Оглавление

[ВВЕДЕНИЕ 2](#_Toc483785226)

[ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 4](#_Toc483785227)

[1.1. Общая информация о проекте 5](#_Toc483785228)

[1.1.1. Описание проекта 5](#_Toc483785229)

[1.1.2. Описание основных компонентов и библиотек 5](#_Toc483785230)

[1.1.3. Инверсия контроля и внедрение зависимостей 6](#_Toc483785231)

[1.2. Организация ввода-вывода и программного интерфейса 7](#_Toc483785232)

[1.2.1. Входные и выходные данные 7](#_Toc483785233)

[1.2.2. Взаимодействие с базой данных 8](#_Toc483785234)

[1.2.3. Взаимодействие с сервером исходящей почты 9](#_Toc483785235)

[1.2.4. Структурная схема взаимодействия с API 9](#_Toc483785236)

[2. ПРАКТИЧЕСКАЯ ЧАСТЬ 11](#_Toc483785237)

[2.1. Обоснование выбора языка программирования 12](#_Toc483785238)

[2.2. Выбор стиля, методов и средств программирования 12](#_Toc483785239)

[2.3. Описание программы 14](#_Toc483785240)

[2.3.1. Общие сведения 14](#_Toc483785241)

[2.3.2. Функциональное назначение программы 14](#_Toc483785242)

[2.3.3. Описание логической структуры 15](#_Toc483785243)

# ВВЕДЕНИЕ

Проблема образования в нашем мире является одной из самой обсуждаемых. Многие родители хотят, чтобы их дети получили релевантное образование, но не знают, каким образом можно повлиять на ребенка и стимулировать его к обучению.

Последнее и было главной задумкой при создании дипломного проекта. Стимулирование через интересные задания, красивое оформление и оплату мобильного телефона должны привлечь детей к обучению, а автоматизация проверки и ведение статистики позволят родителям наблюдать за успехами детей не тратя время на проверку и составление тестов.

Конечно, для удобства задания распределены по темам, а темы по предметам. В свою очередь у каждого предмета имеется атрибут «Класс» («schoolclass»). Этот атрибут был введен для регулирования сложности заданий и тем относительно друг друга. На данный момент программа содержит предметы для 11 различных классов, что соответствует российской учебной программе вплоть до полного среднего образования. Несмотря на всё это диапазон классов не является статичным и может быть расширен в случае, например, введения программы в пользование в стране с иной системой образования

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Общая информация о проекте

### Описание проекта

Проект представляет из себя распределенное приложение, написанное на языке Java в соответствии со спецификацией Java Platform Enterprise Edition (Java EE) и запускаемое на сервере приложений Jboss (Wildfly), или любом другом, поддерживающем спецификацию Java EE в качестве серверной части (backend).

В качестве клиентской части (frontend) может выступать любое клиентское приложение, которое поддерживает обмен данными по протоколу HTTP и способно взаимодействовать в соответствии с архитектурным стилем REST.

### Описание основных компонентов и библиотек

Spring Framework – универсальный фреймворк для Java-платформы, интегрированный для упрощения реалиации инверсии управления (IoC), упрощения реализации паттерна программирования MVC (Spring MVC), упрощения доступа к данным (Spring Data), упрощения организации информационной безопасности (Spring Security).

Apache Maven – фреймворк для автоматизации сборки проекта на основе описания его структуры на языке POM (Project Object Model) в файле pom.xml

Log4j – библиотека журналирования (logging) Java-программ.

Java Platform Enterprise Edition (Java EE) API – набор различных программных интерфейсов, соответствующих спецификации Java EE, таких как JPA, Validation, JavaMail и других

Hibernate – реализация Java Persistence API. Является технологией ORM.

FasterXML/Jackson – библиотека для представления объектов в качестве JSON/XML строки или файла.

Apache Velocity – библиотека для создания шаблонов на языке Velocity Template Language.

OkHttp – библиотека для создания HTTP-клиентов и выполнения HTTP-запросов.

### Инверсия контроля и внедрение зависимостей

Внедрение зависимостей (Dependency Injection, DI) – одна из реализаций инверсии контроля (Inversion of Control, IoC) – процесс предоставления внешней зависимости программному компоненту.

