|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатики и систем управления»

КАФЕДРА «Информационные системы и телекоммуникации»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ**

**НА ТЕМУ:**

Портирование веб-сервисов DeviceManager и

GeofenceManager и компонентов пользовательского интерфейса системы Traccar на OSGi сервис и портлет платформы Liferay с реализацией их взаимодействия

Студент группы ИУ3-72 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ И. В. Музланов

(подпись, дата)

Руководитель курсовой работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.М. Иванов

(подпись, дата)

*2018 г.*

Содержание

[1 Исследовательская часть 3](#_Toc507225047)

[1.1 Обзор предметной области 3](#_Toc507225048)

[1.2 Разрабатываемая система как «черный ящик» в окружении внешних систем 3](#_Toc507225049)

[1.3 Техническое задание 5](#_Toc507225050)

[1.4 Интересы заинтересованных сторон по отношению к системе 6](#_Toc507225051)

[2 Конструкторская часть 7](#_Toc507225052)

[2.1 Технические решения, позволяющие удовлетворить интересы заинтересованных сторон 7](#_Toc507225053)

[2.2 Высокоуровневая архитектура системы 8](#_Toc507225054)

[2.2.1 Компонентная декомпозиция системы 8](#_Toc507225055)

[2.3 Детальная архитектура системы 10](#_Toc507225056)

[2.3.1 Файловая структура репозитория системы в GitHub 10](#_Toc507225057)

[2.3.2 Плагины и классы системы 11](#_Toc507225058)

[3 Технологическая часть 19](#_Toc507225059)

[3.1 Особенности создания бинарной сборки системы 19](#_Toc507225060)

[3.2 Особенности запуска разработанной системы 19](#_Toc507225061)

[3.3 Анализ реализации системы 20](#_Toc507225062)

[3.3.1 Анализ исходного кода с помощью метрик качества 20](#_Toc507225063)

[3.3.2 Анализ зависимостей в коде системы 22](#_Toc507225064)

[3.4 Тестирование на корректность работы 26](#_Toc507225065)

[3.4.1 Проверка взаимодействия DeviceManager и GeofenceManager 26](#_Toc507225066)

[4 Выводы 34](#_Toc507225067)

[Список используемой литературы 35](#_Toc507225068)

1. Исследовательская часть

## 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.

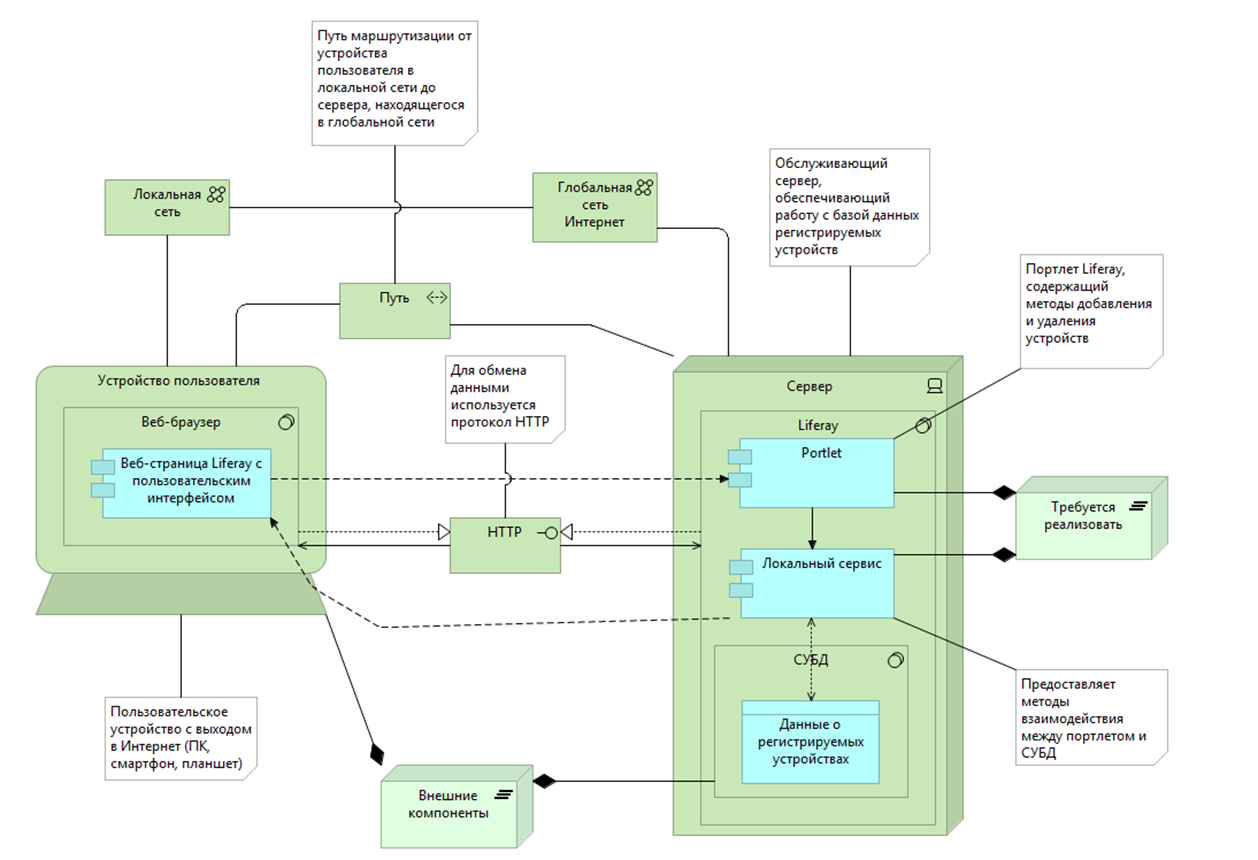
Liferay Portal — программный продукт, представляющий собой корпоративный портал, то есть решение, предназначенное для централизованного доступа к нескольким различным корпоративным приложениям в одном месте. Liferay иногда описывается как система управления содержимым (CMS) или платформу для веб-приложений.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

Traccar – программное обеспечение для отслеживания GPS.

Разрабатываемая система как «черный ящик» в окружении внешних систем

Перед тем, как приступать 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 DeviceManager предназначен для добавления, удаления и изменения информации о подключаемых устройствах в базе данных GPS-трекера Traccar. 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 осуществляет ввод необходимых данных для добавления или удаления информации об устройстве и вызывает. Портируемый веб-сервис GeofenceManager предназначен для добавления, удаления и изменения информации о геозонах в базе данных GPS-трекера Traccar. Его структура аналогична веб-сервису DeviceManager.



*Рисунок 1 - Система в окружении смежных систем*

Техническое задание

Портирование веб-сервисов DeviceManager и GeofenceManager и компонентов пользовательского интерфейса системы Traccar на OSGi сервис и портлет платформы Liferay с реализацией их взаимодействия.

Тема индивидуальных заданий:

* изучить DeviceManager и GeofenceManager и их графические интерфейсы в Traccar;
* реализовать взаимодействие веб-сервисов;
* реализовать хранение данных в БД (функционал должен быть инкапсулирован);
* 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.

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 зарегистрированных устройств и геозон.  П2 Удобное взаимодействие пользователя с системой. |
| Разработчик (новый разработчик) | Р1 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 устройств, добавленных в базу данных. |

# Конструкторская часть

## 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 устройств и геозон.  П2 Удобное взаимодействие пользователя с системой. | Визуализатор данных из БД.  Эргономичный интерфейс реализованных функций взаимодействия с базой данных устройств. |
| 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 устройств, 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. |

## Высокоуровневая архитектура системы

### Компонентная декомпозиция системы

Разрабатываемая система состоит из следующих основных компонентов:

– DeviceManager;

* GeofenceManager.

