МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева»

(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра геоинформатики и информационной безопасности

КУРСОВОЙ ПРОЕКТ ПО ДИСЦИПЛИНЕ

«Инженерия программного обеспечения»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

«Анализатор программ, написанных на С++, на уязвимости»

|  |  |
| --- | --- |
| Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* | Н.М. Фролов |
|  |  |
| Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* | В.А. Бердников |
|  |  |

САМАРА 2022

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева»

(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра геоинформатики и информационной безопасности

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

Студенту ***Фролову Никите Максимовичу*** группы 6412-100503D

Тема проекта: ***«*Анализатор программ, написанных на С++, на уязвимости*»***

|  |  |  |
| --- | --- | --- |
| Планируемые результаты освоения образовательной программы (компетенции) | Планируемые результаты | Содержание задания |
| ПК-5 | **Знать:**  Этапы оценки и управления рисками, современные методики оценки рисков, смысловое содержание численных показателей оценок рисков.  **Уметь:**  определять необходимую и достаточную совокупность средств защиты информации, организационных мер направленных на снижение рисков ИБ, разработать архитектуру СИБ, направленную на снижение рисков информационной безопасности автоматизированных систем.  **Владеть:**  навыками определения приемлемого для организации уровня риска; идентификации, анализа и оценки рисков; ранжирование рисков; принятия решения по рискам и разработка плана реагирования на риски; реализации мероприятий и оценки эффективности реализованных мер. | Знать способы оценки рисков при реализации основных функций программы и смысловое содержание численных показателей этих оценок. Уметь определять средства защиты программного обеспечения. Владеть навыками определения приемлемого уровня риска для организации, принятия решения по рискам и разработка плана реагирования на риски. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата выдачи задания | 8 февраля 2022 г. | | | |
| Срок представления на кафедру пояснительной записки | | |  | |
| Руководитель курсового проекта профессор кафедры ГИиИБ, д.т.н. | |  | | В.А.Бердников |
|  | | *(подпись)* | |  |
| Задание принял к исполнению  студент группы № 6412-100503D | |  | | Н.М. Фролов |
|  | | *(подпись)* | |  |

содержание

[РЕФЕРАТ 5](#_Toc102553847)

[РАСПРЕДЕЛЕНИЕ ОБЯЗАННОСТЕЙ 7](#_Toc102553847)

[ВВЕДЕНИЕ 9](#_Toc102553847)

[1 БИБЛИОТЕКИ/ПРОГРАММНЫЕ СРЕДСТВА, ВЫБОР ЯЗЫКА ПРОГРАММИРОВАНИЯ И СРЕДЫ РАЗРАБОТКИ 11](#_Toc102553848)

[2.1 Библиотеки и программные средства 11](#_Toc102553849)

[2.2 Выбор средств разработки и системных программных средств 12](#_Toc102553850)

[2.3 Требования к разрабатываемому программному обеспечению 13](#_Toc102553851)

[3 ИНСТРУКЦИЯ ПО УСТАНОВКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 14](#_Toc102553852)

[4 ЗАПУСК ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 14](#_Toc102553853)

[5 ФУНКЦИОНАЛЬНОЕ ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 15](#_Toc102553854)

[6 ДЕТАЛЬНОЕ ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 18](#_Toc102553855)

[7 НАЧАЛО РАБОТЫ С ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ 27](#_Toc102553856)

[7.1 Описание графического пользовательского интерфейса 27](#_Toc102553857)

[7.1.1 Клиентское приложение 27](#_Toc102553858)

[7.1.2 Серверное приложение 29](#_Toc102553859)

[7.2 Подготовка приложения к работе 32](#_Toc102553860)

[7.3 Запуск работы основного функционала 33](#_Toc102553861)

[8 ВЫВОДЫ И РЕЗУЛЬТАТ 34](#_Toc102553862)

[9 КОМПЛЕКСНАЯ ОТЛАДКА 34](#_Toc102553863)

[СПИСОК ИСПОЛЬЗОВАННХ ИСТОЧНИКОВ 37](#_Toc102553864)

[ПРИЛОЖЕНИЕ А 38](#_Toc102553865)

[ПРИЛОЖЕНИЕ Б 206](#_Toc102553866)

РЕФЕРАТ

**Пояснительная записка к курсовому проекту:**

распределение обязанностей

**Громов В.В. –** разработал GUI клиент программы. Сделал привязку GUI клиента к разработанной библиотеке анализа кода. Оформил документацию для курсового проекта на git.

**Гафаров И.Н. –** создал механизм, позволяющий хранить уязвимости и список правил для них на кластерном сервере MongoDB, что позволило облегчить процесс взаимодействия с базой данных. Участвовал в создании функционала, позволяющего автоматизировать функциональное тестирование. Также реализовал алгоритмы, позволяющие выявить уязвимости утечка информации, некорректный доступ к файлам, случайные числа криптографического качества. Написал junit тесты, для разрабатываемого программного обеспечения.

**Кузнецов В.А. –** реализовал функционал, позволяющий выявлять уязвимости внедрение SQL команд, внедрение команд, гонки.

**Брагин Р.А. –** реализовал функционал, позволяющий выявлять уязвимости пренебрежение обработкой ошибок, пренебрежение безопасным хранением данных, переполнение целых чисел.

ВВЕДЕНИЕ

1 БИБЛИОТЕКИ/ПРОГРАММНЫЕ СРЕДСТВА, ВЫБОР ЯЗЫКА ПРОГРАММИРОВАНИЯ И СРЕДЫ РАЗРАБОТКИ

## Библиотеки и программные средства

Для реализации функционала программного обеспечения использовался следующий список библиотек:

1. Mongo-java-driver – библиотека, позволяющая работать с базой данных MongoDB, с помощью использования Java кода;
2. Velocity – движок для обработки шаблонов (template engine), который позволяет напрямую обращаться к методам и полям Java классов. В данной работе использовалась для создания результирующих отчетов;
3. Junit – библиотека для модульного тестирования программного обеспечения на языке Java;
4. Log4j – быстрая и гибкая среда ведения журналов (логирования), написанная на Java;
5. Maven-antrun-plugin – плагин, который позволяет запускть ant задачи, с помощью maven. Ant задачи могут быть описаны внутри pom.xml, использовался для автоматизации функционального тестирования;
6. JavaFX – представляет инструментарий для создания кроссплатформенных графических приложений на платформе Java;

## Выбор средств разработки и системных программных средств

Для реализации поставленной задачи был выбран язык программирования Java. Главными преимуществами данного языка являются простота изучения, объектно-ориентированный подход в разработке программного обеспечения, кросплатформенность, высокопроизводительность.

В качестве среды разработки было принято решение использовать IntelliJ IDEA – то IDE, интегрированная среда разработки (комплекс программных средств, который используется для написания, исполнения, отладки и оптимизации кода) для Java, JavaScript, Python и других языков программирования от компании JetBrains. Отличается обширным набором инструментов для рефакторинга (перепроектирования) и оптимизации кода.

* 1. **Требования к разрабатываемому программному обеспечению**

Разрабатываемая программа должна отвечать следующим требованиям, обусловленным ее назначением и сферой применения:

1. Поиск уязвимостей в программах, написанных на С++ путем анализа их текстов при помощи специально разработанной программы - анализатора;
2. Составление отчета по найденным уязвимостям, при этом должно быть указано в какой строчке кода найдена уязвимость, описание уязвимости и способ её устранения;
3. Наличие графического интерфейса, для удобного взаимодействия с программой;

2 МИНИМАЛЬНЫЕ СИСТЕМНЫЕ ТРЕБОВАНИЯ

Требования к рабочей станции для установки данного программного обеспечения:

* Процессор — Intel Core 2 Duo или другой схожий по производительности x86-совместимый процессор с количеством ядер 2 и более;
* Объем оперативной памяти — не менее 2 ГБайт (рекомендуется 4 Гбайт);
* Свободное место на жестком диске — не менее 150 Мбайт (рекомендуется 250 Мбайт);
* Сетевой адаптер или модем;
* Операционная система — Windows 7 / Windows 8 / Windows 8.1 / Windows 10 / Windows 11;
* Postman;
* Java 11;
* Java Development Kit 11.

3 ИНСТРУКЦИЯ ПО УСТАНОВКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В архиве с программным обеспечением предоставлены два JAR-архива: клиентский и серверный. Каждый архив представляет собой готовое для запуска приложение.

Шаги по установке программного обеспечения:

1. Проверьте, что дистрибутив языка программирования Java версии 11 (или выше) установлен на вашей рабочей станции. Если он отсутствует в системе, пожалуйста, установите Java, скачав дистрибутив с официального сайта: <https://www.java.com/ru/download/manual.jsp> ;
2. Проверьте, что установлен JDK (Java Development Kit) 11 (или выше). Обычно JDK располагается в C:\Program Files\Java\jdk-X, где X – версия JDK. Если он отсутствует в системе, пожалуйста, установите JDK:

<https://www.oracle.com/java/technologies/downloads/>

1. Установите переменную среды JAVA\_HOME на директорию установленного JDK. Для этого в поиске (в системе) необходимо ввести «Изменение системных переменных среды» и открыть появившуюся в окне результата утилиту. В открывшемся окне необходимо выбрать вкладку «Дополнительно», а затем нажать на кнопку «Переменные среды…». В нижней части («Системные переменные») нового открывшегося окна необходимо нажать на кнопку «Создать». Во вновь открывшемся диалоговом окне нужно указать название переменной «JAVA\_HOME» и указать полный путь до JDK (см. рисунок 1);

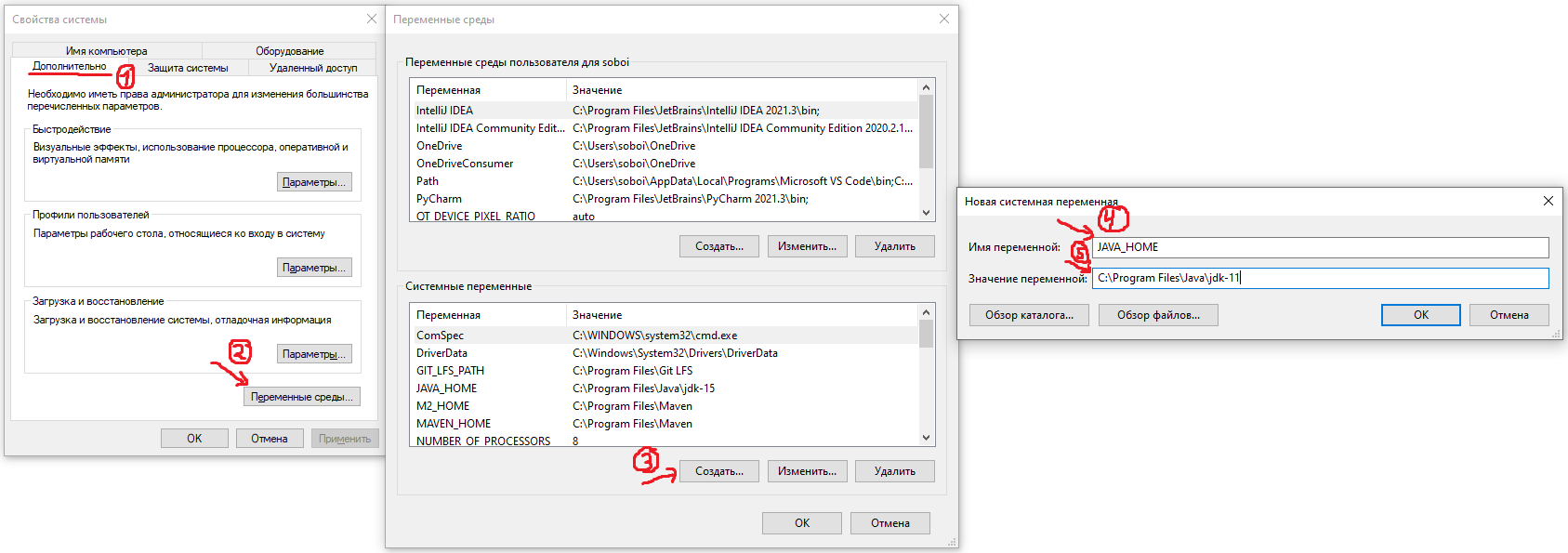


Рисунок 1. Установка переменной среды JAVA\_HOME

1. Проверьте, что в переменной среды PATH существует директория System32 (обычно находится по пути C:\Windows\System32). Если такого пути нет в переменной среды, добавьте его;
2. Запустите командную строку и введите java -version. Если результатом является версия Java – все шаги выполнены верно и язык программирования Java и JDK установлены;
3. Распакуйте архив с программным обеспечением в любую директорию;

4 ЗАПУСК ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Строго рекомендуется запускать программное обеспечение через командную строку в системе Windows, поскольку в таком случае весь журнал событий (лог) будет отображаться именно в командной строке. Если журнал событий не нужен, программное обеспечение может быть запущено простым двойным нажатием левой кнопкой мыши по JAR-архиву. Запуск программного обеспечения через командную строку происходит при выполнении следующей команды:

java -jar <полное имя JAR-архива>

Запуск клиента ничем не отличается от запуска сервера.

5 ФУНКЦИОНАЛЬНОЕ ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В разделе описывается основной принцип работы программного обеспечения и некоторые теоретические основы.

Программное обеспечение представляет собой систему, эмулирующую передачу конфиденциальных данных. Система способна включать и отключать передачу данных согласно предоставляемому расписанию оператором, который взаимодействует с сервером. Данные передаются в зашифрованном виде. Шифрование происходит на стороне клиента-передатчика, а дешифрование на стороне клиента-приемника. Технически программное обеспечение представляет собой совокупность элементов, на которых строится абсолютно любое веб-приложение. Все модули приложения написаны на языке программирования Java.

Общий принцип работы ПО: существует сервер и множество клиентов (2 и более). Между клиентами осуществляется передача данных в определенном формате. Между клиентами стоит ретранслятор (сервер). То есть, передача данных происходит через посредника. Сервер используется как посредник для обеспечения дополнительной надежности при передаче данных (описано ниже). Так же сервер используется для предотвращения конфликтов расписаний, поскольку каждый из клиентов имеет свое собственное расписание. Передача данных включает в себя 3 этапа:

1. Получение передаваемой информации из базы данных одного клиента;
2. Передача информации на сервер;
3. Передача данных с сервера на другой клиент.

Для обеспечения жизненного цикла программного обеспечения серверу необходимо передать расписание, в течение которого возможна передача данных от одного клиента к другому.

Информация передается по принципу перемещения: передаваемые данные копируются для передачи на клиента-получателя, а затем удаляются из клиента-отправителя.

Перед передачей информации с клиента на сервер происходит аутентификация клиента путем передачи некоторого ключевого слова. Ключевое слово создается на этапе запуска программного обеспечения. После создания ключевого слова создается его хэш-значение, получаемое с помощью хэш-функции SHA512. Ключевое слово передается на сервер в обычном открытом виде, который затем проходит на нем проверку. Проверка включает в себя использование хэш-функции SHA512 и сравнение полученного хэш-значения с сохраненным значением в базе данных. Если хэш-значения совпадают, то клиент считается доверенным отправителем конфиденциальной информации. При этом сервер так же должен знать об этом ключевом слове, так как используется как ретранслятор при передаче данных между двумя клиентами.

Часть конфиденциальной информации непосредственно шифруется перед передачей. Алгоритмом шифрования был выбран AES в режиме CBC. Секретный ключ может быть создан автоматически (самостоятельно) программным обеспечением. В результате получается псевдослучайный ключ длиной 256 бит. Ключ может быть использован повторно, если пользователь укажет путь к файлу с ключом при запуске программного обеспечения. Если же путь до файла не указан, будет создан новый ключ. Для успешной передачи информации с сохранением целостности необходимо присутствие одного и того же секретного ключа на двух клиентах. В противном случае информация может быть потеряна, поскольку не существует никакого механизма подтверждения о получении и верном расшифровании данных, то есть, отправка данных напопинает использование протокола UDP: отправка данных свидетельствует о возможности старта удаления отправленных данных на передающей стороне. Однако в случае недоступности сервера или клиента-приемника, данные из клиента-передатчика удалены не будут, так как при передаче все же производится проверка статуса отправки http-запроса с данными в теле этого запроса. Если статус запроса не является успешным – данные отправлены и удалены не будут.

Поскольку сервер является посредником при передаче конфиденциальной информации между клиентами, он получает данные в зашифрованном виде. Расшифровать данные на сервере не получится, поскольку сервер ничего не знает о секретном ключе. Таким образом, обеспечивается конфиденциальность данных на ретрансляторе.

В случае потери связи с клиентом-получателем, сервер временно сохраняет данные в своей базе данных, при этом добавляя информацию об отправителе и получателе. Сервер периодически пытается установить соединение с клиентом-получателем для повторной отправки данных. В случае если связь установлена – начинается передача данных с сервера на клиент-получатель. Если связь с клиентом-получателем не установлена – сервер попытается установить сеанс связи через некоторый промежуток времени.

Данные приходят на клиент-получатель в зашифрованном виде. У клиента-получателя так же проверяется ключевое слово, отправленное при передаче данных с сервера. Клиент-получатель расшифровывает зашифрованную часть полученной информации и сохраняет ее в свою локальную базу данных.

Данные могут передаваться клиентами только в одном направлении. То есть, у каждого клиента-отправителя может быть только один клиент-получатель. При этом существует возможность обеспечения полнодуплексного режима передачи информации.

6 ДЕТАЛЬНОЕ ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Средства сборки, язык, каркас и модули программы

Программное обеспечение написано на языке Java версии 18. Для сборки программного обеспечения был использован Maven (фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM, являющемся подмножеством XML).

Программное обеспечение было разделено на 3 модуля: common, client и server. В модулях server и client описаны классы, специфичные для соответствующих экземпляров (сервера и клиента соответственно). Модуль common используется как общая часть (зависимость) для двух других модулей.

Программное обеспечение является приложением, основанным на Spring Boot, то есть, использует особенности и возможности каркаса Spring. Ярким примером использования особенностей этого каркаса является механизм внедрения зависимостей (Dependency Injection).

В процессе разработки были задействованы такие технологии как Hibernate, Mockito, Junit, Java Persistence API, Rest API и Slf4j.

