

Федеральное государственное бюджетное образовательное учреждение  
высшего образования



«Московский государственный технический университет  
им. Н.Э. Баумана (национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ – Информатика и управления  
КАФЕДРА – Информационные системы и телекоммуникации

## **РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе на тему

Портирование веб-сервиса NotificationManager и  
компонента пользовательского интерфейса системы Traccar на OSGi сервис и портлет  
платформы Liferay с сохранением протокола взаимодействия клиента с сервером

Студент группы ИУЗ-73

(подпись)

А.Ю. Арапов

Руководитель курсовой работы

(подпись)

А. М. Иванов

Москва, 2017

## Содержание

1	Исследовательская часть .....	4
1.1	Обзор предметной области .....	4
1.2	Разрабатываемая система как «черный ящик» в окружении внешних систем .....	4
1.3	Техническое задание .....	6
1.4	Интересы заинтересованных сторон по отношению к системе .....	7
2	Конструкторская часть .....	8
2.1	Технические решения, позволяющие удовлетворить интересы заинтересованных сторон .....	8
2.2	Высокоуровневая архитектура системы .....	9
2.2.1	Компонентная декомпозиция системы .....	9
2.2.2	Модульная декомпозиция системы .....	11
2.3	Детальная архитектура системы.....	12
2.3.1	Файловая структура репозитория системы в GitHub .....	12
2.3.2	Плагины и классы системы .....	13
3	Технологическая часть .....	21
3.1	Особенности создания бинарной сборки системы .....	21
3.2	Особенности запуска разработанной системы.....	22
3.3	Анализ реализации системы .....	22
3.3.1	Анализ исходного кода с помощью метрик качества .....	22
3.3.2	Анализ зависимостей в коде системы .....	26
3.4	Тестирование на корректность работы .....	28
3.4.1	Проверка NotificationManager .....	29
3.4.2	Корректный ввод данных и добавление нового уведомления.....	32
4	Выводы .....	34
	Список используемой литературы .....	35

# 1 Исследовательская часть

## 1.1 Обзор предметной области

Портлет — подключаемый, сменный компонент пользовательского интерфейса веб-портала (элемент веб-страницы). Портлет выдаёт фрагменты разметки, которые встраиваются в страницу портала.

OSGi (Open Services Gateway Initiative) — спецификация динамической модульной системы и сервисной платформы для Java-приложений, разрабатываемая консорциумом OSGi Alliance.

Liferay Portal — программный продукт, представляющий собой корпоративный портал, то есть решение, предназначенное для централизованного доступа к нескольким различным корпоративным приложениям в одном месте. Liferay иногда описывается как система управления содержимым (CMS) или платформу для веб-приложений. Написан на языке Java и распространяется под двумя видами лицензий: свободной и проприетарной, используя бизнес-модель двойного лицензирования.

Traccar – программное обеспечение для отслеживания GPS.

## 1.2 Разрабатываемая система как «черный ящик» в окружении внешних систем

Перед тем, как приступить к разработке и реализации системы, необходимо определить ее общую структуру, связи между компонентами, функционирование и определить, что необходимо разработать, а что будет относиться к внешним компонентам. Для этого необходимо рассмотреть систему как «черный ящик» в окружении внешних смежных систем (см. рисунок 1) и определить общие связи и принципы функционирования.

Портируемый веб-сервис NotificationManager предназначен для добавления, удаления и изменения информации об уведомлениях в базе данных GPS-трекера Трассар. Основными физическими компонентами являются устройство пользователя, обеспечивающее веб-интерфейс взаимодействия, обрабатывающий сервер, который реализует доступ к базе данных, и сетевое пространство между ними. Основной механизм работы сервиса заключается в следующем: пользователь с помощью предоставленного интерфейса своими действиями осуществляет ввод необходимых данных для добавления или удаления информации об уведомлении. Все изменения сохраняются на сервере, кроме того, по запросу сервер показывает список всех текущих уведомлений.

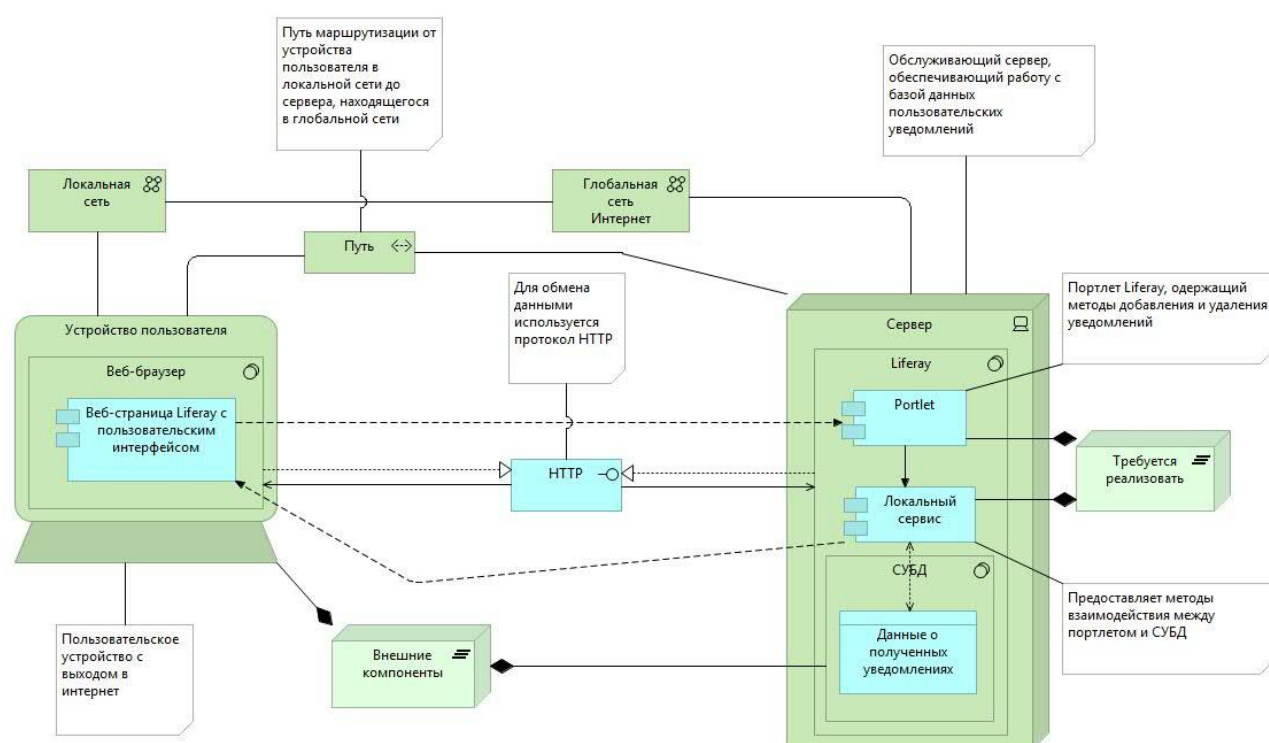


Рисунок 1 - Система в окружении смежных систем

### 1.3 Техническое задание

Портирование веб-сервиса NotificationManager и компонента пользовательского интерфейса системы Traccar на OSGI сервис и портлет платформы Liferay с сохранением протокола взаимодействия клиента с сервером.