В свою очередь каждый из этих компонентов рассмотрим подробнее. DeviceManager состоит из следующих компонентов:

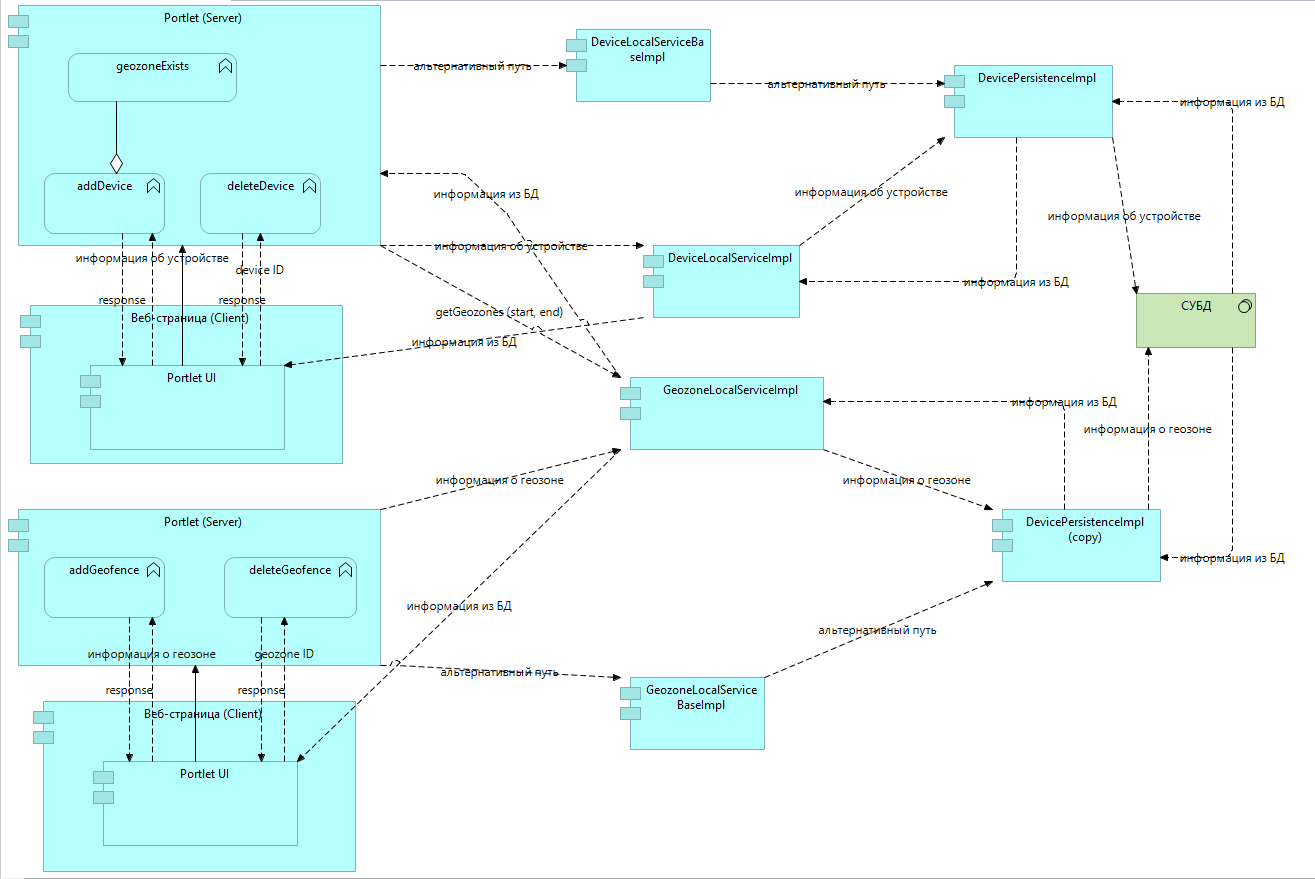
* DevicePersistenceImpl – компонент, обеспечивающий прямой доступ к базе данных для методов CRUD (create, read, update, delete —четыре базовые функции, используемые при работе с персистентными хранилищами данных).
* DeviceLocalServiceImpl – реализация локального сервиса – один из главных классов в проекте, определяющий функционал работы серверной части. В этом классе можно добавлять пользовательскую бизнес-логику.
* DeviceLocalServiceBaseImpl – абстрактный класс, характеризующий реализацию базы сервиса, может применяться в качестве альтернативы DeviceLocalServiceImpl.
* DevicesManagerPortlet – непосредственно, сам портлет, в котором реализованы три метода:
* addDevice – метод добавления устройства в базу данных;
* deleteDevice – метод удаления устройства из базу данных;
* geozoneExists – метод определения существования введенной для устройства геозоны (реализует непосредственно взаимодействие между веб-сервисами);
* Веб-страница, на которой развёртывается пользовательский веб-интерфейс портлета в виде html-страницы, с которой взаимодействует пользователь.

Пользователь взаимодействует с веб-интерфейсом портлета – вводит информацию в поля, нажимает на кнопки. После нажатия кнопки производиться обработка соответствующего события – вызывается соответствующий метод, описанный в DeviceManagerPortlet. 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 устройстве передается в локальный сервис, который в свою очередь вызывает соответствующий метод Persistence. Методы, описанные в DevicePersistane осуществляют доступ к базе данных.

Отображение информации об зарегистрированных в базе данных устройствах осуществляется с помощью компонента DeviceLocalServiceUtil, который предоставляет удаленный доступ к локальному сервису. Далее процедура получения данных аналогична процедуре записи, описанной выше. Полученные данные из базы данных, через компонент DeviceLocalServiceUtil отображаться на веб-странице.

Компонент GeofenceManager имеет аналогичную структуру, за исключением самого портлета: в нем реализованы только два метода – addGeofence и deleteGeofence.

На диаграмме (см. рисунок 2) можно проследить основные потоки данных в системе. Взаимодействие между DeviceManager и GeofenceManager осуществляется при добавлении нового устройства: проверяется факт существования указанного для устройства ID геозоны. Если геозоны с указанным ID не существует, то устройство нельзя будет добавить.



*Рисунок 2 – Диаграмма компонентов*

Детальная архитектура системы

### Файловая структура репозитория системы в GitHub

Разработанный проект расположен в репозитории GitHub [3].

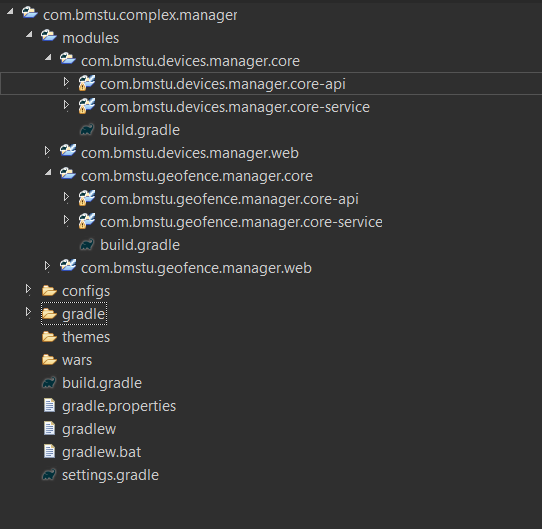
Com.bmstu.complex.manager – файл с кодом проекта, который состоит из:

* 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-e в системе, на которой разворачивается Liferay через системные переменные, а так же прописывается путь к сервису Gradle;
* Modules – основная папка проекта, в котором находятся классы проекта – сервисные модули core и модули реализации web для DeviceManager и GeofenceManager;

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.

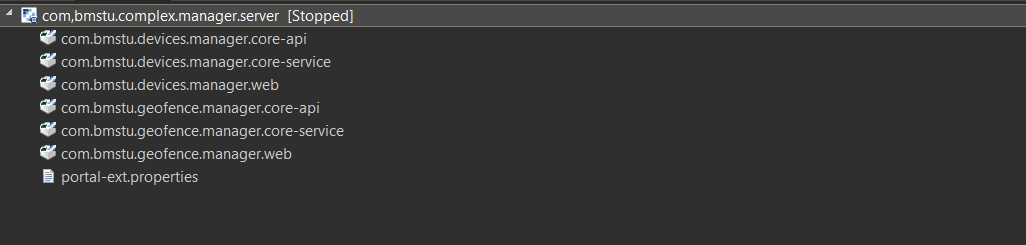
### Плагины и классы системы

Проект состоит из четырех основных модулей: com.bmstu.devices.manager.core – модуль, где описаны интерфейсы взаимодействия, сервисы, структуры для работы с базами данных и т.д. для DeviceManager, com.bmstu.devices.manager.web – где описан функционал портлета веб-приложения и структура пользовательского интерфейса DeviceManager, а также аналогичные модули com.bmstu.geofence.manager.core и com.bmstu.geofence.manager.web для GeofenceManager. В свою очередь, модуль com.bmstu.devices.manager.core разбивается на com.bmstu.devices.manager.core-api, который содержит API для проекта, и com.bmstu.devices.manager.core-service, в котором содержится реализация интерфейсов, определенных в core-api модуле. Эти интерфейсы предоставляют сервисы OSGI для экземпляра портала, в котором развертывается разрабатываемое приложение. Модуль com.bmstu.geofence.manager.core разбивается аналогично.