Проще говоря, построение объектов производится внешним механизмом, который исходя из настроек построения внедряет те, или иные зависимости в соответствующие компоненты.

По такому принципу, например, работает Spring Framework. При развертывании приложения существует механизм, который исходя из XML-контекста, или аннотаций в Java-коде создает компоненты, содержащие ссылки на другие компоненты (внедрение зависимостей) и использующие их для реализации собственного функционала. См. Рис 1 и 2.

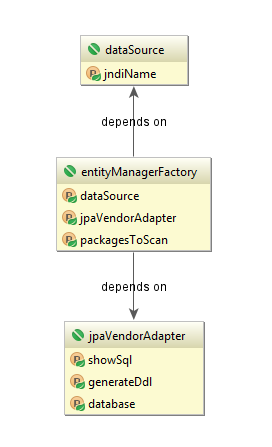
Рис. 1. Простейший пример внедрения зависимостей

Рис. 2. То же самое в схематичном виде

## Организация ввода-вывода и программного интерфейса

### Входные и выходные данные

Входными данными приложения являются HTTP-запросы, а выходные, соответственно, HTTP-ответы в соответствии со спецификацией RFC 2616 HTTP/1.1.

Внимание! Для использования кросс-доменных запросов убедитесь в том, что приложение настроено для их принятия и дает ответ с соответствующими заголовками ответа (response headers) на предварительный запрос методом OPTIONS.

Приложение может использовать в качестве данных для обработки такие части запроса, как переменные путей, параметры запроса (path variables и request parameters, представляющие из себя query string) и тело запроса (request body).

Сервер приложений принимает строку, определяет контекст к которому сделан запрос, соотносит это с приложением, которое находится в этом контексте и направляет его соответствующему сервлету (сервлеты объявлены в web.xml). Данное приложение построено на основе Spring MVC Framework, так что в данном случае управление по выбору обработчика обычно предоставляют сервлету dispathcerServlet (класс org.springframework.web.servlet.DispatcherServlet) см. Рис 3.

DispatcherServlet

Request

Response

Handler Mapping

RestController

Рис 3. Схема обработки запроса внутри приложения Spring MVC

Как можно увидеть из вышеприведенной схемы, DispatcherServlet является посредником между приложением основанном на технологии Spring MVC и сервером и занимается решением задач связанных с обработкой запроса. Так, например, в случае нашего приложения, где каждая функция обработки помечена аннотацией RestController, автоматически результат функции будет помечен как ResponseBody, что является поводом для преобразования возвращаемого объекта в JSON библиотекой jackson (или в любой другой медиа-тип, обозначенный в параметре produces аннотации RequestMapping). Если же не будет найдено аннотации RestController у класса, или ResponseBody у результата функции, то DispatcherServlet, по умолчанию действующий в соответствии с шаблоном MVC постарается обратиться к стандартной реализации ViewResolver и найти представление, сооответствующее возвращаемому String, Model, или ModelAndView значению. Как можно заметить из описания, сервлет-диспетчер действительно говорящее название для данного класса.

### Взаимодействие с базой данных

Приложение по умолчанию пытается найти базу данных в контексте ресурсов сервера по имени «java:/PostgresDS» (технология JNDI). Для того чтобы изменить способ подключения необходимо в конфигурационном файле «applicationContext.xml» указать для параметра «jndiName» бина (bean) «dataSource» иное JNDI имя, или создать новый бин класса, реализующего интерфейс DataSource, сконфигурировать его и добавить в качестве значения свойства «dataSource» в «entityManagerFactory».

Для взаимодействия с базой данных используется технология ORM (Object-Relation Mapping – объектно-реляционное отображение), позволяющая работать с записями в базе данных как с объектами Java. Для работы с базой данных по этой технологии в контексте данного приложения используется JPA (Java Persistence API) и реализация этой спецификации Hibernate.

