INVITRO Assistant

Информационная система

Автоматизированное рабочее место процедурной сестры. Инструкция по установке.

2023, Москва

Содержание

1 Введение

Настоящий документ содержит описание работ по развертыванию автоматизированного рабочего места процедурной сестры (далее по тексту – системы).

1.1 Назначение и функции системы.

Система предназначена для автоматизации процессов медицинского офиса по регистрации заявок для лаборатории, автоматизации документооборота с клиентами и пациентами, автоматизации выдачи результатов, автоматизации проведения продажи и контроля кассовой дисциплины.

Основными задачами и бизнес-функциями системы являются:

Работа с карточкой пациента, включая следующие бизнес-функции:

- Поиск пациента;
- Создание карточки анонимного пациента;
- Создание карточки пациента;
- Корректировка данных карточки пациента;
- Печать амбулаторной карты;
- Подключение и отключение от ЭДО;
- Доступ к истории и работа с подписанными пациентом юридическими документами;
- Доступ к истории заказов пациента;
- Подключение пациента к программе лояльности;
- Персонификация чекапов;
- Применение продуктов чекапа к заказу;
- Возврат непримененных продуктов чекапа.

Работа с новым заказом, включая следующие бизнес-функции:

- Изменение у заказа заказчика и отражение этих изменений на корзине заказа;
- Назначение контактов пациентом и заказчиком в заказ;
- Поиск продуктов в продуктовом каталоге и добавление их в корзину заказа;
- Работа с приоритетами продуктов в корзине;
- Изменение типа оплаты у заказа;
- Заполнение медицинской информации по заказу и медицинских опросников;
- Настройка правил доставки результатов клиенту;
- Настройка правил доставки результатов в госорганы;

- Передача пациенту информации о продуктах заказа на почту или в распечатанном виде;
- Внесение анкет;
- Ввод результатов по тестам до и после оплаты;
- Вывод информации о рекомендациях к продуктам заказа;
- Отображение и распечатка документов и бланков по заказау;
- Работа с юридическими документами по заказу;
- Подписание документов через ЭДО;
- Применение правил лояльности к заказу;
- Создание заказов из ранее созданных предзаказов;
- Оплата заказа наличными, картой, СБП;
- Оформление заказа в режиме CITO.

Работа с ранее созданным заказом, включая следующие бизнес-функции:

- Поиск истории заказов по офису офису;
- Поиск истории заказов по пациенту;
- Отображение информации по ранее созданному заказу;
- Формирование дозаказа;
- Повторение заказа;
- Возврат части или всего заказа;
- Доступ к кассовым чекам по заказу;
- Доступ к документам заказа.

Дополнительно система выполняет сопутствующие вышеуказанным процессам бизнес-функции:

- Поиск и работа с документами-результатами;
- Формирование отчетов о работе офиса;
- Печать амбулаторной карты пациента;
- Подключение пациента к системе ЭДО;
- Продажа, персонификация, применение, возврат продуктов чекапа;
- Доступ к истории заказов, где контакт выступал как пациент и плательщик;
- Доступ к чекам, документам, этикеткам по ранее созданному заказу;
- Массовое создание заказов.

1.2 Компоненты системы.

Система состоит из следующих базовых компонентов, запуск которых обеспечивает возможность корректной работы пользователя в медицинском офисе:

- **АРМПС2 Frontend -** компонент предназначен для работы пользователя по созданию, редактированию и оплаты заказа из медицинского офиса планирования услуг, получения результатов лабораторных исследования, а также предоставления клиенту копий платежных и иных документов по программируемым правилам;
- **АРМПС2 Backend -** компонент предназначен для управления работой клиентского приложения по программируемой логике;
- **DB АРМПС2** – база данных АРМПС2;
- **Rabbit MQ** – программный брокер сообщений;
- **Redis** – программное обеспечение для потоковой обработки данных при использовании очереди;
- **NVITRO Hardware (HWS)** – вспомогательное программное обеспечение для работы пользователя с оборудованием в локальной сети. Устанавливается на рабочее место пользователя.