Тема индивидуальных заданий:

- изучить соответствующий Manager и его графический интерфейс в Traccar;
- спроектировать интерфейс компонента;
- реализовать хранение данных в БД (функционал должен быть инкапсулирован);
- разделение модели данных и бизнес логики; провести тестирование;
- описать требования, конструкцию, особенности сборки и запуска в документации;
- реализовать визуализацию данных в GUI;
- обработка событий GUI и отправка команд;
- использование CSS стилей и шаблонов.

#### 1.4 Интересы заинтересованных сторон по отношению к системе

В таблице ниже представлены результаты выявления и начального анализа заинтересованных сторон (ЗС) и их интересов по отношению к системе.

Таблица 1- Заинтересованные стороны и их интересы по отношению к системе

Заинтересованные стороны	Интересы заинтересованных сторон
Пользователь	П1 Удобное отображение всех полученных уведомлений.  П2 Удобное взаимодействие пользователя с системой
Разработчик (новый разработчик)	P1 Использование распространенных средств разработки.
Владелец проекта (project owner)	В1 Быстрая и полная передача исходного кода, настроек, документов.  В2 Возможность в дальнейшем расширять систему, например, добавляя методы изменения полей уведомлений, добавленных в базу данных

## 2 Конструкторская часть

### 2.1 Технические решения, позволяющие удовлетворить интересы заинтересованных сторон

В таблице ниже представлены результаты выбора технических решений, позволяющие удовлетворить интересы заинтересованных сторон по отношению сторон.

Таблица 2 - Технические решения, удовлетворяющие интересам ЗС

Интересы заинтересованных сторон	Технические решения
П1 Удобное отображение всех уведомлений. П2 Удобное взаимодействие пользователя с системой.	Визуализатор данных из БД. Эргономичный интерфейс реализованных функций взаимодействия с базой данных уведомлений.
Р1 Использование распространенных средств разработки.	Хорошо документированный код, и наличие файлов конфигурации.
В1 Быстрая и полная передача исходного кода, настроек, документов.  В2 Возможность в дальнейшем расширять систему, например, добавляя методы изменения полей уведомлений, добавленных в базу данных.	Код и настройки разрабатываемой системы будут находиться в системе GitHub. Контроль версий будет производиться с использованием системы Git. Использование Gradle как инструмент сборки. Использование Gradle как инструмент управления зависимостями для расширяемости. Для обеспечения расширяемости код разрабатываемой системы будет разбит на модули, зависимости между которыми будут только через стандартные интерфейсы взаимодействия с БД.

## 2.2 Высокоуровневая архитектура системы

### 2.2.1 Компонентная декомпозиция системы

Разрабатываемая система состоит из следующих основных компонентов:

- NotificationPersistenceImpl – компонент, обеспечивающий прямой доступ к базе данных для методов CRUD (create, read, update, delete —четыре базовые функции, используемые при работе с персистентными хранилищами данных);
- NotificationLocalServiceImpl – реализация локального сервиса – один из главных классов в проекте, определяющий функционал работы серверной части. В этом классе можно добавлять пользовательскую бизнес-логику.
- NotificationLocalServiceBaseImpl – абстрактный класс, характеризующий реализацию базы сервиса, может применяться в качестве альтернативы
- NotificationsManagerPortlet – непосредственно, сам портлет, в котором реализованы два метода:
  - addNotification – метод добавления уведомления в базу данных;
  - deleteNotification – метод удаления уведомления из базы данных;
- Веб-страница, на которой развёртывается пользовательский веб-интерфейс портлета в виде html-страницы, с которой взаимодействует пользователь.



Вышеперечисленные компоненты, информационные потоки данных, а также основные связи показаны на диаграмме компонентов (рисунок 2).

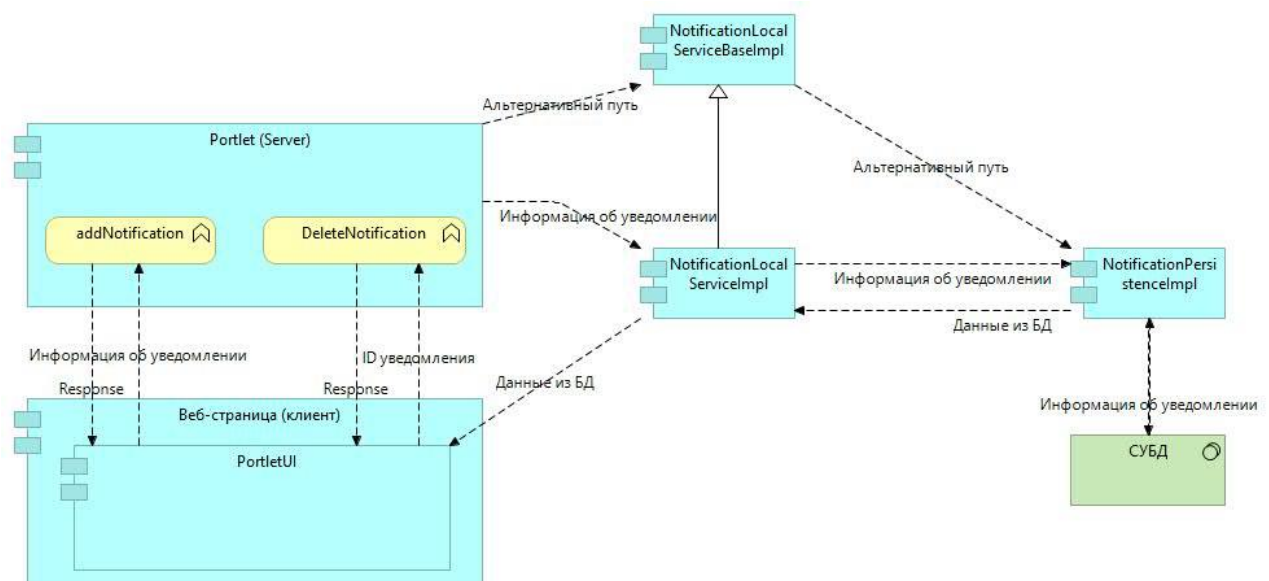


Рисунок 2 - Диаграмма компонентов

На данной диаграмме можно проследить основные движения потоков данных в системе. Пользователь взаимодействует с веб-интерфейсом портлета: вводит информацию в поля, нажимает на кнопки. После нажатия кнопки производится обработка соответствующего события – вызывается соответствующий метод, описанный в **NotificationManagerPortlet**. Введенная пользователем информация передается в портлет с помощью запроса **ActionRequest**, в ответ **ActionResponse** веб-интерфейс сообщает пользователю об успешном или неуспешном выполнении действия. Далее информация об уведомлении передается в локальный сервис, который в свою очередь вызывает соответствующий метод **Persistence**. Методы, описанные в **NotificationPersistence** осуществляют доступ к базе данных.

Отображение информации об зарегистрированных в базе данных уведомлениях осуществляется с помощью компонента **NotificationLocalServiceUtil**, который предоставляет удаленный доступ к локальному сервису. Далее процедура получения данных аналогична процедуре записи, описанной выше.

Полученные данные из базы данных, через компонент **NotificationLocalServiceUtil** отображаются на веб-странице.

### 2.2.2 Модульная декомпозиция системы

Портированный NotificationManager можно разделить на три основных группы модулей:

- Внешние классы и интерфейсы Liferay (com.liferay.portal.kernel): MVCPortlet, PersistedModelLocalService, BaseLocalService, BaseLocalServiceImpl, BaseModel – эти модули предоставляют стандартные структуры и интерфейсы для взаимодействия с Liferay, начиная с портлета и заканчивая работой с базой данных;
- Классы и интерфейсы, описывающие конкретную реализацию менеджера: NotificationManagerPortlet, NotificationLocalService, NotificationLocalServiceImpl, NotificationLocalServiceBaseImpl, NotificationLocalServiceUtil, NotificationPersistence, NotificationPersistenceImpl;
- Модули веб-интерфейса: веб-страница и размещенный на ней пользовательский интерфейс, реализуемый jsp-файлами.