Spring Data JPA – часть Spring Framework отвечающая за удобное взаимодействие с базой данных с помощью JPA. Spring Data JPA позволяет быстро и легко создать репозитории со стандартными CRUD-операциями. Для этого необходимо создать интерфейс и расширить с помощью него интерфейс JpaRepository, указав при этом класс сущности, с которой будет произоводиться взаимодействие и тип первичного ключа (Id). Кроме того, можно описывать собственные специфичные запросы к БД различными способами, такими как:

* Описание запроса в названии функции в интерфейсе
* Описание запроса на языке HQL в аннотации Query
* Описание запроса на языке SQL в аннотации Query (для этого необходимо установить значение аттрибута nativeQuery как true)

### Взаимодействие с сервером исходящей почты

Взаимодействие приложения с сервером исходящей почты производится по протоколу SMTP с помощью реализации интерфейса JavaMailSender – класса org.springframework.mail.javamail.JavaMailSenderImpl.

По умолчанию используется бин с идентификатором «mailSender», настроенный на взаимодействие с почтовым сервером Gmail (smtp.gmail.com). Изменить это и другие настройки можно в конфигурационном файле приложения «applicationContext.xml».

### Структурная схема взаимодействия с API

Сервер определяет приложение, которому необходимо направить запрос исходя из строки запроса, в то время как приложение, принимающее запрос проводит его через слои. Каждый слой может обратиться к слою под собой, или вернуть результат в случае ошибки во времени выполнения.

Запрос

GET localhost/api/myapi

Приложение 2 [/demo]

Приложение 1 [/api]

Контекст развертывания

Контроллер

Сервисный слой

Доменный слой

Слой базы данных

Сервер приложений

[localhost]

Ответ

{

“message” : “Hello world!”

}

Рис 4. Схематичное описание взаимодействия с API

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## Обоснование выбора языка программирования

При выборе языка программирования были учтены следующие факторы:

* + - Статическая типизация – многие ошибки исключаются на стадии компиляции
    - Скорость
    - Кроссплатформенность – переносимость на другие платформы

Таким образом если для первого и второго пункта подходила платформа .NET с языком C# и Java со спецификацией Java EE (различные PHP и подобные были сразу же исключены), то для третьего пункта победителем стал язык Java.

Кроме того, на фоне иных малоизвестных языков программирования Java обладает огромным сообществом, что, несомненно, положительно влияет на скорость решения проблем при разработке приложений.

## Выбор стиля, методов и средств программирования

При выборе стиля и методов для проекта было необходимо учесть следующие критерии:

* + - Слабые связи между компонентами
    - Разделение компонентов по функциональному назначению
    - Сопровождаемость

В итоге для реализации слабых связей между компонентами был выбран Spring Framework и реализация инверсии контроля – внедрение зависимостей. Интерфейсы языка Java также сослужили отличную службу для реализации слабых связей. Таким образом, для разрыва сильной связи класс-класс (когда один класс использует методы другого конкретного класса) между ними вводятся интерфейсы. См. Рис. 5 и Рис. 6.

**class UserService**

-addUser(String username, password)

**class UserController**

-addUser(User user)

Рис. 5. Сильные связи между двумя классами

Рис. 6. Сильная (прямая) зависимость разорвана интерфейсом

**class UserService** *implements IUserService*

-addUser(String username, password)

**class UserController**

-addUser(User user)

*interface IUserService*

-addUser(String username, password)

Как видно из схемы на Рис. 5. Класс UserController напрямую вызывает класс UserService, что означает, что класс UserController сильно связан с классом UserService, к которому он обращается и подмена последнего в определенной степени вызовет проблему при интеграции его в класс UserController (особенно, если новая реализация будет нести в себе иные названия функций, аргументы методов и др.).

Для того, чтобы этого избежать было принято решение разорвать зависимость между классами, используя интерфейс IUserService, как видно на Рис. 6. Таким образом, класс UserController не будет знать о том, как именно будет выполняться то, или иное дейсвие – он просто вызовет необходимую функцию интерфейса, не беспокоясь о том, какой именно класс содержит реализацию. Интерфейс в данном контексте служит для того, чтобы определить некоторое соглашение, которое будет выполнять класс, который служит его реализацией. То есть реализуя определенный интерфейс, класс *обязан* реализовать все функции этого интерфейса. Этот механизм взаимодействия позволяет разрывать связи между классами и легко подменять компоненты.