Внутренние сервисы системы:

- **АРМПС2 Frontend Admin -** компонент клиент-серверного взаимодействия системы для осуществления управления настройками администратора (для демонстрационного стенда не требуется);
- **Сервис отчетов АРМПС2 -** компонент предназначен для формирования отчета по программируемой логике;
- **Сервис работы с цифровой платформой для АРМПС2 -** компонент предназначен для взаимодействия системы с цифровой платформой с целью наложения на документ цифровой подписи;
- **Сервис документов АРМПС2 -** компонент предназначен для формирования требуемых документов в соответствии с шаблоном по программируемой логике;
- **Сервис получения данных в АРМПС2 -** компонент предназначен для получения информации по подписке из интеграционной шины данных (для демонстрационного стенда не требуется, поскольку является интеграционным сервисом);
- **Сервис отправки данных из АРМПС2 -** компонент предназначен для публикации информации по схеме в интеграционную шину данных (для демонстрационного стенда не требуется, поскольку является интеграционным сервисом).

1.3 Термины и определения

INVITRO Hardware (HWS) – Вспомогательное программное обеспечение системы, необходимое для взаимодействия с локальным оборудованием (касса, платежные терминалы, термо-принтеры и принтеры А4). Программное обеспечение устанавливается локально на компьютер пользователя в медицинском офисе.

Kubernetes – Открытое программное обеспечение для оркестровки контейнеризованных приложений – автоматизации их развертывания, масштабирования и координации в условиях кластера.

Logback/Log4J – Программная платформа с настройкой для логирования приложений на языках Java.

PostgreSQL – Свободная объектно-реляционная система управления базами данных.

Rabbit MQ – Программный брокер сообщений на основе стандарта AMPQ.

Redis – Резидентная система управления базами данных класса NoSQL с открытым исходным кодом.

Docker-образ – Шаблон для создания Docker-контейнеров.

Docker — Программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений.

Лог (log) — Текстовый файл, куда автоматически записывается важная информация о работе системы или программы.

Репозиторий – Виртуальное пространство, предназначенное для хранения исходников программного кода компонентов/приложения/системы/комплекса.

2 Требования для установки комплекса

Для установки и настройки системы существуют обязательные минимально необходимые требования.

2.1 Требования к квалификации специалиста

Настройку и установку всех компонентов и подсистем, входящих в систему должен выполнять технический специалист, имеющий соответствующую квалификацию по установке серверных систем.

Технический специалист должен отвечать следующим квалификационным требованиям:

- иметь опыт администрирования Windows, Linux, Oracle;
- владеть знаниями в области сетевых технологий;
- иметь знания в области Java-технологий;
- иметь опыт разворачивания приложений в Kubernetes;
- владеть методами системного администрирования.

2.2 Требования к аппаратному обеспечению (не ниже)

АРМПС2 Frontend Admin:

- Процессор Intel Xeon-P 8270 FIO Kit for DL560 G10 (рекомендуемое);
- Оперативная память: 512 Мб;
- Ядра: 1 CPU.

АРМПС2 Frontend:

- Процессор Intel Xeon-P 8270 FIO Kit for DL560 G10 (рекомендуемое);
- Оперативная память: 512 Мб;
- Ядра: 1 CPU.

АРМПС2 Backend:

- Процессор Intel Xeon-P 8270 FIO Kit for DL560 G10 (рекомендуемое);
- Оперативная память: 4096 Мб;
- Диск SSD: 50Гб;
- Ядра: 4 vCPU.

DB АРМПС2:

- Процессор Intel Xeon-P 8270 FIO Kit for DL560 G10 (рекомендуемое);
- Оперативная память: 4096 Мб;
- Диск SSD: 100Гб;
- Ядра: 4 vCPU.

Сервис отчетов АРМПС2:

- Процессор Intel Xeon-P 8270 FIO Kit for DL560 G10 (рекомендуемое);
- Оперативная память: 1024 Мб;
- Ядра: 1 CPU.

Сервис работы с цифровой платформой для АРМПС2:

- Процессор Intel Xeon-P 8270 FIO Kit for DL560 G10 (рекомендуемое);
- Оперативная память: 1024 Мб;
- Ядра: 1 CPU.

