SRS Анонимайзер

Руководство Администратора

Дисклеймер: Программное обеспечение является встраиваемым компонентом и не имеет графического пользовательского интерфейса. Конечный пользователь ПО – разработчики программного обеспечения, системные администраторы.

Содержание

[Аннотация 2](#_Toc80881071)

[Назначение ПО 2](#_Toc80881072)

[Назначение программы 2](#_Toc80881073)

[Функции программы 2](#_Toc80881074)

[Архитектура ПО 3](#_Toc80881075)

[Назначение (контекст) 3](#_Toc80881076)

[Архитектура на уровне контейнера 3](#_Toc80881077)

[Архитектура на уровне компонентов 4](#_Toc80881078)

[Установка ПО и Средств Доступа к ПО 5](#_Toc80881079)

[Требования к необходимым техническим средствам 5](#_Toc80881080)

[Установка, выполнение и доступ в ПО в среде JRE 6](#_Toc80881081)

[Установка, выполнение и доступ в ПО в среде Docker (Podman) 6](#_Toc80881082)

[Настройка параметров ПО 7](#_Toc80881083)

[Процедуры диагностики и устранения ошибок ПО 9](#_Toc80881084)

[Приложения 9](#_Toc80881085)

[Приложение А – Изменение настроек ПО 9](#_Toc80881086)

[Приложение Б – Файл базовых параметров логирования 10](#_Toc80881087)

[Приложение C – Сведения о схеме базы данных 11](#_Toc80881088)

# Аннотация

Настоящий документ содержит руководство для администратора по работе с программным обеспечением «Анонимайзер» (далее, компонент анонимайзер, анонимайзер или ПО). Назначением документа является описание назначения, архитектуры, доступа, настроек и диагностирования/устранения ошибок ПО.

# Назначение ПО

## Назначение программы

SRS Анонимайзер предоставляет собой компонент, реализующий набор стратегий для обработки сообщений ряда стандартных и частных протоколов, используемых в секторе коммерческой авиации и основанных на интернет-протоколе HTTP (SOAP и REST). Обработка состоит в идентификации, удалении (маскировании) и восстановлении персональных данных, содержащихся в таких сообщениях. ПО реализует предметно-ориентированный язык (DSL) для упрощения работы с сообщениями и понижения требований к квалификации персонала, обслуживающего и дополняющего данное ПО.

## Функции программы

SRS Анонимайзер (Далее – Анонимайзер) реализован на базе компонента «SRS Прокси» (далее – Прокси) и позволяет предотвратить утечку персональных данных за пределы периметра определяемого топологией развертывания ПО.

Компонент Прокси принимает потоки данных на порты, заданные в конфигурации. При получении входного пакета данных Прокси определяет поддерживает ли Анонимайзер данный тип сообщения. При положительном ответе Прокси передает содержание запроса Анонимайзеру для обработки. Получив обработанный (анонимизированный) запрос, Прокси направляет его по назначению и ждет ответа. Получив (анонимизированный) ответ, Прокси передаёт его Анонимайзеру для де-анонимизации и, в заключение, отправляет обработанный ответ к источнику запроса.

В случае, когда Анонимайзер не поддерживает анализ и модификацию запроса и ответа полученного пакета данных, Прокси пересылает запросы и ответы между источником и назначением без какого-либо анализа и модификации, используя при этом наиболее эффективный алгоритм для пересылки (streaming).

Анонимайзер использует библиотеку стратегий анализа и обработки потоков данных. Библиотека стратегий состоит из файлов в формате YAML, которые Анонимайзер загружает и активирует в процессе запуска ПО. Каждый такой файл использует предметно-ориентированный язык (DSL) для выбора элементов в структуре сообщения для обработки, а также конкретные операции обработки. Реализация предметно-ориентированного языка (DSL) также «знает» как извлечь (или сохранить) из внешней базы данных информацию необходимую для вышеупомянутой обработки.