Модульная декомпозиция системы представлена на модульной диаграмме (рисунок 3).

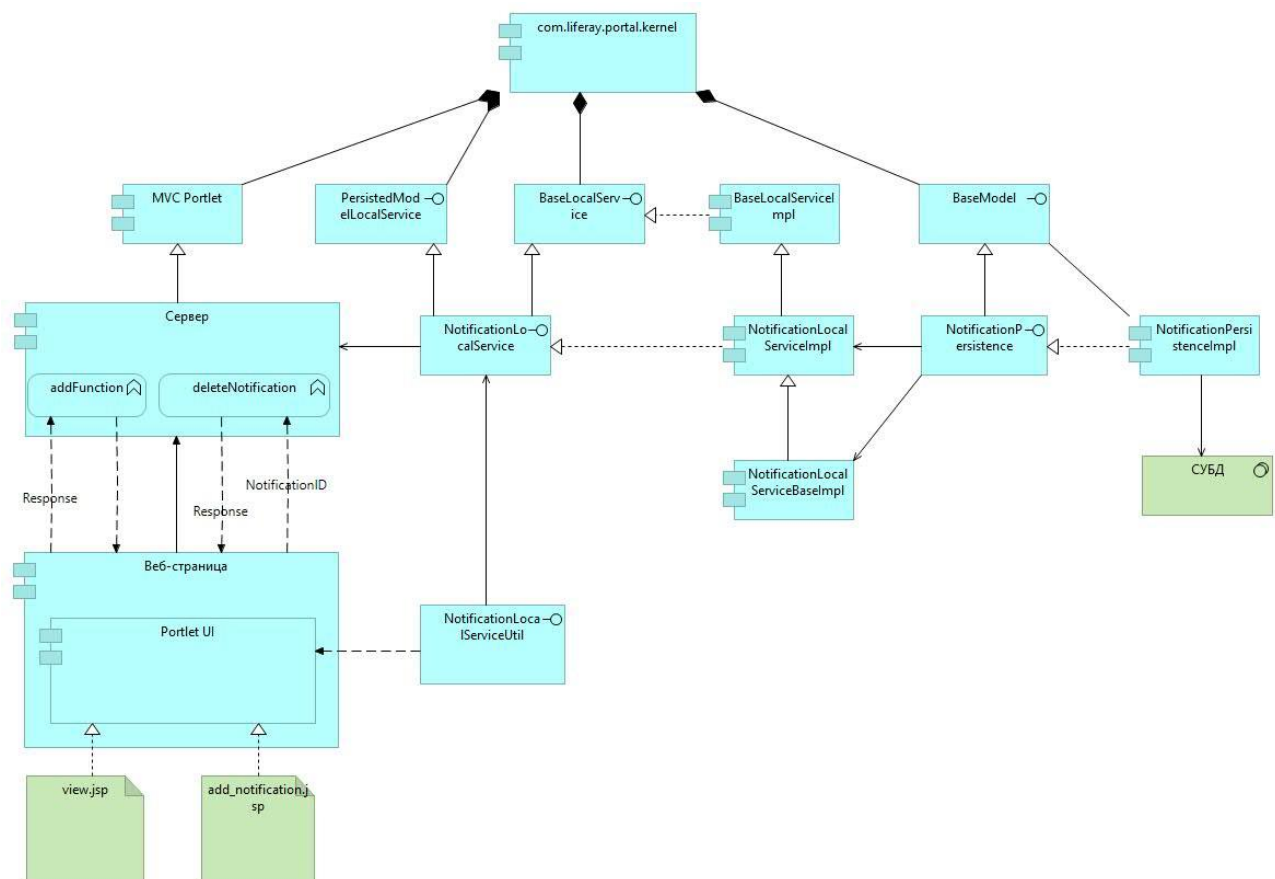


Рисунок 3 - Модульная диаграмма

## 2.3 Детальная архитектура системы

### 2.3.1 Файловая структура репозитория системы в GitHub

Разработанный проект расположен в репозитории GitHub [3].

Структура проекта следующая:

- `.metadata` – метаданные проекта Eclipse с информацией о проекте и воркспейсе.
- `.recommenders` – служебный файл.
- `com.bmstu.notifications.manager` – файл с кодом проекта, который состоит из:
  - `Configs` – папка конфигурации Liferay, с помощью которой Liferay IDE настраивает запуск системы;
  - `Gradle/wrapper` – папка системы автоматической сборки, описанная в п.3.1, описывающая расположение Gradle-е в системе, на которой разворачивается Liferay через системные переменные, а также прописывается путь к сервису Gradle;

- Modules – основная папка проекта, в котором находятся классы проекта – сервисный модуль core и модуль реализации web.

Gradle Wrapper является предпочтительным способом для начала Gradle сборки. Он содержит bat-скрипты для Windows и shell-скрипты для OS X и Linux. Эти скрипты позволяют вам запускать сборку с Gradle без необходимости установки самого Gradle в систему.

### 2.3.2 Плагины и классы системы

Проект состоит из двух основных модулей: `com.bmstu.notifications.manager.core` – модуль, где описаны интерфейсы взаимодействия, сервисы, структуры для работы с базами данных и т.д., и `com.bmstu.notifications.manager.web` – где описан функционал портлета веб-приложения и структура пользовательского интерфейса. В свою очередь, модуль `com.bmstu.notifications.manager.core` разбивается на `com.bmstu.notifications.manager.core-api`, который содержит API для проекта, и `com.bmstu.notifications.manager.core-service`, в котором содержится реализация интерфейсов, определенных в `core-api` модуле. Эти интерфейсы предоставляют сервисы OSGI для экземпляра портала, в котором разворачивается разрабатываемое приложение.

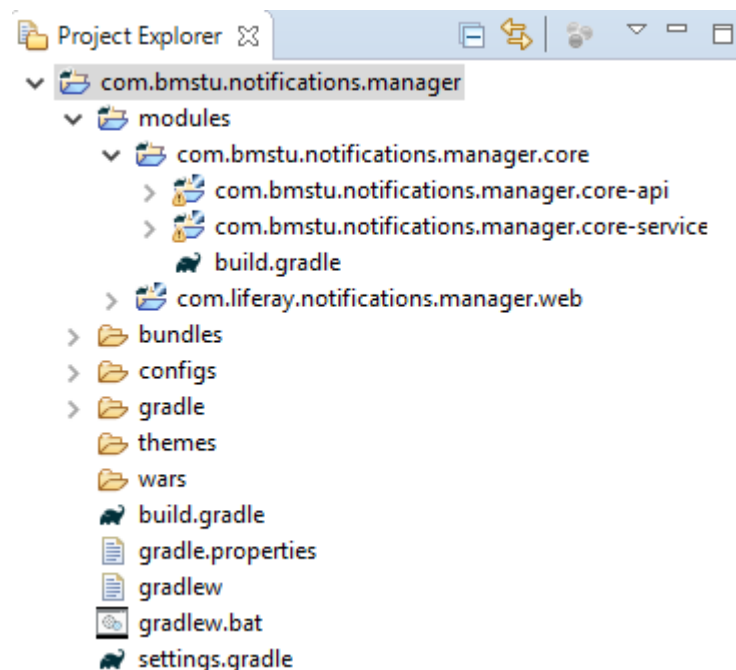


Рисунок 4 - Структура проекта с 3 базовыми модулями

Именно эти три основных модуля и станут Liferay OSGI бандлами данного проекта, которые будут сгенерированы в jar-файлы и развернуты в Liferay:

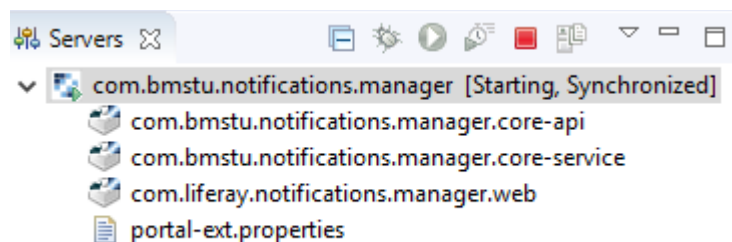


Рисунок 5 - Компоненты OSGI

Модули core-api и core-service создавались с помощью Service-Builder – это инструмент генерации кода, созданный Liferay, который позволяет разработчикам определять пользовательские модели, называемые объектами.

Service Builder создает сервисный уровень с помощью технологии объектно-реляционного сопоставления (ORM), которая обеспечивает чистое разделение между вашей объектной моделью и кодом для базы данных. Это освобождает от добавления необходимой бизнес-логики для приложения. Service Builder берет XML-файл в качестве входных данных и генерирует необходимую модель, уровни обслуживания для приложения. Service Builder генерирует большую часть общего кода, необходимого для реализации операций создания, чтения, обновления, удаления и поиска в базе данных, что позволяет сосредоточиться на аспектах дизайна более высокого уровня.

Service Builder создает проект по инструкции, написанной в service.xml.

В данном файле в теле тега <entity></entity> описывается сущность, которая станет основой для создания таблицы базы данных, поэтому поля сущности описываются также, как и поля базы данных.

Структура данной сущности для данного проекта выглядит следующим образом:

- Entity local-service=«true» name=«Notification»
  - Первичный ключ для таблицы – notificationId;
  - Поле для поиска – groupId;
  - Аудитные поля:
    - companyId;
    - userId;
    - username;
    - createDate;
    - modifiedDate;
  - Статусные поля:
    - status;
    - statusByUserId;

- statusByUserName;
- statusDate;
- Другие поля (взяты из спецификации для REST-методов traccar для NotificationManager):
  - type;
  - always;
  - mails;
  - sms;
  - web;
  - notificationAttributes;
- Теги finder для генерации поисковых методов по заданным полям:

### 2.3.2.1 Модуль core-api

Модуль содержит API для разрабатываемого проекта. Все классы и интерфейсы в core-api модуле упакованы в jar-файл, который генерируется при каждой компиляции и развертке файла. При развертывании этого JAR в Liferay доступны необходимые интерфейсы для определения API сервиса.

Рассмотрим структуру модуля core-api:

- Persistence
  - NotificationPersistence – интерфейс, который определяет методы CRUD, такие, как create, remove, countAll, find, findAll и т.д.
  - NotificationUtil – класс, который обортывает NotificationPersistenceImpl и обеспечивает прямой доступ к базе данных для методов CRUD (create, read, update, delete — четыре базовые функции, используемые при работе с персистентными хранилищами данных). Эта утилита должна использоваться только с сервисом, поэтому используются NotificationLocalServiceUtil или NotificationServiceUtil;
- Local Service:
  - NotificationLocalService – локальный интерфейс обслуживания – главное звено в проекте;
  - NotificationLocalServiceUtil – локальный класс утилиты, который обортывает NotificationLocalServiceImpl и служит в качестве основной локальной точки доступа к сервисному уровню;
  - NotificationLocalServiceWrapper – оболочка локального сервиса, которая реализует NotificationLocalService. Этот класс предназначен для расширения и позволяет разработчикам настраивать локальные службы;
- Remote Service:
  - NotificationService – интерфейс удаленного сервиса;

- NotificationServiceUtil – утилита, которая обертывает NotificationServiceImpl и служит основной точкой удаленного доступа к сервисному уровню;
- NotificationServiceWrapper – оболочка удаленного сервиса, которая обертывает NotificationService. Этот класс предназначен для расширения и позволяет разработчикам настраивать службы удаленного объекта;
- NotificationSoap – базовая реализация модели SOAP (Simple Object Access Protocol — простой протокол доступа к объектам);
- Model:
  - NotificationModel – интерфейс базовой модели. Этот интерфейс и его реализация (NotificationModelImpl) служат только контейнером для объектов со свойствами по умолчанию;
  - Notification – интерфейс, который расширяется NotificationModel;
  - NotificationWrapper – оболочка Notification.

#### 2.3.2.2 Модуль core-service

Модуль core-service содержит реализацию интерфейсов, описанных в модуле core-api. Эти интерфейсы предоставляют службы OSGi для экземпляра портала, в котором разворачивается разрабатываемое приложение.

Рассмотрим структуру данного модуля:

- Persistence:
  - NotificationPersistenceImpl – класс реализации NotificationPersistence;
- Local Service:
  - NotificationLocalServiceImpl – реализация локального сервиса – один из главных классов в проекте, определяющий функционал работы серверной части. В этом классе можно добавлять пользовательскую бизнес-логику.
  - NotificationLocalServiceBaseImpl – абстрактный класс, характеризующий реализацию базу сервиса.
- Remote Service
  - NotificationServiceImpl – реализация интерфейса удаленного сервиса. Здесь можно написать код, который добавляет дополнительные проверки безопасности и вызывает локальные сервисы. Для любых настраиваемых методов, добавленных здесь, Service Builder добавляет соответствующие методы в NotificationService интерфейс при следующем запуске
  - NotificationServiceBaseImpl – абстрактный класс удаленного сервиса.
- Model
  - NotificationModelImpl – реализация интерфейса базовой модели.

- NotificationImpl – реализация модели. Данный класс можно использовать для добавления вспомогательных методов или логики приложения в разрабатываемую модель (без добавления доступны только сеттеры полей и автогенераторы). Всякий раз, когда происходит добавление специальных методов в этот класс, Service Builder добавляет соответствующие методы в Notification интерфейс при следующем запуске.

Также в модуле core-service лежит сам service.xml, использующийся как входные данные для Service Builder и файл service.properties с настройками сервиса.

### 2.3.2.3 Модуль web

Модуль web описывает структуру виджета пользовательского интерфейса портлета (form, portlet, liferay-ui) и реакции на действия и события пользовательского интерфейса (onClick), а также связь обработки событий пользовательского интерфейса с методами локального сервиса.

Рассмотрим структуру данного модуля:

- constants – в данном разделе лежит класс NotificationManagerPortletKeys с описанием констант, которые используются в портлете, в данном случае, только одной константы – public static final String NotificationsManager = “NotificationsManager”, используемой для задания пути;
- portlet – собственно, описание самого портлета: функциональное описание используемых методов. Данный модуль содержит класс NotificationsManagerPortlet, в котором определены два метода:
  - public void addNotification( ActionRequest request, ActionResponse response) – метод добавления уведомления в базу данных;
  - public void deleteNotification( ActionRequest request, ActionResponse response) – метод удаления уведомления из базу данных.