Сервис документов АРМПС2:

- Процессор Intel Xeon-P 8270 FIO Kit for DL560 G10 (рекомендуемое);
- Оперативная память: 1024 Мб;
- Ядра: 1 CPU.

Сервис получения данных в АРМПС2:

- Процессор Intel Xeon-P 8270 FIO Kit for DL560 G10 (рекомендуемое);
- Оперативная память: 1024 Мб;
- Ядра: 1 vCPU.

Сервис отправки данных из АРМПС2:

- Процессор Intel Xeon-P 8270 FIO Kit for DL560 G10 (рекомендуемое);
- Оперативная память: 1024 Мб;
- Ядра: 1 vCPU.

INVITRO Hardware (HWS)

- Оперативная память: 8192 Мб;
- Диск SSD: 20Гб;
- Расположение - рабочее место пользователя.

2.3 Необходимое ПО

Следующее ПО необходимо установить перед основной установкой основных компонентов системы:

на уровне сервера приложений: Ubuntu 22.04 или выше;

на уровне сервера базы данных, вспомогательного сервера баз данных: Ubuntu 22.04 или выше;

для HWS (перечень операционных систем, на которых протестирована работа сервиса):

- Windows 7 - x86, x64;
- Corporate, Windows 8 - x64;
- Corporate, Windows 10 - x64;
- Corporate, Professional;

- Ultimate Windows 11 - x64 – PRO.

на уровне систем управления базами данных: PostgreSQL 14 или выше;

на уровне браузера: Google Chrome 75.0 и выше;

на уровне кластера Kubernetes: Kubernetes 1.22 или выше;

на уровне управления контейнеризацией: Docker 22.0 или выше;

на уровне брокера сообщений: RabbitMQ 3.10. или выше;

на уровне управления кэшем: Redis 7.0 или выше.

2.4 Требования к каналам связи

Способность систем к обмену данными должна учитывать возможность использования каналов со скоростью передачи не выше 100 Мбит/с.

2.5 Комплектность инсталляционного пакета

Инсталляционный пакет состоит из:

- Установки и запуска конфигураций PostgreSQL;
- Установки и запуска конфигураций RabbitMQ;
- Установки и запуска конфигураций Redis;
- Установки и запуска конфигураций Docker;
- Установки и запуска конфигураций Kubernetes;
- Установки и запуска компонентов системы;
- Установки и запуска INVITRO Hardware (HWS) на рабочем месте пользователя.

3 Настройка системы

3.1 Просмотр версии ОС

Информация о выпуске:

Версия: ARMPS-3.1.0, (v.2.0.0);

Окружение: Windows 10, Chrome 112.0;

Разрядность ОС: Ubuntu 22.04X64.

3.2 Установка и конфигурация PostgreSQL

Установить PostgreSQL 14 в типовой конфигурации по официальной инструкции, расположенной по адресу: https://www.postgresql.org/download/.

Далее, необходимо создать базу данных с именем *armpsdb* и пользователя с именем *armpsuser*, установить произвольный пароль к ресурсу.

Далее, в поле *Resource name="jdbc/armps*"* конфигурационного файла *armps2.xml* необходимо прописать адрес хоста, имя базы данных, имя пользователя и пароль.

Опционально:

- настроить кластер postgres (master/slave);
- установить pgbouncer.

3.3 Установка и запуск конфигураций RabbitMQ

Установить RabbitMQ в минимальной требуемой версии по официальной инструкции, расположенной по адресу: https://www.rabbitmq.com/#support.

Далее, в конфигурационный файл *armps2.xml* необходимо прописать имя пользователя *armpsuser* и установить произвольный пароль.

3.4 Установка и запуск конфигураций Redis

Установить Redis cluster в минимальной требуемой версии по официальной инструкции, расположенной по адресу: https://redis.io/docs/management/scaling/.

Далее, в значение переменной *Environment name="ru.invitro.armps.redis.cluster.nodes"* конфигурационного файла *armps2.xml* необходимо прописать адреса и порты нод кластера.

3.5 Установка и запуск конфигураций Docker-образов

Предусловия: установка производится на виртуальную машину с ОС Ubuntu Server 20.04 x64.

