователя:

```
Successfully added user: {
        "user" : "sier",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "sier"
                }
        ]
}
```

2.3.17. Создаем БД рендера:

```
use renderdb
```

Вывод консоли при успешном переключении на БД `renderdb`:

```
switched to db renderdb
```

2.3.18. Создаем пользователя для подключения к базе данных:

```
db.createUser(
   {
     user: "renderdb",
     pwd: "renderdb123",
     roles: [ {role: "readWrite", db: "renderdb"} ]
   }
)
```

Вывод консоли при успешном добавлении пользователя:

```
Successfully added user: {
        "user" : "renderdb",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "renderdb"
                }
        ]
}
```

2.3.19. Отключаемся от консоли базы данных:

```
exit
```

Вывод консоли при успешной операции:

```
bye
```

2.4. Конфигурирование MongoDb replica set для распределенных отказоустойчивых конфигураций на Centos 7.

Перед настройкой Replica Set на все серверы нужно установить и настроить MongoDB, согласно п.п.2.1-2.2. Пункты с созданием БД и добавлением пользователя для доступа к ней на второй и последующих репликах можно пропустить, т.к. при первичной инициализации replica set база данных и параметры авторизации будут реплицированы с первичного сервера.

Для обеспечения защищенного внутреннего взаимодействия между инстансами реплик необходимо сгенерировать ключевой файл, который после генерации необходимо скопировать на сервер каждой из реплик, независимо от того является ли он Primary, Secondary или Arbiter.

В текущей инструкции считается, что группа серверов состоит из трех инстансов – Двух инстансов БД и арбитра.

Примечание: MongoDB Replica-set может работать и без арбитра, но для быстрой смены Primary в случае падения текущего арбитр необходим. В его отсутствие смена Primary-сервера может занимать от нескольких десятков секунд до нескольких минут. Кроме того, есть сведения, что в конфигурациях без арбитра, существует ненулевая вероятность потерять часть данных при падении первичного сервера.

2.4.1. Подключаемся консольным клиентом:

```
mongo
```

Вывод консоли при успешном подключении:

```
MongoDB shell version v3.6.4
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.4
```

2.4.2. Переключаемся на служебную базу данных:

```
use admin
```

Вывод консоли при успешной операции:

```
switched to db admin
```

2.4.3. Авторизуемся рутом:

```
db.auth("root","pa$$w0rd")
```

Вывод консоли при успешной авторизации:

```
1
```

2.4.4. Добавляем роли, разрешающие управление репликацией для администратора MongoDB:

```
db.grantRolesToUser(
  "root",
   [
     { role: "clusterAdmin", db:"admin"},
     { role: "clusterManager", db:"admin"},
     { role: "clusterMonitor", db:"admin"},
     { role: "hostManager", db:"admin"}
```

```
    ]
)
```

2.4.5. Отключаемся от консоли базы данных:

```
exit
```

Вывод консоли при успешной операции:

```
bye
```

2.4.6. Генерим ключ при помощи OpenSSL:

```
openssl rand -base64 756 > /opt/mongokey/keyfile
```

2.4.7. Меняем разрешения доступа к файлу и владельца. Это ограничение MongoDB для *nix-систем, он не принимает ключевые файлы, доступ к которым возможен для неограниченного числа лиц:

```
chmod 400 /opt/mongokey/keyfile
```

```
chown mongod:mongod /opt/mongokey/keyfile
```

2.4.8. Открываем файл конфигурации для редактирования:

```
vi /etc/mongod.conf
```

2.4.9. Добавляем в секцию security путь до ключевого файла, раскомментируем секцию #replication и добавляем в нее произвольное имя, которое нужно присвоить replica-set:

```
security:
  authorization: enabled
  keyFile: /opt/mongokey/keyfile
replication:
  replSetName: sierrepl
```

2.4.10. Сохраняем изменения и выходим:

```
<Esc>
:wq
<Enter>
```

2.4.11. Перезапускаем инстанс MongoDB для принятия изменений конфигурации:

systemctl restart mongod

2.4.12. Копируем сертификат, сгенеренный в п.2.3.6 на сервер второго и последующего инстансов MongoDB, назначаем права доступа к нему, изменяем конфигурацию всех инстансов согласно п.п.2.3.7-2.3.11.