Основными элементами приложения являются стандартные элементы слоев веб-приложения: контроллеры, сервисы и репозитории. Репозитории используются как «драйвера» или «адаптеры» между приложением и базой данных. Этот тип объектов используется исключительно для доступа к данным и позволяет выполнять самые простые действия с ними такие как сохранение (создание), получение, редактирование и удаление. Сервисы используются как обертки для репозиториев, в рамки которых входят различного рода проверки данных на наличие и корректность. Под контроллерами понимаются REST-контроллеры, которые используют сервисы для обработки полученных данных и получения результата на основе этой обработки. Контроллеры используются в основном для запуска сервисов и выдачи ответов пользователю. Ответ включает в себя http-статус и иногда сообщение (например, описание ошибки).

## Инициализация ресурсов

При запуске программного обеспечения происходит инициализация исходных ресурсов с последующей обработкой конфигураций, написанных разработчиком. В этом приложении используется конфигурация для настройки объекта доступа к базе данных (все необходимые сведения для подключения), а также конфигурация настройки сервлетов (специальных объектов, предназначенных для обработки запросов пользователя и выдачи ответов).

## Описание сущностей

В программном обеспечении используются простые POJO (Plain Old Java Object) объекты для представления сущностей. Сущности включают в себя совокупность полей, каждое из которых реплицируется в соответствующую колонку таблицы. Каждая колонка таблицы имеет соответствующий полю сущности тип данных.

В приложении существует 5 типов сущностей: ключевое слово, расписание, интервал времени, передаваемая и получаемая единицы конфиденциальной информации. Каждая сущность является сериализуемой (то есть, может быть представлена в удобном для передачи виде) и имеет свой уникальный идентификатор. Идентификаторы заполняются автоматически при сохранении сущностей в базу данных.

Ключевое слово используется для аутентификации клиентов при передаче информации. Это слово передается вместе с конфиденциальными данными в заголовке http-запроса. Сами же конфиденциальные данные отправляются телом http-запроса.

Расписание представляет собой совокупность упорядоченных интервалов времени, а также содержит поле – IP-адрес клиента, которому принадлежит текущее расписание. Расписание изначально присылается оператором на сервер в JSON-формате с заранее определенными временными интервалами. В случае если данные в расписании являются неактуальными или некорректными, они отбрасываются проверяющим сервисом, который выдает ошибку и посылает сигнал в контроллер, который затем отправляет соответствующий статус пользователю и выводит сообщение в журнал событий.

Интервал времени помимо идентификатора содержит в себе два поля: время начала и время конца промежутка времени. Эти поля играют роли границ временного интервала.

Принимаемая и получаемая конфиденциальная информация – это по содержанию одна и та же сущность, однако их было создано две с целью налаживания работы с технологией ORM, реализацией которой в данном случае является Hibernate. Чтобы предотвратить отправку полученных данных в таблицу, предназначенную исключительно для данных, подлежащих передаче, для них была создана отдельная таблица, при этом использование технологии ORM предполагает использование отдельных таблиц для каждой сущности приложения.

## Репозитории, сервисы и контроллеры

Каждой сущности принадлежит соответствующий ей репозиторий, который помогает приложению работать с ней и с предназначенной для нее таблицей в базе данных. На сервере достаточно 3-х сущностей: конфиденциальные данные для отправки, расписание и интервал времени. На клиенте используются все 5.

Помимо сервисов, являющихся покрытием для репозиториев, существуют и другие сервисы, необходимые для достижения других целей. Таким образом, можно разделить сервисы на 3 группы: сервисы для покрытия репозиториев, сервисы для выполнения команд и сервисы для выполнения бизнес-логики.

Сервисами для покрытия репозиториев являются обычные сервисы, количество которых равно количеству репозиториев (за исключением одного, который из-за особенностей технологии Hibernate был не нужен – сервиса для интервалов времени).

Сервисы для выполнения команд представляют собой классы, которые используют написанную утилиту запуска команд в командной строке Windows. В эти сервисы входят: сервис для работы с блокировкой и разблокировкой IP-адресов и сервис для блокировки и разблокировки сетевых интерфейсов.

К сервисам для другой бизнес-логики относятся следующие: сервис для планирования задач и сервис для проверки корректности расписания.

## Планируемые задачи

Для контроля запуска и завершения передачи данных используются задачи, которые запускаются в отдельных демон-потоках в определенное время. За запуск задач в назначенное время несет ответственность планировщик задач, реализованный на основе существующего стандартного планировщика задач из JDK (Java Development Kit). Каждая задача представляет собой некоторую часть кода, которая должна начать свое выполнение в назначенное время.

## Шифрование данных и ключ авторизации

Перед передачей данных с клиента-отправителя данные проходят стадию шифрования. В качестве алгоритма шифрования был выбран алгоритм AES и режим CBC. Режим CBC был выбран исходя из двух его главных достоинств: постоянная скорость обработки блоков и возможность распараллеливания шифрования. Это позволяет быстро и эффективно шифровать данные. Дешифрация же данных происходит в однопоточном режиме. Секретный ключ состоит из 256 битов (32 байта). Секретные ключи могут быть сгенерированы с помощью внутреннего механизма приложения (случайный ключ), либо могут быть загружены из файла уже с готовым ключом. Используемые файлы с секретными ключами имеют расширение pckey. Для успешных шифрования и дешифрования ключ должен совпадать на передающей и приемной сторонах. В противном случае информация может быть утеряна на приемной стороне. Данные на сервер приходят уже в зашифрованном виде, при этом сторона сервера ничего не знает о ключе и об алгоритме шифрования. Таким образом, если сервер получает данные с передающей стороны и по какой-либо причине не может передать полученные данные на получающую сторону, данные временно сохранятся на сервере в зашифрованном виде. В случае компрометации зашифрованные данные практически невозможно расшифровать (исходя из постепенного устаревания данных).

## Пользовательский интерфейс

Для удобства использования приложение оснащено удобным пользовательским интерфейсом, в котором можно настроить все необходимые входные данные, включая данные об используемой базе данных, используемые порты для установки на них слушателей http-сервера. Так же необходимо указать ключевое слово и в том, и в другом интерфейсе. Пользовательские интерфейсы различаются для клиентского и серверного программных обеспечений. Примеры графических интерфейсов для клиента и сервера изображены на рисунока 4, 5 и 6.

На клиенте пользователю необходимо настроить адреса (IP-адрес и порт) сервера и клиента-приемника, а также полное название файла с секретным ключом, который будет использоваться при шифровании конфиденциальных данных.

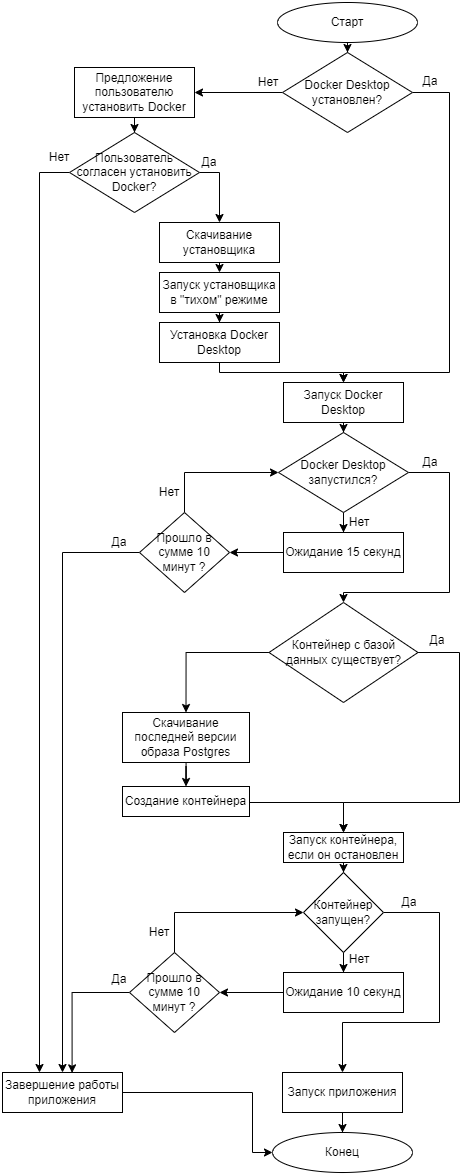
На стороне сервера необходимо настроить адреса клиентов, для которых могут быть отправлены расписания работы. В списке адресов должен быть хотя бы один адрес клиента.

## Развертка базы данных

В представляемом программном обеспечении используется база данных на основе Postgres. Для работы программного обеспечения экземпляр базы данных должен быть развернут в Docker-контейнере.

## Установщик приложения

Поскольку для использования программного обеспечения необходимы некоторые дополнительные инструменты, установить и настроить необходимые компоненты поможет установщик приложения, который запускается при старте работы программы. Если все необходимые инструменты установлены на рабочей станции, установщик проигнорирует все свои шаги и передаст управление стартеру приложения. Установщик имеет несколько этапов работы, что показано на рисунке 2. Первым шагом установщика является проверка наличия установленного программного обеспечения Docker Desktop. Если оно не установлено в системе, пользователю предлагается его установить и автоматически настроить. Если пользователь откажется от установки, приложение завершит свою работу. Иначе происходят следующие этапы: скачивание установщика с официального сайта через инструмент wget, поставляемый вместе с программным обеспечением. Затем запускается установщик в «тихом» режиме, то есть без использования графического пользовательского интерфейса.

Рисунок 2. Жизненный цикл установщика ПО

После всех перечисленных действий происходит запуск установленного приложения. Запуск производится с таймаутом, выставленным в 10 минут. В течение этого времени программа периодически (раз в 15 секунд) проверяет запущенные в фоновом режиме процессы и ищет среди них Docker Desktop. Если в течение 10 минут Docker Desktop не запустился, происходит аварийный выход из программы. После запуска Docker Desktop происходит проверка существования Docker-контейнера в контексте приложения. Если его нет в списке существующих контейнеров, производятся два дополнительных шага, включающие в себя скачивание последней версии образа базы данных Postgres и создание контейнера. Если контейнер создан, производится его запуск с таймаутом в 10 минут. Установщик проверяет каждые 10 секунд статус контейнера, и если контейнер не запустится в течение таймаута, произойдет аварийная остановка приложения. После старта Docker-контейнера происходит запуск основного приложения.

После запуска серверного и клиентских приложений и их преждевременной настройки, работа программного обеспечения начинается с отправки расписаний для клиентов на сервер. Никакого графического интерфейса для этого программное обеспечение не предоставляется. Для работы с программным обеспечением необходимо установить программное обеспечение Postman. После этого оператором генерируются несколько расписаний, предназначенных для определенных клиентов. Эти расписания должны быть представлены в JSON-формате и помещены в тело http-запроса. Пример тела для http-запроса представлен на рисунке 3.

Если для одного клиента предназначено несколько расписаний, то первое из них отправляется клиенту сразу, а остальные остаются на сервере путем сохранения в базу данных, которые будут запрошены в порядке очереди клиентом по окончанию актуальности действующего расписания.

Если расписания будут настроены неверно, например, время окончания временного промежутка будет раньше начала этого же промежутка времени, или если временные промежутки не установлены в хронологическом порядке, то все расписания, передаваемые одним http-запросом, будут отброшены с выводом сообщения об ошибке и способе ее устранения.



Рисунок 3. Пример тела http-запроса с расписаниями для двух клиентов

Идентификаторы, показанные на рисунке 3, совершенно не обязательны для указания. Если они не указаны явно, они будут созданы автоматически.

После получения клиентом корректного расписания, планировщиком задач создаются несколько демон-потоков, которые вызовут процедуры, соответствующие старту и концу передачи конфиденциальной информации согласно указанным на расписании промежуткам времени. Если между промежутками времени существуют интервалы, в эти интервалы информация передаваться не будет. Когда актуальность расписания пропадает (заканчивается последний временной интервал), запускается дополнительная бизнес-логика, которая перед концом передачи информации уничтожает неактуальное расписание и делает http-запрос к серверу, в ответ на который сервер попытается предоставить новое расписание клиенту. Если такое расписание нашлось на сервере, сервер отдает это расписание в ответном http-ответе, удаляя при этом его из своей базы данных. В противоположном случае клиент просто завершает свою работу.

Информация передается непрерывно в период доступного для передачи интервала времени. Если передаваемая конфиденциальная информация на клиенте была передана вся, то клиент переключится в режим ожидания новых данных. Ожидание длится в течение двух минут. В случае если новые данные были получены – передача данных продолжается, а счетчик времени сбросится. Если же информация в течение выделенного времени ожидания не появляется, клиент удаляет все существующие расписания и завершает свою работу.

7 НАЧАЛО РАБОТЫ С ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ

**7.1 Описание графического пользовательского интерфейса**

**7.1.1 Клиентское приложение**

Пример графического интерфейса клиентского приложения представлен на рисунке 4.

Описание полей представлено в таблице 1.

Таблица 1. Описание полей графического интерфейса клиентского приложения

|  |  |  |
| --- | --- | --- |
| Название поля | Значение по умолчанию | Применение |
| Select instance | Client | Отображает текущий тип экземпляра используемого ПО |
| Input header key | private\_channel\_key | Ключевое слово, которое будет использоваться для первичной аутентификации входящих запросов. |
| Application port | 8081 | Порт, на котором будет открыт прослушиватель http-сервера. |
| PG instance name | private\_channel | Название экземпляра базы данных, которое будет использоваться при работе ПО. |
| PG username | postgres | Имя пользователя, которое будет использовано при авторизации в экземпляре базы данных. |
| PG password | postgres | Пароль пользователя, который будет использован при авторизации в экземпляре базы данных. |
| PG instance port | 7431 | Порт, на котором будет запущен экземпляр базы данных |
| Main database | private\_channel | Название используемой базы данных. |
| Using network interface | Первый элемент в списке выводимых сетевых интерфейсов при использовании команды ipconfig в командной строке | Сетевой интерфейс, который будет использован при передаче/приеме данных. |
| Server ip | 127.0.0.1:8080  А точно только 127.0.0.1 Обычно этот адрес зарезервирован для использования «сам на себя» | Полный IP адрес сервера |
| Receiver ip | 127.0.0.1:8081 | Полный IP адрес клиента-приемника |
| Secret key file path |  | Путь до файла с ключом. Может быть оставлен пустым, тогда приложение создаст ключ автоматически и поместит его в файл secret\_key.pckey рядом с JAR архивом ПО. |

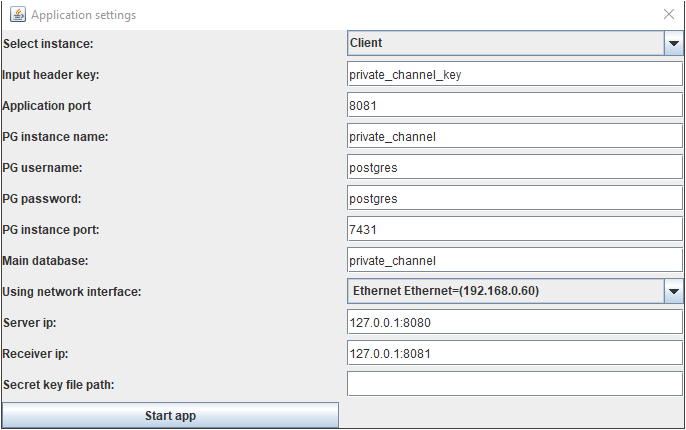


Рисунок 4. Пример графического интерфейса клиентского приложения

**7.1.2 Серверное приложение**

Пример графического интерфейса серверного приложения представлен на рисунках 5 и 6.

Описание полей, представленных в графическом интерфейсе серверного приложения, представлено в таблице 2.

Таблица 2. Описание полей графического интерфейса клиентского приложения

|  |  |  |
| --- | --- | --- |
| Название поля | Значение по умолчанию | Применение |
| Select instance | Client | Отображает текущий тип экземпляра используемого ПО |
| Input header key | private\_channel\_key | Ключевое слово, которое будет использоваться для первичной аутентификации входящих запросов. |
| Application port | 8081 | Порт, на котором будет открыт прослушиватель http-сервера. |
| PG instance name | private\_channel | Название экземпляра базы данных, которое будет использоваться при работе ПО. |
| PG username | postgres | Имя пользователя, которое будет использовано при авторизации в экземпляре базы данных. |
| PG password | postgres | Пароль пользователя, который будет использован при авторизации в экземпляре базы данных. |
| PG instance port | 7431 | Порт, на котором будет запущен экземпляр базы данных |
| Main database | private\_channel | Название используемой базы данных. |
| Using network interface | Первый элемент в списке выводимых сетевых интерфейсов при использовании команды ipconfig в командной строке | Сетевой интерфейс, который будет использован при передаче/приеме данных. |
| Clients Ips for connection |  | Список адресов клиентов, с которыми будет производиться взаимодействие сервера. |



Рисунок 5. Пример графического интерфейса клиентского приложения (основные настройки)

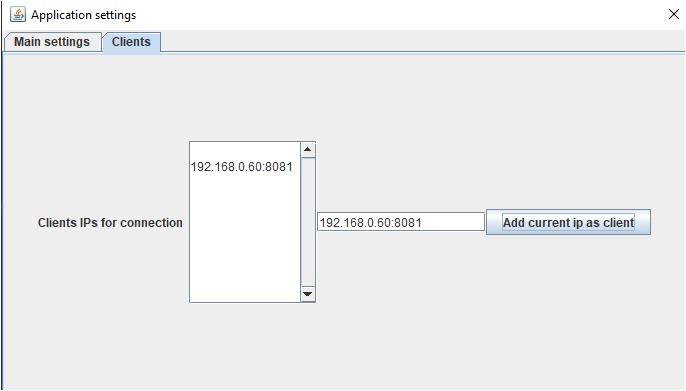


Рисунок 6. Пример графического интерфейса клиентского приложения (адреса клиентов)

Для добавления клиента в список клиентов, с которыми будет установлено взаимодействие сервера, необходимо указать в редактируемом поле полный IP адрес клиента, с которым предполагается работа, а затем необходимо нажать на кнопку “Add current ip as client”.

**7.2 Подготовка приложения к работе**

Перед стартом приложения необходимо заполнить все поля, указанные в графическом пользовательском интерфейсе клиентов. Необходимо корректно указать сетевой интерфейс, который будет использоваться для передачи данных. Если интерфейс будет выбран другой – канал будет уязвим, поскольку физические порты перекроются именно у выбранного сетевого интерфейса, а не у предполагаемого. Настроить сервер, указав использующийся для передачи данных сетевой интерфейс и добавив IP-адреса запущенных ранее клиентов. После этого, работу с приложением можно начинать.

**7.3 Запуск работы основного функционала**

Перед началом работы программы клиенты и сервер находятся в состоянии ожидания расписания. Расписание генерируется оператором и отправляется на сервер http-запросом с помощью программного обеспечения Postman.