*Рисунок 3 - Структура проекта с 6 базовыми модулями*

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



*Рисунок 4 - Компоненты OSGI*

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

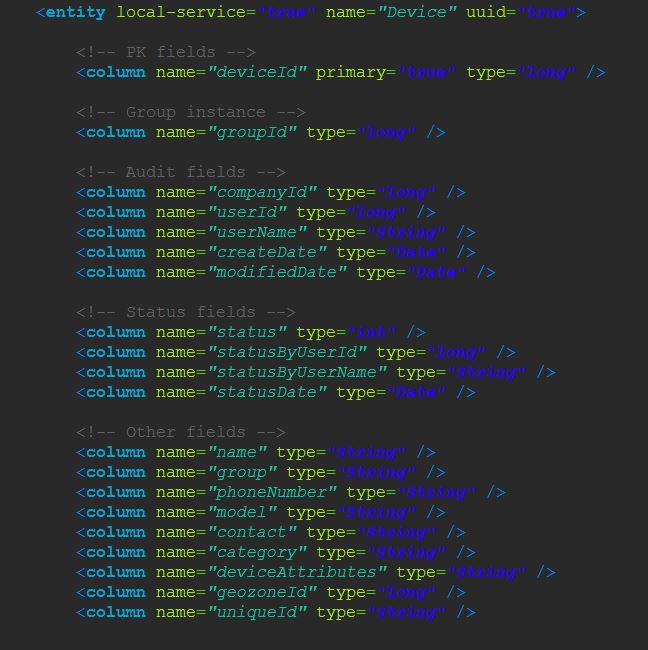
Service Builder создает сервисный уровень с помощью технологии объектно-реляционного сопоставления (ORM), которая обеспечивает чистое разделение между вашей объектной моделью и кодом для базы данных. Это освобождает от добавления необходимой бизнес-логики для приложения. Service Builder берет XML-файл 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 сущностей для данного проекта выглядит следующим образом:



*Рисунок 5 – структура сущности Geozone*



*Рисунок 6 – структура сущности Device*

#### Модули core-api

Модули содержат API для разрабатываемого проекта. Все классы и интерфейсы в core-api модулях упакованы в jar-файлы, которые генерируются при каждой компиляции и развертке файлов. При развертывании этих JAR в Liferay доступны необходимые интерфейсы для определения API сервиса.

Рассмотрим структуру модуля devices.manager.core-api:

* Persistence
* DevicePersistence – интерфейс, который определяет методы CRUD, такие, как create, remove, countAll, find, findAll и т.д.
* DeviceUtil – класс, который обертывает DevicePersistenceImpl 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 DeviceLocalServiceUtil или DeviceServiceUtil
* Local Service
* DeviceLocalService – локальный интерфейс обслуживания – главное звено в проекте.
* DeviceLocalServiceUtil – локальный класс утилиты, который обертывает DeviceLocalServiceImpl и служит в качестве основной локальной точки доступа к сервисному уровню.
* DeviceLocalServiceWrapper – оболочка локального сервиса, которая реализует DeviceLocalService. Этот класс предназначен для расширения и позволяет разработчикам настраивать локальные службы.
* Remote Service
* DeviceService – интерфейс удаленного сервиса.
* DeviceServiceUtil – утилита, которая обертывает DeviceServiceImpl и служит основной точкой удаленного доступа к сервисному уровню.
* DeviceServiceWrapper – оболочка удаленного сервиса, которая обертывает DeviceService. Этот класс предназначен дял расширения и позволяет разработчикам настраивать службы удаленного объекта.
* DeviceSoap – базовая реализация модели SOAP (Simple Object Access Protocol — простой протокол доступа к объектам).
* Model
* DeviceModel – интерфейс базовой модели. Этот интерфейс и его реализация (DeviceModelImpl) служат только контейнером для объектов со свойствами по умолчанию.
* Device – интерфейс, который расширяется DeviceModel.
* DeviceWrapper – оболочка Device.

Для geofence.manager.core-api структура аналогична вышеприведенной.

#### Модули core-service

Модули core-service содержат реализацию интерфейсов, описанных в модулях core-api. Эти интерфейсы предоставляют службы OSGi для экземпляров портала, в которых развертывается разрабатываемое приложение.

Рассмотрим структуру devices.manager.core-service:

* Persistence
* DevicePersistenceImpl – класс реализации DevicePersistence.
* Local Service
* DeviceLocalServiceImpl – реализация локального сервиса – один из главных классов в проекте, определяющий функционал работы серверной части. В этом классе можно добавлять пользовательскую бизнес-логику.
* DeviceLocalServiceBaseImpl – абстрактный класс, характеризующий реализацию базу сервиса.
* Remote Service
* DeviceServiceImpl – 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 DeviceService интерфейс при следующем запуске
* DeviceServiceBaseImpl – абстрактный класс удаленного сервиса.
* Model
* DeviceModelImpl – реализация интерфейса базовой модели.
* DeviceImpl – 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 Device 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.

Структура geofence.manager.core-service аналогична вышеприведенной.

#### Модули web

Модули web описывают структуру виджетов пользовательского интерфейса портлетов (form, portlet, liferay-ui) и реакции на действия и события пользовательского интерфейса (onClick), а также связь обработки событий пользовательского интерфейса с методами локальных сервисов.

Рассмотрим структуру devices.manager.web:

* constants – в данном разделе лежит класс DeviceManagerPortletKeys с описанием констант, которые используются в портлете, в данном случае, только одной константы – public static final String DevicesManager = “DevicesManager”, используемой для задание пути.
* portlet – собственно, описание самого портлета: функциональное описание используемых методов. Данный модуль содержит класс DevicesManagerPortlet, в котором определены два метода:
* public void addDevice( ActionRequest request, ActionResponse response) – метод добавления устройства в базу данных;
* public void deleteDevice( ActionRequest request, ActionResponse response) – метод удаления устройства из базу данных;
* private boolean geozoneExists(long geozoneId) – метод проверки существования геозоны с указанным ID.

Также здесь используются OSGI-аннотации создания компонента портлета, т.к. класс DevicesManagerPortlet 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 = "-") указвает на то, что при отсоединения сервиса, если не определен метод unbind ничего не делать.

Аннотации @Componentи и @Reference позволяют избежать шаблонного кода, связанного с сервис-трекерами. Кроме этого, определены и сами методы привязки сервисов – protected void bindDeviceService(DeviceLocalService deviceLocalService) и protected void bindGeozoneService(GeozoneLocalService geozoneLocalService), которые определены в классе DevicesManagerPortlet.

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 DevicesManagerPortlet.
* edit\_device.jsp – используется для описания UI на странице ввода данных для добавления нового устройства, которая вызывается после нажатия кнопки «AddDevice».

Структура geofence.manager.core аналогична вышеприведенной, за исключением класса GeofenceManagerPortlet, который не содержит метода geozoneExists и метода привязки сервиса deviceLocalService.

# Технологическая часть

Особенности создания бинарной сборки системы

Для сборки данного проекта использовалась система сборки Gradle. Эта система позволила собрать весь проект, а также упаковать все шесть модулей 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 модулей 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 Device и Geozone.

Особенности запуска разработанной системы

После того, как была осуществлена сборка проекта, необходимо загрузить и установить сервер Apachi Tomcat. После скачивания сервера необходимо указать полный путь до папки с установленным сервером в Eclipce 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 упакованы core-service, core-api и, реализующий взаимодействие с web-интерфейсом, портлеты 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 портлетов.