2.4.13. Проводим инициализацию главного мастер-сервера с которого будем инициализировать весь кластер. Подключаемся консольным клиентом:

```
mongo
```

Вывод консоли при успешном подключении:

```
MongoDB shell version v3.6.4
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.4
```

2.4.14. Переключаемся на служебную базу данных:

```
use admin
```

Вывод консоли при успешной операции:

```
switched to db admin
```

2.4.15. Авторизуемся рутом:

```
db.auth("root","pa$$w0rd")
```

Вывод консоли при успешной авторизации:

```
1
```

2.4.16. Проверяем текущий статус репликации:

```
rs.status()
```

Вывод консоли, со статусом 0 и сообщением NotYetInitialized, означающим, что репликация еще не была инициализирована:

```
{
        "info" : "run rs.initiate(...) if not yet done for the
set",
        "ok" : 0,
        "errmsg" : "no replset config has been received",
        "code" : 94,
        "codeName" : "NotYetInitialized",
        "$clusterTime" : {
                "clusterTime" : Timestamp(0, 0),
                "signature" : {
                        "hash"                                    :
BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

2.4.17. Инициализируем главную реплику (замените _id реплик-сета, _id сервера реплики и хост/порт на требуемые в вашей конфигурации):

```
rs.initiate(
  {
    _id : "sierrepl",
    members: [
      { _id : 2, host : "192.168.202.2:27017" }
    ]
  }
```

```
)
```

Вывод консоли при успешной инициализации (статус 1):

```
{
        "ok" : 1,
        "operationTime" : Timestamp(1526041434, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1526041434, 1),
                "signature" : {
                        "hash"                                          :
BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

2.4.18. Добавляем вторую реплику сета (замените хост/порт на требуемые в вашей конфигурации):

```
rs.add("192.168.202.3:27017")
```

Вывод консоли при успешном добавлении второго сервера в Replica Set:

```
{
        "ok" : 1,
        "operationTime" : Timestamp(1526041890, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1526041890, 1),
                "signature" : {
                        "hash"                                          :
BinData(0,"OML4OUVIR6f7ROabQcVUS3nvNSs="),
                        "keyId" : NumberLong("6554298059960877057")
                }
        }
}
```

2.4.19. Добавляем арбитра (замените хост/порт на требуемые в вашей конфигурации):

```
rs.addArb("192.168.202.4:27017")
```

2.4.20. Проверяем конфигурацию:

```
rs.status()
```

Вывод консоли - большой JSON с текущей конфигурацией. Нас интересует наличие статусных полей (state), которые могут принимать значение 1, если все ОК и сервер первичный, либо 2, если все ОК и сервер вторичный, либо 7, если все ОК и сервер – арбитр, либо 5, если еще идет синхронизация данных между репликами (расшифровка статуса будет в поле stateStr):

```
{
        "set" : "sierrepl",
        "myState" : 1,
        ...
```

```
        "members" : [
                {
                        "_id" : 2,
                        "name" : "192.168.202.2:27017",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                             ...
                },
                {
                        "_id" : 3,
                        "name" : "192.168.202.3:27017",
                        "health" : 1,
                        "state" : 5,
                        "stateStr" : "STARTUP2",
                        ...
                },

                ...
        ]
        ...
}
```

2.4.21. Отключаемся от консоли базы данных:

```
exit
```

Вывод консоли при успешной операции:

```
bye
```

2.5. Просмотр статуса репликации, переконфигурирование replica-set, удаление из него ноды.

Все изменения параметров текущей конфигурации желательно делать на ноде, которая в текущий момент времени является первичной. Но в любом случае, при изменении конфигурации, «первичная» нода теряет свой PRIMARY-статус, и реплика-сет проводит «перевыборы» лидера. Для изменения нод необходимо авторизоваться в консольном клиенте `mongo` пользователем, имеющим роль `clusterAdmin`.

Описанные далее команды изменения настроек нужно выполнять в консольном клиенте `mongo` на Primary-сервере. Команды просмотра текущего статуса (например, для поиска текущей PRIMARY) можно выполнить на любом сервере. Определить, является ли текущая нода PRIMARY можно, сразу после запуска консольного клиента, еще до авторизации – в строке приветствия будет выведено имя реплика-сета и статус текущей ноды:

```
mongo
```

Вывод консоли:

```
MongoDB shell version v3.6.4
```

```
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.6.4
sierrepl:SECONDARY>
```

2.5.1. Просмотр текущей конфигурации и статуса репликации:

Просмотр статуса репликации:

```
rs.status()
```

Просмотр текущей конфигурации:

```
rs.conf()
```

2.5.2. Изменение параметров текущей конфигурации.

Для изменения параметров текущей конфигурации, без удаления и создания новой ноды (например, изменение хоста расположения ноды, либо его ip-адреса) необходимо выполнить три действия в консольном клиенте `mongo`

Получаем текущую конфигурацию в качестве переменной:

```
cfg = rs.conf()
```

После выполнения команды получения в консоль будет выведена JSON с текущей конфигурацией.