Также здесь используются OSGI-аннотации создания компонента портлета, т.к. класс NotificationsManagerPortlet определяется как Liferay MVC Portlet Component:

- В аннотации @Component перечислены основные параметры, название разделов в расположении данного портлета и его собственное название, путь к файлу с виджетом UI, языком контекста и определением сервиса;
- Аннотация @Reference(unbind = "-") указывает на то, что при отсоединения сервиса, если не определен метод unbind ничего не делать.



Аннотации `@Component` и `@Reference` позволяют избежать шаблонного кода, связанного с сервис-трекерами. Кроме этого, определен и сам метод привязки сервиса – `protected void bindDeviceService(NotificationLocalService notificationsLocalService)`, который определен в классе `NotificationsManagerPortlet`.

Также в модуле `web`, в разделе `resource`, есть Java Server Pages файлы:

- `init.jsp` – для стартовой инициализации (`import` и `uri`)
- `view.jsp` – файл описания виджета UI – описание кнопок, форм, полей и привязка событий к методам, определенным в классе `NotificationsManagerPortlet`.
- `add_notification.jsp` – используется для описания UI на странице ввода данных для добавления нового уведомления, которая вызывается после нажатия кнопки «Add Notification».

## 3 Технологическая часть

### 3.1 Особенности создания бинарной сборки системы

Для сборки данного проекта использовалась система сборки Gradle. Эта система позволила собрать весь проект, а также упаковать все три модуля нашего проекта в jar файл. Для того, чтобы собрать проект, необходимо запустить стандартную задачу Gradle, которая называется deploy. Формат команды выглядит следующим образом: «gradle deploy».

После этого Gradle подгрузит все зависимости проекта. Главная зависимость – зависимость модуля web от подмодулей core-api и core-service.

В данных модулях содержатся все необходимые модели и интерфейсы для создания объектов базы данных и взаимодействия с базой данных, а также модели описания объектов типа Notification.

### 3.2 Особенности запуска разработанной системы

После того, как была осуществлена сборка проекта, необходимо загрузить и установить сервер Apache Tomcat. После скачивания сервера необходимо указать полный путь до папки с установленным сервером в Eclipse Servers. Так как Gradle позволяет собрать и упаковать проект в jar-файлы, то созданный сервер (с подсказкой Eclipse) увидит созданные Gradle-ом jar-файлы, в которых упакован core-service, core-api и, реализующий взаимодействие с web-интерфейсом, портлет web.

После этого необходимо запустить сервер. Будут подтянуты бандлы, реализующий функционал Laifray портала, наши jar-файлы и откроется веб страница Liferay портала. После добавления портлета на страницу портала из раздела Applications □ Traccar, можно пользоваться описанным функционалом п портлета.

### 3.3 Анализ реализации системы

#### 3.3.1 Анализ исходного кода с помощью метрик качества

На рисунке 6 отображен список всех метрик по разделам. Всего имеется четыре раздела:

- метрики количества (Count);
- метрики сложности (Complexity);
- метрики Роберта Мартина (Robert C. Martin);
- метрики Чидамбера-Кеммерера (Chidamber & Kemerer).

Первый раздел с метриками количества (Count) содержит следующие метрики:

- количество классов верхнего уровня (Unit);
- среднее число внутренних классов на класс (Classes / Class); среднее число методов в классе (Methods / Class);
- среднее число полей в классе (Fields / Class); число строчек кода (ELOC);
- число строчек кода на модуль (ELOC / Unit).

Второй раздел с метриками сложности (Complexity) содержит всего три различных метрики:

- средняя циклическая сложность (CC);
- метрика Fat (Fat);
- средняя зависимость компонентов между модулями (ACD - Unit).

Третий раздел с метриками Роберта Мартина содержит следующие метрики:

- нормализованное расстояние от основной последовательности (D);
- абстрактность (A);
- нестабильность (I);
- число афферентных соединений (Ca);
- число эфферентных соединений (Ce).

fx Metrics	
Category/Metric	Value
Count	
Libraries	5
Packages	12
Units	27
Units / Package	2.25
Classes / Class	0
Methods / Class	22.52
Fields / Class	5
ELOC	4141
ELOC / Unit	153.37
Complexity	
CC	1.23
Fat - Libraries	2
Fat - Packages	18
Fat - Units	53
Tangled - Libraries	0.00%
ACD - Library	10.00%
ACD - Package	17.42%
ACD - Unit	13.96%
Robert C. Martin	
D	-0.19
D	0.33
Chidamber & Kemerer	
WMC	27.81
DIT	1.19
NOC	0.16
CBO	n/a
RFC	n/a
LCOM	n/a

Рисунок 6 - Значение метрик

Последний раздел с метриками Чидамбера-Кемерера содержит следующие метрики:

- средняя длина метода на класс (WMC);
- средняя глубина наследования (DIT);
- среднее количество классов-наследников (NOC);
- среднее число соединений класса (CBO);
- среднее число методов, которые потенциально могут быть выполнены в ответ на сообщение, полученное объектом этого класса (RFC);
- отсутствие единства методов (LCOM).

Основные метрики, характеризующие удовлетворение интересов заинтересованных сторон: CC, eLOC/Unit, ACD – Library/Package/Unit, D и CBO.

Метрика CC характеризует количество линейно независимых маршрутов через программный код. Цикломатическая сложность программы для обеспечения лучшего понимания кода не должна превышать 10 [6, стр 121]. Согласно рисунку 6, данное требование удовлетворено, а значит, интерес разработчика к легкой поддержке и пониманию кода проекта.

Метрика eLOC - это количество всех строк, которые не являются комментариями, пробелами или отдельными фигурными скобками или скобками. Этот показатель наиболее более точно отражает количество выполненной работ. Показатель eLOC/Unit – отношение количества эффективных строк кода к файлам, в которых они находятся, должно быть не больше 200 согласно принятому промышленному стандарту [7, 8]. В данном проекте данное требование удовлетворено, а значит, удовлетворен интерес разработчика к легкому пониманию и поддержке проекта.

Метрика ACD – Library/Package/Unit – процент зависимости между соответствующими модулями системы. Согласно принятому промышленному стандарту, должен быть не больше 20% [7, 8]. Согласно рисунку 6, данное требование удовлетворено, а значит, удовлетворен интерес разработчика к легкой поддержке и расширяемости проекта.

Метрика D показывает, насколько сборка проекта сбалансирована относительно её абстрактности и стабильности. Согласно рисунку 6,  $D = -0.19$ , а значит, сборка сбалансирована относительно её абстрактности и стабильности. Таким образом, удовлетворен интерес разработчика к расширяемости проекта.

Метрика CBO дает возможность определить количество классов, с которыми связан данный класс. Низкое значение данной метрики (рисунок 6) улучшает модульность и содействует инкапсуляции проекта, а значит, удовлетворен интерес разработчика к расширяемости проекта.

### 3.3.2 Анализ зависимостей в коде системы

На рисунке 7 представлена диаграмма классов. Диаграмма состоит из двух основных компонентов:

- web(com.bmstu.notifications.manager.web);
- core(com.bmstu.notifications.manager.core).

Как и в описании плагинов и классов (п.2.3), модуль core делится на два модуля: service и api, соответствующие модулям программы.

В модуле `service` выделены два основных класса, описывающие реализацию функционала локального сервиса: `NotificationLocalServiceImpl` и `NotificationLocalServiceBaseImpl`, также в полях этих классов выделены основные методы и атрибуты, которые используются в проекте:

Атрибуты:

- `notificationLocalService`;
- `notificationPersistence`;
- `counterLocalService`;
- `resourceLocalService`;

и др.