Структура запроса:

* http-метод запроса: POST;
* Request url: {server\_ip}/api/v1/server/schedules;
* Body type: raw;
* Body: {schedules array in JSON format}

После отправки запроса в журнале клиентов, которым были предназначены отправленные расписания, можно найти записи о времени старта работы. Запуск работы будет воспроизведен согласно первому промежутку времени актуального расписания. Запуск работы подразумевает собой запуск передачи данных из одного клиента на другой, то есть из базы данных клиента-передатчика в базу данных клиента-приемника. Если в расписании между указанными временными промежутками существуют промежутки времени, которые не относятся к допустимым интервалам для передачи информации, то в эти промежутки времени передача информации не будет осуществляться. Передача данных будет завершена на начале «нерабочего» промежутка и начала (может «начата») после него.

Если расписание потеряет свою актуальность (последний промежуток времени закончился), клиенты автоматически запросят у сервера новые расписания для работы. Если расписания, предназначенные обращающимся клиентам, на сервере отсутствуют, клиенты завершают свою работу. Сервер остается в рабочем состоянии в режиме ожидания новых расписаний от оператора.

Для успешной передачи данных с клиента-передатчика на клиент-приемник, передаваемые кофниденциальные данные должны храниться в базе данных клиента-передатчика в таблице conf\_info. Если данные отсутствуют после старта передачи данных в течение 60 секунд – клиенты завершают свою работу. Если данные присутствуют, будет осуществлена их передача с клиента-передатчика на клиент-приемник. Переданные данные будут храниться в базе данных клиента-приемника таблице received\_info.

# 8 ВЫВОДЫ И РЕЗУЛЬТАТ

В результате выполнения курсовой работы был написан программный продукт, позволяющий передавать конфиденциальную информацию через физически отключаемый канал. Результаты тестов показали, что разработанная программа успешно справляется с поставленной задачей. Все созданные сервисы были проверены и отлажены, что значительно минимизировало вероятность возникновения нештатной ситуации и обеспечило работоспособность программы.

# 9 КОМПЛЕКСНАЯ ОТЛАДКА

Отладка проходила в несколько этапов. Первым этапом служила проверка передачи данных от оператора на сервер (расписаний). Расписания в отладочной информации присутсвовали без потерь. Далее проверялись отправка первого расписания на клиентов и сохранение других расписаний в базе данных сервера. Были отлажены все планируемые задачи и проконтроллировано их создание при получении расписания клиентами, старте и окончании передачи данных, а также планируемая задача, отвечающая за запрос нового расписания клиентами у сервера.

В ходе работы были проверены и отлажены сервисы для работы с командной строкой и IP-адресами (блокировка и разблокировка).

Были отлажены работы сервисов хеширования данных алгоритмом SHA512 и шифрования данных алгоритмом AES.

Была отлажена и настроена работа сервиса, отвечающего за включение и отключение сетевого интерфейса.

Была проверена работа скриптов, создающих структуру базы данных в Docker-контейнере.

Также в ходе работы был отлажен установщик программы с проверкой наличия приложения Docker, установленного на рабочую станцию, и существования Docker-контейнера.

ЗАКЛЮЧЕНИЕ

В результате работы над курсовым проектом были изучены механизмы отключения физического канала, были опробованы работы с IP-адресами, сетевыми интерфейсами, связкой приложения с базой данных, а также создание планируемых задач. Таким образом, на выходе получился готовый программный продукт, который можно использовать в целях обеспечения безопасности передающейся информации.

СПИСОК ИСПОЛЬЗОВАННХ ИСТОЧНИКОВ

1. Козмина Ю. Spring 5 для профессионалов // 5-е издание. – 2020. – 626 – 677 с.   
2. @Component vs @Repository and @Service in Spring [Электронный ресурс]: Вэб-сайт: <https://www.baeldung.com/spring-component-repository-service>.  
3. Документация Postgres [Электронный ресурс]: <https://www.postgresql.org/docs/14/index.html>.  
4. Документация hibernate [Электронный ресурс]: <https://hibernate.org/orm/documentation/6.0/>.

5. Документация netsh [Электронный ресурс]: <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/netsh>.

ПРИЛОЖЕНИЕ А

Код программы

**Модуль** common

**Пакет** org.ssau.privatechannel.constants

public interface Endpoints {

String API\_V1\_SERVER = "/api/v1/server";

String API\_V1\_CLIENT = "/api/v1/client";

String UPLOAD\_DATA = "/upload-data";

String GENERATE\_DATA = "/generate-data/{count}";

String GENERATE\_SCHEDULE = "/generate-schedule/{duration}";

String SCHEDULES = "/schedules";

String SCHEDULE = "/schedule";

String GET\_NEW\_SCHEDULE = "/getNewSchedule";

}

public interface FirewallRuleNames {

String

BLOCK\_IP = "BlockIp",

UNBLOCK\_IP = "UnblockIp";

}

public interface Parameters {

Long MAX\_ID = 10000000L;

}

public interface SystemProperties {

String INSTANCE = "instance";

String HEADER\_KEY = "header\_key";

String RECEIVER\_IP = "receiver\_ip";

String SERVER\_IP = "server\_ip";

String CURRENT\_IP = "current\_ip";

String DB\_URL = "db\_url";

String DB\_USER = "db\_user";

String DB\_PASSWORD = "db\_password";

String DB\_PORT = "db\_port";

String MAIN\_DB = "main\_db";

String DB\_INSTANCE = "db\_instance";

String NETWORK = "network";

String APP\_PORT = "app\_port";

}

public interface UrlSchemas {

String HTTP = "http://";

}

**Пакет** org.ssau.privatechannel.exception

public class BadAlgorythmLengthException extends Exception {

public BadAlgorythmLengthException(String message) {

super(message);

}

}

public class DockerMissingException extends Exception {

public DockerMissingException(String message) {

super(message);

}

}

public class HeaderKeyNotActualException extends Exception {

public HeaderKeyNotActualException(String message) {

super(message);

}

}

public class InvalidInstanceTypeException extends Exception {

public InvalidInstanceTypeException(String message) {

super(message);

}

}

public class ValidationException extends Exception {

public ValidationException(String message) {

super(message);

}

}

**Пакет** org.ssau.privatechannel.firetasks

@Slf4j

public class StartDataTransferringTask extends TimerTask {

public static final String THREAD\_NAME = "DataTransferringFromClient";

private static final String SEND\_DATA\_ENDPOINT = Endpoints.API\_V1\_SERVER + Endpoints.UPLOAD\_DATA;

private static final String SCHEMA = "http://";

private final ConfidentialInfoService infoService;

private final IpService ipService;

private final NetworkAdapterService networkAdapterService;

private final RestTemplate restTemplate;

private String receiverIp;

private static final Integer WAIT\_DELAY = 10;

private static final Integer WAIT\_TIMEOUT\_SECONDS = 600;

private static final Integer DELAY\_BETWEEN\_BATCHES = 1;

public StartDataTransferringTask(ConfidentialInfoService infoService,

RestTemplate restTemplate,

IpService ipService,

NetworkAdapterService networkAdapterService) {

this.infoService = infoService;

this.restTemplate = restTemplate;

this.ipService = ipService;

this.networkAdapterService = networkAdapterService;

}

@SneakyThrows

@Override

public void run() {

ipService.deleteRuleByName(FirewallRuleNames.BLOCK\_IP);

String currentInterface = SystemContext.getProperty(SystemProperties.NETWORK);

networkAdapterService.enableInterfaces(currentInterface);

String senderIp = SystemContext.getProperty(SystemProperties.CURRENT\_IP);

Thread thread = new Thread(() -> {

log.info("Data transferring started between clients with ips {} and {}", senderIp, receiverIp);

int currentWaitTime = 0;

while (true) {

List<ConfidentialInfo> batch = infoService.nextBatch();

try {

batch = AESUtil.encryptBatchInfo(batch, KeyHolder.getKey(), KeyHolder.getIv());

} catch (Exception e) {

log.error("Error during data encryption", e);

break;

}

if (batch.isEmpty()) {

if (currentWaitTime >= WAIT\_TIMEOUT\_SECONDS) {

log.info("No more information from client with IP={}. Exiting program", senderIp);

log.info("Transferring data between clients with ips {} and {} completed", senderIp, receiverIp);

System.exit(0);

}

log.info("All information sent. Waiting for new info...");

currentWaitTime += WAIT\_DELAY;

try {

Thread.sleep(TimeUnit.SECONDS.toMillis(WAIT\_DELAY));

continue;

} catch (InterruptedException e) {

log.error("Interrupting thread. Reason: {}", e.getMessage());

break;

}

}

currentWaitTime = 0;

for (ConfidentialInfo next : batch) {

next.setSenderIP(senderIp);

if (Objects.isNull(receiverIp)) {

log.warn("Receiver IP not provided. Trying to get this IP from data");

if (Objects.isNull(next.getReceiverIP())){

log.error("Could not get receiver IP from data [ID = {}]. Exiting", next.getId());

return;

}

}

else

next.setReceiverIP(receiverIp);

}

String serverIp = SystemContext.getProperty(SystemProperties.SERVER\_IP);

String serverAddress = SCHEMA + serverIp + SEND\_DATA\_ENDPOINT;

HttpHeaders headers = new HttpHeaders();

headers.add(SystemProperties.HEADER\_KEY, SystemContext.getProperty(SystemProperties.HEADER\_KEY));

HttpEntity<List<ConfidentialInfo>> entity = new HttpEntity<>(batch, headers);

ResponseEntity<String> response;

try {

Thread.sleep(TimeUnit.SECONDS.toMillis(DELAY\_BETWEEN\_BATCHES));

response = restTemplate.postForEntity(serverAddress, entity, String.class);

}

catch (Throwable e) {

log.error("Could not send data to server", e);

break;

}

if (response.getStatusCode() != HttpStatus.OK) {

log.error("Could not send data to server: server returned status {}", response.getStatusCode());

break;

}

infoService.deleteBatch(batch);

}

});

ThreadsHolder.addAndRunThread(THREAD\_NAME, thread);

}

public void setReceiverIp(String receiverIp) {

this.receiverIp = receiverIp;

}

}

**Пакет** org.ssau.privatechannel.model

@Entity

@Table(name = Tables.AUTHORIZATION\_KEY)

@NamedQuery(

name = QueryNames.GET\_KEY,

query = Queries.GET\_KEY)

public class AuthorizationKey {

@Id

@Column(name = Columns.ID, nullable = false)

private Long id;

@Column(name = Columns.HASH, nullable = false)

private String hash;

public AuthorizationKey(Long id, String key) {

this.id = id;

this.hash = Sha512EncoderService.getHash(key);

}

public AuthorizationKey() {

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getHash() {

return hash;

}

public void setHash(String hash) {

this.hash = hash;

}

public static abstract class QueryNames {

public static final String GET\_KEY = "AuthorizationKey.get";

}

public static abstract class Queries {

public static final String GET\_KEY = "select key from AuthorizationKey key";

}

public static abstract class Tables {

public static final String AUTHORIZATION\_KEY = "authorization\_key";

}

private static abstract class Columns {

public static final String ID = "id";

public static final String HASH = "hash";

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;

AuthorizationKey that = (AuthorizationKey) o;

return id != null && Objects.equals(id, that.id);

}

@Override

public int hashCode() {

return getClass().hashCode();

}

}

@Entity

@Table(name = ConfidentialInfo.Tables.CONFIDENTIAL\_INFORMATION)

@NamedQuery(

name = QueryNames.GET\_ALL\_INFO,

query = Queries.GET\_ALL\_INFO)

@NamedQuery(

name = QueryNames.GET\_ALL\_INFO\_BY\_IDS,

query = Queries.GET\_ALL\_INFO\_BY\_IDS)

@NamedNativeQuery(

name = QueryNames.GET\_BATCH\_INFO,

query = Queries.GET\_BATCH\_INFO,

resultClass = ConfidentialInfo.class)

@TypeDef(name = "jsonb", typeClass = JsonBinaryType.class)

@ToString

@Builder

@AllArgsConstructor

@Getter

@Setter

public class ConfidentialInfo implements Serializable {

@Id

@Column(name = Columns.RECORD\_ID, nullable = false)

private Long id;

@Type(type = "jsonb")

@Column(name = Columns.TEXT\_DATA, columnDefinition = "json")

private Map<String, Object> data;

@Column(name = Columns.SENDER\_IP)

private String senderIP;

@Column(name = Columns.RECEIVER\_IP)

private String receiverIP;

public ConfidentialInfo() {

}

public String getSenderIP() {

return senderIP;

}

public void setSenderIP(String senderIP) {

this.senderIP = senderIP;

}

public String getReceiverIP() {

return receiverIP;

}

public void setReceiverIP(String receiverIP) {

this.receiverIP = receiverIP;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public Map<String, Object> getData() {

return data;

}

public void setData(Map<String, Object> data) {

this.data = data;

}

public static abstract class Queries {

public static final String GET\_ALL\_INFO = "select info from ConfidentialInfo info";

public static final String GET\_ALL\_INFO\_BY\_IDS = "select info from ConfidentialInfo info where info.id in (:ids)";

public static final String GET\_BATCH\_INFO = "select inf.record\_id, inf.text\_data, inf.receiver\_ip, inf.sender\_ip from conf\_info inf limit 10";

}

public static abstract class QueryNames {

public static final String GET\_ALL\_INFO = "ConfidentialInfo.findAll";

public static final String GET\_ALL\_INFO\_BY\_IDS = "ConfidentialInfo.findAllByIds";

public static final String GET\_BATCH\_INFO = "ConfidentialInfo.getBatch";

}

public static abstract class Tables {

public static final String CONFIDENTIAL\_INFORMATION = "conf\_info";

}

private static abstract class Columns {

public static final String RECORD\_ID = "record\_id";

public static final String SENDER\_IP = "sender\_ip";

public static final String RECEIVER\_IP = "receiver\_ip";

public static final String TEXT\_DATA = "text\_data";

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;

ConfidentialInfo that = (ConfidentialInfo) o;

return id != null && Objects.equals(id, that.id);

}

@Override

public int hashCode() {

return getClass().hashCode();

}

}

@Entity

@Table(name = ReceivedInformation.Tables.CONFIDENTIAL\_INFORMATION)

@NamedQuery(

name = QueryNames.GET\_ALL\_INFO,

query = Queries.GET\_ALL\_INFO)

@NamedNativeQuery(

name = QueryNames.GET\_BATCH\_INFO,

query = Queries.GET\_BATCH\_INFO,

resultClass = ReceivedInformation.class)

@TypeDef(name = "jsonb", typeClass = JsonBinaryType.class)

@ToString

@AllArgsConstructor

@Builder

@Getter

@Setter

public class ReceivedInformation {

@Id

@Column(name = Columns.RECORD\_ID, nullable = false)

private Long id;

@Type(type = "jsonb")

@Column(name = Columns.TEXT\_DATA, columnDefinition = "json")

private Map<String, Object> data;

@Column(name = Columns.SENDER\_IP, nullable = false)

private String senderIP;

@Column(name = Columns.RECEIVER\_IP, nullable = false)

private String receiverIP;

public ReceivedInformation() {

}

public String getSenderIP() {

return senderIP;

}

public void setSenderIP(String senderIP) {

this.senderIP = senderIP;

}

public String getReceiverIP() {

return receiverIP;

}

public void setReceiverIP(String receiverIP) {

this.receiverIP = receiverIP;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public Map<String, Object> getData() {

return data;

}

public void setData(Map<String, Object> data) {

this.data = data;

}

public static abstract class Queries {

public static final String GET\_ALL\_INFO = "select info from ReceivedInformation info";

public static final String GET\_BATCH\_INFO = "select inf.record\_id, inf.text\_data from conf\_info inf limit 10";

}

public static abstract class QueryNames {

public static final String GET\_ALL\_INFO = "ReceivedInformation.findAll";

public static final String GET\_BATCH\_INFO = "ReceivedInformation.getBatch";

}

public static abstract class Tables {

public static final String CONFIDENTIAL\_INFORMATION = "received\_info";

}

private static abstract class Columns {

public static final String RECORD\_ID = "record\_id";

public static final String SENDER\_IP = "sender\_ip";

public static final String RECEIVER\_IP = "receiver\_ip";

public static final String TEXT\_DATA = "text\_data";

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;

ReceivedInformation that = (ReceivedInformation) o;

return id != null && Objects.equals(id, that.id);

}

@Override

public int hashCode() {

return getClass().hashCode();

}

}

@Entity

@Table(name = Schedule.Tables.SCHEDULE)

@NamedQuery(name = QueryNames.FIND\_ALL,

query = Queries.FIND\_ALL)

@NamedQuery(name = QueryNames.FIND\_FIRST\_BY\_IP,

query = Queries.FIND\_FIRST\_BY\_IP)

@NoArgsConstructor

@ToString

@AllArgsConstructor

@Builder

@Getter

@Setter

public class Schedule implements Serializable {

@Id

@Column(name = Columns.SCHEDULE\_ID, nullable = false)

private Long id;

@OneToMany(mappedBy = "id", cascade = CascadeType.ALL, fetch = FetchType.EAGER)

private List<TimeFrame> timeFrames;

@Column(name = Columns.CLIENT\_IP)

private String clientIp;

public List<TimeFrame> getTimeFrames() {

return timeFrames;

}

public void setTimeFrames(List<TimeFrame> time\_frames) {

this.timeFrames = time\_frames;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getClientIp() {

return clientIp;

}

public void setClientIp(String clientIp) {

this.clientIp = clientIp;

}

public static abstract class Queries {

public static final String FIND\_ALL =

"select distinct s from Schedule s";

public static final String FIND\_FIRST\_BY\_IP =

"select distinct s from Schedule s where s.clientIp = :ip";

}

public static abstract class QueryNames {

public static final String FIND\_ALL = "Schedule.findAll";

public static final String FIND\_FIRST\_BY\_IP = "Schedule.findFirstByIp";

}

public static abstract class Tables {

public static final String SCHEDULE = "schedule";

}

public static abstract class Columns {

public static final String SCHEDULE\_ID = "schedule\_id";

public static final String CLIENT\_IP = "client\_ip";

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;

Schedule schedule = (Schedule) o;

return id != null && Objects.equals(id, schedule.id);

}

@Override

public int hashCode() {

return getClass().hashCode();

}

}

@Entity

@Table(name = TimeFrame.Tables.TIME\_FRAME)

@NamedNativeQuery(name = QueryNames.SELECT\_TIMEFRAMES\_FOR\_SCHEDULE, query = Queries.SELECT\_TIMEFRAMES\_FOR\_SCHEDULE)