Меняем в ней какие-либо параметры требуемой реплики:

```
cfg.members[0].host = "newhost.mymongo.ru:27027"
```

```
cfg.members[0].priority = 2
```

Реконфигурируем кластер:

```
rs.reconfig(cfg)
```

2.5.3. Удаление ноды из кластера

Для удаления ноды достаточно выполнить следующую команду:

```
rs.remove("newhost.mymongo.ru:27027")
```

3. Установка Docker.

3.1. Установка последней стабильной версии Docker на Centos 7.

3.1.1. Обновляем индекс пакетов:

```
sudo yum check-update
```

3.1.2. Запускаем скрипт установки докера с официального сайта:

```
curl -fsSL https://get.docker.com/ | sh
```

Будет очень большой вывод в консоль, финальные строки при успешной установке:

```
…

If you would like to use Docker as a non-root user, you should now
consider
adding your user to the "docker" group with something like:

  sudo usermod -aG docker your-user

Remember that you will have to log out and back in for this to take
effect!

WARNING: Adding a user to the "docker" group will grant the ability
to run
         containers which can be used to obtain root privileges on
the
         docker host.
         Refer                                                  to
https://docs.docker.com/engine/security/security/#docker-daemon-
attack-surface
         for more information.
```

3.1.3. Запускаем демон докера:

```
sudo systemctl start docker
```

3.1.4. Добавляем демон в автозапуск:

```
sudo systemctl enable docker
```

3.1.5. Опционально можно проверить, что докер работает, запустив образ «hello world»:

```
docker run hello-world
```

Часть вывода консоли при успешном запуске:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest:
sha256:f5233545e43561214ca4891fd1157e1c3c563316ed8e237750d59bde7336
1e77
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working
correctly.
…
```

3.2. Установка последней стабильной версии Docker на Ubuntu Server 18.04 LTS

3.2.1. Устанавливаем некоторые дополнительные пакеты:

```
apt-get install apt-transport-https software-properties-common ca-certificates curl wget
```

3.2.2. Импортируем сертификат репозитория:

```
wget https://download.docker.com/linux/ubuntu/gpg

apt-key add gpg
```

3.2.3. Добавляем репозиторий докера:

```
echo "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable" | sudo tee /etc/apt/sources.list.d/docker.list
```

3.2.4. Обновляем кэш репозиториев:

```
apt-get update
```

3.2.5. Устанавливаем докер:

```
apt-get install docker-ce
```

3.2.6. Запускаем демон докера:

```
systemctl start docker
```

3.2.7. Добавляем демон в автозапуск:

```
systemctl enable docker
```

3.2.8. Опционально можно проверить, что докер работает, запустив образ «hello world»:

```
docker run hello-world
```

Часть вывода консоли при успешном запуске:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9bb5a5d4561a: Pull complete
Digest:
sha256:f5233545e43561214ca4891fd1157e1c3c563316ed8e237750d59bde73361e77
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
…
```

3.3. Настройка Docker Swarm кластера.

В данном разделе описана подготовка Docker к работе в режиме отказоустойчивого кластера. Если требуется запуск и работа приложения в режиме «одного сервера» действия, описанные в данном разделе выполнять не требуется.

Перед выполнением действий описанных в этом разделе необходимо установить Docker на все серверы, которые будут участвовать в организации кластера, согласно п.3.1.

3.3.1. Инициализируем Docker Swarm (измените хост на текущий, на котором вы инициализируете ноду. По этому имени, либо IP хоста другие ноды в дальнейшем будут искать текущую) в режиме «менеджера» для текущей ноды:

docker swarm init --advertise-addr 192.168.202.2

Вывод консоли при успешной инициализации (Docker сгенерирует команду по которой к текущему менеджеру можно подключать воркеры, в текущей инструкции данная ссылка не пригодится, т.к. мы настраиваем конфигурацию с двумя менеджерами):

```
Swarm initialized: current node (vius41kyvl7ieopjjh9t2pfhr) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3sufroc44i5ek1a6i2wo1g4vj2szkx2661cwpqwndvdc6e77j4-9vubs71o8014tssqdffn3u9xc 192.168.202.2:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

3.3.2. Получаем команду для подключения второй ноды в качестве второго менеджера:

```
docker swarm join-token manager
```

Вывод консоли при успешной операции:

```
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3sufroc44i5ek1a6i2wo1g4vj2szkx2661cwpqwndvdc6e77j4-bbncfj8trmgs5pl8ryq78tyxn 192.168.202.2:2377
```

3.3.3. Выполняем полученную команду на втором сервере, чтобы добавить его в кластер (замените приведенный пример на полученное в предыдущем пункте!):

```
docker swarm join --token SWMTKN-1-3sufroc44i5ek1a6i2wo1g4vj2szkx2661cwpqwndvdc6e77j4-bbncfj8trmgs5pl8ryq78tyxn 192.168.202.2:2377
```

Вывод консоли при успешной операции:

```
This node joined a swarm as a manager.
```

4. Развертывание системы в docker-контейнерах.

Для упрощения управления и оркестровки набора микросервисов рекомендуется разворачивание системы в условиях контейнеризации. Предполагается, что docker уже установлен в системе и настроен на работу либо в одиночном режиме, либо в режиме кластера (swarm).

Для создания контейнеров предоставляется набор готовых скриптов (Dockerfile) и скомпилированных микросервисов для создания образов. Все образы строятся из «базового» образа, включающего в себя только JDK8, располагающегося в нем по пути /opt/jdk8 (это сделано для того, чтобы не занимать место на диске копируя в каждый создаваемый образ JDK8, размером в пятьсот с лишним мегабайт).

Базовый образ в предоставляемой конфигурации основан на контейнере Ubuntu 16.04. В виду того, что для работы большинства сервисов системы нужен только JDK8, окружение ОС не принципиально - можно использовать любой другой базовый контейнер (Centos, Debian, Alpine и пр.). Для смены базового контейнера на требуемый нужно отредактировать файл /base/Dockerfile, изменив в нем строчку «FROM ubuntu:16.04» изменив имя базового образа на необходимый. Не рекомендуется использовать «пустой» базовый контейнер, т.к. в нем нет утилит для управления содержимым контейнера, которые могут потребоваться для его администрирования.

Файлы конфигурации необходимо расположить в определенной директории, которая будет «проброшена» в образ сервиса конфигурации. Это необходимо для того, чтобы была возможность редактирования конфигурационных файлов без пересборки образа и переразворачивания контейнера с сервисом конфигурации. (Если используется swarm-кластер, лучшим решением будет создание отдельного тома (volume) для хранения конфигурационных файлов и подключения его к контейнеру сервиса конфигурации).

4.1. Создаем образы контейнеров сервисов системы:

В описываемых далее скриптах предполагается, что набор Docker-скриптов распакован в какую-либо временную директорию, пользователь работает под root и текущее положение пользователя – корень этой временной директории.

4.1.1. Создаем базовый образ. Распаковываем JDK в директорию `/base/opt`:

```
tar -xvzf jdk8.tar.gz -C ./base/opt
```

Для создания этого и любого последующего образа необходимо находиться внутри директории, в которой расположен Dockerfile, чтобы докер не сканировал и буферизировал всю файловую систему от текущего положения до DockerFile, поэтому первым действием переходим в соответствующую директорию, вторым – создаем образ:

```
cd base/

docker build -t sier/base .
```

(точка в конце этого и дальнейших скриптов создания образов принципиальна, это указатель на текущую директорию)

4.1.2. Создаем образ сервиса конфигурации:

```
cd ../config/

docker build -t sier/config:v1 .
```

4.1.3. Создаем образ сервиса оркестровки:

```
cd ../discovery/
```

```
docker build -t sier/discovery:v1 .
```

4.1.4. Создаем образ сервиса маршрутизации:

```
cd ../routing/

docker build -t sier/routing:v1 .
```

4.1.5. Создаем образ сервиса мониторинга:

```
cd ../hystrix-dashboard/

docker build -t sier/hystrix-dashboard:v1 .
```

4.1.6. Создаем образ сервиса ядра:

```
cd ../core/

docker build -t sier/core:v1 .
```

4.1.7. Создаем образ сервиса бизнес-логики:

```
cd ../business/

docker build -t sier/business:v1 .
```

4.1.8. Создаем образ сервиса API файлохранилища:

```
cd ../filestorage/

docker build -t sier/filestorage:v1 .
```

4.1.9. Создаем образ сервиса нумерации:

```
cd ../numerator/

docker build -t sier/numerator:v1 .
```

4.1.10. Создаем образ сервиса рендеринга:

```
cd ../rendering/

docker build -t sier/rendering:v1 .
```

4.1.11. Создаем образ рендера:

Ввиду того, что сервис рендера форм требует установленных шрифтов для поддержки кириллицы, его нужно собирать из расширенного базового образа с установленными шрифтами. Скрипт установки шрифтов невозможно добавить в Dockerfile, т.к. при их установке требуется интерактивно согласиться с ELUA. Поэтому предварительно создаем расширенный базовый образ.

4.1.11.1. Запускаем контейнер с именем «baseimage» из «базового» образа:

```
docker run --name baseimage -d -t -i sier/base /bin/bash
```

подключаемся к его консоли:

```
docker exec -i -t baseimage /bin/bash
```

выполняем установку пакета шрифтов:

```
apt-get install ttf-mscorefonts-installer
```

пересоздаем кэш шрифтов:

```
fc-cache -f -v
```

в выводе команды обязательно нужно проверить наличие строчки, показывающей, что установилось много шрифтов (~60 шт.) в