Методы:

- `addNotification(long userId, other field..., ServiceContext serviceContext)`;
- `getNotification(long NotificationId)`;
- `createNotification(long NotificationId)`;
- `deleteNotification(long NotificationId)` ;
- `updateNotification(Notification Notification)`.

В модуле `api` на диаграмме показан только основной интерфейс `NotificationLocalService`, который реализуется классом `NotificationLocalServiceBaseImpl`, который, в свою очередь, наследуется классом `NotificationLocalServiceImpl`.

Модуль `web`, отвечающий за UI, полностью представлен двумя составляющими его классами: `NotificationsManagerPortletKeys` в модуле `constants`, который описывает используемые константы, и класс `NotificationsManagerPortlet` в модуле `portlet`, который описывает компонент Liferay MVC Portlet.

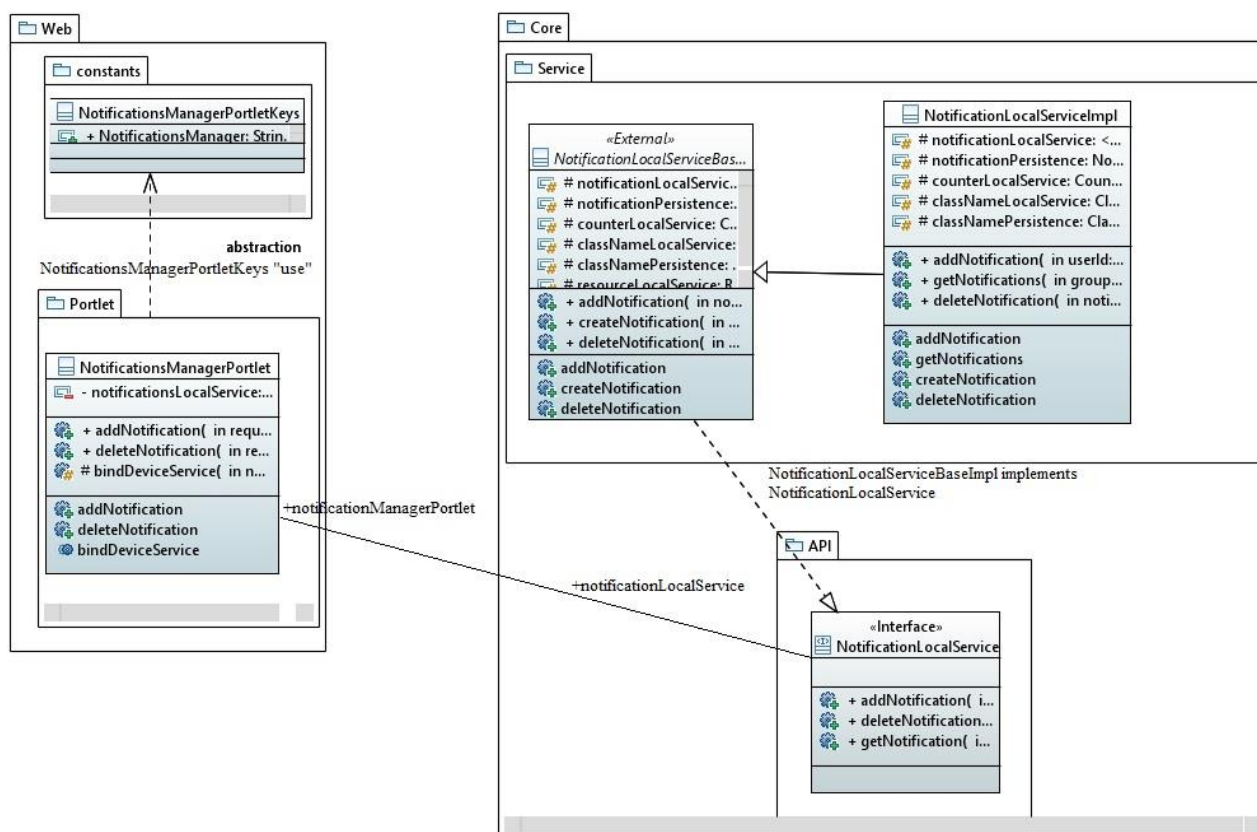


Рисунок 7 - Диаграмма классов

Рисунки 8-9 характеризуют зависимости между пакетами, реализующих портлет и модель Notification, отвечающие принципам SOLID: разделенные интерфейсы, зависимости на абстракциях, открытые для расширения модули и др.