@NoArgsConstructor

@AllArgsConstructor

@ToString

@Builder

@Getter

@Setter

public class TimeFrame implements Serializable {

private static final String DATE\_PATTERN = "dd-MM-yyyy HH:mm:ss";

private static final String TIMEZONE = "Europe/Samara";

@Id

@Column(name = Columns.TIME\_FRAME\_ID, nullable = false)

private Long id;

@Column(name = Columns.START\_TIME)

@JsonFormat(shape = JsonFormat.Shape.STRING, pattern = DATE\_PATTERN, timezone = TIMEZONE)

@JsonSerialize(using = LocalDateTimeSerializer.class)

@JsonDeserialize(using = LocalDateTimeDeserializer.class)

private LocalDateTime startTime;

@Column(name = Columns.END\_TIME)

@JsonFormat(shape = JsonFormat.Shape.STRING, pattern = DATE\_PATTERN, timezone = TIMEZONE)

@JsonSerialize(using = LocalDateTimeSerializer.class)

@JsonDeserialize(using = LocalDateTimeDeserializer.class)

private LocalDateTime endTime;

@JsonIgnore

@ManyToOne

@JoinColumn(name = "schedule\_id")

private Schedule schedule;

public boolean isIntersectsWith(TimeFrame timeFrame) {

return isMomentInTimeFrame(timeFrame.startTime) || isMomentInTimeFrame(timeFrame.endTime) ||

timeFrame.isMomentInTimeFrame(startTime) || timeFrame.isMomentInTimeFrame(endTime);

}

public boolean isMomentInTimeFrame(LocalDateTime moment) {

return moment.isAfter(startTime) && moment.isBefore(endTime);

}

public LocalDateTime getStartTime() {

return startTime;

}

public void setStartTime(LocalDateTime startTime) {

this.startTime = startTime;

}

public LocalDateTime getEndTime() {

return endTime;

}

public void setEndTime(LocalDateTime endTime) {

this.endTime = endTime;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public Schedule getSchedule() {

return schedule;

}

public void setSchedule(Schedule schedule) {

this.schedule = schedule;

}

public static abstract class Queries {

public static final String SELECT\_TIMEFRAMES\_FOR\_SCHEDULE =

"select \* from time\_frame where schedule\_id = :schedule\_id";

}

public static abstract class QueryNames {

public static final String SELECT\_TIMEFRAMES\_FOR\_SCHEDULE = "TimeFrame.findAllWithSchedule";

}

public static abstract class Tables {

public static final String TIME\_FRAME = "time\_frame";

}

private static abstract class Columns {

public static final String TIME\_FRAME\_ID = "time\_frame\_id";

public static final String START\_TIME = "start\_time";

public static final String END\_TIME = "end\_time";

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)) return false;

TimeFrame timeFrame = (TimeFrame) o;

return id != null && Objects.equals(id, timeFrame.id);

}

@Override

public int hashCode() {

return getClass().hashCode();

}

}

**Пакет** org.ssau.privatechannel.repository

public abstract class AbstractRepository {

@Autowired

protected EntityManager entityManager;

}

@Slf4j

@Repository

public class AuthKeyRepository extends AbstractRepository {

public AuthorizationKey get() {

try {

return entityManager.createNamedQuery(QueryNames.GET, AuthorizationKey.class).getSingleResult();

}

catch (Exception e) {

log.warn("Header key not placed in database");

return null;

}

}

@Transactional

public void set(AuthorizationKey info) {

entityManager.persist(info);

}

@Transactional

public void delete(AuthorizationKey info) {

entityManager.remove(info);

}

private static abstract class QueryNames {

public static final String GET = "AuthorizationKey.get";

}

}

@Repository

public class ConfidentialInfoRepository extends AbstractRepository {

public List<ConfidentialInfo> findAllByIds(List<Long> ids) {

return entityManager.createNamedQuery(NamedQueries.FIND\_ALL\_BY\_IDS,

ConfidentialInfo.class).setParameter(QueryParams.IDS, ids).getResultList();

}

public List<ConfidentialInfo> nextBatch() {

return entityManager.createNamedQuery(NamedQueries.GET\_BATCH,

ConfidentialInfo.class).getResultList();

}

@Transactional

public void deleteBatch(List<ConfidentialInfo> batch) {

List<Long> ids = new ArrayList<>();

for (ConfidentialInfo record : batch) {

ids.add(record.getId());

}

List<ConfidentialInfo> allByIds = findAllByIds(ids);

for (ConfidentialInfo currentRecord : allByIds) {

entityManager.remove(currentRecord);

}

}

@Transactional

public void addAll(List<ConfidentialInfo> info) {

for (ConfidentialInfo currentRecord : info) {

entityManager.merge(currentRecord);

}

}

public int getInfoCount() {

return ((BigInteger)

(entityManager.createNativeQuery("select count(\*) from conf\_info").getSingleResult())).intValue();

}

private static class NamedQueries {

public static final String FIND\_ALL\_BY\_IDS = "ConfidentialInfo.findAllByIds";

public static final String GET\_BATCH = "ConfidentialInfo.getBatch";

}

private static abstract class QueryParams {

public static final String IDS = "ids";

}

}

@Repository

public class ScheduleRepository extends AbstractRepository {

public Schedule findById(Long id) {

return entityManager.find(Schedule.class, id);

}

public Schedule findNextForIp(String ip) {

try {

List<Schedule> resultList = entityManager.createNamedQuery(NamedQueries.FIND\_FIRST\_BY\_IP, Schedule.class)

.setParameter(QueryParams.IP, ip).getResultList();

if (Objects.isNull(resultList) || resultList.isEmpty()) {

return null;

}

return resultList.get(0);

} catch (NoResultException e) {

return null;

}

}

@Transactional

public void add(Schedule schedule) {

entityManager.persist(schedule);

}

@Transactional

public void addAll(List<Schedule> schedules) {

for (Schedule currentRecord : schedules) {

entityManager.persist(currentRecord);

}

}

@Transactional

public void delete(Schedule schedule) {

Schedule scheduleForDelete = findById(schedule.getId());

entityManager.remove(scheduleForDelete);

}

private static class NamedQueries {

public static final String FIND\_FIRST\_BY\_IP = "Schedule.findFirstByIp";

}

private static class QueryParams {

public static final String IP = "ip";

}

}

@Repository

public class TimeFrameRepository extends AbstractRepository {

@SuppressWarnings("unchecked")

public List<TimeFrame> findAllForSchedule(Long scheduleId) {

return entityManager.createNativeQuery("select \* from time\_frame " +

"where schedule\_id = :schedule\_id", TimeFrame.class)

.setParameter(QueryParams.SCHEDULE\_ID, scheduleId).getResultList();

}

private static class QueryParams {

public static final String SCHEDULE\_ID = "schedule\_id";

}

public void deleteAll(List<TimeFrame> timeFrames) {

for (TimeFrame timeFrame : timeFrames)

entityManager.remove(timeFrame);

}

}

**Пакет** org.ssau.privatechannel.service

@Slf4j

@Service

public class AuthKeyService {

private final AuthKeyRepository authKeyRepository;

private static final Random RANDOMIZER = new Random();

@Autowired

public AuthKeyService(AuthKeyRepository authKeyRepository) {

this.authKeyRepository = authKeyRepository;

}

public AuthorizationKey get() {

return authKeyRepository.get();

}

public void set(AuthorizationKey key) {

if (Objects.isNull(key.getId())) {

key.setId(Math.abs(RANDOMIZER.nextLong()) % Parameters.MAX\_ID);

}

authKeyRepository.set(key);

}

public void delete(AuthorizationKey info) {

authKeyRepository.delete(info);

}

public boolean isActual(String key) {

AuthorizationKey actualKey = authKeyRepository.get();

AuthorizationKey expectedKey = new AuthorizationKey(actualKey.getId(), key);

return actualKey.equals(expectedKey);

}

public void generateNewHeaderKey() {

String headerKey = SystemContext.getProperty(SystemProperties.HEADER\_KEY);

AuthorizationKey previousKey = get();

if (Objects.isNull(previousKey))

log.warn("Auth service returned empty result. Key will be created");

else

delete(previousKey);

AuthorizationKey newKey = new AuthorizationKey(null, headerKey);

set(newKey);

}

}

@Service

public class ConfidentialInfoService {

private final ConfidentialInfoRepository infoRepository;

private static final Random RANDOMIZER = new Random();

@Autowired

public ConfidentialInfoService(ConfidentialInfoRepository infoRepository) {

this.infoRepository = infoRepository;

}

public List<ConfidentialInfo> findAllByIds(List<Long> ids) {

return infoRepository.findAllByIds(ids);

}

public List<ConfidentialInfo> nextBatch() {

return infoRepository.nextBatch();

}

public void deleteBatch(List<ConfidentialInfo> batch) {

infoRepository.deleteBatch(batch);

}

public void addAll(List<ConfidentialInfo> info) {

for (ConfidentialInfo confidentialInfo : info) {

if (Objects.isNull(confidentialInfo.getId())) {

confidentialInfo.setId(Math.abs(RANDOMIZER.nextLong()) % Parameters.MAX\_ID);

}

}

infoRepository.addAll(info);

}

public int getInfoCount() {

return infoRepository.getInfoCount();

}

}

@Slf4j

@Service

public class IpService {

public Map<String, String> getAllInternalRegisteredIp() throws IOException {

Map<String, String> result = new HashMap<>();

List<String> consoleOutput = CommandRunner.runQuietWithReturn(Commands.GET\_ALL\_INTERFACES\_INFO);

List<String> interfaces = new ArrayList<>();

Pattern adapterNamePattern = Pattern.compile("[^.]+[A-z\\d-() ]+:");

for (String resString : consoleOutput) {

boolean isInterfaceFound;

if (resString.matches(adapterNamePattern.pattern()) && !resString.isEmpty()) {

Matcher matcher = Pattern.compile("[A-z0-9-() ]+").matcher(resString);

isInterfaceFound = matcher.find();

if (isInterfaceFound && !matcher.group().equals(" ")) {

interfaces.add(matcher.group());

}

}

Matcher ipv4Matcher = Pattern.compile("IPv4-").matcher(resString);

if (ipv4Matcher.find()) {

Matcher matcher = Pattern.compile("[0-9]+.[0-9]+.[0-9]+.[0-9]+").matcher(resString);

boolean isIpV4Found = matcher.find();

if (isIpV4Found) {

result.put(interfaces.get(interfaces.size() - 1), matcher.group());

}

}

}

return result;

}

public void blockIP(IpAddress ipAddress, String ruleName) throws IOException {

String command = String.format(Commands.BLOCK\_IP\_ADDRESS, ruleName, Ports.HTTP, ipAddress.getIp());

CommandRunner.run(command);

log.info(String.format("IP address \"%s\" blocked with rule name = \"%s\"", ipAddress.getIp(), ruleName));

}

public void deleteRuleByName(String ruleName) throws IOException {

String command = String.format(Commands.DELETE\_RULE, ruleName);

CommandRunner.run(command);

log.info(String.format("Rule name \"%s\" deleted", ruleName));

}

@Data

@AllArgsConstructor

public static class IpAddress {

private String ip;

}

private static abstract class Commands {

public static final String

GET\_ALL\_INTERFACES\_INFO = "ipconfig",

BLOCK\_IP\_ADDRESS = "netsh advfirewall firewall add rule name=\"%s\" protocol=TCP "

+ "localport=%s action=block dir=IN remoteip=%s",

DELETE\_RULE =

"netsh advfirewall firewall delete rule name=\"%s\"";

}

private static abstract class Ports {

public static final String

HTTP = "8080";

}

}

@Slf4j

@Service

public class NetworkAdapterService {

public void enableInterfaces(String netInterface) throws IOException {

String command = String.format(Commands.ENABLE\_INTERFACE, netInterface);

CommandRunner.runQuiet(command);

log.info(String.format("%s interface now enabled", netInterface));

}

public void disableInterfaces(String netInterface) throws IOException {

String command = String.format(Commands.DISABLE\_INTERFACE, netInterface);

CommandRunner.runQuiet(command);

log.info(String.format("\"%s\" interface now disabled", netInterface));

}

private static abstract class Commands {

public static final String

DISABLE\_INTERFACE = "netsh interface set interface \"%s\" disable",

ENABLE\_INTERFACE = "netsh interface set interface \"%s\" enable";

}

}

@Slf4j

@Service

public class ScheduleService {

private final ScheduleRepository scheduleRepository;

private final TimeFrameRepository timeFrameRepository;

private static final Random RANDOMIZER = new Random();

@Autowired

public ScheduleService(ScheduleRepository scheduleRepository,

TimeFrameRepository timeFrameRepository) {

this.scheduleRepository = scheduleRepository;

this.timeFrameRepository = timeFrameRepository;

}

public Schedule findNextByIp(String ip) {

Schedule schedule = scheduleRepository.findNextForIp(ip);

if (Objects.isNull(schedule)) {

log.info("New schedule for client with ip = {} not found", ip);

return null;

}

List<TimeFrame> timeFrames = timeFrameRepository.findAllForSchedule(schedule.getId());

schedule.setTimeFrames(timeFrames);

log.info("New schedule for client with IP = {} is: {}", ip, schedule);

return schedule;

}

public void add(Schedule schedule) {

if (Objects.isNull(schedule.getId())) {

schedule.setId(Math.abs(RANDOMIZER.nextLong()) % Parameters.MAX\_ID);

}

for (TimeFrame timeFrame : schedule.getTimeFrames()) {

timeFrame.setSchedule(schedule);

}

scheduleRepository.add(schedule);

}

public void addAll(List<Schedule> schedules) {

for (Schedule schedule : schedules) {

if (Objects.isNull(schedule.getId())) {

schedule.setId(Math.abs(RANDOMIZER.nextLong()) % Parameters.MAX\_ID);

}

for (TimeFrame timeFrame : schedule.getTimeFrames()) {

timeFrame.setSchedule(schedule);

}

}

scheduleRepository.addAll(schedules);

}

@Transactional

public void delete(Schedule schedule) {

List<TimeFrame> timeFrames = timeFrameRepository.findAllForSchedule(schedule.getId());

timeFrameRepository.deleteAll(timeFrames);

scheduleRepository.delete(schedule);

}

public boolean isActualSchedule(Schedule schedule) {

for (TimeFrame timeFrame : schedule.getTimeFrames()) {

if (timeFrame.getStartTime().isBefore(LocalDateTime.now()))

return false;

}

return true;

}

}

@Slf4j

@Service

public class TimerService {

public void createTask(TimerTask task, LocalDateTime time) {

Date fireMoment = Date.from(time.atZone(ZoneId.systemDefault()).toInstant());

new Timer().schedule(task, fireMoment);

}

}

**Пакет** org.ssau.privatechannel.ui

@Slf4j

public class StartPage {

public static final String KEY\_FILE\_NAME = "secret\_key.pckey";

public static void show() throws IOException {

JDialog settingsWindow = new JDialog(new JFrame(), "Application settings", true);

settingsWindow.setSize(DefaultParams.WINDOW\_SIZE);

settingsWindow.setLocation(DefaultParams.LOCATION\_POINT);

settingsWindow.addWindowListener(new WindowAdapter() {

@Override

public void windowClosing(WindowEvent e) {

System.exit(0);

}

});

// Вспомогательная панель

JPanel grid = new JPanel();

GridLayout layout;

if (SystemContext.getProperty(SystemProperties.INSTANCE).equals(Instances.CLIENT)) {

layout = new GridLayout(DefaultParams.ROWS,

DefaultParams.COLUMNS,

DefaultParams.H\_GAP,

DefaultParams.V\_GAP);

} else {

layout = new GridLayout(DefaultParams.ROWS - 2,

DefaultParams.COLUMNS,

DefaultParams.H\_GAP,

DefaultParams.V\_GAP);

}

grid.setLayout(layout);

String[] instances = {SystemContext.getProperty(SystemProperties.INSTANCE)};

IpService ipService = new IpService();

Map<String, String> allInternalRegisteredIp = ipService.getAllInternalRegisteredIp();

List<String> interface2Ip = new ArrayList<>();

for (Map.Entry<String, String> currentPair : allInternalRegisteredIp.entrySet()) {

interface2Ip.add(currentPair.getKey() + "=(" + currentPair.getValue() + ")");

}

String[] interface2IpArray = interface2Ip.toArray(String[]::new);

JComboBox<String> instancesComboBox = new JComboBox<>(instances);

instancesComboBox.setEditable(false);

JComboBox<String> interfacesComboBox = new JComboBox<>(interface2IpArray);

// Selecting instance

grid.add(new JLabel("Select instance:"));

grid.add(instancesComboBox);

grid.add(new JLabel("Input header key:"));

JTextField keyPlaceholder = new JTextField("private\_channel\_key");

grid.add(keyPlaceholder);

// Setup application port

grid.add(new JLabel("Application port"));

JTextField appPort = instances[0]

.equals(Instances.SERVER)?new JTextField(DefaultParams.DEFAULT\_APP\_PORT):new JTextField("8081");

grid.add(appPort);

// DB settings

grid.add(new JLabel("PG instance name:"));

JTextField dbInstanceName = new JTextField("private\_channel");

grid.add(dbInstanceName);

grid.add(new JLabel("PG username:"));

JTextField pgUser = new JTextField("postgres");

grid.add(pgUser);

grid.add(new JLabel("PG password:"));

JTextField pgPassword = new JTextField("postgres");

grid.add(pgPassword);

grid.add(new JLabel("PG instance port:"));

JTextField dbPort = instances[0].equals(Instances.SERVER)?new JTextField("7430"):new JTextField("7431");

grid.add(dbPort);

grid.add(new JLabel("Main database:"));

JTextField mainDatabase = new JTextField("private\_channel");

grid.add(mainDatabase);

// Network interface

grid.add(new JLabel("Using network interface:"));

grid.add(interfacesComboBox);

JTextField serverIp = new JTextField("127.0.0.1:8080");

JTextField receiverIp = new JTextField("127.0.0.1:8081");

JTextField keyFilePath = new JTextField();

keyFilePath.setToolTipText("If empty then will be created a new one");

// Server and other client IPs (only for clients)

if (instances[0].equals(Instances.CLIENT)) {

grid.add(new JLabel("Server ip:"));

grid.add(serverIp);

grid.add(new JLabel("Receiver ip:"));

grid.add(receiverIp);

grid.add(new JLabel("Secret key file path:"));

grid.add(keyFilePath);

}

JButton startButton = new JButton("Start app");

grid.add(startButton);

// =====================================================

JScrollPane clientsIps = null;

JTextField currentClientIp = new JTextField();

JTextArea clientsIpsTextArea = new JTextArea();

currentClientIp.setToolTipText("Write here IP of client...");

currentClientIp.setColumns(15);

if (instances[0].equals(Instances.SERVER)) {

JPanel clientsPanel = new JPanel();

GridBagLayout clientsLayout = new GridBagLayout();

clientsPanel.setLayout(clientsLayout);

clientsIpsTextArea.setRows(10);

clientsIpsTextArea.setColumns(10);

clientsIps = new JScrollPane(clientsIpsTextArea,

JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS, JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER);

clientsIpsTextArea.setEditable(false);

clientsPanel.add(new JLabel("Clients IPs for connection "));

clientsPanel.add(clientsIps);

clientsPanel.add(currentClientIp);

JButton addClientButton = new JButton("Add current ip as client");

Set<String> uniqueIps = new HashSet<>();

ActionListener listener = ae -> {

String currentIp = currentClientIp.getText();

if (uniqueIps.contains(currentIp)) {

return;

}

try {

validateAddress(currentIp);

} catch (ValidationException e) {

throw new RuntimeException(e);

}

uniqueIps.add(currentIp);

String currentText = clientsIpsTextArea.getText();

clientsIpsTextArea.setText(String.format("%s\n%s", currentText, currentClientIp.getText()));

};

addClientButton.addActionListener(listener);

clientsPanel.add(addClientButton);

JTabbedPane pane = new JTabbedPane();

pane.addTab("Main settings", grid);

pane.addTab("Clients", clientsPanel);

settingsWindow.getContentPane().add(pane);

} else {

settingsWindow.getContentPane().add(grid);

}

settingsWindow.pack();

ActionListener listener = ae -> {

// Environment setup...