Для разделения компонентов по функциональному назначению был использован не только паттерн (шаблон программирования) Модель-Представление-Контроллер (MVC), но и другие, такие как DTO (Объект передачи данных), DAO (Объект доступа к данным), ValueObject (Объект-значение), Делегаты, POJO (Старый добрый объект Java).

Это, в свою очередь, позволило разделить программу на своеобразные «слои», как показано на Рис.4 (пункт 1.2.4). Такой подход позволяет «наращивать» слои сверху, или снизу в то же время не ограничивая компоненты игнорировать некоторые из них (как например это сделано в API для менеджера содержимого в приложении – контроллер обращается напрямую к DAO, игнорируя сервис-слой).

Сопровождаемость проекта обусловлена общеизвестными структурами и паттернами программирования, подобно вышеописанным DAO, MVC и другими. Кроме того, инверсия контроля позволяет запросто подменять зависимости в компонентах, которые были описаны в XML-файлах. Но конфигурация приложения выполнена не только с помощью XML, но и с помощью аннотаций, которые присутствуют в самом Java-коде. При выборе способа конфигурации приложения был сделан упор на семантическую логичность конфигурации – те компоненты, которые являются основополагающими функциональному назначению приложения были сконфигурированы аннотациями (такими как Autowired, Qualifier), те же, которые могут быть подменены (например email и sms-сервисы),

## Описание программы

### Общие сведения

Программа представляет из себя серверное приложение в некотором адресном контексте хоста. Приложение может отвечать на запросы, переданные в его контекст, при этом совершая определенные операции с данными (выполняя бизнес-логику) и взаимодействуя с иными серверами в сети с помощью программного интерфейса (API).

### Функциональное назначение программы

Как было описано раннее, программа имеет определенный программный интерфейс в некотором адресном контексте сервера, на который оно загружено. С помощью этого интерфейса клиентские приложения (например представление, которое реализует запросно-ответный функционал по технологии REST) «общаются» с приложением.

Функциональное назначение программы, как единого целого, представляет из себя механизм обработки запросов таких приложений для реализации функционального назначения проекта.

Функциональное назначение проекта заключается в реализации нескольких разделов функционала разделенного по ролям пользователя, который будет использовать приложение.

Существуют следующие роли и закрепленный за ними функционал:

* + - Администратор – приложение реализует административный функционал, такой, например как, менеджмент учетных записей пользователя, ответы на сообщения обратной связи и др.
    - Менеджер содержимого (Контент-менеджер) – приложение реализует функционал для управления содержимым сайта (CMS), например, добавление и редактирование вопросов, тем, и предметов.
    - Родитель – приложение реализует функционал управления профилем пользователя, профилями детей, которых этот пользователь зарегистрировал, функционал пополнения счёта и просмотра статистики.
    - Ребенок – приложение реализует функционал выполнения заданий, получения результата выполнения и наград.
    - Анонимный пользователь – приложение реализует функционал авторизации, аутентификации и обратной связи.

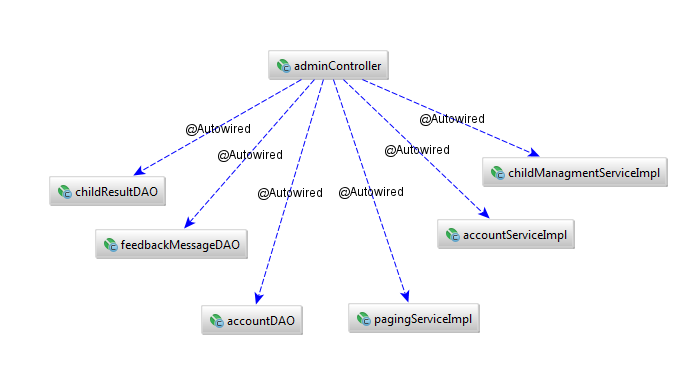
### Описание логической структуры

Описание структуры взаимодействия с API было приведено в разделе 1.2.4.