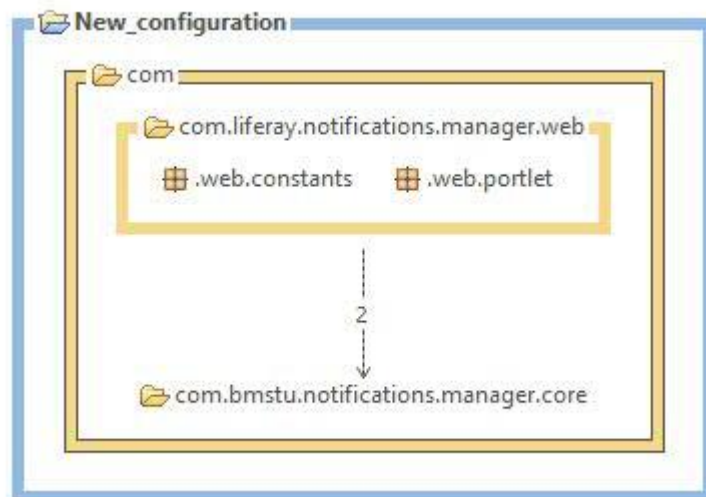


Рисунок 8 - Зависимости между основными модулями

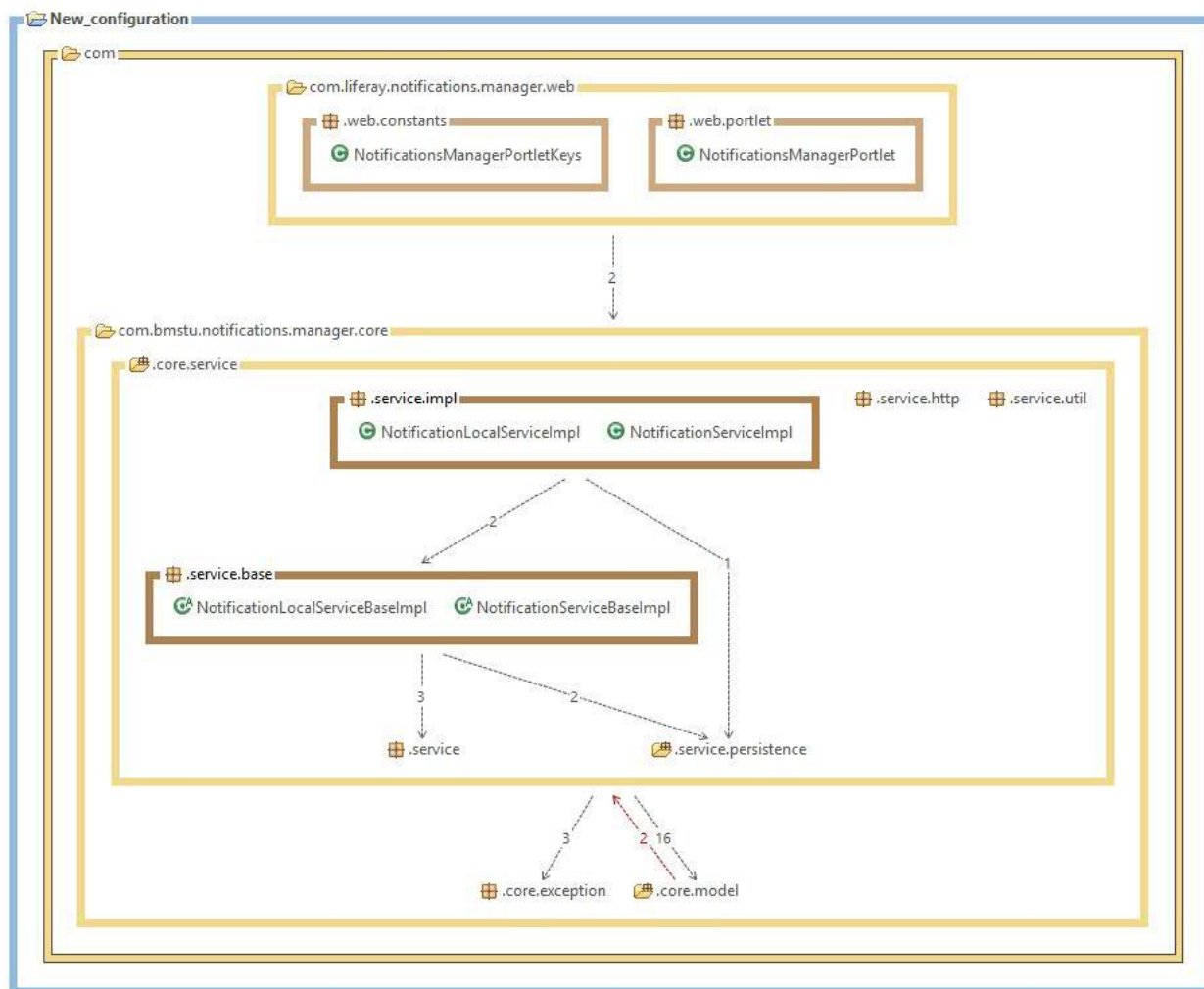


Рисунок 9 - Зависимости между основными модулями



### 3.4 Тестирование на корректность работы

Для проведения тестирования был выбран ручной метод. Для этого будет проверяться выполнение всех требований технического задания (в упрощенном варианте, объединяя схожие компоненты), а именно:

- корректный ввод данных и добавление нового уведомления;
- корректный ввод данных и удаление уведомления;

### 3.4.1 Проверка NotificationManager

Изначально Service Builder конфигурирует файл view.jsp файл, который определяет структуру виджета. В процессе разработки в этот файл были добавлены реализации форм добавления и удаления из базы данных уведомления по полю notificationId. Структура контейнера поиска представлена на рисунке 10.

```
<liferay-ui:search-container
total="<%=NotificationLocalServiceUtil.getNotificationsCount()%>">
<liferay-ui:search-container-results
results="<%=NotificationLocalServiceUtil.getNotifications(scopeGroupId.longValue(),
searchContainer.getStart(), searchContainer.getEnd())%>" />

<liferay-ui:search-container-row
className="com.bmstu.notifications.manager.core.model.Notification"
modelVar="notification">

<liferay-ui:search-container-column-text property="notificationId" name="ID"/>
<liferay-ui:search-container-column-text property="type" value="Some Type"/>
<liferay-ui:search-container-column-text property="always" name="All Devices"/>
<liferay-ui:search-container-column-text property="mail" name="Send by Mail"/>
<liferay-ui:search-container-column-text property="sms" name="Send by SMS"/>
<liferay-ui:search-container-column-text property="web" name="Send by Web"/>
<liferay-ui:search-container-column-text property="notificationAttributes" name="Attributes"/>

</liferay-ui:search-container-row>

<liferay-ui:search-iterator />

</liferay-ui:search-container>
```

Рисунок 10 - Структура контейнера

Первоначальный вид пользовательского интерфейса представлен на рисунке 11 и состоит из двух кнопок: «Add Notification» и «Delete», а также статическое представление базы данных и поля ввода параметра notificationId.

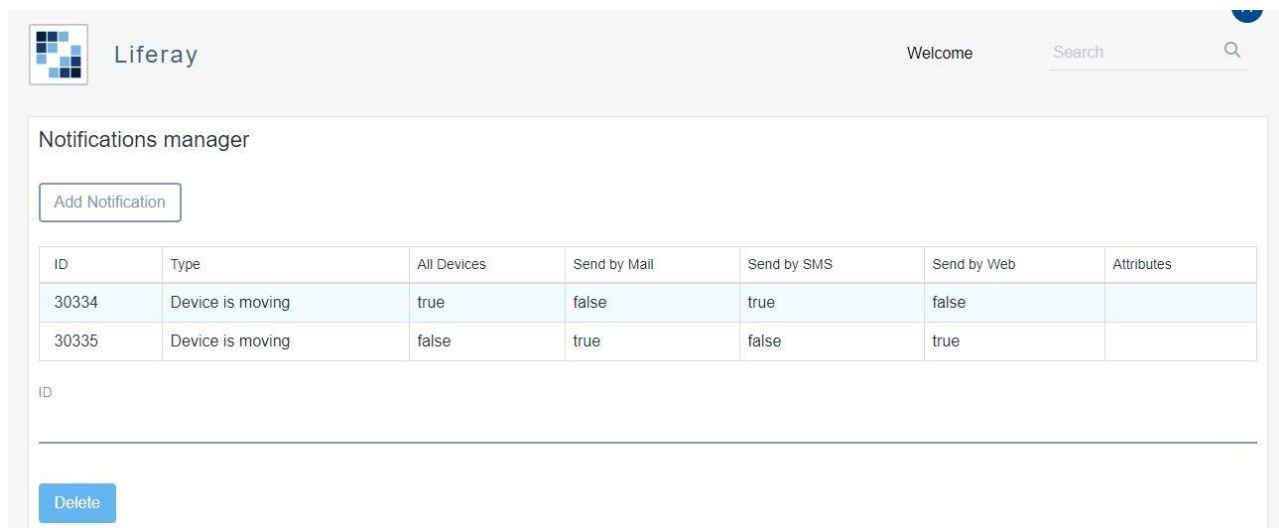


Рисунок 11 - Пользовательский интерфейс

При нажатии кнопки «Add Notification» открывается новая JSP страница `add_notification.jsp`, которая состоит из полей ввода, определенных структурой контейнера (рисунок 12) и двух кнопок: «Save» и «Cancel».

The screenshot shows the 'add\_notification.jsp' form. It has a 'Type' dropdown menu with 'Device is moving' selected. Below it are four checkboxes: 'All Devices', 'Send by Mail', 'Send by SMS', and 'Send by Web'. There is also an 'Attributes' text input field. At the bottom are 'Save' and 'Cancel' buttons.