Предварительно требуется установить docker и docker-compose.

Установить Docker в минимальной требуемой версии по официальной инструкции, расположенной по адресу: https://docs.docker.com/compose/install/.

Образы поставляются в виде tar.gz архивов, которые требуется загрузить в локальный docker-репозиторий.

Код скрипта:

$ ls

armps-armps-v29924.tar.gz

armps-frontend-v30169.tar.gz

В имени каждого файла образа содержится версия его сборки. Например, v29924 - означает сборку с номером 29924.

Чтобы загрузить каждый образ, необходимо для каждого файла образа выполнить команду *docker load -i <имя файла>*.

Код скрипта*:*

docker load -i armps-armps-v29924.tar.gz

docker load -i armps-frontend-v30169.tar.gz

Далее, в локальном репозитории станут доступны образы *$ docker images* в представлении:

armps-frontend, 30169, 543ada2f6be9, 7 days ago, 156MB;

armps-armps, 29924, be6d0a3aafe0, 8 days ago, 1.09GB.

3.6 Установка базовых компонентов системы

3.6.1 Конфигурация серверного компонента системы

Создать директории для конфигурационных файлов и файлов с логами системы.

```
mkdir /etc/armps2
mkdir -p /srv/armps2/temp /srv/armps2/logs
```

Создать файл */srv/armps2/docker-compose.yml* с содержимым:

```
version: "3.4"
services:
  armps:
    image: armps-armps:<version>
    restart: always
    env_file:
      - /etc/armps2/armps2.env
    ports:
      - 8088:8080
    volumes:
      - /etc/armps2:/etc/armps2
      - /srv/armps2/logs:/usr/local/tomcat/logs
      - /srv/armps2/temp:/usr/local/tomcat/temp
      - /srv/armps2/conf/Catalina/localhost:/usr/local/tomcat/conf/Catalina/localhost
```

Создать файл */srv/armps2/docker-compose-migrations.yml* с содержимым:

```
version: "3.4"
services:
  migrations:
```

```
    image: database-updater:<version>
    #env_file:
    #  - /etc/armps2/version.env
    volumes:
      - /etc/armps2:/etc/armps2
      - /srv/armps2/conf/Catalina/localhost:/usr/local/tomcat/conf/Catalina/localhost
```

Создать файл */etc/armps2/armps2.env* с содержимым:

```
ENVIRONMENT_NAME=test
# JAVA_OPTS_MEMORY=-Xms4096m -Xmx35200m
```

Создать файл */etc/armps2/logging.properties* с содержимым:

```
handlers = 1catalina.org.apache.juli.FileHandler,
2localhost.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler
.handlers = 1catalina.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler
############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################
1catalina.org.apache.juli.FileHandler.level = FINE
1catalina.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
1catalina.org.apache.juli.FileHandler.prefix = catalina.
2localhost.org.apache.juli.FileHandler.level = FINE
2localhost.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
2localhost.org.apache.juli.FileHandler.prefix = localhost.
java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
############################################################
# Facility specific properties.
# Provides extra control for each logger.
############################################################
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers =
2localhost.org.apache.juli.FileHandler
# For example, set the com.xyz.foo logger to only log SEVERE
# messages:
#org.apache.catalina.startup.ContextConfig.level = FINE
#org.apache.catalina.startup.HostConfig.level = FINE
#org.apache.catalina.session.ManagerBase.level = FINE
#org.apache.catalina.core.AprLifecycleListener.level=FINE
```

Создать файл */srv/armps2/conf/Catalina/localhost/armps.xml* с содержимым:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/armps">
<Resource name="jdbc/armps"
      auth="Container"
      dataSourceName="armps"
      factory="org.postgresql.ds.common.PGObjectFactory"
      type="org.postgresql.ds.PGSimpleDataSource"
      targetServerType="master"
```