# Архитектура ПО

## Назначение (контекст)

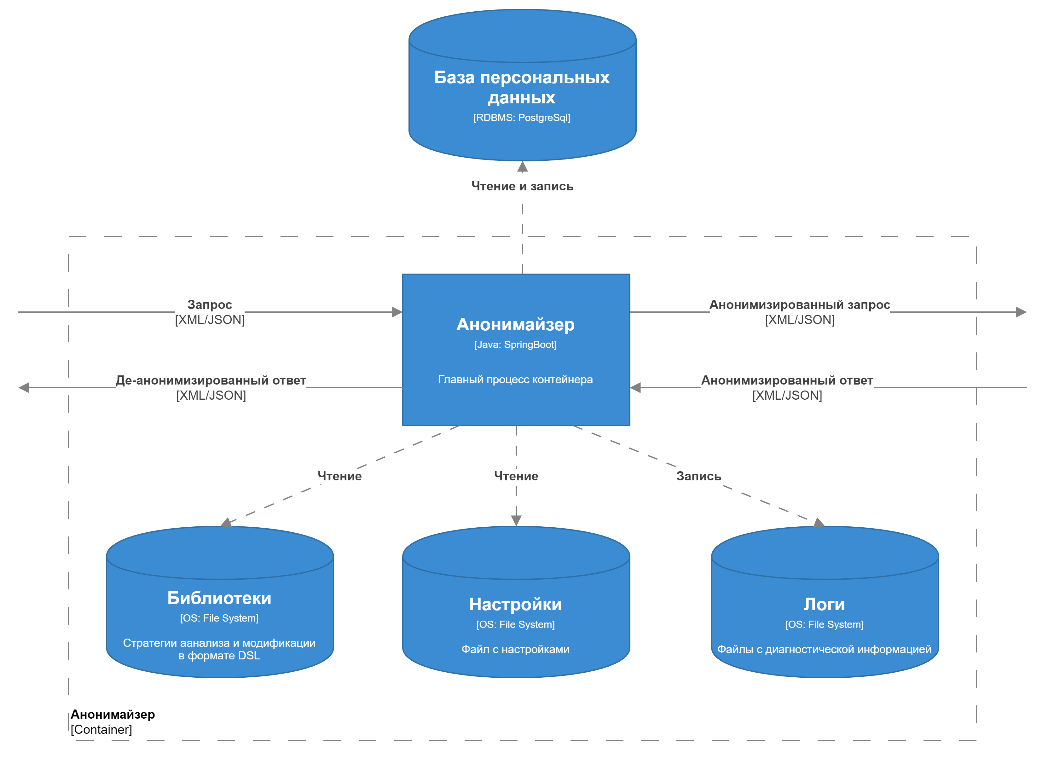
SRS Анонимайзер, вместе с компонентом SRS Прокси, реализует мини-платформу для перехвата, анализа и модификации потоков данных, используемых в секторе коммерческой авиации и основанный на протоколе HTTP, под-протоколах SOAP (XML) и REST (JSON).



Системы потребители (источники потоков данных) взаимодействуют с системами обслуживания (назначение потоков данный) через ПО вместо того, чтобы взаимодействовать с друг другом непосредственно. Наличии ПО в этой цепочки коммуникаций делает возможными анализ и модификацию потока данных. Анонимайзер упрощает вышеуказанные анализ и модификацию за счет предоставления библиотеки стратегий для обработки сообщений, принадлежащих к популярным протоколам сектора коммерческой авиации, а также путем предоставления предметно-ориентированный языка (DSL) для разработки новых стратегий.