String applicationPort = appPort.getText();

String databaseInstanceName = dbInstanceName.getText();

String headerKey = keyPlaceholder.getText();

String postgresUser = pgUser.getText();

String postgresPassword = pgPassword.getText();

String postgresPort = dbPort.getText();

String postgresDb = mainDatabase.getText();

String network = Objects.requireNonNull(interfacesComboBox.getSelectedItem()).toString();

String serverIpAddress;

String receiverIpAddress;

if (instances[0].equals(Instances.CLIENT)) {

serverIpAddress = serverIp.getText();

receiverIpAddress = receiverIp.getText();

try {

validateAddress(serverIpAddress);

validateAddress(receiverIpAddress);

} catch (ValidationException e) {

log.error("Error during addresses validation", e);

throw new RuntimeException(e);

}

SystemContext.setProperty(SystemProperties.SERVER\_IP, serverIpAddress);

SystemContext.setProperty(SystemProperties.RECEIVER\_IP, receiverIpAddress);

if (!keyFilePath.getText().isEmpty()) {

byte[] key;

try {

key = Files.readAllBytes(Paths.get(keyFilePath.getText()));

KeyHolder.setKey(key);

KeyHolder.setIv(AESUtil.generateIv());

log.debug("Secret key picked up from file");

} catch (IOException e) {

throw new RuntimeException(e);

}

}

else {

try {

KeyHolder.setKey(AESUtil.generateKey(DefaultParams.SECRET\_KEY\_LENGTH));

KeyHolder.setIv(AESUtil.generateIv());

writeToFile(KEY\_FILE\_NAME, KeyHolder.getKey().getEncoded());

log.info("New secret key generated and saved to file {}", KEY\_FILE\_NAME);

} catch (NoSuchAlgorithmException | BadAlgorythmLengthException | IOException e) {

log.error("Error during secret key generation", e);

throw new RuntimeException(e);

}

}

} else {

try {

String clients = clientsIpsTextArea.getText().substring(1);

ClientsHolder.addAllClients(clients.split("\n"));

} catch (Exception e) {

String errorMessage =

"Something wrong during parsing clients ips. " +

"May be you forgot to add at least one client IP?";

showErrorDialog(errorMessage);

throw e;

}

}

try {

validatePort(applicationPort);

validateParameter(SystemProperties.DB\_INSTANCE, databaseInstanceName);

validateParameter(SystemProperties.DB\_USER, postgresUser);

validateParameter(SystemProperties.DB\_PASSWORD, postgresPassword);

validatePort(postgresPort);

validateParameter(SystemProperties.MAIN\_DB, postgresDb);

} catch (ValidationException e) {

throw new RuntimeException(e);

}

SystemContext.setProperty(SystemProperties.APP\_PORT, applicationPort);

SystemContext.setProperty(SystemProperties.DB\_INSTANCE, databaseInstanceName);

SystemContext.setProperty(SystemProperties.HEADER\_KEY, headerKey);

SystemContext.setProperty(SystemProperties.DB\_USER, postgresUser);

SystemContext.setProperty(SystemProperties.DB\_PASSWORD, postgresPassword);

SystemContext.setProperty(SystemProperties.DB\_PORT, postgresPort);

SystemContext.setProperty(SystemProperties.MAIN\_DB, postgresDb);

SystemContext.setProperty(SystemProperties.NETWORK, network.split("=")[0].split(" ")[1]);

Matcher matcher = Pattern.compile("[0-9]+.[0-9]+.[0-9]+.[0-9]+").matcher(network);

boolean isNetworkProvided = matcher.find();

if (isNetworkProvided) {

String currentIp = matcher.group();

try {

validateIp(SystemProperties.CURRENT\_IP, currentIp);

} catch (ValidationException e) {

throw new RuntimeException(e);

}

SystemContext.setProperty(SystemProperties.CURRENT\_IP, currentIp);

}

SystemContext.printAllProperties();

settingsWindow.setVisible(false);

};

startButton.addActionListener(listener);

settingsWindow.setVisible(true);

}

private static void validateParameter(String parameterName, String parameter) throws ValidationException {

if (parameter.isBlank() || parameter.isEmpty()) {

String errorMessage = String.format("Parameter %s not specified", parameterName);

showErrorDialog(errorMessage);

throw new ValidationException(errorMessage);

}

if (parameter.length() > DefaultParams.MAX\_PARAMETER\_LENGTH) {

String errorMessage = String.format("Parameter is too long. Must be not bigger than %s",

DefaultParams.MAX\_PARAMETER\_LENGTH);

showErrorDialog(errorMessage);

throw new ValidationException(errorMessage);

}

}

private static void validateIp(String parameterName, String ip) throws ValidationException {

validateParameter(parameterName, ip);

String[] parts = ip.split("\\.");

if (parts.length == 0) {

String errorMessage =

String.format("Invalid ip structure. Must be [0-255].[0-255].[0-255].[0-255]. Got: %s", ip);

showErrorDialog(errorMessage);

throw new ValidationException(errorMessage);

}

try {

for (String part : parts) {

int parsed = Integer.parseInt(part);

if (parsed < 0 || parsed > 255) {

String errorMessage =

String.format("Invalid part of ip address. Got ip: %s; Invalid part: %s", ip, part);

showErrorDialog(errorMessage);

throw new ValidationException(errorMessage);

}

}

} catch (NumberFormatException e) {

String errorMessage =

String.format("Invalid part of ip address. Ip must include only digits and dots. Got ip: %s", ip);

showErrorDialog(errorMessage);

throw new ValidationException(errorMessage);

}

}

private static void validatePort(String port) throws ValidationException {

validateParameter(SystemProperties.DB\_PORT, port);

try {

int parsed = Integer.parseInt(port);

if (parsed < DefaultParams.MIN\_PORT || parsed > DefaultParams.MAX\_PORT) {

String errorMessage = String.format("Invalid port. Must be in range [%s-%s]",

DefaultParams.MIN\_PORT, DefaultParams.MAX\_PORT);

showErrorDialog(errorMessage);

throw new ValidationException(errorMessage);

}

} catch (NumberFormatException e) {

String errorMessage = String.format("Invalid port. Port must include only digits. Got: %s", port);

showErrorDialog(errorMessage);

throw new ValidationException(errorMessage);

}

}

private static void showErrorDialog(String errorMessage) {

log.error(errorMessage);

JFrame jFrame = new JFrame();

JOptionPane.showMessageDialog(jFrame, errorMessage, "Error", JOptionPane.ERROR\_MESSAGE);

}

private static void validateAddress(String address) throws ValidationException {

String[] ipAndPort = address.split(":");

if (ipAndPort.length != 2) {

String errorMessage = "Invalid address: " + address;

showErrorDialog(errorMessage);

throw new ValidationException(errorMessage);

}

try {

validateIp("Client IP", ipAndPort[0]);

} catch (ValidationException e) {

String errorMessage = "Invalid ip address: " + ipAndPort[0];

showErrorDialog(errorMessage);

throw new ValidationException(errorMessage);

}

try {

validatePort(ipAndPort[1]);

} catch (ValidationException e) {

String errorMessage = "Invalid port for address: " + ipAndPort[1];

showErrorDialog(errorMessage);

throw new ValidationException(errorMessage);

}

}

private static abstract class DefaultParams {

// Window settings

public static final Dimension WINDOW\_SIZE = new Dimension(320, 320);

public static final Point LOCATION\_POINT = new Point(100, 100);

// Grid settings

public static final Integer ROWS = 13;

public static final Integer COLUMNS = 2;

public static final Integer H\_GAP = 8;

public static final Integer V\_GAP = 5;

// Validation constants

public static final int MAX\_PARAMETER\_LENGTH = 50;

public static final int MIN\_PORT = 7000;

public static final int MAX\_PORT = 9000;

// Default params

public static final String DEFAULT\_APP\_PORT = "8080";

public static final int SECRET\_KEY\_LENGTH = 256;

}

private static abstract class Instances {

public static final String CLIENT = "Client";

public static final String SERVER = "Server";

}

private static void writeToFile(String filename, byte[] array) throws IOException {

FileUtils.writeByteArrayToFile(new File(filename), array);

}

}

**Пакет** org.ssau.privatechannel.utils

@Slf4j

public class AESUtil {

private static final List<Integer> VALID\_KEY\_LENGTHS = Arrays.asList(128, 192, 256);

public static final String ALGORITHM = "AES/CBC/PKCS5Padding";

public static SecretKey generateKey(int n) throws NoSuchAlgorithmException, BadAlgorythmLengthException {

if (!VALID\_KEY\_LENGTHS.contains(n)) {

String logMessage = String.format("Incorrect secret key length: %s. Must be one of %s",

n, VALID\_KEY\_LENGTHS);

log.error(logMessage);

throw new BadAlgorythmLengthException(logMessage);

}

KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");

keyGenerator.init(n);

return keyGenerator.generateKey();

}

public static IvParameterSpec generateIv() {

byte[] iv = new byte[16];

new SecureRandom().nextBytes(iv);

return new IvParameterSpec(iv);

}

public static String encrypt(String input, SecretKey key, IvParameterSpec iv)

throws NoSuchPaddingException,

NoSuchAlgorithmException,

InvalidAlgorithmParameterException,

InvalidKeyException,

BadPaddingException,

IllegalBlockSizeException {

Cipher cipher = Cipher.getInstance(ALGORITHM);

cipher.init(Cipher.ENCRYPT\_MODE, key, iv);

byte[] cipherText = cipher.doFinal(input.getBytes());

return Base64.getEncoder()

.encodeToString(cipherText);

}

public static String decrypt(String cipherText, SecretKey key, IvParameterSpec iv)

throws NoSuchPaddingException,

NoSuchAlgorithmException,

InvalidAlgorithmParameterException,

InvalidKeyException,

BadPaddingException,

IllegalBlockSizeException {

Cipher cipher = Cipher.getInstance(ALGORITHM);

cipher.init(Cipher.DECRYPT\_MODE, key, iv);

byte[] plainText = cipher.doFinal(Base64.getDecoder()

.decode(cipherText));

return new String(plainText);

}

public static SealedObject encryptObject(Serializable object,

SecretKey key,

IvParameterSpec iv) throws NoSuchPaddingException,

NoSuchAlgorithmException, InvalidAlgorithmParameterException,

InvalidKeyException, IOException, IllegalBlockSizeException {

Cipher cipher = Cipher.getInstance(ALGORITHM);

cipher.init(Cipher.ENCRYPT\_MODE, key, iv);

return new SealedObject(object, cipher);

}

public static Serializable decryptObject(SealedObject sealedObject,

SecretKey key, IvParameterSpec iv) throws NoSuchPaddingException,

NoSuchAlgorithmException, InvalidAlgorithmParameterException, InvalidKeyException,

ClassNotFoundException, BadPaddingException, IllegalBlockSizeException,

IOException {

Cipher cipher = Cipher.getInstance(ALGORITHM);

cipher.init(Cipher.DECRYPT\_MODE, key, iv);

return (Serializable) sealedObject.getObject(cipher);

}

public static ConfidentialInfo encryptConfInfo(ConfidentialInfo info, SecretKey key, IvParameterSpec iv)

throws NoSuchPaddingException,

NoSuchAlgorithmException,

InvalidAlgorithmParameterException,

InvalidKeyException,

BadPaddingException,

IllegalBlockSizeException {

return ConfidentialInfo.builder()

.id(info.getId())

.senderIP(info.getSenderIP())

.receiverIP(info.getReceiverIP())

.data(encryptMap(info.getData(), key, iv))

.build();

}

public static List<ConfidentialInfo> encryptBatchInfo(List<ConfidentialInfo> info,

SecretKey key,

IvParameterSpec iv)

throws NoSuchPaddingException,

NoSuchAlgorithmException,

InvalidAlgorithmParameterException,

InvalidKeyException,

BadPaddingException,

IllegalBlockSizeException {

List<ConfidentialInfo> result = new ArrayList<>();

for (ConfidentialInfo record : info) {

result.add(encryptConfInfo(record, key, iv));

}

return result;

}

public static ReceivedInformation decryptConfInfo(ReceivedInformation info,

SecretKey key,

IvParameterSpec iv)

throws NoSuchPaddingException,

NoSuchAlgorithmException,

InvalidAlgorithmParameterException,

InvalidKeyException,

BadPaddingException,

IllegalBlockSizeException {

return ReceivedInformation.builder()

.id(info.getId())

.senderIP(info.getSenderIP())

.receiverIP(info.getReceiverIP())

.data(decryptMap(info.getData(), key, iv))

.build();

}

public static List<ReceivedInformation> decryptBatchInfo(List<ReceivedInformation> info,

SecretKey key,

IvParameterSpec iv)

throws NoSuchPaddingException,

NoSuchAlgorithmException,

InvalidAlgorithmParameterException,

InvalidKeyException,

BadPaddingException,

IllegalBlockSizeException {

List<ReceivedInformation> result = new ArrayList<>();

for (ReceivedInformation record : info) {

result.add(decryptConfInfo(record, key, iv));

}

return result;

}

public static Map<String, Object> encryptMap(Map<String, Object> map,

SecretKey key,

IvParameterSpec iv)

throws NoSuchPaddingException,

NoSuchAlgorithmException,

InvalidAlgorithmParameterException,

InvalidKeyException,

BadPaddingException,

IllegalBlockSizeException {

Map<String, Object> result = new HashMap<>();

for (Map.Entry<String, Object> record : map.entrySet()) {

result.put(

encrypt(record.getKey(), key, iv),

encrypt(record.getValue().toString(), key, iv)

);

}

return result;

}

public static Map<String, Object> decryptMap(Map<String, Object> map,

SecretKey key,

IvParameterSpec iv)

throws NoSuchPaddingException,

NoSuchAlgorithmException,

InvalidAlgorithmParameterException,

InvalidKeyException,

BadPaddingException,

IllegalBlockSizeException {

Map<String, Object> result = new HashMap<>();

for (Map.Entry<String, Object> record : map.entrySet()) {

result.put(

decrypt(record.getKey(), key, iv),

decrypt(record.getValue().toString(), key, iv)

);

}

return result;

}

}

@Slf4j

public class ApplicationInstaller {

public static void run(String instance) throws IOException, InterruptedException,

DockerMissingException, InvalidInstanceTypeException {

DockerInstaller.run();

SystemContext.setProperty(SystemProperties.DB\_URL, DefaultDbParams.DEFAULT\_DB\_URL);

SystemContext.setProperty(SystemProperties.DB\_USER, DefaultDbParams.DEFAULT\_DB\_USERNAME);

SystemContext.setProperty(SystemProperties.DB\_PASSWORD, DefaultDbParams.DEFAULT\_DB\_PASSWORD);

log.info("Single database will be installed");

if (Instances.SERVER.equals(instance) || Instances.CLIENT.equals(instance)) {

DbClusterInstaller.singleInstall(instance);

} else {

log.error("Invalid instance type provided during database deployment: {}. Must be {} or {}",

instance, Instances.SERVER, Instances.CLIENT);

throw new InvalidInstanceTypeException("Invalid instance type provided during application installation");

}

}

private static abstract class DefaultDbParams {

public static final String DEFAULT\_DB\_URL = "jdbc:postgresql://localhost:%s/private\_channel";

public static final String DEFAULT\_DB\_USERNAME = "postgres";

public static final String DEFAULT\_DB\_PASSWORD = "postgres";

}

}

public class ClientsHolder {

private static final List<String> CLIENTS\_IPS = new ArrayList<>();

public static void addAllClients(String[] clients) {

List<String> newClients = Arrays.asList(clients);

CLIENTS\_IPS.addAll(newClients);

}

public static List<String> getAllClients() {

return CLIENTS\_IPS;

}

}

public class CommandRunner {

public static void run(String command) throws IOException {

runWithReturn(command);

}

public static List<String> runQuietWithReturn(String command) throws IOException {

ProcessBuilder builder = new ProcessBuilder(

"cmd.exe", "/c", command);

builder.redirectErrorStream(true);

Process process = builder.start();

InputStreamReader inputStreamReader = new InputStreamReader(process.getInputStream());

BufferedReader r = new BufferedReader(inputStreamReader);

List<String> fullConsoleOutput = new ArrayList<>();

String line;

while (true) {

line = r.readLine();

if (Objects.isNull(line)) {

break;

}

fullConsoleOutput.add(line);

}

return fullConsoleOutput;

}

public static List<String> runWithReturn(String command) throws IOException {

ProcessBuilder builder = new ProcessBuilder(

"cmd.exe", "/c", command);

builder.redirectErrorStream(true);

Process process = builder.start();

InputStreamReader inputStreamReader = new InputStreamReader(process.getInputStream());

BufferedReader r = new BufferedReader(inputStreamReader);

List<String> fullConsoleOutput = new ArrayList<>();

String line;

while (true) {

line = r.readLine();

if (Objects.isNull(line)) {

break;

}

fullConsoleOutput.add(line);

System.out.println(line);

