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

Студент группы ИУ3-72 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Нгуен Тху Нга

(подпись, дата)

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

(подпись, дата)

*2018 г.*

Содержание

[1 Исследовательская часть 4](#_Toc505598383)

[1.1 Обзор предметной области 4](#_Toc505598384)

[1.2 Разрабатываемая система как «черный ящик» в окружении внешних систем 4](#_Toc505598385)

[1.3 Техническое задание 6](#_Toc505598386)

[1.4 Интересы заинтересованных сторон по отношению к системе 7](#_Toc505598387)

[2 Конструкторская часть 8](#_Toc505598388)

[2.1 Технические решения, позволяющие удовлетворить интересы заинтересованных сторон 8](#_Toc505598389)

[2.2 Высокоуровневая архитектура системы 9](#_Toc505598390)

[2.2.1 Компонентная декомпозиция системы 9](#_Toc505598391)

[2.2.2 Модульная декомпозиция системы 11](#_Toc505598392)

[2.3 Детальная архитектура системы 12](#_Toc505598393)

[2.3.1 Файловая структура репозитория системы в GitHub 12](#_Toc505598394)

[2.3.2 Плагины и классы системы 13](#_Toc505598395)

[3 Технологическая часть 21](#_Toc505598396)

[3.1 Особенности создания бинарной сборки системы 21](#_Toc505598397)

[3.2 Особенности запуска разработанной системы 22](#_Toc505598398)

[3.3 Анализ реализации системы 22](#_Toc505598399)

[3.3.1 Анализ исходного кода с помощью метрик качества 22](#_Toc505598400)

[3.3.2 Анализ зависимостей в коде системы 26](#_Toc505598401)

[3.4 Тестирование на корректность работы 28](#_Toc505598402)

[3.4.1 Проверка GeofenceManager 29](#_Toc505598403)

[3.4.2 Проверка уникальности первичного ключа geozoneId 31](#_Toc505598404)

[3.4.3 Корректный ввод данных и добавление нового устройства 32](#_Toc505598405)

[4 Выводы 34](#_Toc505598406)

[Список используемой литературы 35](#_Toc505598407)

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

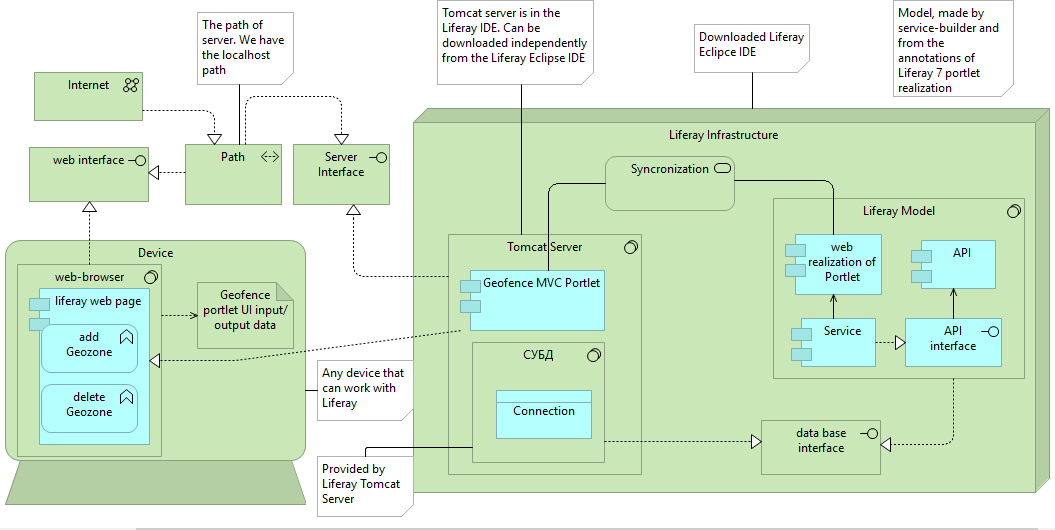
LiferayPortal — программный продукт, представляющий собой корпоративный портал, то есть решение, предназначенное для централизованного доступа к нескольким различным корпоративным приложениям в одном месте. Liferay иногда описывается как система управления содержимым (CMS) или платформу для веб-приложений. 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 определитьее 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.

Для понимания реализации системы необходимо понять принципы ее работы. Так как Liferay представляет собой портал с множеством приложений, с которым взаимодействует пользователь, то важной частью системы будет внешняя UI-оболочка. С помощью нее пользователь будет вводить данные реализуемого нами менеджера. Далее эти данные будут поступать в компонент, который будет реализовывать заполнение стандартной модели полями, которые были взять с сайта с помощью request-методов. Далее эти данные будут загружаться в базу данных, доступ к которой будет предоставляться по интерфейсу взаимодействия с базой данных.