## Архитектура на уровне контейнера

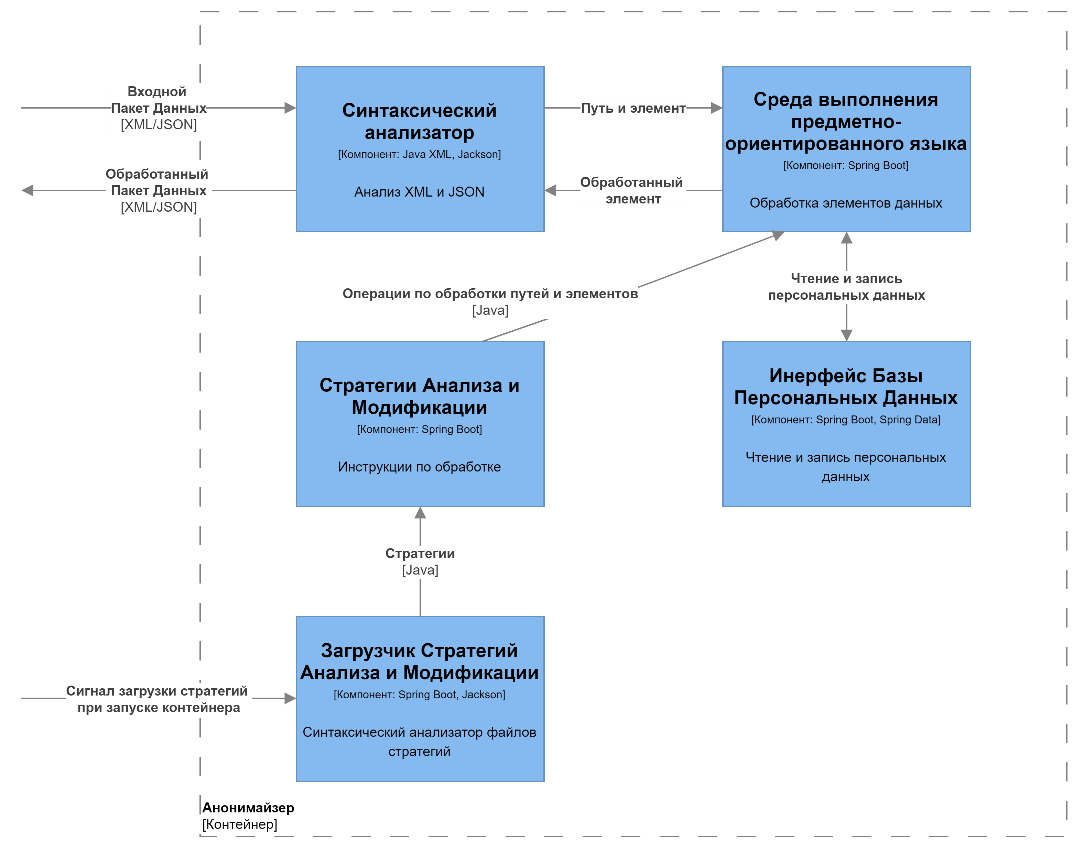
В контексте контейнера ПО состоит из главного процесса анонимайзера, который использует базовые (системные) библиотеки наряду с библиотеками стратегий, файл конфигурации параметров ПО и систему логирования с ее собственным подмодулем конфигурации.



ПО также взаимодействует со внешней базой данных для получения и сохранения информации необходимой для обработки запросов и ответов.

## Архитектура на уровне компонентов

Контейнер ПО включает в себя компоненты, отвечающие за синтаксический анализ пакетов данных, среды выполнения предметно-ориентированного языка, а также дополнительные модули для чтения стратегий обработки и взаимодействия с базой персональных данных.



Все компоненты используют SpringBoot для сборки и управления зависимостями, в то время как синтаксические анализаторы используют библиотеки Java XML (встроенную в Java Virtual Machine Runtime) и Jackson FasterXML.

# Установка ПО и Средств Доступа к ПО

Компонент анонимайзер может быть установлен и выполнен следующими методами:

1. Посредством выполнения в среде контейнеризатора приложений, например Docker или Podman. Этот метод рекомендуется для приложений, которые не предоставляют повышенных требований к надежности предоставляемых служб.
2. Посредством выполнения в среде Java Runtime Environment (JRE). Этот метод рекомендуется для поддержки процессов разработки и отладки.
3. Посредством выполнения в среде оркестровки контейнеризированных приложение с поддержкой автоматизации их развертывания, масштабирования и координации в условиях кластера, например Kubernetes.

Следующие подразделы предоставляют детальные инструкции по установке, выполнению и доступа к среде выполнения для первого и второго из вышеуказанных методов. Производитель ПО предоставляет хостингу в среде Kubernetes в качестве отдельных услуг.