}

return fullConsoleOutput;

}

public static void runQuiet(String command) throws IOException {

ProcessBuilder builder = new ProcessBuilder(

"cmd.exe", "/c", command);

builder.redirectErrorStream(true);

Process process = builder.start();

InputStreamReader inputStreamReader = new InputStreamReader(process.getInputStream());

BufferedReader r = new BufferedReader(inputStreamReader);

String line;

while (true) {

line = r.readLine();

if (line == null) {

break;

}

}

}

}

@Slf4j

public class DbClusterInstaller {

private static final String DEFAULT\_CONTAINER\_PREFIX = "pc";

public static void singleInstall(String instance) throws IOException, InvalidInstanceTypeException, InterruptedException {

if (!instance.equals(Instances.SERVER) && !instance.equals(Instances.CLIENT)) {

log.error("Invalid instance type provided during database deployment: {}. Must be {} or {}",

instance, Instances.SERVER, Instances.CLIENT);

throw new InvalidInstanceTypeException("DB Installation must be only for server or client instance");

}

String dbUrl = SystemContext.getProperty(SystemProperties.DB\_URL);

String dbPort = SystemContext.getProperty(SystemProperties.DB\_PORT);

String containerName = String.format("%s\_%s", DEFAULT\_CONTAINER\_PREFIX, dbPort);

SystemContext.setProperty(SystemProperties.DB\_URL, String.format(dbUrl, dbPort));

List<String> consoleOutput =

CommandRunner.runWithReturn(String.format(Commands.GET\_CONTAINER\_INFO, containerName));

if (consoleOutput.size() > 1) {

log.info("Database container \"{}\" already exist", containerName);

String containerInfo = consoleOutput.get(1);

runContainerIfStopped(containerInfo);

} else {

log.info("Database container \"{}\" not exist. Starting database installation...", containerName);

String port = SystemContext.getProperty(SystemProperties.DB\_PORT);

PostgresInstaller.run(containerName, port);

}

}

private static void runContainerIfStopped(String containerInfo) throws IOException {

if (!containerInfo.contains(ContainerStatuses.EXITED)) {

log.info("Container already running");

return;

}

log.info("Container exist but not running. Starting container...");

String containerId = containerInfo.split(" ")[0];

CommandRunner.run(String.format(Commands.START\_CONTAINER\_BY\_ID, containerId));

}

private static class Commands {

public static final String GET\_CONTAINER\_INFO = "docker ps -f name=%s -a";

public static final String START\_CONTAINER\_BY\_ID = "docker start %s";

}

public static class Instances {

public static final String SERVER = "server\_db";

public static final String CLIENT = "client\_db";

}

private static class ContainerStatuses {

public static final String EXITED = "Exited";

}

}

@Slf4j

public class DockerInstaller {

private static final String DOCKER\_DESKTOP\_DOWNLOAD\_LINK =

"https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe";

private static final String INSTALLER\_PATH = "installation/docker/";

private static final String INSTALLER\_FILE\_NAME = "Docker Desktop Installer.exe";

private static final String DOCKER\_TASK\_NAME = "Docker Desktop.exe";

private static final Integer MAX\_WAIT\_TIME\_SECONDS = 600;

private static final Integer DEFAULT\_DELAY\_IN\_SECONDS = 15;

public static void run() throws IOException, InterruptedException, DockerMissingException {

log.info("Docker starting with timeout {} ...", MAX\_WAIT\_TIME\_SECONDS);

boolean dockerStarted = startDocker();

// Docker not installed

if (!dockerStarted) {

log.info("Docker Desktop not installed on current PC. Installation of docker desktop will be offer");

boolean isDockerCanBeInstalled = showOfferDockerInstallationPopUp();

if (isDockerCanBeInstalled) {

log.info("Starting docker desktop installation...");

CommandRunner.runQuiet(Commands.DOWNLOAD\_DOCKER\_DESKTOP); // Work

CommandRunner.run(Commands.DOCKER\_QUIET\_INSTALL); // Work

CommandRunner.run(Commands.DELETE\_DOCKER\_INSTALLER); // Work

startDocker();

} else {

log.error("Docker will not be installed on current PC: user cancelled installation. Exiting...");

throw new DockerMissingException("Docker-Desktop is required for private-channel application");

}

}

log.info("Docker successfully started");

}

private static boolean startDocker() throws InterruptedException, IOException {

final boolean[] isDockerInstalled = {true};

Thread thread = new Thread() {

@SneakyThrows

@Override

public void run() {

String result = CommandRunner.runWithReturn(Commands.RUN\_DOCKER).get(0);

if (!result.equals("Successful"))

isDockerInstalled[0] = false;

}

};

thread.start();

int timeSpent = 0;

while (true) {

Thread.sleep(TimeUnit.SECONDS.toMillis(DEFAULT\_DELAY\_IN\_SECONDS));

timeSpent += DEFAULT\_DELAY\_IN\_SECONDS;

if (!isDockerInstalled[0]) {

return false;

}

log.info("Wait for docker started...");

List<String> consoleOutput =

CommandRunner.runQuietWithReturn(Commands.GET\_PROCESSES\_LIST);

for (String task : consoleOutput) {

if (task.contains(DOCKER\_TASK\_NAME)) {

thread.interrupt();

return true;

}

}

if (timeSpent >= MAX\_WAIT\_TIME\_SECONDS) {

String errorMessage = "Timeout reached while trying to start docker";

log.error(errorMessage);

throw new InterruptedException(errorMessage);

}

}

}

private static boolean showOfferDockerInstallationPopUp() {

JFrame jFrame = new JFrame();

int result = JOptionPane.showConfirmDialog(jFrame,

"Docker not installed on your computer. Would you like to install docker? (Docker is required)");

return result == 0;

}

private static abstract class Commands {

public static final String RUN\_DOCKER = "\"C:\\Program Files\\Docker\\Docker\\Docker Desktop.exe\" & echo Successful";

public static final String DOWNLOAD\_DOCKER\_DESKTOP =

String.format("cd %s & wget %s", INSTALLER\_PATH, DOCKER\_DESKTOP\_DOWNLOAD\_LINK);

public static final String DOCKER\_QUIET\_INSTALL =

String.format("cd %s & \"%s\" install --quiet", INSTALLER\_PATH, INSTALLER\_FILE\_NAME);

public static final String DELETE\_DOCKER\_INSTALLER =

String.format("cd %s & del \"%s\"", INSTALLER\_PATH, INSTALLER\_FILE\_NAME);

public static final String GET\_PROCESSES\_LIST = "tasklist.exe";

}

}

public class KeyHolder {

private static SecretKey key;

private static IvParameterSpec iv;

public static SecretKey getKey() {

return key;

}

public static void setKey(SecretKey key) {

KeyHolder.key = key;

}

public static void setKey(byte[] key) {

KeyHolder.key = new SecretKeySpec(key, AESUtil.ALGORITHM);

}

public static IvParameterSpec getIv() {

return iv;

}

public static void setIv(IvParameterSpec iv) {

KeyHolder.iv = iv;

}

}

@Slf4j

public class PostgresInstaller {

private static final String DB\_INSTALLATION\_JAR\_PATH = "installation/db\_installation-1.0.jar";

private static final String SETTINGS\_FILE\_PATH = "installation/default\_settings.json";

private static final String DEFAULT\_USERNAME = "postgres";

private static final String DEFAULT\_PASSWORD = "postgres";

private static final String DEFAULT\_DB\_NAME = "private\_channel";

private static final Integer MAX\_WAIT\_TIME\_SECONDS = 600;

private static final Integer DEFAULT\_DELAY\_IN\_SECONDS = 10;

public static void run(String containerName, String port) throws IOException, InterruptedException {

log.info("Start container [name = {}] creation with timeout = {}", containerName, MAX\_WAIT\_TIME\_SECONDS);

Thread thread = new Thread() {

@SneakyThrows

@Override

public void run() {

createSettingsFile(containerName, port);

String command = String.format(Commands.START\_CONTAINER\_INSTALLATION,

DB\_INSTALLATION\_JAR\_PATH,

SETTINGS\_FILE\_PATH);

CommandRunner.run(command);

}

};

thread.start();

int timeSpent = 0;

while (true) {

Thread.sleep(TimeUnit.SECONDS.toMillis(DEFAULT\_DELAY\_IN\_SECONDS));

timeSpent += DEFAULT\_DELAY\_IN\_SECONDS;

log.info("Wait for container {} started...", containerName);

List<String> consoleOutput =

CommandRunner.runQuietWithReturn(String.format(Commands.GET\_CONTAINER\_INFO, containerName));

if (consoleOutput.size() > 1) {

log.info("Database container \"{}\" started", containerName);

thread.interrupt();

break;

}

if (timeSpent >= MAX\_WAIT\_TIME\_SECONDS) {

String errorMessage = "Timeout reached during container creation";

log.error(errorMessage);

throw new InterruptedException(errorMessage);

}

}

log.info("Container \"{}\" created and started", containerName);

}

private static void createSettingsFile(String dbName, String port) throws IOException {

ContainerSettings settings = getDefaultSettings(dbName, port);

String settingsJson = new ObjectMapper().writeValueAsString(settings);

FileWriter writer = new FileWriter(SETTINGS\_FILE\_PATH);

writer.write(settingsJson);

writer.close();

}

private static ContainerSettings getDefaultSettings(String instanceName, String port) {

ContainerSettings settings = new ContainerSettings();

settings.setInstanceName(instanceName);

settings.setUser(DEFAULT\_USERNAME);

settings.setPassword(DEFAULT\_PASSWORD);

settings.setDb(DEFAULT\_DB\_NAME);

settings.setPort(port);

return settings;

}

private static abstract class Commands {

public static final String START\_CONTAINER\_INSTALLATION

= "java -jar %s -settingsFile=%s";

public static final String GET\_CONTAINER\_INFO = "docker ps -f name=%s";

}

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

private static class ContainerSettings {

private String instanceName;

private String user;

private String password;

private String db;

private String port;

}

}

@Slf4j

public class Sha512EncoderService {

public static String getHash(String input) {

try {

MessageDigest md = MessageDigest.getInstance("SHA-512");

byte[] messageDigest = md.digest(input.getBytes());

BigInteger no = new BigInteger(1, messageDigest);

StringBuilder hashtext = new StringBuilder(no.toString(16));

while (hashtext.length() < 32) {

hashtext.insert(0, "0");

}

return hashtext.toString();

} catch (NoSuchAlgorithmException e) {

log.error("Data encryption failed", e);

throw new RuntimeException(e);

}

}

}

@Slf4j

public class SystemContext {

public static final Map<String, String> context = new HashMap<>();

public static void setProperty(String propertyName, String value) {

context.put(propertyName, value);

}

public static String getProperty(String propertyName) {

return context.get(propertyName);

}

public static void printAllProperties() {

for (Map.Entry<String, String> property : context.entrySet()) {

String prop = String.format("Property [%s] = [%s]", property.getKey(), property.getValue());

log.info(prop);

}

}

}

public abstract class ThreadsHolder {

private static final Map<String, Thread> RUNNING\_THREADS = new HashMap<>();

public static synchronized void addAndRunThread(String id, Thread thread) {

thread.start();

RUNNING\_THREADS.put(id, thread);

}

public static synchronized void removeAndStopById(String id) {

if (RUNNING\_THREADS.containsKey(id)) {

Thread thread = RUNNING\_THREADS.get(id);

thread.interrupt();

}

}

}

**Тесты:**

class AESKeyGeneratorTest {

private static final String TEST\_IP = "10.20.30.40";

private final Random RANDOMIZER = new Random();

@BeforeEach

void setUp() {

}

@Test

void givenString\_whenEncrypt\_thenSuccess()

throws NoSuchAlgorithmException,

IllegalBlockSizeException,

InvalidKeyException,

BadPaddingException,

InvalidAlgorithmParameterException,

NoSuchPaddingException,

BadAlgorythmLengthException {

String input = UUID.randomUUID().toString();

SecretKey key = AESUtil.generateKey(128);

IvParameterSpec ivParameterSpec = AESUtil.generateIv();

String cipherText = AESUtil.encrypt(input, key, ivParameterSpec);

String plainText = AESUtil.decrypt(cipherText, key, ivParameterSpec);

Assertions.assertEquals(input, plainText);

}

@Test

void givenObject\_whenEncrypt\_thenSuccess()

throws NoSuchAlgorithmException,

IllegalBlockSizeException,

InvalidKeyException,

InvalidAlgorithmParameterException,

NoSuchPaddingException,

IOException,

BadPaddingException,

ClassNotFoundException,

BadAlgorythmLengthException {

ConfidentialInfo info = ConfidentialInfo.builder()

.id(RANDOMIZER.nextLong())

.receiverIP(TEST\_IP)

.data(generateRandomData())

.build();

SecretKey key = AESUtil.generateKey(128);

IvParameterSpec ivParameterSpec = AESUtil.generateIv();

SealedObject sealedObject = AESUtil.encryptObject(info, key, ivParameterSpec);

ConfidentialInfo object = (ConfidentialInfo) AESUtil.decryptObject(sealedObject, key, ivParameterSpec);

Assertions.assertSame(info, object);

}

private Map<String, Object> generateRandomData() {

int MAX\_DATA\_ROWS\_COUNT = 10;

int rowsCount = 1 + (Math.abs(RANDOMIZER.nextInt()) % (MAX\_DATA\_ROWS\_COUNT));

Map<String, Object> result = new HashMap<>();

for (int i = 0; i < rowsCount; ++i) {

result.put(UUID.randomUUID().toString(), UUID.randomUUID() + UUID.randomUUID().toString());

}

return result;

}

}

**Модуль** client

**Пакет** org.ssau.privatechannel.config

@Configuration

@ComponentScan(basePackages = "java")

@EnableTransactionManagement

public class DataSourceConfig {

private static final String POSTGRES\_DIALECT = "org.hibernate.dialect.PostgreSQL94Dialect";

@Bean

public DataSource getDataSource() {

String url = SystemContext.getProperty(SystemProperties.DB\_URL);

String username = SystemContext.getProperty(SystemProperties.DB\_USER);

String password = SystemContext.getProperty(SystemProperties.DB\_PASSWORD);

DataSourceBuilder<?> dataSourceBuilder = DataSourceBuilder.create();

dataSourceBuilder.url(url);

dataSourceBuilder.username(username);

dataSourceBuilder.password(password);

return dataSourceBuilder.build();

}

@Bean

public Properties hibernateProperties() {

Properties hibernateProp = new Properties();

hibernateProp.put(HibernateProps.DIALECT, POSTGRES\_DIALECT);

hibernateProp.put(HibernateProps.SHOW\_SQL, Boolean.FALSE.toString());

hibernateProp.put(HibernateProps.AUTO\_DDL, "update");

hibernateProp.put(HibernateProps.SQL\_FORMAT, Boolean.TRUE.toString());

return hibernateProp;

}

@Bean

public PlatformTransactionManager transactionManager() {

return new JpaTransactionManager(entityManagerFactory());

}

@Bean

public JpaVendorAdapter jpaVendorAdapter() {

return new HibernateJpaVendorAdapter();

}

@Bean

public EntityManagerFactory entityManagerFactory() {

LocalContainerEntityManagerFactoryBean factoryBean = new LocalContainerEntityManagerFactoryBean();

factoryBean.setPackagesToScan("org/ssau/privatechannel/model");

factoryBean.setDataSource(getDataSource());

factoryBean.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

factoryBean.setJpaProperties(hibernateProperties());

factoryBean.setJpaVendorAdapter(jpaVendorAdapter());

factoryBean.afterPropertiesSet();

return factoryBean.getNativeEntityManagerFactory();

}

private static abstract class HibernateProps {

public static final String DIALECT = "hibernate.dialect";

public static final String SHOW\_SQL = "hibernate.show\_sql";

public static final String AUTO\_DDL = "hibernate.hbm2ddl.auto";

public static final String SQL\_FORMAT = "hibernate.format\_sql";

}

}

@Configuration

public class RestTemplateConfig {

@Bean

public RestTemplate getRestTemplate() {

return new RestTemplate();

}

}

@Configuration

public class ServletConfig implements WebServerFactoryCustomizer<ConfigurableWebServerFactory> {

@Override

public void customize(ConfigurableWebServerFactory factory) {

String appPort = SystemContext.getProperty(SystemProperties.APP\_PORT);

factory.setPort(Integer.parseInt(appPort));

}

}

**Пакет** org.ssau.privatechannel.controller

@Slf4j

@RestController

@RequestMapping(value = Endpoints.API\_V1\_CLIENT)

public class ConfidentialDataController {

private final ReceivedInfoService receivedInfoService;

private final AuthKeyService authKeyService;

@Autowired

public ConfidentialDataController(ReceivedInfoService receivedInfoService,

AuthKeyService authKeyService) {

this.receivedInfoService = receivedInfoService;

this.authKeyService = authKeyService;

}

@PostMapping(value = Endpoints.UPLOAD\_DATA, consumes = MediaType.APPLICATION\_JSON\_VALUE)

public ResponseEntity<?> uploadData(@RequestHeader(SystemProperties.HEADER\_KEY) String headerKey,

@RequestBody List<ReceivedInformation> confidentialInfo) {

try {

checkHeaderKey(headerKey);

}

catch (HeaderKeyNotActualException e) {

return new ResponseEntity<>(HttpStatus.EXPECTATION\_FAILED);

}

try {

confidentialInfo = AESUtil.decryptBatchInfo(confidentialInfo,

KeyHolder.getKey(),

KeyHolder.getIv());

} catch (Exception e) {

log.error("Error during data decryption", e);

return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);

}

log.info("Received data [from IP={}]: {}",

confidentialInfo.get(0).getSenderIP(), confidentialInfo);

receivedInfoService.addAll(confidentialInfo);

return new ResponseEntity<>(HttpStatus.OK);

}

private void checkHeaderKey(String key) throws HeaderKeyNotActualException {

if (!authKeyService.isActual(key)) {

String errorMessage = String.format("Key %s not actual and not persisted in database", key);

log.error(errorMessage);

throw new HeaderKeyNotActualException(errorMessage);

}

}

}

@Slf4j

@RestController

@RequestMapping(value = Endpoints.API\_V1\_CLIENT)

public class GenerateDataController {

private final ConfidentialInfoService infoService;

private final RandomDataGenerator dataGenerator;

@Autowired

public GenerateDataController(ConfidentialInfoService infoService,

RandomDataGenerator dataGenerator) {

this.infoService = infoService;

this.dataGenerator = dataGenerator;