Анализ реализации системы

### 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-Кемерера (Chidamber & Kermerer).

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

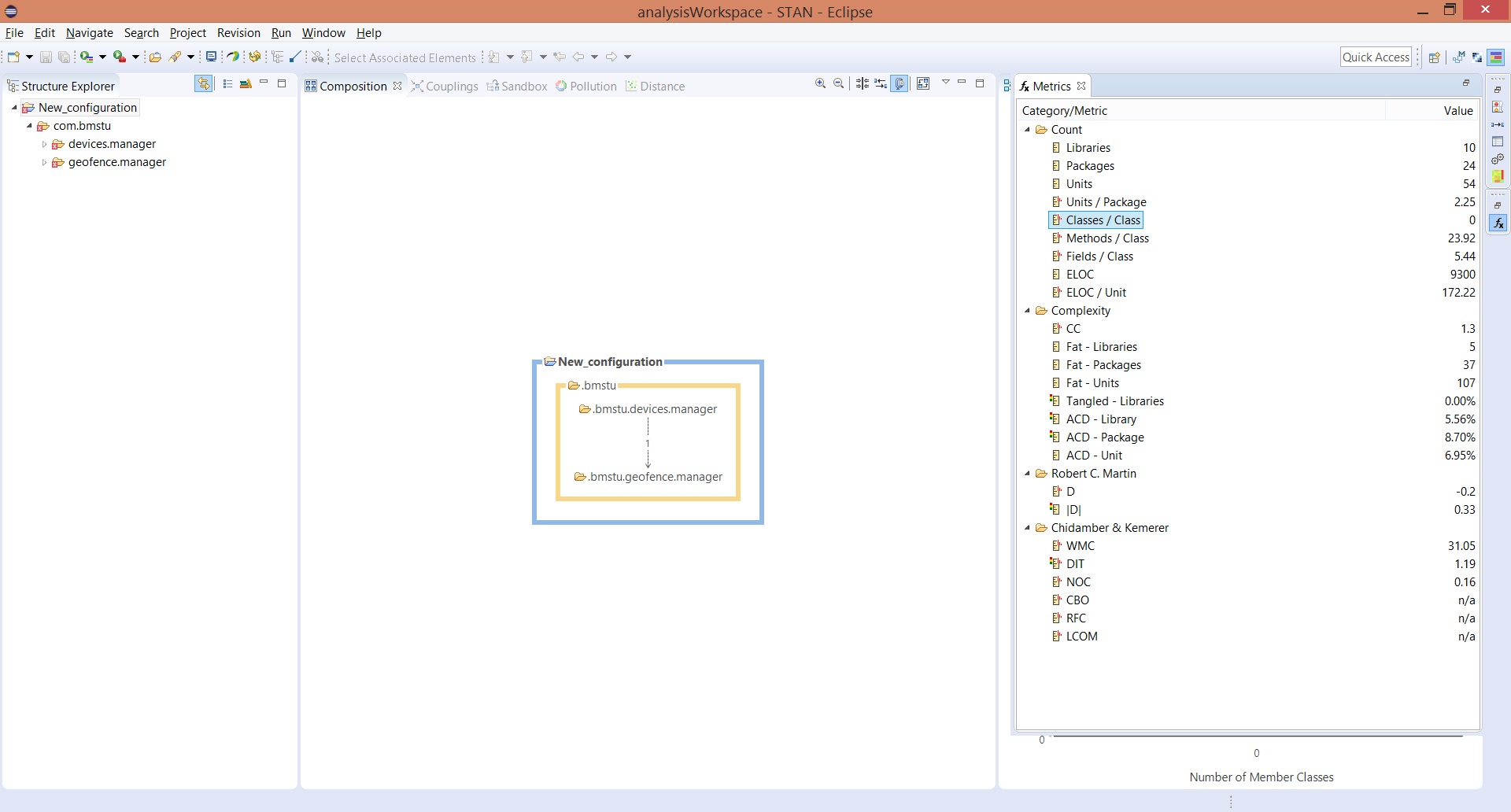
* количество классов верхнего уровня (Unit);
* среднее число внутренних классов на класс (Classes / Class);
* среднее число методов в классе (Methods / Class);
* среднее число полей в классе (Fields / Class);
* число строчек кода (ELOC);
* число строчек кода на модуль (ELOC / Unit).

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).



*Рисунок 7 - 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 и D.

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 7, данное требование удовлетворено, а значит, удовлетворен 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 7, 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 7, D = -0.2, а значит, сборка сбалансирована относительно её абстрактности и стабильности. Таким образом, удовлетворен интерес разработчика к расширяемости проекта.

### Анализ зависимостей в коде системы

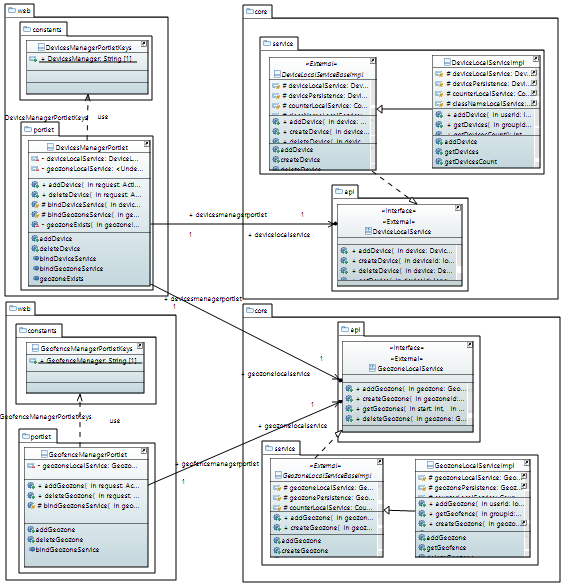
На рисунке 8 представлена диаграмма классов. Диаграмма состоит из четырех основных компонентов: web(com.bmstu.devices.manager.web и com.bmstu.geofence.manager.web) и core(com.bmstu.devices.manager.core и com.bmstu.geofence.manager.core). Как и в описании плагинов и классов (п.2.3), модули core делятся на два модуля: service и api, соответствующие модулям программы. В модуле bmstu.devices.manager.core-service выделены два основных класса, описывающие реализацию функционала локального сервиса: DeviceLocalServiceImpl и DeviceLocalServiceBaseImpl, также в полях этих классов выделены основные методы и атрибуты, которые используются в проекте:

* Атрибуты:
* deviceLocalService
* devicePersistence
* counterLocalService
* resourceLocalService
* и др.
* Методы:
* аddDevice(long userId, other field…, ServiceContext serviceContext)
* getDevice(long deviceId)
* createDevice(long deviceId)
* deleteDevice(long deviceId)
* updateDevice(Device device)

В модуле api на диаграмме показан только основной интерфейс DeviceLocalService, который реализуется классом DeviceLocalServiceBaseImpl, который, в свою очередь, наследуется классом DeviceLocalServiceImpl.