## Требования к необходимым техническим средствам

В случае каждого из трех вышеупомянутые методы установки, для обеспечения функционирования компонента прокси необходимо убедиться в доступности нижеперечисленных технических средств для каждого процесса ПО:

* Двух виртуальных процессоров с частотой 2.7 МГц,
* ОЗУ – не менее 4 Гб,
* ЖМД – не менее 10 Гб или в зависимости от конфигурации подсистемы логирования.

## 

## Установка, выполнение и доступ в ПО в среде JRE

Для выполнения ПО в среде JRE, в дополнении к необходимым техническим средствам, требуется наличие[[1]](#footnote-1):

1. Операционной системы CentOS 7.2, Red Hat Enterprise Linux Server 7.2 или Ubuntu 18.04 LTS
2. Open JDK версия 11 доступная в пути среды операционной системы. Например, на операционной системе Ubuntu 18.04 LTS, установка Open JDK 11 производится посредством выполнения: sudo apt install default-jdk.
3. Компонент прокси **anonymizer-proxy-${version}.jar** предоставленной производителем ПО в виде библиотеки JAR содержащую все необходимые библиотеки поддержки,
4. Файл **override.yaml** с настройками ПО которые отличаются от настроек поставляемых с компонентом прокси по умолчанию (см. Настройка Параметров ПО)

Само выполнение ПО осуществляется при помощи интерфейса командной строки (или с помощью скрипта) посредством следующей команды:

java -jar anonymizer-proxy-${version}.jar --spring.config.additional-location=file:${path-to-config}/override.yaml

где,

**${version}** – версия библиотеки компонента прокси, в настоящее время 1.0.0

**${path-to-config}** – путь к файлу с настройками ПО

При успешном старте компонент прокси выведен на консоль следующее сообщение:

Started Application in X.Y seconds (JVM running for X.Y)

После этого компонент прокси готов к получению запросов на порты, открытые во время запуска. В процессе работы компонент будет выводить всю отладочную информацию на консоль.

## Установка, выполнение и доступ в ПО в среде Docker (Podman)

Для выполнения ПО в среде Docker, в дополнении к необходимым техническим средствам, требуется наличие1:

1. Операционной системы CentOS 7.2 или Red Hat Enterprise Linux Server 7.2,
2. Подсистема Docker (версия 19.x или более недавняя) или Podman (версия 1.6.x или более недавняя) доступная в среде операционной системы,
3. Архив изображения ПО **anonymizer-proxy-${version}.tar** предоставленной производителем ПО в виде архива TAR,
4. Файл **override.yaml** с настройками ПО которые отличаются от настроек поставляемых с компонентом прокси по умолчанию (см. [Настройка Параметров ПО](#_Настройка_Параметров_ПО))

Само выполнение ПО осуществляется при помощи интерфейса командной строки (или с помощью скрипта) посредством следующей команды:

docker run -p ${port}:{port} -v /tmp/config:/tmp/config \

-e JAVA\_ARGS=--spring.config.additional-location=file: ${path-to-config}/override.yaml \

anonymizer-proxy:${version}

где,

**${version}** – версия библиотеки компонента прокси, в настоящее время 1.0.0,

**${path-to-config}** – путь к файлу с настройками ПО

**${port}** – порт слушателя согласно конфигурации По. При необходимости опция повторяется для других портов, указанных в конфигурации.

При успешном старте контейнер прокси выведен на консоль следующее сообщение:

Started Application in X.Y seconds (JVM running for X.Y)

После этого контейнер прокси готов к получению запросов на порты, открытые во время запуска. В процессе работы контейнер будет выводить всю отладочную информацию на консоль.