```
/usr/share/fonts/truetype/msttcorefonts
```

должны быть строки вида:

```
…
/usr/share/fonts/truetype/dejavu: caching, new cache contents: 6 fonts, 0 dirs
/usr/share/fonts/truetype/liberation: caching, new cache contents: 16 fonts, 0 dirs
/usr/share/fonts/truetype/msttcorefonts: caching, new cache contents: 60 fonts, 0 dirs
…
```

4.1.11.2. Отключаемся от консоли образа:

```
exit
```

4.1.11.3. Останавливаем контейнер:

```
docker stop baseimage
```

4.1.11.4. Получаем хеш контейнера:

```
docker ps -a | grep baseimage
```

в выводе консоли на первом месте будет искомый хэш, пример вывода:

```
15ada4685d86 sier/base "/bin/bash" 1 minutes ago Exited (0) 2 minutes ago baseimage
```

4.1.11.5. Коммитим изменения, сделанные в контейнере, в образ с именем «sier/base:render», т.к. это имя прописано в Dockerfile рендера (замените хеш на полученный в выводе предыдущей команды):

```
docker commit 15ada4685d86 sier/base:render
```

4.1.11.6. Проверяем, что образ создался успешно:

```
docker images | grep sier/base
```

вывод консоли должен содержать информацию о двух имеющихся образах с тегами «latest» и «render»:

```
sier/base render e2bb272873e2 1 minutes ago 684MB
```

```
sier/base        latest        9ee60acb70bf      5 minutes ago
575MB
```

4.1.11.7. Удаляем контейнер, который мы создали из базового образа, т.к. он нам больше не требуется (замените хеш на полученный ранее):

```
docker rm 15ada4685d86
```

4.1.11.8. Создаем образ рендера:

```
cd ../render/

docker build -t sier/render:v1 .
```

4.1.12. Создаем образ СМЭВ-клиента:

```
cd ../smevclient/

docker build -t sier/smevclient:v1 .
```

4.1.13. Создаем образ сервиса интеграции со СПЭР-3:

```
cd ../sper3/

docker build -t sier/sper3integration:v1 .
```

4.1.14. Создаем образ «статики»:

```
cd ../static/

docker build -t sier/static:v1 .
```

4.1.15. Создаем образ «архиватора»:

```
cd ../archive/

docker build -t sier/archive:v1 .
```

Имени каждого образа присвоили тег v1, это сделано для упрощения дальнейшего версионирования обновлений – когда будет приходить очередное обновление можно будет делать образ с тегом v2, v3, v4 и т.д, чтобы можно было бы в любой момент откатиться на предыдущую версию контейнера.

4.2. Создаем контейнеры на основе образов и запускаем их в работу (single-server-mode).

В текущем разделе описано разворачивание системы в режиме «одного сервера». Если требуется развернуть систему в режиме отказоустойчивого Swarm-кластера, текущий раздел нужно пропустить и выполнить действия, описанные в п.4.3.

Предполагается, что конфигурационные файлы расположены в /opt/config-server/ , подготовлены, отредактированы и в них указаны верные настройки подключения к СУБД, файловому хранилищу и шине интеграции для соответствующих сервисов.

Опционально. Все дальнейшие действия можно производить как через консоль, так и через любой веб-интерфейс докера. Наиболее распространенный и один из самых легковесных (~35Мб размер образ и ~15Мб потребление ОЗУ в работе) – Portainer. Он поставляется сразу образом докера, поэтому установить и запустить его можно следующими командами:

docker volume create portainer_data

```
docker run -d -p 9000:9000 --name portainer --restart always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer
```

После этого веб-интерфейс будет доступен на хосте, где установлен докер по порту 9000 (http://<хост >:9000). Через веб-интерфейс можно создавать образы, управлять контейнерами (создавать/запускать/останавливать/удалять), следить за их текущим состоянием, подключаться к консоли любого контейнера для управления содержимым, просматривать логи, мониторить текущую нагрузку на ЦПУ и ОЗУ, создаваемую каждым контейнером, а также контролировать параметры кластера, если Docker запущен в режиме swarm.

4.2.1. **Создаем «внутреннюю» сеть докера, в которой будет работать система.** Сеть базируется на адаптере bridge, чтобы у контейнеров была возможность видеть внешние узлы сети хоста:

```
docker network create -d bridge sier-bridge
```

Создаем контейнеры:

4.2.2. **Создаем и запускаем контейнер с сервисом конфигурации**. В него нужно прокинуть директорию с конфигурационными файлами (или «том», в случае swarm), для этого используется параметр `-v`. Кроме того, текущий образ и все далее создаваемые должны работать в одной «внутренней» сети, ее определяем параметром `--network`. Имя контейнера передается в параметре `--name`. Параметр `-d` определяет то, что контейнер должен запуститься в режиме «демона», не «занимая» консоль.

Опционально: т.к. сервис конфигурации – первая точка отказа системы, можно сконфигурировать контейнер на автоматический перезапуск его после остановки по любой причине (от падения от внутренней ошибки приложения до физического перезапуска хоста), для этого в скрипт запуска нужно добавить параметр `--restart always`

*Для сервиса конфигурации **обязательно** нужно назначить имя контейнера «cloud-config-service», т.к. именно по этому имени его ищут все микросервисы на порту 8887*:

```
docker run --network=sier-bridge --name cloud-config-service -v /opt/config-server:/opt/config-server -d sier/config:v1
```

4.2.3. **Создаем и запускаем контейнер с сервисом оркестровки**. Имя контейнера сервиса оркестровки должно совпадать с тем, которое указано в качестве хоста в конфигурационном файле `/opt/config-server/application.properties` в параметре `eureka.client.serviceUrl.defaultZone`, т.к. по этому имени, получаемому из конфигурационного файла, сервис оркестровки будут искать все остальные микросервисы:

Опционально: т.к. сервис оркестровки – вторая точка отказа системы, можно сконфигурировать контейнер на автоматический перезапуск его после остановки по любой причине (от падения от внутренней ошибки приложения до физического перезапуска хоста), для этого в скрипт запуска нужно добавить параметр `--restart always`

```
docker run --network=sier-bridge --name cloud-discovery-service -d sier/discovery:v1
```

4.2.4. **Создаем и запускаем контейнер с сервисом маршрутизации**. Для сервиса маршрутизации необходим проброс двух портов из внутренней сети докера в сеть хоста (8080 и 9080, если есть потребность, их можно изменить в файле routing-service.properties и пробросить измененные). Первый порт – это точка входа в приложение, второй – для съема нагрузочной метрики:

```
docker run --network=sier-bridge --name cloud-routing-service -p 8080:8080 -p 9080:9080 -d sier/routing:v1
```

4.2.5. **Создаем и запускаем сервис мониторинга**. Для подключения к нему из внешней сети также необходим проброс порта (8800):

```
docker run --network=sier-bridge --name cloud-hystrix-dashboard-service -p 8800:8800 -d sier/hystrix-dashboard:v1
```

4.2.6. **Создаем и запускаем сервис ядра**. Для этого и последующих сервисов для их обычной работы не требуется проброса каких-либо портов во внешнюю сеть, пробрасывать можно опционально для каких-либо отладочных действий, но не обязательно (в приведенных скриптах пробросов назначено не будет):

```
docker run --network=sier-bridge --name sier-core -d sier/core:v1
```

4.2.7. **Создаем и запускаем сервис бизнес-логики**:

```
docker run --network=sier-bridge --name sier-business -d sier/business:v1
```

4.2.8. **Создаем и запускаем сервис API файлохранилища**:

```
docker run --network=sier-bridge --name sier-filestorage -d sier/filestorage:v1
```

4.2.9. **Создаем и запускаем сервис нумерации**.

Опционально: этот сервис является третьей точкой отказа системы, поэтому, при желании в скрипт запуска можно также добавить опцию `--restart always`

```
docker run --network=sier-bridge --name sier-numerator -d sier/numerator:v1
```

4.2.10. **Создаем и запускаем сервис рендеринга**:

```
docker run --network=sier-bridge --name sier-rendering -d sier/rendering:v1
```

4.2.11. **Создаем и запускаем рендер**:

```
docker run --network=sier-bridge --name sier-render -d sier/render:v1
```

4.2.12. **Создаем и запускаем СМЭВ-клиент**:

```
docker run --network=sier-bridge --name sier-smevclient -d sier/smevclient:v1
```

4.2.13. **Создаем и запускаем сервис интеграции со СПЭР-3**:

```
docker run --network=sier-bridge --name sier-sper3integration -d sier/sper3integration:v1
```

4.2.14. **Создаем и запускаем контейнер с сервисом архивации**:

```
docker run --network=sier-bridge --name sier-archive -d sier/archive:v1
```

4.2.15. **Создаем и запускаем контейнер со статикой**:

Имя контейнера со статикой используется в качестве хоста для поиска его сервисом маршрутизации. В конфиге routing-service.properties по-умолчанию адрес указан вида http://sier.static/ Если требуется задать другое имя контейнера со статикой, необходимо поправить настройки маршрутизации до контейнера в routing-service.properties, указав в нем требуемое имя контейнера, либо непосредственно ip-адрес, где развернута статика.