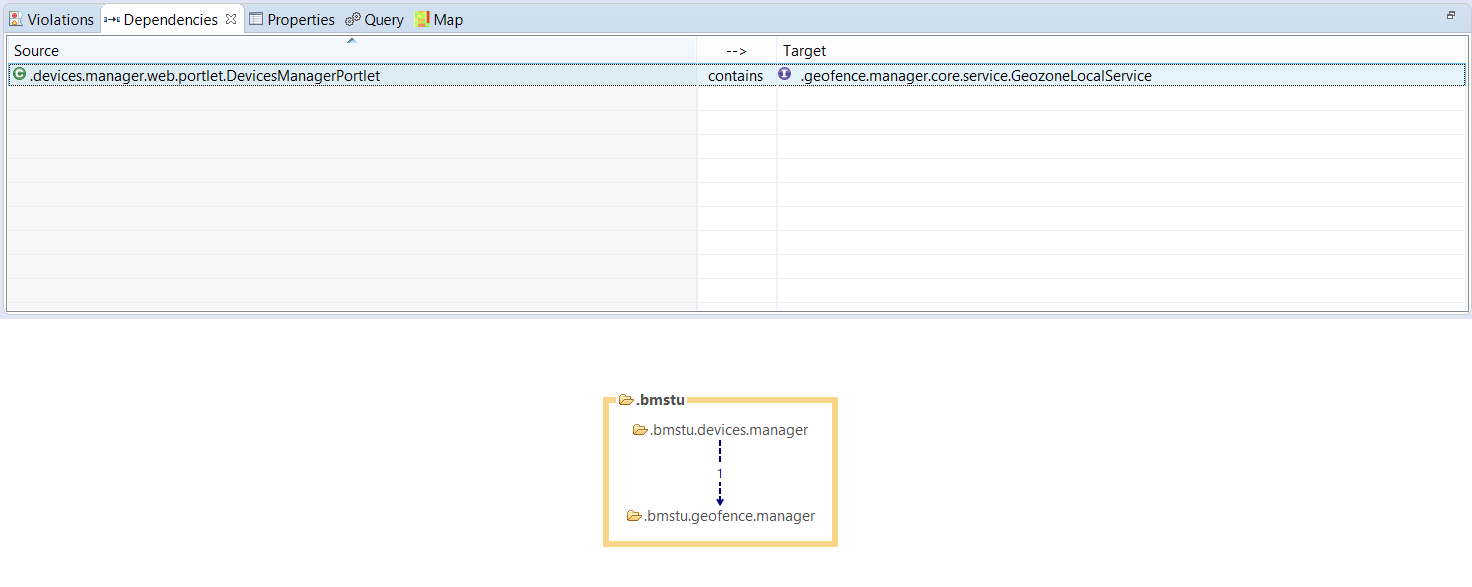
Модули com.bmstu.geofence.manager.core-api и com.bmstu.geofence.manager.core-service реализованы аналогично вышеописанной реализации.

Модуль bmstu.devices.manager.web, отвечающий за UI, полностью представлен двумя составляющими его классами: DevicesManagerPortletKeys в модуле constants, который описывает используемые константы, и класс DevicesManagerPortlet в модуле portlet, который описывает компонент Liferay MVC Portlet. Модуль bmstu.geofence.manager.web реализован аналогично.

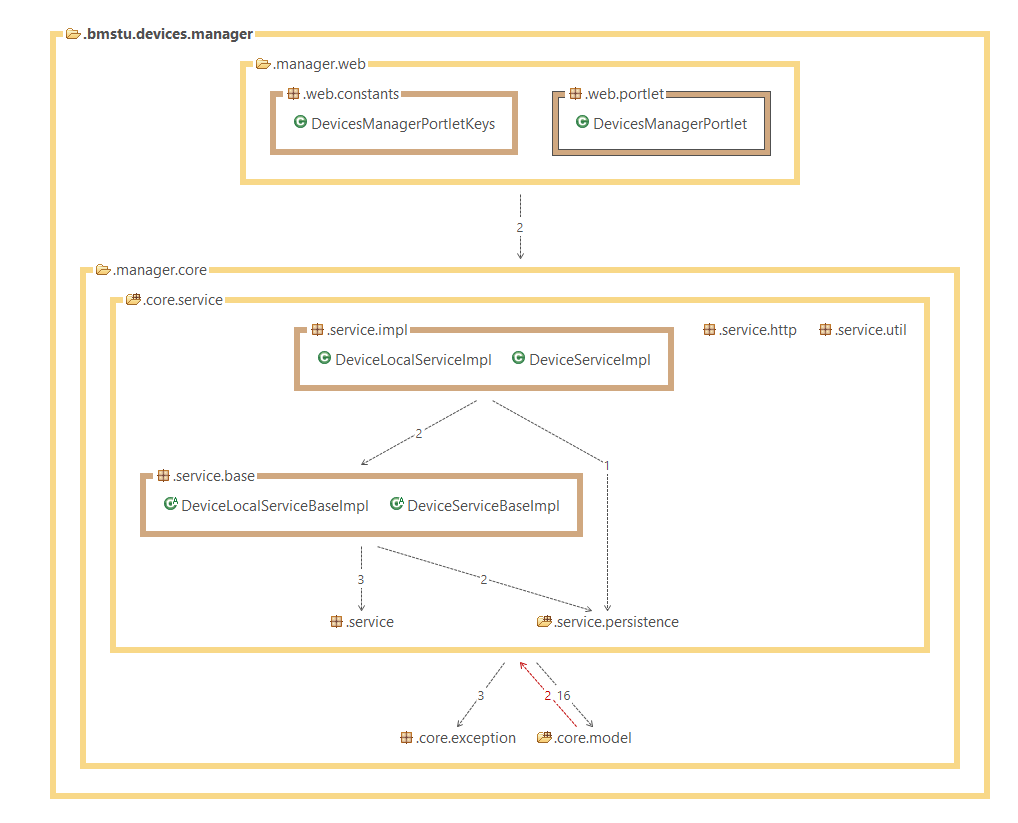


*Рисунок 8 - Диаграмма классов*

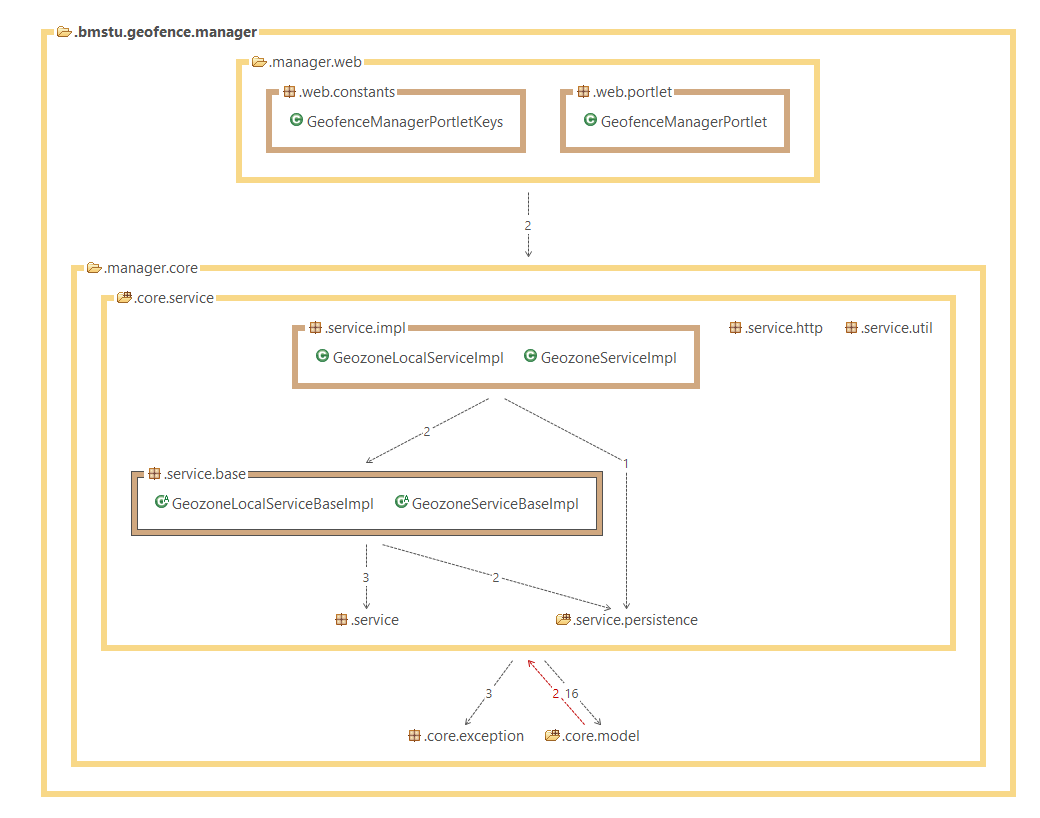
Рисунки 9-11 характеризуют зависимости между пакетами, реализующими портлеты и модели Device и Geozone, отвечающие принципам SOLID: разделенные интерфейсы, зависимости на абстракциях, открытые для расширения модули и др.