# Настройка параметров ПО

ПО хранит все базовые настройки в файле **application.yaml,** который находится в корневом директории библиотеки JAR компонента. Пользователи ПО могут запросить полную версию **application.yaml** с каталогом всех настроек и их допустимых значений у производителя ПО. Для изменения параметров ПО пользователи могут создать дополнительный файл в формате YAML (например **override. yaml**) и поместить все желаемые настройку в данный файл, например:

anonymizer:

db-driver-class-name: org.postgresql.Driver

db-url: jdbc:postgresql://localhost:5432/postgres

db-username: postgres

db-password: \*\*\*\*\*

server-shutdown-timeout: 5000ms

pass-through: false

processing-queue-size: 10

idle-channel-timeout: 5m

routes:

- name: 'SOAP'

ingressPort: 2220

resolver: 'Generic'

services:

- name: 'SOAP Service'

baseUrl: 'https://some-service.com:443'

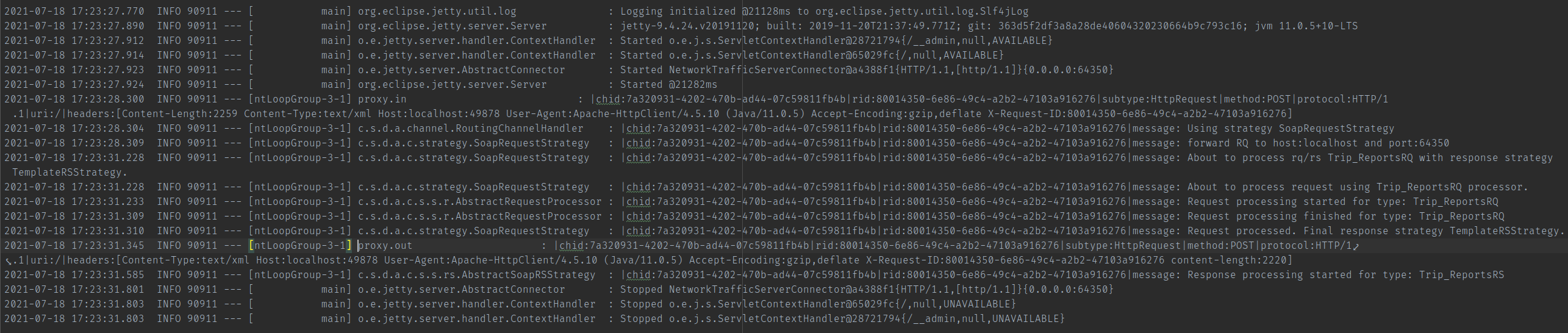
path: '/custom'

Затем, пользователи могут указать на дополнительный файл настроек через опцию компонента --spring.config.additional-location как объяснено в [Установка, Выполнение и Доступ к ПО в Среде JRE](#_Установка,_Выполнение_и).

Для логирования ПО использует библиотеку SLF4J и настройка уровней логирования осуществляется способом который является стандартным для Spring Boot: <https://www.baeldung.com/spring-boot-logging>. Производитель предоставляет базовую конфигурацию компонента логирования по отдельному запросу.

# Процедуры диагностики и устранения ошибок ПО

Подсистема логирования служит основным источником для диагностики ошибок. По умолчанию, подсистема логирования выводит основную информацию о потоках данных и стратегий по их обработкам:



Неправильная конфигурация, нерабочее состояние систем назначения поток данных и проблемы со стратегиями обработки потоков данных являются основными источниками проблем данного ПО. Во всех этих случаях ПО выводит отладочную информацию на уровне ERROR или WARNING. Дополнительно, лог содержит трассировки стека для всех исключительных ситуаций на уровне JVM и ПО.

В случае, когда необходима дополнительная отладочная информация, рекомендуется поменять уровень логирования с INFO на DEBUG.

# Приложения

## Приложение А – Изменение настроек ПО

Как было объяснено в [Установка ПО и Средств Доступа к ПО](#_Установка_ПО_и), базовые настройки ПО могут быть переопределены с командной строки с помощью файла **override.yaml** который имеет следующую структуру:

anonymizer:

db-driver-class-name: org.postgresql.Driver

db-url: jdbc:postgresql://localhost:5432/postgres

db-username: postgres

db-password: \*\*\*\*\*

server-shutdown-timeout: 5000ms

pass-through: false

processing-queue-size: 10

idle-channel-timeout: 5m

routes:

- name: 'SOAP'

ingressPort: 2220

resolver: 'Generic'

services:

- name: 'SOAP Service'

baseUrl: 'https://some-service.com:443'

path: '/custom'

Основные изменения и добавления к **override.yaml** обычно состоят в определение новых или модификации существующих маршрутов (routes).