```
        prepareThreshold="0"
        serverName="<postgres host>"
        portNumber="<postgres port>"
        databaseName="<postgres db name>"
        user="<postgres username>"
        password="<postgres password>"
        />
<Resource name="jdbc/armpsRO"
        auth="Container"
        dataSourceName="armps"
        factory="org.postgresql.ds.common.PGObjectFactory"
        type="org.postgresql.ds.PGSimpleDataSource"
        targetServerType="master"
        prepareThreshold="0"
        serverName="<postgres host>"
        portNumber="<postgres port>"
        databaseName="<postgres db name>"
        user="<postgres username>"
        password="<postgres password>"
        />
<Resource name="jdbc/armpsROCache"
  auth="Container"
        dataSourceName="armps"
        factory="org.postgresql.ds.common.PGObjectFactory"
        type="org.postgresql.ds.PGSimpleDataSource"
        targetServerType="master"
        prepareThreshold="0"
        serverName="<postgres host>"
        portNumber="<postgres port>"
        databaseName="<postgres db name>"
        user="<postgres username>"
        password="<postgres password>"
        />
    <Environment name="ru.invitro.armps.integration.loyalty.url"
                value ="<loyalty service url>"
                type="java.lang.String" />
    <Environment
name="ru.invitro.armps.integration.central.impl.general.OrderItemNumberProviderServi
ce.INZ_POOL_SIZE"
                value ="500"
                type="java.lang.Integer" />
    <Loader
loaderClass="org.springframework.instrument.classloading.tomcat.TomcatInstrumenta
bleClassLoader" />
    <Environment name="spring.profiles.active"
                value="armps-online-master-profile"
                type="java.lang.String" />
    <Environment name="ru.invitro.armps.integration.vnd.url"
                value="<vnd service url>"
                type="java.lang.String"/>
```

```
<Environment name="ru.invitro.armps.integration.rmq.hostname"
        value="<rabbitmq hostname>"
        type="java.lang.String"/>
<Environment name="ru.invitro.armps.integration.rmq.username"
        value="<rabbitmq username>"
        type="java.lang.String"/>
<Environment name="ru.invitro.armps.integration.rmq.password"
        value="<rabbitmq password>"
        type="java.lang.String"/>
<Environment name="ru.invitro.armps.integration.lk.url"
        value="<lk2 url>"
        type="java.lang.String"/>
<Environment name="ru.invitro.armps.integration.lk.auth.url"
        value="<lk2 sp url>"
        type="java.lang.String"/>
<Environment name="ru.invitro.armps.integration.corporate.order.url"
        value="<corporate order service url>"
        type="java.lang.String" />
<Environment name="ru.invitro.armps.redis.cluster.nodes"
        value="host1:port1,host2:port2"
        type="java.lang.String" />
<Environment name="ru.invitro.armps.integration.contact.history.url"
        value="<contact history service url>"
        type="java.lang.String"/>
<Environment name="ru.invitro.armps.integration.contact.medical.card.number.url"
        value="<contact medical card number service url>"
        type="java.lang.String" />
<Environment name="ru.invitro.armps.tele.soap.url"
        value="<teleapi service url>"
        type="java.lang.String" />
<Environment name="ru.invitro.armps.integration.loyalty.timeout"
        value ="120"
        type="java.lang.Integer" />
<Environment name="ru.invitro.armps.integration.esb.publication.service.url"
        value="<ESB publish url>"
        type="java.lang.String" />
<Environment name="ru.invitro.armps.integration.esb.visit.publication.service.url"
        value="<ESB visit publish url>"
        type="java.lang.String" />
<Environment name="ru.invitro.armps.integration.discrepancy.url"
        value="http://<discrepancy host>"
        type="java.lang.String" />
<Environment name="ru.invitro.armps.common.security.ldap.url"
        value="ldap://<AD server ip>"
        type="java.lang.String" />
<Environment name="ru.invitro.armps.common.security.ldap.domain" value=""
type="java.lang.String" />
<Environment name="ru.invitro.armps.integration.schedule.widget.url"
        value="https://<doctorwidget booking hostname>/widget"
        type="java.lang.String" />
```