*Рисунок 9 - Зависимости между основными модулями*



*Рисунок 10 - Зависимости между основными модулями*



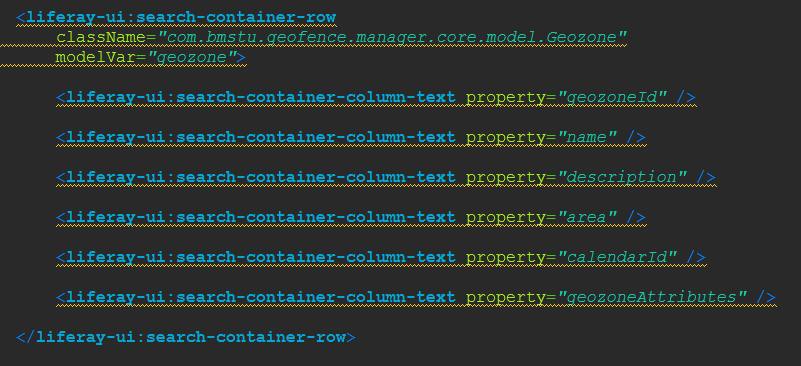
*Рисунок 11 - Зависимости между основными модулями*

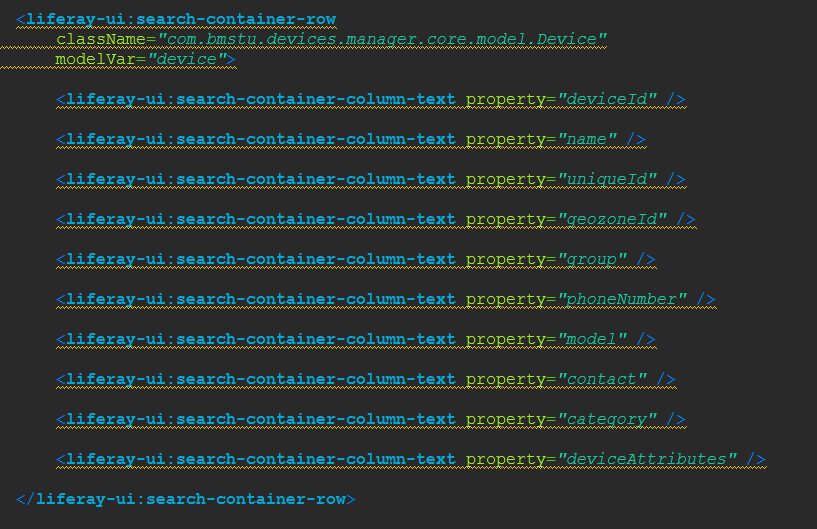
Тестирование на корректность работы

Для проведения тестирования был выбран ручной метод. Для этого будет проверяться выполнение всех требований технического задания (в упрощенном варианте, объединяя схожие компоненты), а именно: корректность добавления нового устройства в соответствии с указанным geozone-id.

### Проверка взаимодействия DeviceManager и GeofenceManager

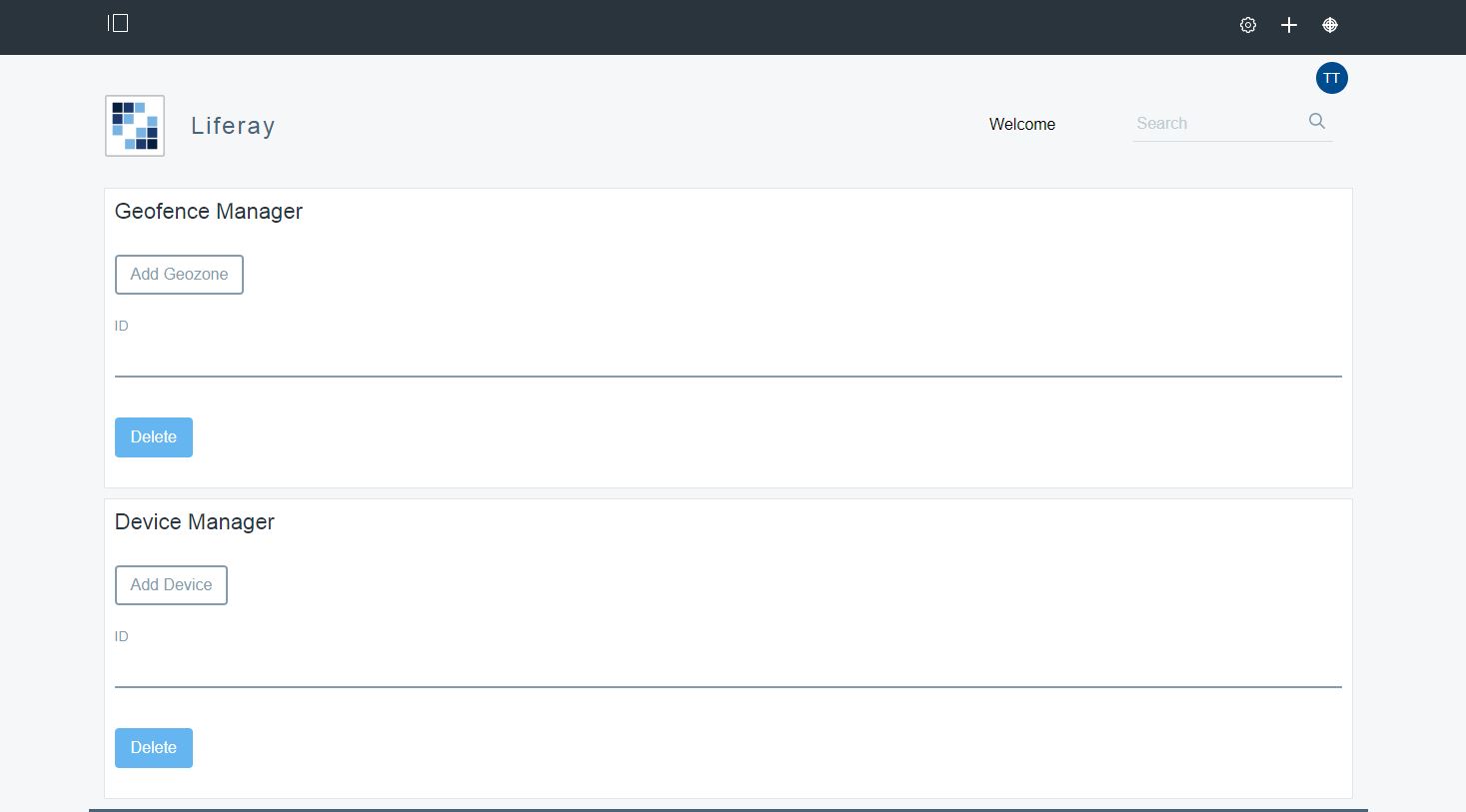
Изначально Service Builder конфигурирует файлы view.jsp, которые определяют структуру виджетов DeviceManager и GeofenceManager (см. рисунки 12-13)