```
docker run --network=sier-bridge --name sier-static -d sier/static:v1
```

После вышеописанных действий должны быть доступны следующие ресурсы:

http://<хост>:8080

http://<хост>:9080/actuator/hystrix.stream

http://<хост>:8800/hystrix/

4.3. Создаем Swarm-сервисы на основе образов и запускаем их в работу (Swarm-cluster).

В текущем разделе описано разворачивание системы в режиме отказоустойчивого кластера. Если требуется развернуть систему в режиме «одного сервера», действия, описанные в этом разделе, выполнять не требуется.

Опционально. Все дальнейшие действия можно производить как через консоль, так и через любой веб-интерфейс докера. Наиболее распространенный и один из самых легковесных (~35Мб размер образ и ~15Мб потребление ОЗУ в работе) – Portainer. Он поставляется сразу образом докера, поэтому установить и запустить его можно следующими командами:

```
docker volume create portainer_data
```

```
docker service create -p 9000:9000 --name portainer --replicas 1 --mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock --mount src=portainer_data,dst=/data portainer/portainer
```

После этого веб-интерфейс будет доступен на всех хостах, где установлен Docker и настроен Swarm по порту 9000 (http://<хост>:9000). Через веб-интерфейс можно создавать образы, управлять контейнерами (создавать/запускать/останавливать/удалять), следить за их текущим состоянием, подключаться к консоли любого контейнера для управления содержимым, просматривать логи, мониторить текущую нагрузку на ЦПУ и ОЗУ, создаваемую каждым контейнером, а также контролировать параметры Swarm кластера.

Перед разворачиванием сервисов, если не используется какое-либо внешнее хранилище образов, необходимо создать образы на каждом сервере, входящем в кластер (т.е. выполнить пункты раздела 4.1. на каждом сервере кластера). Если этого не сделать, SWARM сможет поднять контейнеры с сервисами только на одном сервере кластера – на текущем – т.к. только на нем он сможет найти необходимые ему контейнеры. При попытках запуска сервисов на других серверах кластера будет возвращаться ошибка «image not found».

4.3.1. **Создаем внутреннюю overlay подсеть (замените подсети на требуемые)**:

```
docker network create --driver overlay --subnet=172.7.0.0/16 --gateway=172.7.0.1 sier-net
```

Создаем сервисы:

4.3.2. **Создаем и запускаем сервис конфигурации**. В него нужно прокинуть директорию с конфигурационными файлами, для этого используется параметр `--mount` тип монтирования `bind` определяет, что прокинута дожна быть локальная директория, а не «том». Кроме того, текущий образ и все далее создаваемые должны работать в одной «внутренней» overlay-сети, единой для всех серверов кластера, ее определяем параметром `--network`. Имя сервиса передается в параметре `--name`. Количество требуемых реплик в параметре `--replicas`.

*Для сервиса конфигурации **обязательно** нужно назначить имя «cloud-config-service», т.к. именно по этому имени его ищут все микросервисы на порту 8887*:

```
docker service create --replicas 1 --name cloud-config-service --network=sier-net --mount type=bind,src=/opt/syncstore/config-server,dst=/opt/config-server sier/config:v1
```

4.3.3. **Создаем и запускаем сервис оркестровки**. Имя сервиса оркестровки должно совпадать с тем, которое указано в качестве хоста в конфигурационном файле `/opt/config-server/application.properties` в параметре `eureka.client.serviceUrl.defaultZone`, т.к. по этому имени, получаемому из конфигурационного файла, сервис оркестровки будут искать все остальные микросервисы:

```
docker service create --replicas 1 --name cloud-discovery-service --network=sier-net sier/discovery:v1
```

4.3.4. **Создаем и запускаем сервис маршрутизации**. Для сервиса маршрутизации необходим проброс двух портов из внутренней сети докера в сеть хоста (8080 и 9080, если есть потребность, их можно изменить в файле routing-service.properties и пробросить измененные). Первый порт – это точка входа в приложение, второй – для съема нагрузочной метрики:

```
docker service create --replicas 1 --network=sier-net --name cloud-routing-service -p 8080:8080 -p 9080:9080 sier/routing:v1
```

4.3.5. **Создаем и запускаем сервис мониторинга**. Для подключения к нему из внешней сети также необходим проброс порта (8800):

```
docker service create --replicas 1 --network=sier-net --name cloud-hystrix-dashboard-service -p 8800:8800 sier/hystrix-dashboard:v1
```

4.3.6. **Создаем и запускаем сервис ядра**. Для этого и последующих сервисов для их обычной работы не требуется проброса каких-либо портов во внешнюю сеть, пробрасывать можно опционально для каких-либо отладочных действий, но не обязательно (в приведенных скриптах пробросов назначено не будет):

```
docker service create --replicas 1 --network=sier-net --name sier-core sier/core:v1
```

4.3.7. **Создаем и запускаем сервис бизнес-логики**:

```
docker service create --replicas 1 --network=sier-net --name sier-business sier/business:v1
```

4.3.8. **Создаем и запускаем сервис API файлохранилища**:

```
docker service create --replicas 1 --network=sier-net --name sier-filestorage sier/filestorage:v1
```

4.3.9. **Создаем и запускаем сервис нумерации**.