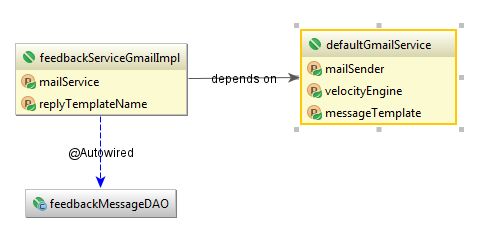
Как было описано выше, точкой вхождения запроса в контекст приложения является dispatcherServlet, который передает управление одному из контроллеров:

* + - AdminController
    - ChildController
    - ContentManagerController
    - ExceptionHandlingController
    - HomeController
    - MoneyController
    - ParentController

Контроллеры представляют из себя классы-синглтоны, обрабатывающие запрос и делегирующие его параметры (как правило) сервисам (см. Рис. 7).

Рис. 7 Контроллер «adminController» взаимодействует с сервисами и объектами доступа к данным

Сервисы, представляющие из себя объекты-обработчики данных, в свою очередь, могут так же взаимодействовать с другими сервисами и DAO. Таким образом, можно сказать, что вся бизнес-логика приложения находится в его сервис-слое (см Рис. 8).

Рис. 8 Сервис «feedbackServiceGmailImpl» взаимодействует с сервисом «defaultGmailService» и объектом доступа к данным «feedbackMessageDAO»

Объекты доступа к данным (DAO) представляют из себя не конкретную реализацию, а интерфейс, на основе названий функций (или аннотаций) которого Spring Framework создает соответствующие реализации при развертывании приложения. Это стало возможным благодаря фреймворку Spring Data JPA.

Модели, которыми оперируют сервисы, контроллеры и DAO делятся на три вида:

* + - Entity – сущности, необходимы для объектно-ориентированного взаимодействия с базой данных. Представляют из себя POJO.
    - DTO – объект передачи данных, необходимы для компоновки и урезания данных. Такие данные возвращаются сервером в виде ответа.
    - ValueObject – объект-значение. Необходим для удобного принятия данных (в том числе с форм) и валидации их значений.

Если с сущностями всё просто и понятно, то способы формирования и использования в приложении DTO и ValueObject могут показаться нестандартными.

В приложении DTO также встречаются двух видов: статичные и формируемые.

Статичные представляют из себя класс, который заполняется значениями с помощью функций или конструктора и возвращается в виде отображения в заданном формате.

Формируемые DTO представляют из себя некоторый результат выполнения функции getDTO() некоторой реализации интерфейса DTOFormingService. Ниже приведена схема формирования DTO с помощью реализации интерфейса DTOFormingServiceHandler – DTOFormingServiceHandlerImpl (см. Рис. 9).

Рис. 9. Схема сервиса формирования DTO и делегата-руководителя

**class DTOFormingServiceHandlerImpl<T>** *implements DTOFormingServiceHandler<T>*

-Map<Class, DTOFormingService> serviceMap

-Object getDTO(T obj)

-Object getCollection(Collection<T> obj)

-Object getPage(Page<T> page)

**class DTOFormingServiceImplA** *implements DTOFormingService<Object,SomeObjA>*

-Object getDTO(SomeObjA obj)

**class DTOFormingServiceImplB** *implements DTOFormingService<Object,SomeObjB>*

-Object getDTO(SomeObjB obj)

Реализация DTOFormingServiceHandlerImpl содержит в себе карту ключ-значение serviceMap, в которой ключ – это объект типа Class, а значение – ссылка на реализацию интерфейса DTOFormingService и использует рефлексию Java для принятия решения, которому известному сервису-обработчику отдать объект, по сути, играя роль делегата.

Так как единственное назначение DTO – быть преобразованным с помощью библиотеки jackson и отосланным куда-либо, и из-за того, что данная библиотека использует рефлексию Java для реализации маппинга, данную конструкцию можно использовать, несмотря на то, что она использует raw-данные во время возвращения их из DTOFormingServiceImpl. В случае, если во время выполнения сервис получит в качестве аргумента объект, который он не сможет опознать, обработка завершится исключение SchemeNotFoundException. Схемы (объекты реализующие DTOFormingService) можно настроить в файле dtoContext.xml

В данном приложении объекты-значения (ValueObject) используются для проверки присланных значений из предполагаемых форм с помощью библиотеки javax.validation и аннотации Spring @Validated, что позволяет привязывать валидацию к форме, а не к сущности, хранимой в БД. Польза от такого использования наглядно представлена в данном приложении:

Существуют два объекта-значения ParentAccountRegistrationValueObject и ChildAccountRegistrationValueObject. В каждом из них реализована своя проверка введенных значений, но, тем не менее, для сохранения в базу данных используется одна и та же сущность Account.