 *Рисунок 12 – структура контейнера geozone*



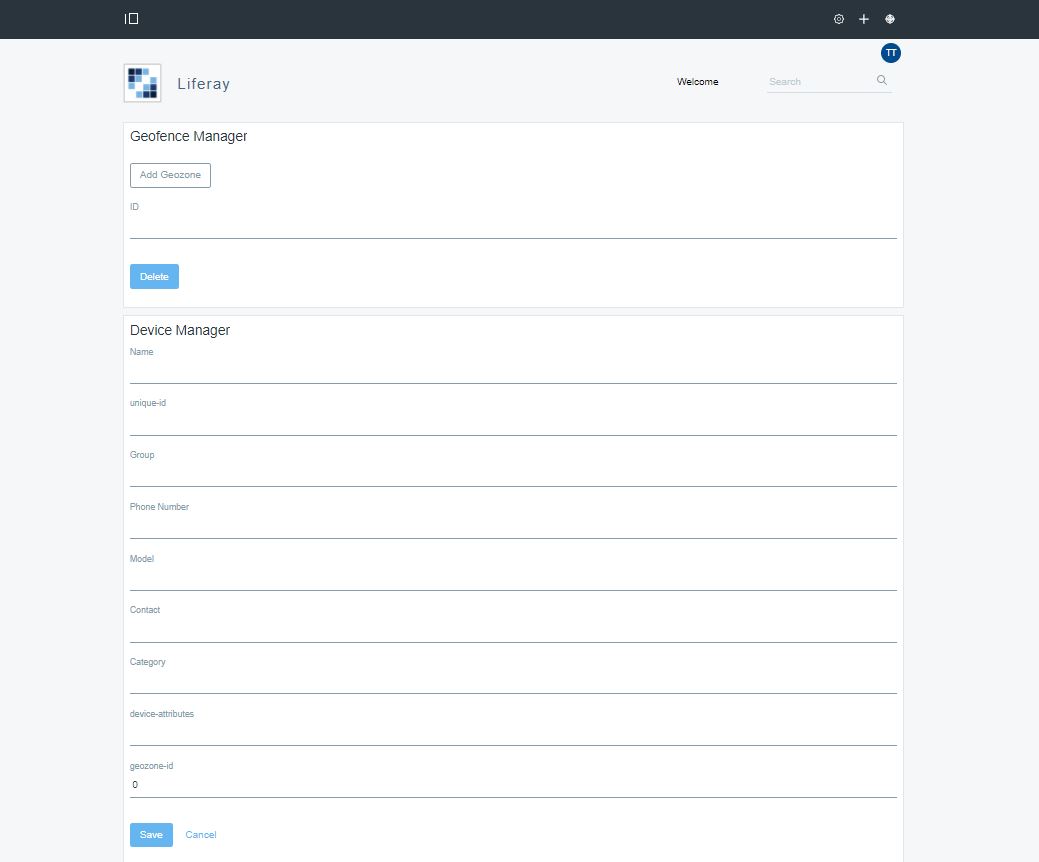
*Рисунок 13 – структура контейнера device*

Первоначальный вид пользовательского интерфейса представлен на рисунке 14 и состоит из двух портлетов:



*Рисунок 14 - Пользовательский интерфейс*

При нажатии кнопки «Add Device» открывается новая JSP страница edit\_device.jsp, которая состоит из полей ввода, определенных структурой контейнера device, (рисунок 15) и двух кнопок: «Save» и «Cancel».



*Рисунок 15 - Интерфейс ввода данных для добавления нового устройства*

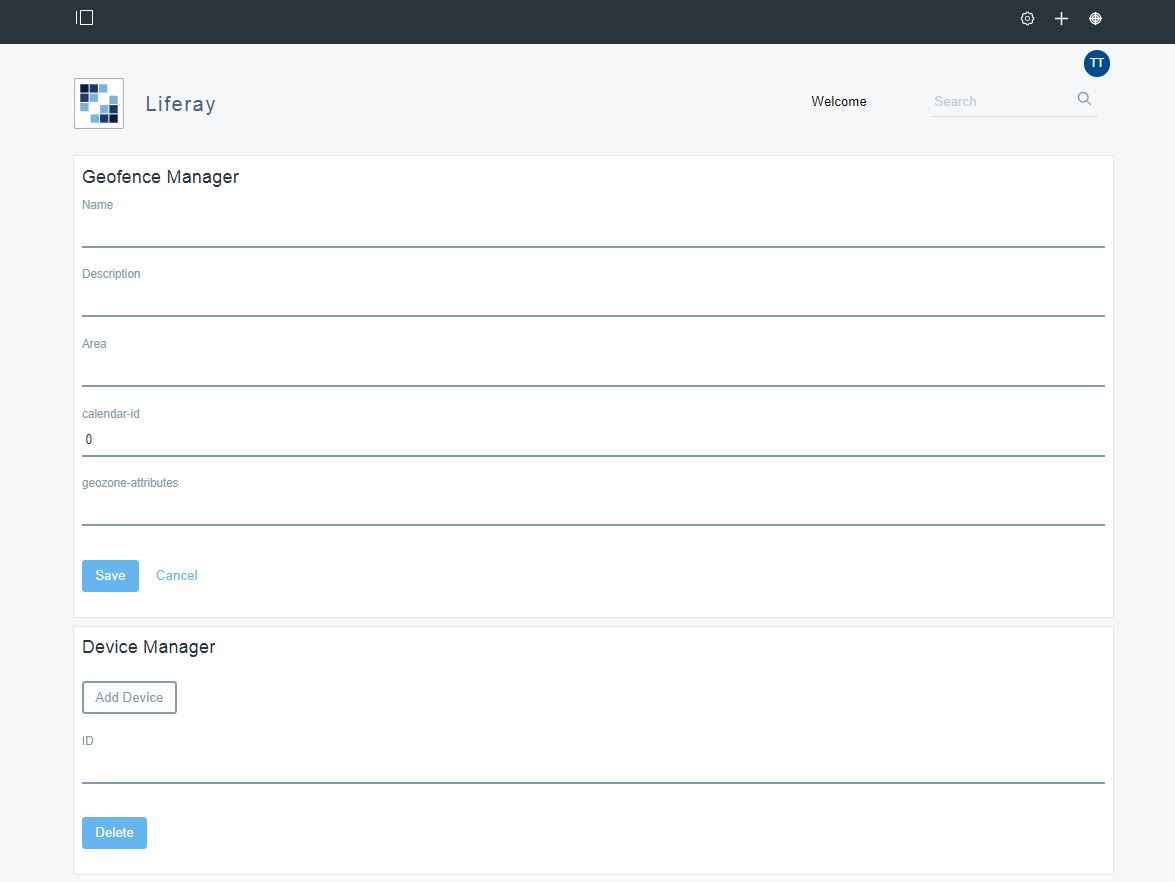
Заполняем поля ввода данных (рисунок 16).



*Рисунок 16 - Заполненная форма ввода данных*

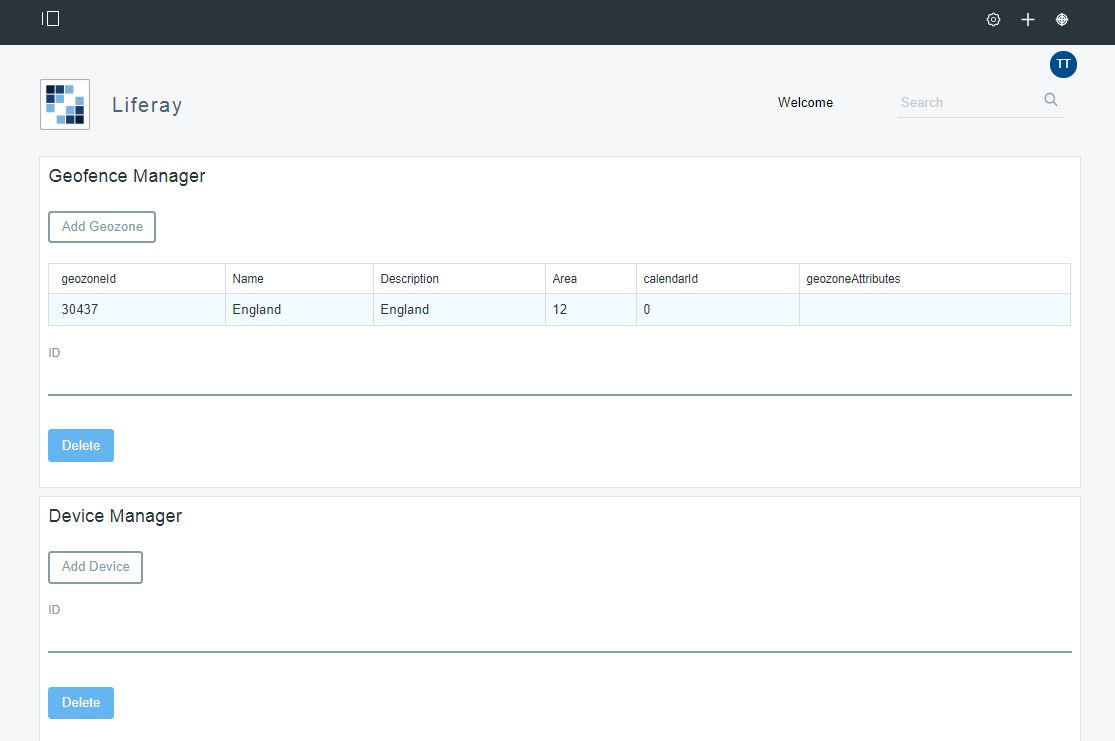
Добавление нового устройства не происходит, так как геозона с указанным ID пока не создана. Выводится соответствующее сообщение об ошибке.