```
<Environment name="ru.invitro.armps.integration.hardware.url"
             value="http://127.0.0.1:8091"
             type="java.lang.String"/>
<Environment name="ru.invitro.armps.integration.lk3.url"
            value="https://<lk3 server hostname>/rest/api"
            type="java.lang.String"/>
<Environment name="ru.invitro.armps.integration.ldap.url"
             value="ldap://<AD server ip>"
             type="java.lang.String" />
<Environment type="java.lang.String"
            name="ru.invitro.armps.integration.ldap.rootDn"
            value="dc=invitro,dc=ru" />
<Environment type="java.lang.String"
            name="ru.invitro.armps.integration.ldap.userDn"
            value="<AD server login>" />
<Environment type="java.lang.String"
            name="ru.invitro.armps.integration.ldap.password"
            value="<AD server password>" />
<Environment type="java.lang.String"
            name="ru.invitro.armps.integration.booking.service.url"
            value="<booking service url>" />
<Environment type="java.lang.String"
            name="ru.invitro.armps.integration.booking.service.login"
            value="<booking service username>" />
<Environment type="java.lang.String"
             name="ru.invitro.armps.logic.blank.url"
             value="<document/blank service url>"/>
<Environment type="java.lang.String"
             name="ru.invitro.armps.logic.blank.publicUrl"
             value="<armps blank service url>"/>
<Environment name="ru.invitro.armps.integration.dp.signature.url"
            value="<digital platform signature service url>"
            type="java.lang.String"/>
<Environment name="ru.invitro.armps.integration.check-print-service.url"
            value="<check print service url>"
            type="java.lang.String"/>
</Context>
```

Применить команду миграции образов:

```
cd /srv/armps2 && docker-compose -f docker-compose-migrations.yml up
```

Запустить приложение командой миграции образов:

```
cd /srv/armps2 && docker-compose up -d
```

Проверить работоспособность сервиса командой:

```
curl http://127.0.0.1:8088/armps/test/ping
```

3.6.2 Конфигурация клиентского компонента системы

Для запуска клиентской составляющей системы (АРМПС2 Frontend) требуется среда кластера Kubernetes с настроенным ingress-контроллером.

Подробности по установке Kubernetes расположены в официальном источнике по адресу: https://kubernetes.io/ru/.

Предварительно необходимо загрузить образ «АРМПС2 Frontend» в приватный репозиторий, доступный из кластера Kubernetes.

Далее, требуется создать файл с именем *frontend-armps.yaml* со следующим содержимым:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/instance: armps-frontend
    app.kubernetes.io/name: armps-frontend
  name: armps-frontend
  namespace: armps
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/instance: armps-frontend
      app.kubernetes.io/name: armps-frontend
  template:
    metadata:
      creationTimestamp: null
      labels:
        app.kubernetes.io/instance: armps-frontend
        app.kubernetes.io/name: armps-frontend
    spec:
      containers:
      - envFrom:
        - configMapRef:
            name: armps-frontend-env
        image: <docker-registry>/armps-frontend:30169
        imagePullPolicy: IfNotPresent
        livenessProbe:
          failureThreshold: 1
          httpGet:
            path: /
            port: http
            scheme: HTTP
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        name: armps-frontend
        ports:
        - containerPort: 80
          name: http
          protocol: TCP
---
kind: ConfigMap
```

```
apiVersion: v1
metadata:
  labels:
    app.kubernetes.io/instance: armps-frontend
    app.kubernetes.io/name: armps-frontend
  name: armps-frontend-env
  namespace: armps
data:
  EXTERNAL_ADAPTER_FRONTEND_NGINX_UPSTREAM: <digital platform adapter
url>
  EXTERNAL_FRONTEND_NGINX_UPSTREAM: <armps backend url>
  EXTERNAL_LOYALTY_FRONTEND_NGINX_UPSTREAM: <loyalty backend url>
  EXTERNAL_SERVICE_CHECK_NGINX_UPSTREAM: <check api url>
---
kind: Ingress
metadata:
  labels:
      app.kubernetes.io/instance: armps-frontend
      app.kubernetes.io/name: armps-frontend
  name: armps-frontend
  namespace: armps
spec:
    ingressClassName: nginx
    rules:
    - host: armps-frontend.<ingress domain>
      http:
        paths:
        - backend:
            service:
              name: armps-frontend
              port:
                number: 80
          path: /
          pathType: ImplementationSpecific
```

Далее необходимо заменить значения в файле в скобках <> на значения, соответствующие среде запуска экземпляра системы.