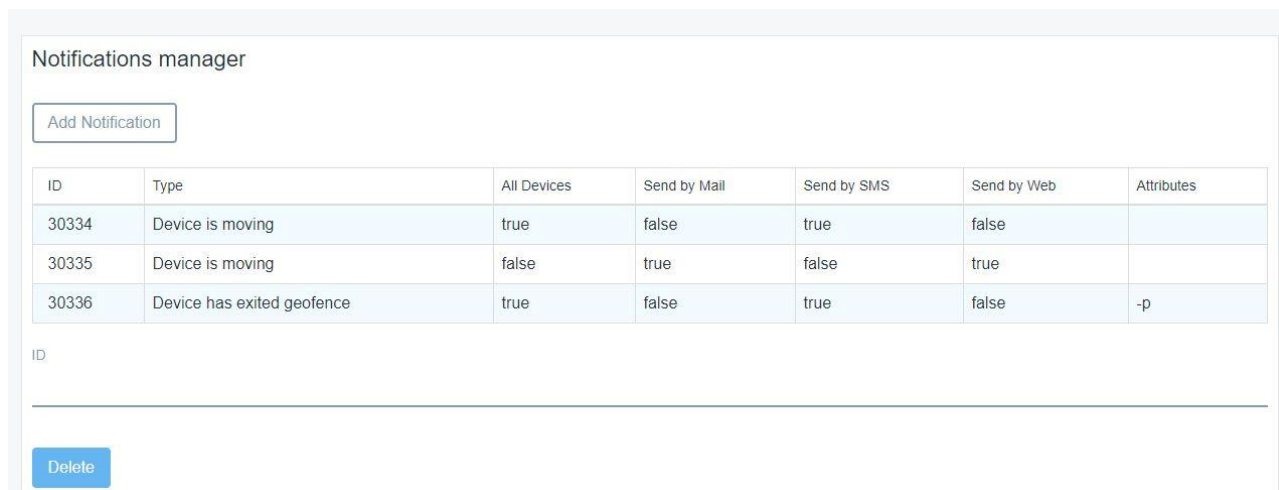
Рисунок 12 - Интерфейс ввода данных для добавления нового уведомления

Заполняем поля ввода данных (рисунок 13).

The screenshot shows the 'add\_notification.jsp' form with the following filled-in data: 'Type' is 'Device has exited geofence', 'All Devices' is checked, 'Send by SMS' is checked, and 'Attributes' is '-p|'. The 'Save' and 'Cancel' buttons are at the bottom.

Рисунок 13 - Заполненная форма ввода данных

Нажимаем кнопку «Save» (рисунок 14):



The screenshot shows a web interface titled "Notifications manager". At the top left is a button labeled "Add Notification". Below it is a table with the following data:

ID	Type	All Devices	Send by Mail	Send by SMS	Send by Web	Attributes
30334	Device is moving	true	false	true	false	
30335	Device is moving	false	true	false	true	
30336	Device has exited geofence	true	false	true	false	-p

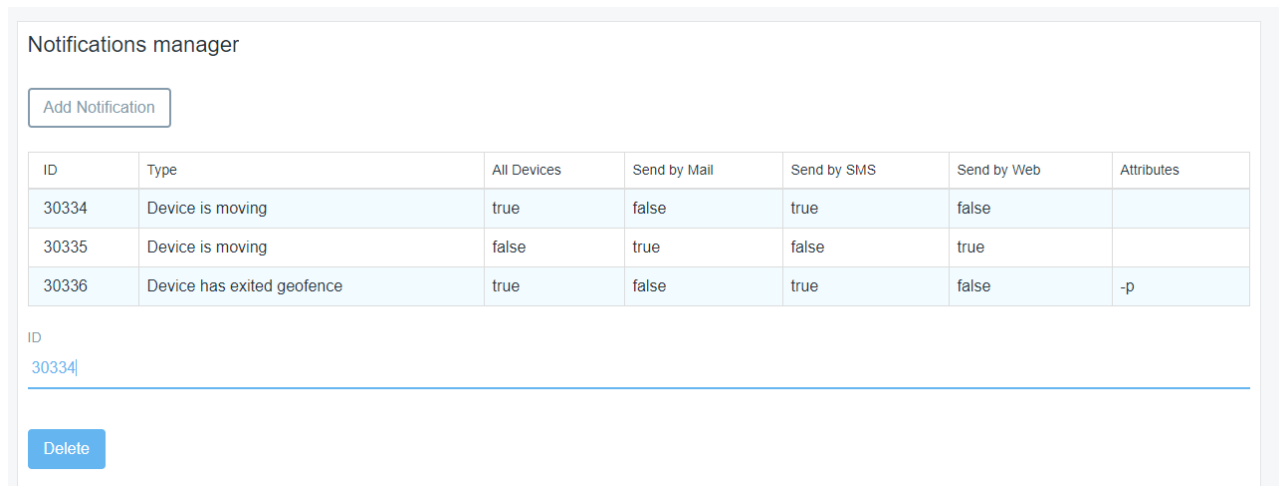
Below the table is a form with a label "ID" and an empty input field. At the bottom left is a blue button labeled "Delete".

Рисунок 14 - Результат добавления уведомления

Добавление нового уведомления прошло успешно – на рисунке 14, в представлении таблицы БД можно увидеть строку добавленного уведомления.

### 3.4.2 Корректный ввод данных и добавление нового уведомления

Теперь попробуем удалить какую-нибудь запись из БД по полю ID:



The screenshot shows the same "Notifications manager" interface as Figure 14. The table contains the same three rows. In the form below the table, the "ID" field now contains the value "30334". The blue "Delete" button at the bottom left is highlighted.

Рисунок 16 - Удаление уведомления из БД

Введем в поле ID значение того уведомления, которое нужно удалить (рисунок 16).

Notifications manager

Add Notification

ID	Type	All Devices	Send by Mail	Send by SMS	Send by Web	Attributes
30335	Device is moving	false	true	false	true	
30336	Device has exited geofence	true	false	true	false	-p

ID

Delete

Рисунок 17 - Результат удаления уведомления

Теперь попробуем ввести значение ID, которого нет в БД:

Notifications manager

Your request failed to complete.

×

ID	Type	All Devices	Send by Mail	Send by SMS	Send by Web	Attributes
30335	Device is moving	false	true	false	true	
30336	Device has exited geofence	true	false	true	false	-p

ID

Delete

Рисунок 18 - Результат удаления несуществующего уведомления

В результате, т.к. уведомление с введенным значением поля не найдено в БД, было выведено сообщение об ошибке (рисунок 18).

## 4 Выводы

В ходе данной курсовой работы веб-сервис NotificationManager и компонент пользовательского интерфейса системы Трассар были успешно портированы на платформу Liferay с использованием технологии OSGi DS с сохранением протокола взаимодействия клиента с сервером.

Данный веб-сервис был протестирован методом ручного тестирования. Данные тесты показали, что полностью реализует базовые функции, указанные в техническом задании и в технической документации. Также проверена уникальность индивидуального идентификатора.

Данный проект можно использовать как встраиваемый модуль для более удобного отображения в других системах GPS-трекинга, навигации и т.п. Также данный веб-сервис имеет возможность масштабирования, расширения функционала или же он может быть быстро перестроен для работы с другими данными.

## Список используемой литературы

[1] Документация по Traccar: [Электронный ресурс]. Режим доступа: <https://www.traccar.org/documentation/>.

[2] Документация по LifeRay: [Электронный ресурс]. Режим доступа: <https://dev.liferay.com/develop/tutorials/>.

[3] GitHub – NotificationManager [Электронный ресурс]. Режим доступа: <https://github.com/Letargon/com.bmstu.notifications.manager.git>

[4] Liferay 7 Features [Электронный ресурс]. Режим доступа: <http://www.liferay savvy.com/2016/03/liferay-7-features.html>

[5] Traccar API-reference [Электронный ресурс]. Режим доступа: <https://www.traccar.org/api-reference/>

[6] Черников Б.В., Поклонов Б.Е. Оценка качества программного обеспечения: Практикум. М.: ИД «ФОРУМ» — ИНФРА-М, 2012 — 400 с.

[7] ГОСТ Р ИСО/МЭК 9126-93 Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению.

[8] ГОСТ 28195-89 Оценка качества программных средств. Общие положения