Теперь создадим требуемую геозону. При нажатии на кнопку «Add Geozone» открывается новая JSP страница edit\_geozone.jsp, которая состоит из полей ввода, определенных структурой контейнера geozone, (рисунок 17) и двух кнопок: «Save» и «Cancel».



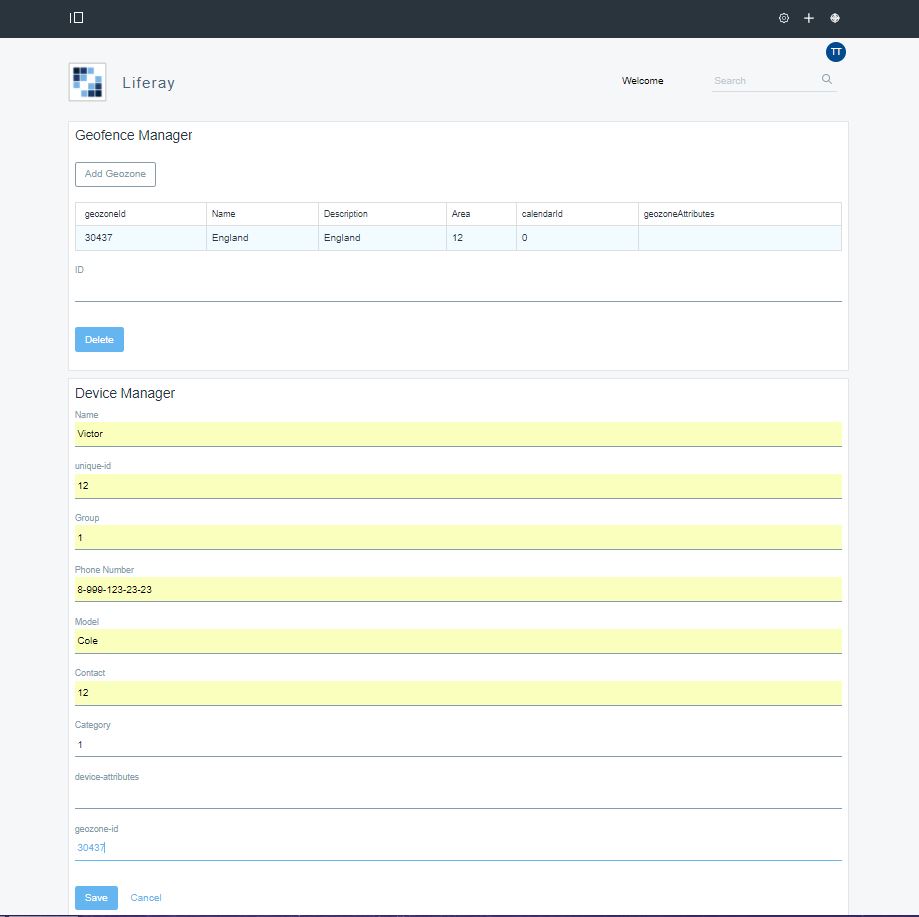
*Рисунок 17 – Интерфейс ввода данных для добавления новой геозоны*

Заполняем поля и нажимаем кнопку «Save» (рисунок 18):



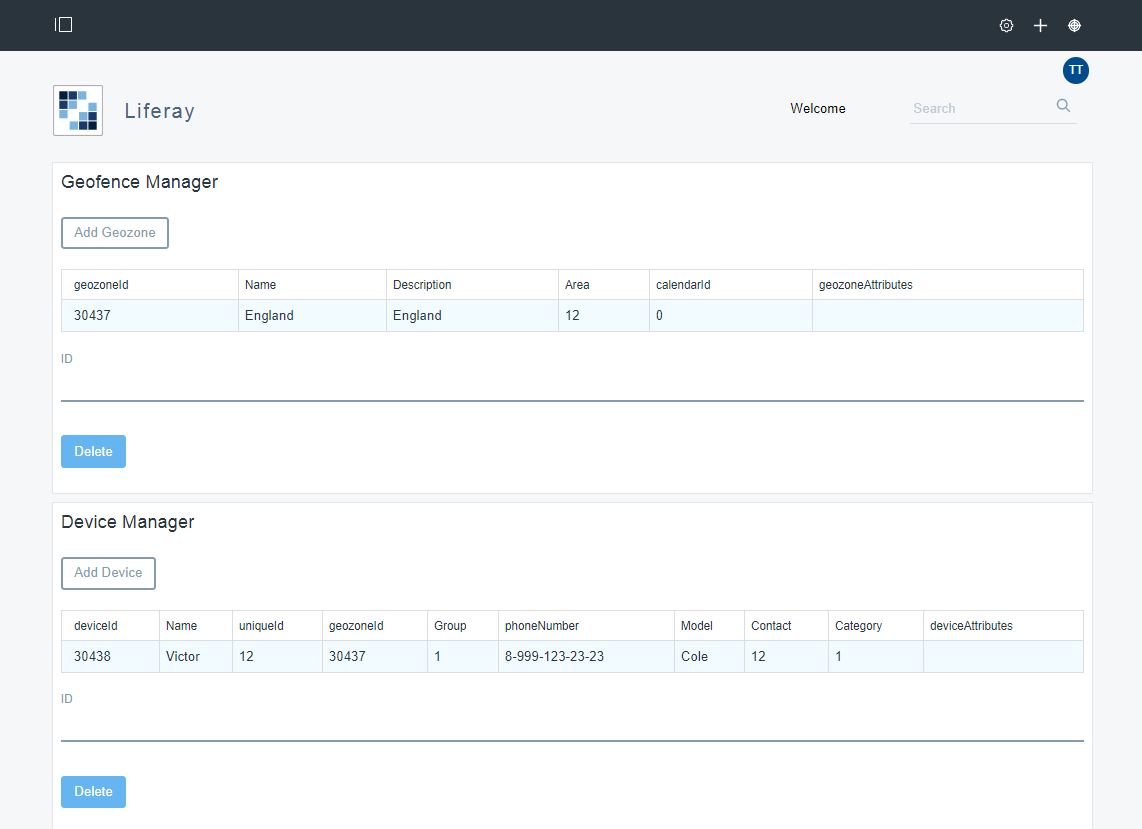
*Рисунок 18 – Добавление новой геозоны*

Теперь, когда в базе данных есть геозона с реальным ID, попроуем добавить устройство для данной геозоны (рисунок 19):



*Рисунок 19 – Добавление устройства для существующей геозоны*

Все поля заполняем аналогично предыдущей попытке добавления устройства, за исключением поля geozone-id – там указываем ID добавленной нами ранее геозоны (30437). Нажимаем кнопку «Save» и видим, что устройство успешно добавлено (рисунок 20):



*Рисунок 20 - Результат добавления устройства с существующим geozone-id*

Таким образом взаимодействие DeviceManager и GeofenceManager реализовано корректно.

# Выводы

В ходе данной курсовой работы веб-сервисы DeviceManager и GeofenceManager и компоненты пользовательского интерфейса системы Traccar были успешно портированы на платформу Liferay с использованием технологии OSGi DS, а также было реализовано взаимодействие данных веб-сервисов.

Реализация данного взаимодействия была протестирована методом ручного тестирования. Данные тесты показали, что взаимодействие реализовано корректно.

Данный проект можно использовать как встраиваемый модуль для более удобного отображения в других системах GPS-трекинга, навигации и т.п. Также данный комплекс из веб-сервисов 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 – ComplexManager [Электронный ресурс]. Режим доступа: <https://github.com/igorechek9012/ComplexManager>

[4] GitHub – DeviceManager [Электронный ресурс]. Режим доступа: <https://github.com/mikleSD/RPO_CW>

[5] GitHub – GroupsManager [Электронный ресурс]. Режим доступа: <https://github.com/vldrshv/RPO_course_work>

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

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

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

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

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