## Приложение Б – Файл базовых параметров логирования

Нижеприведенный базовый файл настроек логирования показывает ряд логгеров, уровень которых можно менять в случае необходимости получения более детальной отладочной информации.



Наиболее полезные логгеры и их уровни:

|  |  |  |
| --- | --- | --- |
| Логгер | Уровень | Краткое описание диагностической информации |
| anonymizer.in,  anonymizer.proxy.in | INFO | Детали части начала запроса (метод, URI, протокол, заголовки) |
|  | DEBUG | Трассирование частей содержания и концовки запроса |
| anonymizer.out,  anonymizer.proxy.out | INFO | Детали части начала ответа (метод, URI, протокол, заголовки) |
|  | DEBUG | Трассирование частей содержания и концовки ответа |
| anonymizer.request.content | DEBUG | Содержание запроса |
| anonymizer.response.content | DEBUG | Содержание ответа |

## Приложение C – Сведения о схеме базы данных

Инструкции/скрипты по созданию таблиц и индексов:

CREATE SEQUENCE personal\_data\_id\_seq

INCREMENT 1

START 1

MINVALUE 1

MAXVALUE 9223372036854775807

CACHE 1;

CREATE SEQUENCE pnr\_info\_id\_seq

INCREMENT 1

START 1

MINVALUE 1

MAXVALUE 9223372036854775807

CACHE 1;

CREATE SEQUENCE segment\_info\_id\_seq

INCREMENT 1

START 1

MINVALUE 1

MAXVALUE 9223372036854775807

CACHE 1;

CREATE SEQUENCE passenger\_info\_id\_seq

INCREMENT 1

START 1

MINVALUE 1

MAXVALUE 9223372036854775807

CACHE 1;

CREATE SEQUENCE passenger\_additional\_info\_id\_seq

INCREMENT 1

START 1

MINVALUE 1

MAXVALUE 9223372036854775807

CACHE 1;

CREATE TABLE pnr\_info

(

id bigint NOT NULL DEFAULT nextval('pnr\_info\_id\_seq'),

pnr character varying NOT NULL,

creation\_date\_time timestamp with time zone NOT NULL,

domestic\_traffic boolean NOT NULL,

PRIMARY KEY (id),

UNIQUE(pnr)

);

CREATE TABLE segment\_info

(

id bigint NOT NULL DEFAULT nextval('segment\_info\_id\_seq'),

pnr\_id bigint NOT NULL,

segment\_identifier character varying NOT NULL,

creation\_date\_time timestamp with time zone NOT NULL,

departure\_date date,

departure\_airport character varying,

arrival\_date date,

arrival\_airport character varying,

flight\_number character varying,

UNIQUE(pnr\_id, segment\_identifier),

PRIMARY KEY (id),

CONSTRAINT fk\_pnr\_info FOREIGN KEY(pnr\_id) REFERENCES pnr\_info(id)

);

CREATE TABLE passenger\_info

(

id bigint NOT NULL DEFAULT nextval('passenger\_info\_id\_seq'),

pnr\_id bigint NOT NULL,

creation\_date\_time timestamp with time zone NOT NULL,

passenger\_name\_number character varying NOT NULL,

passenger\_full\_id character varying,

UNIQUE(pnr\_id, passenger\_name\_number),

PRIMARY KEY (id),

CONSTRAINT fk\_pnr\_info FOREIGN KEY(pnr\_id) REFERENCES pnr\_info(id)

);