Далее требуется загрузить «манифест» в Kubernetes, используя команду:

```
$ kubectl apply -f frontend-armps.yaml
```

По завершению описанного алгоритма действий работоспособность системы проверяется в браузере по адресу, указанному в поле *host ingress.*

3.7 Установка внутренних сервисов системы

3.7.1 Конфигурация сервиса по работе с документами в системе

Для запуска внутреннего сервиса системы по формированию документов требуется создать файл */srv/blank-service/docker-compose.yml* с содержанием:

```
version: "3"
services:
```

```
  blank-service:
    image: "docker-registry.invitro.ru/development/invitro-java/invitro-armps-blank-service/armps-blank-api:28371"
    restart: unless-stopped
    environment:
      - JAVA_OPTS=-Dfile.encoding=UTF-8 -Dspring.config.location=/etc/subscriber-service/application.yaml -Xms1024m -Xmx3072m
      - LOG_FILE=/var/log/blank-service.log
    volumes:
      - ./log:/var/log
      - ./config:/etc/subscriber-service
    ports:
      - "8093:8080"
```

Далее требуется прописать маршрутизацию:

```
# https://github.com/docker/compose/issues/4336
networks:
  default:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet:  192.168.200.1/24
```

По завершению описанного алгоритма необходимо выполнить команду docker-compose up-d.

3.7.2 Конфигурация сервиса по работе с цифровой платформой в системе

Для запуска внутреннего сервиса системы по взаимодействию с цифровой платформой для наложения простой электронной подписи на документы, генерируемые в системе, требуется создать файл *dp-signature-adapter.yaml* следующего содержания:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dp-signature-adapter-extra
  labels:

    helm.sh/chart: dp-signature-adapter-0.1.31386
    app.kubernetes.io/name: dp-signature-adapter
    app.kubernetes.io/instance: dp-signature-adapter
    app.kubernetes.io/version: "31365"
    app.kubernetes.io/managed-by: Helm
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dp-signature-adapter
  labels:
    helm.sh/chart: dp-signature-adapter-0.1.31386
```

```
    app.kubernetes.io/name: dp-signature-adapter
    app.kubernetes.io/instance: dp-signature-adapter
    app.kubernetes.io/version: "31365"
    app.kubernetes.io/managed-by: Helm
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: dp-signature-adapter
      app.kubernetes.io/instance: dp-signature-adapter
  template:
    metadata:
      labels:
        app.kubernetes.io/name: dp-signature-adapter
        app.kubernetes.io/instance: dp-signature-adapter
    spec:
      imagePullSecrets:
        - name: global-docker-registry-secret
      serviceAccountName: dp-signature-adapter
      securityContext:
        {}
      terminationGracePeriodSeconds: 420
      volumes:
        - name: config-extra
          configMap:
            name: dp-signature-adapter-extra

      containers:
        - name: dp-signature-adapter
          securityContext:
            {}
          image: "<docker-registry>invitro-dp-signature-adapter-service/invitro-dp-
signature-adapter-service:31365"
          imagePullPolicy: IfNotPresent
          envFrom:
            - configMapRef:
                name: dp-signature-adapter-env
          volumeMounts:
            - mountPath: /etc/config-extra
              name: config-extra

          ports:
            - name: http
              containerPort: 8080
              protocol: TCP
            - name: management
              containerPort: 8090
              protocol: TCP