}

@PostMapping(value = Endpoints.GENERATE\_DATA)

public void generate(@PathVariable("count") Integer count) {

log.debug("Generate test data...");

List<ConfidentialInfo> data = dataGenerator.generate(count);

infoService.addAll(data);

}

}

@Slf4j

@RestController

@RequestMapping(path = Endpoints.API\_V1\_CLIENT)

public class ScheduleController {

private final ScheduleService scheduleService;

private final TasksService tasksService;

private final AuthKeyService authKeyService;

@Autowired

public ScheduleController(ScheduleService scheduleService,

TasksService tasksService,

AuthKeyService authKeyService) {

this.scheduleService = scheduleService;

this.tasksService = tasksService;

this.authKeyService = authKeyService;

}

@PostMapping(value = Endpoints.SCHEDULE, consumes = MediaType.APPLICATION\_JSON\_VALUE)

public ResponseEntity<?> saveSchedule(@RequestBody Schedule schedule) {

authKeyService.generateNewHeaderKey();

scheduleService.add(schedule);

tasksService.plan(schedule);

return new ResponseEntity<>(HttpStatus.ACCEPTED);

}

}

**Пакет** org.ssau.privatechannel.firetasks

@Slf4j

public class AskNewScheduleTask extends TimerTask {

private final RestTemplate restTemplate;

private final IpService ipService;

private final NetworkAdapterService networkAdapterService;

private final ScheduleService scheduleService;

public AskNewScheduleTask(RestTemplate restTemplate,

IpService ipService,

NetworkAdapterService networkAdapterService,

ScheduleService scheduleService) {

this.restTemplate = restTemplate;

this.ipService = ipService;

this.networkAdapterService = networkAdapterService;

this.scheduleService = scheduleService;

}

@Override

public void run() {

String serverIp = SystemContext.getProperty(SystemProperties.SERVER\_IP);

String urlRequestSchedule = UrlSchemas.HTTP +

serverIp +

Endpoints.API\_V1\_SERVER +

Endpoints.GET\_NEW\_SCHEDULE;

String currentIp = SystemContext.getProperty(SystemProperties.CURRENT\_IP);

HttpEntity<String> httpEntity = new HttpEntity<>(currentIp);

log.info("Requesting new schedule from server [url = {}]", urlRequestSchedule);

ResponseEntity<Schedule> scheduleResponseEntity =

restTemplate.postForEntity(urlRequestSchedule, httpEntity, Schedule.class);

if (!scheduleResponseEntity.getStatusCode().is2xxSuccessful()) {

log.error("Could not get schedule from server. Server returned {}",

scheduleResponseEntity.getStatusCode());

System.exit(0);

}

String urlProvideScheduleToClient = UrlSchemas.HTTP +

currentIp +

Endpoints.API\_V1\_CLIENT +

Endpoints.SCHEDULE;

if (Objects.isNull(scheduleResponseEntity.getBody())) {

log.info("Schedule does not exist right now. Exiting program...");

System.exit(0);

}

log.info("New schedule got: {}", scheduleResponseEntity.getBody());

HttpEntity<?> scheduleEntity = new HttpEntity<>(scheduleResponseEntity.getBody());

ResponseEntity<String> response =

restTemplate.postForEntity(urlProvideScheduleToClient, scheduleEntity, String.class);

if (!response.getStatusCode().is2xxSuccessful()) {

log.error("Could not send new schedule to client. Client returned {}",

scheduleResponseEntity.getStatusCode());

System.exit(0);

}

EndDataTransferringTask endDataTransferringTask = new EndDataTransferringTask(

ipService, networkAdapterService, scheduleService, scheduleResponseEntity.getBody());

endDataTransferringTask.run();

}

}

public class EndDataTransferringTask extends TimerTask {

private final IpService ipService;

private final NetworkAdapterService networkAdapterService;

private final ScheduleService scheduleService;

private final Schedule schedule;

public EndDataTransferringTask(IpService ipService,

NetworkAdapterService networkAdapterService,

ScheduleService scheduleService,

Schedule schedule) {

this.ipService = ipService;

this.networkAdapterService = networkAdapterService;

this.scheduleService = scheduleService;

this.schedule = schedule;

}

@SneakyThrows

@Override

public void run() {

ipService.deleteRuleByName(FirewallRuleNames.UNBLOCK\_IP);

String serverIp = SystemContext.getProperty(SystemProperties.SERVER\_IP);

String receiverIp = SystemContext.getProperty(SystemProperties.RECEIVER\_IP);

ipService.blockIP(new IpService.IpAddress(serverIp), FirewallRuleNames.BLOCK\_IP);

ipService.blockIP(new IpService.IpAddress(receiverIp), FirewallRuleNames.BLOCK\_IP);

String networkInterface = SystemContext.getProperty(SystemProperties.NETWORK);

networkAdapterService.disableInterfaces(networkInterface);

scheduleService.delete(schedule);

ThreadsHolder.removeAndStopById(StartDataTransferringTask.THREAD\_NAME);

}

}

**Пакет** org.ssau.privatechannel.repository

@Repository

public class ReceivedInfoRepository extends AbstractRepository {

@Transactional

public void addAll(List<ReceivedInformation> info) {

for (ReceivedInformation currentRecord : info) {

entityManager.merge(currentRecord);

}

}

}

**Пакет** org.ssau.privatechannel.service

@Service

public class ReceivedInfoService {

private final ReceivedInfoRepository infoRepository;

@Autowired

public ReceivedInfoService(ReceivedInfoRepository infoRepository) {

this.infoRepository = infoRepository;

}

public void addAll(List<ReceivedInformation> info) {

infoRepository.addAll(info);

}

}

@Slf4j

@Service

public class TasksService {

private final ConfidentialInfoService infoService;

private final ScheduleService scheduleService;

private final IpService ipService;

private final NetworkAdapterService networkAdapterService;

private final RestTemplate restTemplate;

private final TimerService timerService;

@Autowired

public TasksService(ConfidentialInfoService infoService,

IpService ipService,

TimerService timerService,

NetworkAdapterService networkAdapterService,

RestTemplate restTemplate,

ScheduleService scheduleService) {

this.infoService = infoService;

this.ipService = ipService;

this.networkAdapterService = networkAdapterService;

this.restTemplate = restTemplate;

this.timerService = timerService;

this.scheduleService = scheduleService;

}

public void plan(Schedule schedule) {

List<TimeFrame> timeFrames = schedule.getTimeFrames();

for (TimeFrame timeFrame : timeFrames) {

LocalDateTime startTime = timeFrame.getStartTime();

LocalDateTime endTime = timeFrame.getEndTime();

String currentIp = SystemContext.getProperty(SystemProperties.CURRENT\_IP);

String receiverIp = SystemContext.getProperty(SystemProperties.RECEIVER\_IP);

StartDataTransferringTask startTransferringTask =

new StartDataTransferringTask(infoService,

restTemplate,

ipService,

networkAdapterService);

startTransferringTask.setReceiverIp(receiverIp);

timerService.createTask(startTransferringTask, startTime);

log.info("Transferring data from client [IP={}] to client [IP={}] will be started at {}",

currentIp, receiverIp, startTime);

if (isLastTimeframe(schedule, timeFrame)) {

AskNewScheduleTask askNewScheduleTask = new AskNewScheduleTask(

restTemplate,

ipService,

networkAdapterService,

scheduleService);

log.info("New schedule will be requested at {}", timeFrame.getEndTime());

timerService.createTask(askNewScheduleTask, timeFrame.getEndTime());

break;

}

EndDataTransferringTask endTransferringTask =

new EndDataTransferringTask(ipService, networkAdapterService, scheduleService, schedule);

timerService.createTask(endTransferringTask, endTime);

log.info("Transferring data from client [IP={}] to client [IP={}] will be end at {}",

currentIp, receiverIp, endTime);

}

}

private boolean isLastTimeframe(Schedule schedule, TimeFrame timeFrame) {

List<TimeFrame> timeFrames = schedule.getTimeFrames();

return timeFrames.get(timeFrames.size()-1).equals(timeFrame);

}

}

**Пакет** org.ssau.privatechannel.utils

@Component

public class RandomDataGenerator {

private final Random RANDOMIZER = new Random();

public List<ConfidentialInfo> generate(int count) {

List<ConfidentialInfo> info = new ArrayList<>();

for (int i = 0; i < count; ++i) {

ConfidentialInfo currentRecord = ConfidentialInfo.builder()

.id(null)

.data(generateRandomData())

.build();

info.add(currentRecord);

}

return info;

}

public List<ReceivedInformation> generateReceivedData(int count) {

List<ReceivedInformation> info = new ArrayList<>();

for (int i = 0; i < count; ++i) {

ReceivedInformation currentRecord = ReceivedInformation.builder()

.id(null)

.data(generateRandomData())

.build();

info.add(currentRecord);

}

return info;

}

private Map<String, Object> generateRandomData() {

int MAX\_DATA\_ROWS\_COUNT = 10;

int rowsCount = 1 + (Math.abs(RANDOMIZER.nextInt()) % (MAX\_DATA\_ROWS\_COUNT));

Map<String, Object> result = new HashMap<>();

for (int i = 0; i < rowsCount; ++i) {

result.put(UUID.randomUUID().toString(), UUID.randomUUID() + UUID.randomUUID().toString());

}

return result;

}

}

**Пакет** org.ssau.privatechannel

@Slf4j

@SpringBootApplication

public class Client {

private static final String CURRENT\_INSTANCE = "Client";

public static void main(String[] args) {

SystemContext.setProperty(SystemProperties.INSTANCE, CURRENT\_INSTANCE);

try {

StartPage.show();

} catch (IOException e) {

throw new RuntimeException("Could not show start page", e);

}

try {

ApplicationInstaller.run(Instances.CLIENT);

} catch (Exception e) {

throw new RuntimeException("Something wrong during client start: ", e);

}

SpringApplication.run(Client.class, args);

}

}

**Тесты:**

class ConfidentialDataControllerTest {

private final ReceivedInfoService receivedInfoService = Mockito.mock(ReceivedInfoService.class);

private final AuthKeyService authKeyService = Mockito.mock(AuthKeyService.class);

private final RandomDataGenerator dataGenerator = new RandomDataGenerator();

private final ConfidentialDataController dataController

= new ConfidentialDataController(receivedInfoService,

authKeyService);

private static final String TEST\_HEADER\_KEY = "test\_header\_key";

private static final String TEST\_WRONG\_KEY = "test\_wrong\_key";

private static final int TEST\_DATA\_COUNT = 10;

@BeforeEach

void setUp() {

Mockito.when(authKeyService.isActual(Mockito.anyString())).thenReturn(false);

Mockito.when(authKeyService.isActual(TEST\_HEADER\_KEY)).thenReturn(true);

}

@Test

void uploadData() {

List<ReceivedInformation> info = dataGenerator.generateReceivedData(TEST\_DATA\_COUNT);

ResponseEntity<?> responseEntity = dataController.uploadData(TEST\_HEADER\_KEY, info);

Assertions.assertEquals(responseEntity.getStatusCode(), HttpStatus.OK,

"Expected: " + HttpStatus.OK + ", actual: " + responseEntity.getStatusCode());

}

@Test

void uploadData\_WrongKey() {

List<ReceivedInformation> info = dataGenerator.generateReceivedData(TEST\_DATA\_COUNT);

ResponseEntity<?> responseEntity = dataController.uploadData(TEST\_WRONG\_KEY, info);

Assertions.assertEquals(responseEntity.getStatusCode(), HttpStatus.EXPECTATION\_FAILED,

"Expected: " + HttpStatus.EXPECTATION\_FAILED + ", actual: " + responseEntity.getStatusCode());

}

}

class ScheduleControllerTest {

private final ScheduleService scheduleService = Mockito.mock(ScheduleService.class);

private final TasksService tasksService = Mockito.mock(TasksService.class);

private final AuthKeyService authKeyService = Mockito.mock(AuthKeyService.class);

private final ScheduleController scheduleController = new ScheduleController(

scheduleService,

tasksService,

authKeyService);

private static final Long TEST\_SCHEDULE\_ID = 10L;

private static final Long TEST\_TIMEFRAME\_ID = 20L;

private static final String TEST\_CLIENT\_IP = "10.20.30.40";

private static final List<TimeFrame> TEST\_TIMEFRAMES =

List.of(

TimeFrame.builder()

.id(TEST\_TIMEFRAME\_ID)

.startTime(LocalDateTime.now())

.endTime(LocalDateTime.now().plusSeconds(1))

.build()

);

private static final Schedule TEST\_SCHEDULE = Schedule.builder()

.id(TEST\_SCHEDULE\_ID)

.clientIp(TEST\_CLIENT\_IP)

.timeFrames(TEST\_TIMEFRAMES)

.build();

@BeforeEach

void setUp() {

}

@Test

void saveSchedule() {

ResponseEntity<?> response = scheduleController.saveSchedule(TEST\_SCHEDULE);

Assertions.assertEquals(response.getStatusCode(), HttpStatus.ACCEPTED,

"Expected: " + HttpStatus.ACCEPTED + ", actual: " + response.getStatusCode());

}

}

**Модуль** server

**Пакет** org.ssau.privatechannel.config

@Configuration

@ComponentScan(basePackages = "java")

@EnableTransactionManagement

public class DataSourceConfig {

private static final String POSTGRES\_DIALECT = "org.hibernate.dialect.PostgreSQL94Dialect";

@Bean

public DataSource getDataSource() {

String url = SystemContext.getProperty(SystemProperties.DB\_URL);

String username = SystemContext.getProperty(SystemProperties.DB\_USER);

String password = SystemContext.getProperty(SystemProperties.DB\_PASSWORD);

DataSourceBuilder<?> dataSourceBuilder = DataSourceBuilder.create();

dataSourceBuilder.url(url);

dataSourceBuilder.username(username);

dataSourceBuilder.password(password);

return dataSourceBuilder.build();

}

@Bean

public Properties hibernateProperties() {

Properties hibernateProp = new Properties();

hibernateProp.put(HibernateProps.DIALECT, POSTGRES\_DIALECT);

hibernateProp.put(HibernateProps.SHOW\_SQL, Boolean.FALSE.toString());

hibernateProp.put(HibernateProps.AUTO\_DDL, "update");

hibernateProp.put(HibernateProps.SQL\_FORMAT, Boolean.TRUE.toString());

return hibernateProp;

}

@Bean

public PlatformTransactionManager transactionManager() {

return new JpaTransactionManager(entityManagerFactory());

}

@Bean

public JpaVendorAdapter jpaVendorAdapter() {

return new HibernateJpaVendorAdapter();

}

@Bean

public EntityManagerFactory entityManagerFactory() {

LocalContainerEntityManagerFactoryBean factoryBean = new LocalContainerEntityManagerFactoryBean();

factoryBean.setPackagesToScan("org/ssau/privatechannel/model");

factoryBean.setDataSource(getDataSource());

factoryBean.setJpaVendorAdapter(new HibernateJpaVendorAdapter());

factoryBean.setJpaProperties(hibernateProperties());

factoryBean.setJpaVendorAdapter(jpaVendorAdapter());

factoryBean.afterPropertiesSet();

return factoryBean.getNativeEntityManagerFactory();

}

private static abstract class HibernateProps {

public static final String DIALECT = "hibernate.dialect";

public static final String SHOW\_SQL = "hibernate.show\_sql";

public static final String AUTO\_DDL = "hibernate.hbm2ddl.auto";

public static final String SQL\_FORMAT = "hibernate.format\_sql";

}

}

@Configuration

public class RestTemplateConfig {

@Bean

public RestTemplate getRestTemplate() {

return new RestTemplate();

}

}

@Configuration

public class ServletConfig implements WebServerFactoryCustomizer<ConfigurableWebServerFactory> {

@Override

public void customize(ConfigurableWebServerFactory factory) {

String appPort = SystemContext.getProperty(SystemProperties.APP\_PORT);

factory.setPort(Integer.parseInt(appPort));

}

}

**Пакет** org.ssau.privatechannel.controller

@Slf4j

@RestController

@RequestMapping(Endpoints.API\_V1\_SERVER)

public class InputController {

private final RestTemplate restTemplate;

private final ScheduleService scheduleService;

private final AuthKeyService authKeyService;

private final ScheduleValidatorService validatorService;

private static final Random RANDOMIZER = new Random();

@Autowired

public InputController(RestTemplate restTemplate,

ScheduleService scheduleService,

AuthKeyService authKeyService,

ScheduleValidatorService validatorService) {

this.restTemplate = restTemplate;

this.scheduleService = scheduleService;

this.authKeyService = authKeyService;

this.validatorService = validatorService;

}

@PostMapping(value = Endpoints.SCHEDULES)

public ResponseEntity<?> provideSchedules(@RequestBody List<Schedule> schedules) {

authKeyService.generateNewHeaderKey();

String clientId = schedules.get(0).getClientIp();

if (Objects.nonNull(scheduleService.findNextByIp(clientId))) {

scheduleService.addAll(schedules);

String logMessage = String.format("Schedules for client [IP = %s] already exists. " +

"Schedules will be added in queue", clientId);

log.info(logMessage);

return new ResponseEntity<>(logMessage, HttpStatus.ACCEPTED);

}

provideIds(schedules);

try {

validatorService.validateSchedules(schedules);

} catch (ValidationException e) {

return new ResponseEntity<>(e.getMessage(), HttpStatus.BAD\_REQUEST);

}

HttpEntity<Schedule> scheduleHttpEntity = new HttpEntity<>(schedules.get(0));

List<String> allClients = ClientsHolder.getAllClients();

for (String currentClient : allClients) {

String fullUrl = UrlSchemas.HTTP + currentClient + Endpoints.API\_V1\_CLIENT + Endpoints.SCHEDULE;

ResponseEntity<String> stringResponseEntity =

restTemplate.postForEntity(fullUrl, scheduleHttpEntity, String.class);

boolean isStatusSuccessful = stringResponseEntity.getStatusCode().is2xxSuccessful();

if (!isStatusSuccessful) {

log.error("Cannot send first schedule to client by ip: {}. Client returned {}",

currentClient, stringResponseEntity.getStatusCode());

if (stringResponseEntity.getStatusCode().equals(HttpStatus.BAD\_REQUEST)) //400

return new ResponseEntity<>(

"The server cannot process the request sent by the client", HttpStatus.BAD\_REQUEST);

if (stringResponseEntity.getStatusCode().equals(HttpStatus.NOT\_FOUND)) //404

return new ResponseEntity<>(

"The server did not find the page to which the link leads", HttpStatus.NOT\_FOUND);

if (stringResponseEntity.getStatusCode().equals(HttpStatus.INTERNAL\_SERVER\_ERROR)) //500

return new ResponseEntity<>(

"Server configuration failure", HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

schedules.remove(0);

scheduleService.addAll(schedules);

return new ResponseEntity<>("Success", HttpStatus.OK);