Во время выполнения приложения могут случиться непредвиденные обстоятельства (исключения). Такие исключения называют «RuntimeException», или «Исключение во время выполнения». Такие исключения принято решать, или пробрасывать в вызывающую функцию. При написании данного приложения это было использовано для проверки данных и ответа в случае возникновения каких-либо проблем.

В приложении, как уже было описано выше, существует класс ExceptionHandlerController, функционал которого добавлен во все остальные контроллеры. Этот класс реализует обработку ошибок во время выполнения. Как правило, для большинства ошибок, этот механизм записывает их описание в объект MessageDTO и возвращает его в качестве ответа на запрос.

Кроме всего, в приложении существует контекст безопасности, описанный в разделе 3 «Информационная безопасность».

### Используемые технические средства

Для развертывания данного приложения необходим сервер приложений, поддерживающий спецификацию Java EE. Рекомендуемый выбор: Wildfly 10.

Кроме того, так как приложение взаимодействует с сервисами в сети Интернет необходим выделенный IP адрес.

Также для взаимодействия с сервисом Яндекс.Касса необходимо пробросить внешний порт 80 к серверному порту (8080 по умолчанию) и домен popolnyalka.com с сертификатом безопасности.

### Входные данные

Как было описано выше, входные данные представляют из себя HTTP-запросы. В контексте приложения запросы проходят через ряд обозначенных в нем фильтров и их содержимое или параметры получают представление в виде объектов, или примитивных значений, которые в дальнейшем обрабатываются.

### Выходные данные

Выходными данными приложения являются HTTP-ответы, запросы к СУБД, SMTP серверам и запросы к сервисам в сети.

Содержимое тела HTTP ответов в данном приложении сводится к JSON-строке, сформированной из одного из объектов классов MessageDTO или PageImpl. Ниже приведены поля и описание значений свойств данных объектов.

MessageDTO представляет из себя объект, содержащий следующие поля:

* + - errors – карта ключ-значение – карта, содержимым которой являются ошибки для программиста клиентской части проекта
    - messages – массив (список) сообщений для пользователя. Сообщения формируются исходя из ключа исключения. Описание всех ключей и значений, закрепленных за ними представлено в файле /resources/messages/messages.properties
    - messageType – перечисление, определяющее статус ответа (SUCCESS/ERROR на момент написания). Необходимо для определения статуса ответа. Является перечислением, а не, например, булевой переменной, в свете возможного добавления собственных статусов (WARN, INFO и т. д.).
    - content – может содержать любой объект

PageImpl представляет из себя объект, содержащий массив некоторых объектов и необходим для удобной реализации пейджинга (paging). Содержит следующие поля:

* + - content – массив объектов
    - first – булево значение, верно, если страница первая, в остальных случаях – ложно
    - last – булево значение, верно, если страница последняя, в остальных случаях – ложно
    - number – номер страницы
    - numberOfElements – количество элементов на странице
    - totalElements – количество элементов всего
    - totalPages – количество страниц
    - sort – данные сортировки

Запросы к различным сервисам в Сети зависят от реализации сервиса. Например, к Яндекс.Кассе запросы создаются ввиде XML, к SMS-сервису sms.ru – с помощью параметров запроса, а к SMTP серверам – по протоколу SMTP.

К базе данных запросы делаются в соответствии с конфигурацией приложения. По умолчанию – на диалекте Postgresql. За формирование запросов к БД отвечают библиотеки JPA, ее реализация Hibernate, сами запросы формирует Spring Data JPA, а информация об источнике данных находится на сервере под JNDI-именем «java:/PostgresDS».