CREATE TABLE passenger\_additional\_info

(

id bigint NOT NULL DEFAULT nextval('passenger\_additional\_info\_id\_seq'),

pnr\_id bigint NOT NULL,

creation\_date\_time timestamp with time zone NOT NULL,

segment\_id bigint NOT NULL,

passenger\_id bigint NOT NULL,

seat\_number character varying,

frequent\_flyer\_number character varying,

bag\_tag\_number character varying,

ticket\_number character varying,

UNIQUE(pnr\_id, segment\_id,passenger\_id),

PRIMARY KEY (id),

CONSTRAINT fk\_pnr\_info FOREIGN KEY(pnr\_id) REFERENCES pnr\_info(id),

CONSTRAINT fk\_passenger\_info FOREIGN KEY(passenger\_id) REFERENCES passenger\_info(id),

CONSTRAINT fk\_segment\_info FOREIGN KEY(segment\_id) REFERENCES segment\_info(id)

);

CREATE TABLE personal\_data

(

id bigint NOT NULL DEFAULT nextval('personal\_data\_id\_seq'),

pnr\_id bigint NOT NULL,

segment\_id bigint,

passenger\_id bigint,

owner\_passenger\_id bigint,

sequential\_number integer,

field\_name character varying,

data character varying,

data\_anonymized character varying,

PRIMARY KEY (id),

UNIQUE(pnr\_id, segment\_id, passenger\_id, owner\_passenger\_id, sequential\_number, field\_name),

CONSTRAINT fk\_pnr\_info FOREIGN KEY(pnr\_id) REFERENCES pnr\_info(id),

CONSTRAINT fk\_segment\_info FOREIGN KEY(segment\_id) REFERENCES segment\_info(id),

CONSTRAINT fk\_passenger\_info FOREIGN KEY(passenger\_id) REFERENCES passenger\_info(id),

CONSTRAINT fk\_owner\_passenger\_info FOREIGN KEY(owner\_passenger\_id) REFERENCES passenger\_info(id)

);

CREATE TABLE session\_state

(

security\_token character varying NOT NULL,

pnr character varying NOT NULL,

creation\_date\_time timestamp with time zone NOT NULL,

PRIMARY KEY (security\_token, pnr)

);

CREATE INDEX idx\_pnr\_info\_pnr ON pnr\_info(pnr);

CREATE INDEX idx\_personal\_data\_pnr\_id ON personal\_data(pnr\_id);

CREATE INDEX idx\_session\_state\_session\_token ON session\_state (security\_token);

Инструкции по созданию вспомогательных SQL функций:

CREATE SEQUENCE uid\_id\_seq

INCREMENT 1

START 1

MINVALUE 1

MAXVALUE 2147483647

CACHE 1;

CREATE TABLE uid

(

id integer NOT NULL DEFAULT nextval('uid\_id\_seq'),

uid character varying,

creation\_date\_time timestamp with time zone default now(),

PRIMARY KEY (id),

UNIQUE(uid)

);

CREATE INDEX uid\_index ON uid(uid);

CREATE OR REPLACE FUNCTION public.generate\_uid(

size integer DEFAULT 29)

RETURNS text

LANGUAGE 'plpgsql'

COST 100

VOLATILE

AS $BODY$

DECLARE

characters TEXT := 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';

l INT := length(characters);

i INT := 0;

output TEXT := '';

BEGIN

WHILE i < size LOOP

-- floor(random()\* (high-low + 1) + low)::int gives int from low to high

output := output || substr(characters,floor(random()\* (l) + 1)::int , 1);

i := i + 1;

END LOOP;

RETURN output;

END;

$BODY$;

CREATE OR REPLACE FUNCTION public.get\_uid(

length integer DEFAULT 29)

RETURNS text

LANGUAGE 'plpgsql'

COST 100

VOLATILE

AS $BODY$

DECLARE

res TEXT;

notexists integer := 1;

BEGIN

res := generate\_uid(length);

while notexists = 1 loop

IF EXISTS (SELECT \* FROM uid WHERE uid = res) THEN

res := generate\_uid(length);

ELSE

INSERT INTO uid (uid) VALUES (res);

notexists := 0;

END IF;

end loop;

return res;

END

$BODY$;

1. Для упрощения процесса установки Производитель ПО предоставляет скрипт для установки все необходимых системных компонентов. [↑](#footnote-ref-1)