```
docker service create --replicas 1 --network=sier-net --name sier-numerator sier/numerator:v1
```

4.3.10. **Создаем и запускаем сервис рендеринга**:

```
docker service create --replicas 1 --network=sier-net --name sier-rendering sier/rendering:v1
```

4.3.11. **Создаем и запускаем рендер**:

```
docker service create --replicas 1 --network=sier-net --name sier-render sier/render:v1
```

4.3.12. **Создаем и запускаем СМЭВ-клиент**:

```
docker service create --replicas 1 --network=sier-net --name sier-smevclient sier/smevclient:v1
```

4.3.13. **Создаем и запускаем сервис интеграции со СПЭР-3**:

```
docker service create --replicas 1 --network=sier-net --name sier-sper3integration sier/sper3integration:v1
```

4.3.14. **Создаем и запускаем сервис архивации**:

```
docker service create --replicas 1 --network=sier-net --name sier-archive sier/archive:v1
```

4.3.15. **Создаем и запускаем сервис статики**:

Имя сервиса статики используется в качестве хоста для поиска его сервисом маршрутизации. В конфиге routing-service.properties по-умолчанию адрес указан вида http://sier-static/ Если требуется задать другое имя сервиса статики, необходимо поправить настройки маршрутизации до сервиса в routing-service.properties, указав в нем требуемое имя сервиса, где развернута статика.

```
docker service create --replicas 1 --network=sier-net --name sier-static sier/static:v1
```

После вышеописанных действий должны быть доступны следующие ресурсы:

http://<хост>:8080

http://<хост>:9080/actuator/hystrix.stream

http://<хост>:8800/hystrix/

В виду прозрачной динамической маршрутизации, включенной в SWARM по-умолчанию, хост – это любой сервер, входящий в SWARM-кластер. Трафик будет полностью прозрачно для клиента маршрутизироваться на сервер где в данный момент действительно развернут требуемый сервис.

5. Скрипты для быстрого разворачивания Системы в Swarm.

Для облегчения инсталляции ниже приведены скрипты, которые можно использовать при доступности репозитория АО «Эволента»:

```
docker login harbor.evolenta.ru

docker network create --driver overlay --subnet=172.7.0.0/16 --gateway=172.7.0.1 sier-net

docker service create --with-registry-auth --replicas 1 --name cloud-config-service --network=sier-net --mount type=bind,src=/opt/config-server,dst=/opt/config-server harbor.evolenta.ru/sier/config:v7

docker service create --with-registry-auth --replicas 1 --name cloud-discovery-service --network=sier-net harbor.evolenta.ru/sier/discovery:v7

docker service create --with-registry-auth --replicas 1 --network=sier-net --name cloud-routing-service -p 8080:8080 -p 9080:9080 harbor.evolenta.ru/sier/routing:v7

docker service create --with-registry-auth --replicas 1 --network=sier-net --name sier-core harbor.evolenta.ru/sier/core:v32

docker service create --with-registry-auth --replicas 1 --network=sier-net --name sier-static harbor.evolenta.ru/sier/static:v423

docker service create --with-registry-auth --replicas 1 --network=sier-net --name sier-numerator harbor.evolenta.ru/sier/numerator:v5

docker service create --with-registry-auth --replicas 1 --network=sier-net --name sier-business harbor.evolenta.ru/sier/business:v28

docker service create --with-registry-auth --replicas 1 --network=sier-net --name sier-filestorage harbor.evolenta.ru/sier/filestorage:v14

docker service create --with-registry-auth --replicas 1 --network=sier-net --name sier-rendering harbor.evolenta.ru/sier/rendering:v15
```

docker service create --with-registry-auth --replicas 1 --network=sier-net --name sier-render harbor.evolenta.ru/sier/render:v20

docker service create --with-registry-auth --replicas 1 --network=sier-net --name sier-smevclient harbor.evolenta.ru/sier/smevclient:v31

docker service create --with-registry-auth --replicas 1 --network=sier-net --name sier-sper3integration harbor.evolenta.ru/sier/sper3integration:v26

docker service create --with-registry-auth --replicas 1 --network=sier-net --name sier-archive harbor.evolenta.ru/sier/archive:v11

docker service create --with-registry-auth --replicas 1 --network=sier-net --name sier-kpp -p 9192:9192 -p 80:80 --mount type=bind,src=/opt/kpp/db,dst=/opt/db harbor.evolenta.ru/sier/kpp:v4