}

@PostMapping(value = Endpoints.GENERATE\_SCHEDULE, consumes = MediaType.TEXT\_PLAIN\_VALUE)

public ResponseEntity<?> generateSchedule(@PathVariable("duration") Integer duration,

@RequestBody String clientIp) {

int startDelay = 20;

int delayBetweenSchedules = 60;

List<Schedule> schedules = new ArrayList<>();

Schedule schedule = new Schedule();

TimeFrame timeFrame = new TimeFrame();

timeFrame.setStartTime(LocalDateTime.now().plusSeconds(startDelay));

timeFrame.setEndTime(LocalDateTime.now().plusSeconds(startDelay + duration));

TimeFrame timeFrame2 = new TimeFrame();

timeFrame2.setStartTime(timeFrame.getEndTime().plusSeconds(delayBetweenSchedules));

timeFrame2.setEndTime(timeFrame.getEndTime().plusSeconds(delayBetweenSchedules + duration));

List<TimeFrame> timeFrames = Arrays.asList(timeFrame, timeFrame2);

schedule.setTimeFrames(timeFrames);

schedules.add(schedule);

schedule.setClientIp(clientIp);

return provideSchedules(schedules);

}

private void provideIds(List<Schedule> schedules) {

for (Schedule schedule : schedules) {

if (Objects.isNull(schedule.getId()))

schedule.setId(Math.abs(RANDOMIZER.nextLong()) % Parameters.MAX\_ID);

for (TimeFrame timeFrame : schedule.getTimeFrames()) {

if (Objects.isNull(timeFrame.getId())) {

timeFrame.setId(Math.abs(RANDOMIZER.nextLong()) % Parameters.MAX\_ID);

}

}

}

}

}

@Slf4j

@RestController

@RequestMapping(path = Endpoints.API\_V1\_SERVER)

public class ScheduleController {

private final ScheduleService scheduleService;

@Autowired

public ScheduleController(ScheduleService scheduleService) {

this.scheduleService = scheduleService;

}

@PostMapping(value = Endpoints.GET\_NEW\_SCHEDULE, produces = MediaType.APPLICATION\_JSON\_VALUE)

public ResponseEntity<?> sendSchedule(@RequestBody String requesterIp) {

log.info("Searching schedule for client [ip={}]", requesterIp);

Schedule schedule = scheduleService.findNextByIp(requesterIp);

if (Objects.isNull(schedule)) {

log.info("Schedule for client [ip={}] not found", requesterIp);

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

}

if (!scheduleService.isActualSchedule(schedule)) {

log.error("Schedule not actual on server for client with IP = {}", requesterIp);

return new ResponseEntity<>(HttpStatus.EXPECTATION\_FAILED);

}

log.info("Schedule for client [ip={}] found: {}", requesterIp, schedule);

scheduleService.delete(schedule);

return new ResponseEntity<>(schedule, HttpStatus.OK);

}

}

@Slf4j

@RestController

@RequestMapping(value = Endpoints.API\_V1\_SERVER)

public class ServerController {

private final DataManagementService dataManagementService;

private final AuthKeyService authKeyService;

@Autowired

public ServerController(DataManagementService dataManagementService,

AuthKeyService authKeyService) {

this.dataManagementService = dataManagementService;

this.authKeyService = authKeyService;

}

@PostMapping(value = Endpoints.UPLOAD\_DATA, consumes = MediaType.APPLICATION\_JSON\_VALUE)

public ResponseEntity<?> uploadData(@RequestHeader(SystemProperties.HEADER\_KEY) String headerKey,

@RequestBody List<ConfidentialInfo> confidentialInfo) {

try {

try {

checkHeaderKey(headerKey);

}

catch (HeaderKeyNotActualException e) {

return new ResponseEntity<>(HttpStatus.EXPECTATION\_FAILED);

}

dataManagementService.tryToSendDataToReceiver(confidentialInfo);

} catch (BadRequestException e) {

return new ResponseEntity<>(e.getMessage(), HttpStatus.BAD\_REQUEST);

} catch (NotFoundException e) {

return new ResponseEntity<>(e.getMessage(), HttpStatus.NOT\_FOUND);

} catch (InternalServerErrorException e) {

return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL\_SERVER\_ERROR);

}

return new ResponseEntity<>(HttpStatus.OK);

}

private void checkHeaderKey(String key) throws HeaderKeyNotActualException {

if (!authKeyService.isActual(key)) {

String errorMessage = String.format("Key %s not actual and not persisted in database", key);

log.error(errorMessage);

throw new HeaderKeyNotActualException(errorMessage);

}

}

}

**Пакет** org.ssau.privatechannel.exception

public class BadRequestException extends Exception {

public BadRequestException(String message) {

super(message);

}

}

public class InternalServerErrorException extends Exception {

public InternalServerErrorException(String message) {

super(message);

}

}

public class NotFoundException extends Exception {

public NotFoundException(String message) {

super(message);

}

}

**Пакет** org.ssau.privatechannel.firetasks

public class NewTryToSendDataTask extends TimerTask {

private final RestTemplate restTemplate;

private final ConfidentialInfoService infoService;

private List<Long> ids;

private String serverAddress;

public NewTryToSendDataTask(RestTemplate restTemplate,

ConfidentialInfoService infoService) {

this.restTemplate = restTemplate;

this.infoService = infoService;

}

@Override

public void run() {

List<ConfidentialInfo> data = infoService.findAllByIds(ids);

infoService.deleteBatch(data);

HttpHeaders headers = new HttpHeaders();

headers.add(SystemProperties.HEADER\_KEY, SystemContext.getProperty(SystemProperties.HEADER\_KEY));

HttpEntity<?> entity = new HttpEntity<>(data, headers);

String appPort = SystemContext.getProperty(SystemProperties.APP\_PORT);

serverAddress = String.format(serverAddress, "127.0.0.1:" + appPort);

restTemplate.postForEntity(serverAddress, entity, String.class);

}

public void setIds(List<Long> ids) {

this.ids = ids;

}

public void setServerAddress(String serverAddress) {

this.serverAddress = serverAddress;

}

}

**Пакет** org.ssau.privatechannel.service

@Slf4j

@Service

public class DataManagementService {

private static final String RECEIVER\_URL = UrlSchemas.HTTP + "%s" + Endpoints.API\_V1\_CLIENT + Endpoints.UPLOAD\_DATA;

private static final String SERVER\_URL = UrlSchemas.HTTP + "%s" + Endpoints.API\_V1\_SERVER + Endpoints.UPLOAD\_DATA;

private static final Integer WAIT\_TIME\_IN\_MINUTES = 1;

private final ConfidentialInfoService infoService;

private final TimerService timerService;

private final RestTemplate restTemplate;

private final IpService ipService;

private final NetworkAdapterService networkAdapterService;

private static final Integer TRY\_SEND\_DATA\_DELAY\_SECONDS = 10;

@Autowired

public DataManagementService(ConfidentialInfoService infoService,

TimerService timerService,

RestTemplate restTemplate,

IpService ipService,

NetworkAdapterService networkAdapterService) {

this.infoService = infoService;

this.restTemplate = restTemplate;

this.timerService = timerService;

this.ipService = ipService;

this.networkAdapterService = networkAdapterService;

}

public void tryToSendDataToReceiver(List<ConfidentialInfo> confidentialInfo)

throws BadRequestException, NotFoundException, InternalServerErrorException {

if (infoService.getInfoCount() > 0) {

StartDataTransferringTask startTransferringTask =

new StartDataTransferringTask(infoService,

restTemplate,

ipService,

networkAdapterService);

timerService.createTask(startTransferringTask, LocalDateTime.now().plusSeconds(TRY\_SEND\_DATA\_DELAY\_SECONDS));

log.info("Trying to send remaining data from server");

}

HttpHeaders headers = new HttpHeaders();

headers.set(SystemProperties.HEADER\_KEY, SystemContext.getProperty(SystemProperties.HEADER\_KEY));

HttpEntity<List<ConfidentialInfo>> confidentialInfoHttpEntity = new HttpEntity<>(confidentialInfo, headers);

String receiverIP = confidentialInfo.get(0).getReceiverIP();

String httpAddress = String.format(RECEIVER\_URL, receiverIP);

ResponseEntity<String> stringResponseEntity;

try {

stringResponseEntity =

restTemplate.postForEntity(httpAddress, confidentialInfoHttpEntity, String.class);

} catch (Throwable e) {

e.printStackTrace();

log.error("Could not send data to client [ip = {}]: {}", receiverIP, e.getMessage());

infoService.addAll(confidentialInfo);

LocalDateTime nextTryTime = LocalDateTime.now().plusSeconds(WAIT\_TIME\_IN\_MINUTES \* 20);

scheduleNewTransferringTry(nextTryTime, confidentialInfo);

throw e;

}

boolean isStatusSuccessful = stringResponseEntity.getStatusCode().is2xxSuccessful();

if (!isStatusSuccessful) {

infoService.addAll(confidentialInfo);

if (stringResponseEntity.getStatusCode().equals(HttpStatus.BAD\_REQUEST)) {

log.error("Wrong request to receiver IP. May be body is incorrect");

throw new BadRequestException("Wrong request to receiver IP. May be body is incorrect");

}

if (stringResponseEntity.getStatusCode().equals(HttpStatus.NOT\_FOUND)) {

log.error("Page not found");

throw new NotFoundException("Page not found");

}

if (stringResponseEntity.getStatusCode().equals(HttpStatus.INTERNAL\_SERVER\_ERROR)) {

log.error("Something wrong with client on other side");

throw new InternalServerErrorException("Something wrong with client on other side");

}

}

}

private void scheduleNewTransferringTry(LocalDateTime timestamp,

List<ConfidentialInfo> confidentialInfo) {

NewTryToSendDataTask task = new NewTryToSendDataTask(restTemplate, infoService);

List<Long> ids = new ArrayList<>();

for (ConfidentialInfo currentRecord : confidentialInfo) {

ids.add(currentRecord.getId());

}

task.setIds(ids);

task.setServerAddress(SERVER\_URL);

timerService.createTask(task, timestamp);

}

}

@Slf4j

@Service

public class ScheduleValidatorService {

public void validateSchedules(List<Schedule> schedules) throws ValidationException {

for (Schedule schedule : schedules) {

validateSchedule(schedule);

}

}

private void validateSchedule(Schedule schedule) throws ValidationException {

for (TimeFrame timeFrame : schedule.getTimeFrames()) {

validateTimeframe(timeFrame);

}

checkTimeFramesForIntersects(schedule.getTimeFrames());

checkTimeframesOrder(schedule.getTimeFrames());

}

private void validateTimeframe(TimeFrame timeFrame) throws ValidationException {

if (timeFrame.getStartTime().isBefore(LocalDateTime.now())) {

String errorMessage = String.format("Timeframe %s has start before current time. " +

"It is incorrect", timeFrame);

log.error(errorMessage);

throw new ValidationException(errorMessage);

}

if (timeFrame.getStartTime().isAfter(timeFrame.getEndTime())){

String errorMessage = String.format(

"Schedule is incorrect. Timeframe start must be before timeframe end. Got: %s", timeFrame);

log.error(errorMessage);

throw new ValidationException(errorMessage);

}

}

private void checkTimeFramesForIntersects(List<TimeFrame> timeFrames) throws ValidationException {

for (int i = 0; i < timeFrames.size()-1; ++i) {

for (int j = 1; j < timeFrames.size(); ++j) {

if (timeFrames.get(i).isIntersectsWith(timeFrames.get(j))) {

String errorMessage = String.format("Timeframes %s and %s intersects",

timeFrames.get(i),

timeFrames.get(j));

log.error(errorMessage);

throw new ValidationException(errorMessage);

}

}

}

}

private void checkTimeframesOrder(List<TimeFrame> timeFrames) throws ValidationException {

for (int i = 0; i < timeFrames.size() - 1; ++i) {

LocalDateTime firstTimeframeStartTime = timeFrames.get(i).getStartTime();

LocalDateTime secondTimeframeStartTime = timeFrames.get(i+1).getStartTime();

if (secondTimeframeStartTime.isBefore(firstTimeframeStartTime)) {

String errorMessage = String.format("Timeframes must be ordered. " +

"Timeframe %s must be after and %s", secondTimeframeStartTime, firstTimeframeStartTime);

log.error(errorMessage);

throw new ValidationException(errorMessage);

}

}

}

}

**Пакет** org.ssau.privatechannel

@SpringBootApplication

public class Server {

private static final String CURRENT\_INSTANCE = "Server";

public static void main(String[] args) {

SystemContext.setProperty(SystemProperties.INSTANCE, CURRENT\_INSTANCE);

try {

StartPage.show();

} catch (IOException e) {

throw new RuntimeException("Could not show start page", e);

}

try {

ApplicationInstaller.run(Instances.SERVER);

} catch (Exception e) {

throw new RuntimeException("Something wrong during server start: ", e);

}

SpringApplication.run(Server.class);

}

}

**Тесты:**

class InputControllerTest {

private final RestTemplate restTemplate = Mockito.mock(RestTemplate.class);

private final ScheduleService scheduleService = Mockito.mock(ScheduleService.class);

private final AuthKeyService authKeyService = Mockito.mock(AuthKeyService.class);

private final ScheduleValidatorService validatorService = Mockito.mock(ScheduleValidatorService.class);

private final InputController inputController = new InputController(

restTemplate,

scheduleService,

authKeyService,

validatorService

);

private static final Long TEST\_SCHEDULE\_ID = 10L;

private static final Long TEST\_TIMEFRAME\_ID = 20L;

private static final String TEST\_CLIENT\_IP = "10.20.30.40";

private static final List<TimeFrame> TEST\_TIMEFRAMES =

List.of(

TimeFrame.builder()

.id(TEST\_TIMEFRAME\_ID)

.startTime(LocalDateTime.now())

.endTime(LocalDateTime.now().plusSeconds(1))

.build()

);

private static final Schedule TEST\_SCHEDULE = Schedule.builder()

.id(TEST\_SCHEDULE\_ID)

.clientIp(TEST\_CLIENT\_IP)

.timeFrames(TEST\_TIMEFRAMES)

.build();

private static final List<Schedule> TEST\_SCHEDULES = new ArrayList<>();

static {

TEST\_SCHEDULES.add(TEST\_SCHEDULE);

}

@Test

void provideSchedules() {

ResponseEntity<?> response = inputController.provideSchedules(TEST\_SCHEDULES);

Assertions.assertEquals(HttpStatus.OK, response.getStatusCode(),

"Expected: " + HttpStatus.OK + ", actual: " + response.getStatusCode());

}

}

class ScheduleControllerTest {

private final ScheduleService scheduleService = Mockito.mock(ScheduleService.class);

private final ScheduleController scheduleController = new ScheduleController(scheduleService);

private static final String TEST\_REQUESTER\_IP = "10.20.30.40";

private static final Long TEST\_SCHEDULE\_ID = 10L;

private static final Long TEST\_TIMEFRAME\_ID = 20L;

private static final String TEST\_CLIENT\_IP = "10.20.30.40";

private static final List<TimeFrame> TEST\_TIMEFRAMES =

List.of(

TimeFrame.builder()

.id(TEST\_TIMEFRAME\_ID)

.startTime(LocalDateTime.now())

.endTime(LocalDateTime.now().plusSeconds(1))

.build()

);

private static final Schedule TEST\_SCHEDULE = Schedule.builder()

.id(TEST\_SCHEDULE\_ID)

.clientIp(TEST\_CLIENT\_IP)

.timeFrames(TEST\_TIMEFRAMES)

.build();

@BeforeEach

void setUp() {

Mockito.when(scheduleService.findNextByIp(TEST\_REQUESTER\_IP)).thenReturn(TEST\_SCHEDULE);

Mockito.when(scheduleService.isActualSchedule(TEST\_SCHEDULE)).thenReturn(true);

}

@Test

void sendSchedule() {

ResponseEntity<?> response = scheduleController.sendSchedule(TEST\_REQUESTER\_IP);

Assertions.assertEquals(HttpStatus.OK, response.getStatusCode(),

"Expected: " + HttpStatus.OK + ", actual: " + response.getStatusCode());

}

}

class ServerControllerTest {

private final DataManagementService dataManagementService = Mockito.mock(DataManagementService.class);

private final AuthKeyService authKeyService = Mockito.mock(AuthKeyService.class);

private final Random RANDOMIZER = new Random();

private static final String TEST\_HEADER\_KEY = "test\_header\_key";

private static final String TEST\_WRONG\_KEY = "test\_wrong\_key";

private static final int TEST\_DATA\_COUNT = 10;

private final ServerController serverController = new ServerController(

dataManagementService,

authKeyService

);

@BeforeEach

void setUp() {

Mockito.when(authKeyService.isActual(Mockito.anyString())).thenReturn(false);

Mockito.when(authKeyService.isActual(TEST\_HEADER\_KEY)).thenReturn(true);

}

@Test

void uploadData() {

ResponseEntity<?> response = serverController.uploadData(TEST\_HEADER\_KEY, generate());

Assertions.assertEquals(response.getStatusCode(), HttpStatus.OK,

"Expected: " + HttpStatus.OK + ", actual: " + response.getStatusCode());

}

@Test

void uploadData\_wrong() {

ResponseEntity<?> response = serverController.uploadData(TEST\_WRONG\_KEY, generate());

Assertions.assertEquals(response.getStatusCode(), HttpStatus.EXPECTATION\_FAILED,

"Expected: " + HttpStatus.EXPECTATION\_FAILED + ", actual: " + response.getStatusCode());

}

private List<ConfidentialInfo> generate() {

List<ConfidentialInfo> info = new ArrayList<>();

for (int i = 0; i < ServerControllerTest.TEST\_DATA\_COUNT; ++i) {

ConfidentialInfo currentRecord = ConfidentialInfo.builder()

.id(null)

.data(generateRandomData())

.build();

info.add(currentRecord);

}

return info;

}

private Map<String, Object> generateRandomData() {

int MAX\_DATA\_ROWS\_COUNT = 10;

int rowsCount = 1 + (Math.abs(RANDOMIZER.nextInt()) % (MAX\_DATA\_ROWS\_COUNT));

Map<String, Object> result = new HashMap<>();

for (int i = 0; i < rowsCount; ++i) {

result.put(UUID.randomUUID().toString(), UUID.randomUUID() + UUID.randomUUID().toString());

}

return result;

}

}