          livenessProbe:
```

```
        httpGet:
          path: /actuator/health/liveness
          port: management
        failureThreshold: 5
        periodSeconds: 10
      startupProbe:
        httpGet:
          path: /actuator/health/liveness
          port: management
        failureThreshold: 30
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /actuator/health/readiness
          port: management
      resources:
        limits:
          cpu: 1000m
          memory: 2Gi
        requests:
          cpu: 500m
          memory: 1Gi
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dp-signature-adapter
  labels:
    helm.sh/chart: dp-signature-adapter-0.1.31386
    app.kubernetes.io/name: dp-signature-adapter
    app.kubernetes.io/instance: dp-signature-adapter
    app.kubernetes.io/version: "31365"
    app.kubernetes.io/managed-by: Helm
  annotations:
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "1"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "600"
spec:
  ingressClassName: nginx
  rules:
    - host: "dp-signature-adapter.arm.invitro-dev.k8s"
      http:
        paths:
          - path: /
            pathType: ImplementationSpecific
            backend:
              service:
                name: dp-signature-adapter
                port:
                  number: 8080
```

Далее требуется заменить значения в файле в скобках <> на значения, соответствующие среде запуска экземпляра системы.

Далее требуется загрузить «манифест» в Kubernetes командой:

$ kubectl apply -f dp-signature-adapter.yaml

По завершению описанного алгоритма действий работоспособность компонента системы проверяется в браузере по адресу, указанному в поле *host ingress.*

3.7.3 Конфигурация сервиса по работе с отчетами в системе

Для запуска внутреннего сервиса системы по формированию отчетов необходимо создать файл *report-service.yaml* следующего содержания:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: armps-report
  labels:
    helm.sh/chart: armps-report-0.1.15011
    app.kubernetes.io/name: armps-report
    app.kubernetes.io/instance: armps-report
    app.kubernetes.io/version: "15009"
    app.kubernetes.io/managed-by: Helm
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: armps-report
      app.kubernetes.io/instance: armps-report
  template:
    metadata:
      labels:
        app.kubernetes.io/name: armps-report
        app.kubernetes.io/instance: armps-report
    spec:
      imagePullSecrets:
        - name: global-docker-registry-secret
      serviceAccountName: armps-report
      securityContext:
        {}
      terminationGracePeriodSeconds: 420
      volumes:
        - name: config-extra
          configMap:
            name: armps-report-extra

      containers:
        - name: armps-report
          securityContext:
            {}
          image: "<registry>/invitro-armps-report-service:15009"
```

```
        imagePullPolicy: IfNotPresent
        envFrom:
          - configMapRef:
              name: armps-report-env
        volumeMounts:
          - mountPath: /etc/config-extra
            name: config-extra

        ports:
          - name: http
            containerPort: 8080
            protocol: TCP
          - name: management
            containerPort: 8090
            protocol: TCP

        livenessProbe:
          httpGet:
            path: /actuator/health/liveness
            port: management
          failureThreshold: 5
          periodSeconds: 10
        startupProbe:
          httpGet:
            path: /actuator/health/liveness
            port: management
          failureThreshold: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /actuator/health/readiness
            port: management
        resources:
          limits:
            cpu: 500m
            memory: 1Gi
          requests:
            cpu: 500m
            memory: 1Gi
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: armps-report
  labels:
    helm.sh/chart: armps-report-0.1.15011
    app.kubernetes.io/name: armps-report
    app.kubernetes.io/instance: armps-report
    app.kubernetes.io/version: "15009"
    app.kubernetes.io/managed-by: Helm
```

```
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/proxy-connect-timeout: "1"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "600"
spec:
  rules:
    - host: "report.arm.invitro-dev.k8s"
      http:
        paths:
          - path: /
            backend:
              serviceName: armps-report
              servicePort: 8080
```

Далее требуется заменить значения в файле в скобках <> на значения, соответствующие среде запуска экземпляра системы.

Далее требуется загрузить «манифест» в Kubernetes командой:

```
$ kubectl apply -f report-service.yaml
```

По завершению описанного алгоритма действий работоспособность компонента системы проверяется в браузере по адресу, указанному в поле *host ingress.*

3.8 Установка INVITRO Hardware

На рабочую станцию пользователя требуется установить вспомогательное программное обеспечение системы HWS, необходимое для взаимодействия с локальным оборудованием (касса, платежные терминалы, термо-принтеры и принтеры А4). Программное обеспечение устанавливается локально на компьютер в медицинском офисе.

Информацию по установке HWS приведена в отдельном файле.

4 Список сокращений

Сокращение	Описание
АРМПС2	Автоматизированное рабочее место процедурной сестры, система
ПО	Программное обеспечение
ОС	Операционная система
БД	База данных
ПК	Персональный компьютер