*Рисунок 1 - Система в окружении смежных систем*

Техническое задание

Портирование веб-сервиса GeofenceManager 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.

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 портлета | П1 Удобное отображение всех геозон.  П2 Удобное взаимодействие с базой данных геозон. |
| Разработчик портлета (новый разработчик) | Р1 Использование распространенных средств  разработки. |
| Владелец опенсорсного проекта (project owner) | В1 Быстрая и полная передача исходного кода, настроек, документов.  В2 Возможность в дальнейшем расширять систему, например, добавляя методы изменения полей  геозон, 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 геозон.  П2 Удобное взаимодействие с базой данных геозон. | Визуализатор данных из БД.  Эргономичный (удобное) интерфейс реализованных функций взаимодействия с базой данных геозон. |
| Р1 Использование распространенных средств  разработки. | Хорошо документированный код, наличие обучений и файлов конфигураций. |
| В1 Быстрая и полная передача исходного кода, настроек, документов. | Код и настройки разрабатываемой системы будут находиться в системе GitHub. Контроль версий будет производится с использованием системы Git. |

|  |  |
| --- | --- |
| В2 Возможность в дальнейшем расширять систему, например, добавляя методы изменения полей геозон, добавленных в базу данных. | Будет использоваться система бинарной сборки Gradle.  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

Для того, чтобы разобраться с реализацией системы, необходимо понять, какие поля используются в Traccar Geofence для реализации модели и передачи данных на сервер. После изучения Geofence.java был сделан вывод, что необходимые поля для хранения модели типа Geofence будут следующие:

* + - name типа String;
    - description типа String;
    - area типа String;
    - calendarId типа long;
    - geozoneAttributes типа String.

Для реализации модели нужны методы добавления данных в БД, удаление данных из БД по уникальному идентификатору, проверка на ввод пустых значений в поля ввода данных.

Разрабатываемая система состоит из следующих основных компонентов:

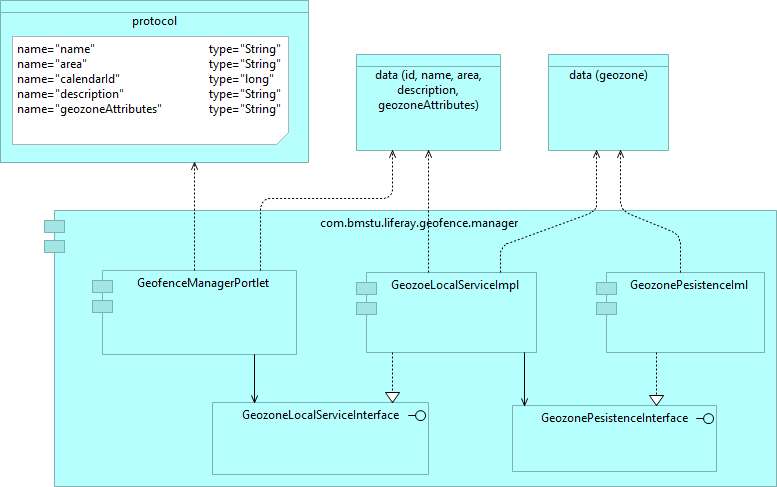
* GeozonePersistenceImpl– компонент, обеспечивающий прямой доступ к базе данных для методов CRUD (create, read, update, delete —четыре базовые функции, используемые при работе с персистентными хранилищами данных).
* GeozoneLocalServiceImpl – реализация локального сервиса – один из главных классов в проекте, определяющий функционал работы серверной части. В этом классе можно добавлять пользовательскую бизнес-логику.
* GeozoneLocalServiceBaseImpl – абстрактный класс, характеризующий реализацию базы сервиса, может применяться в качестве альтернативы GeozoneLocalServiceImpl.
* GeofenceManagerPortlet – непосредственно, сам портлет, в котором реализованы два метода:
* addGeozone – методдобавленияустройствавбазуданных;
* deleteGeozone – методудаленияустройстваизбазуданных;
* 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).

На сервере реализован портлет, который использует протокол передачи данных в виде набора данных id, area, calendarId, name и attributes. С помощью функции добавление геозоны будут передаваться данные в функцию реализации взаимодействия с базой данных, чтобы сформировать объект

данных, который инкапсулирован в базу данных. Таким образом, необходимо заполнить поля объекта типа Geozone и добавить это значение в базу данных.

**

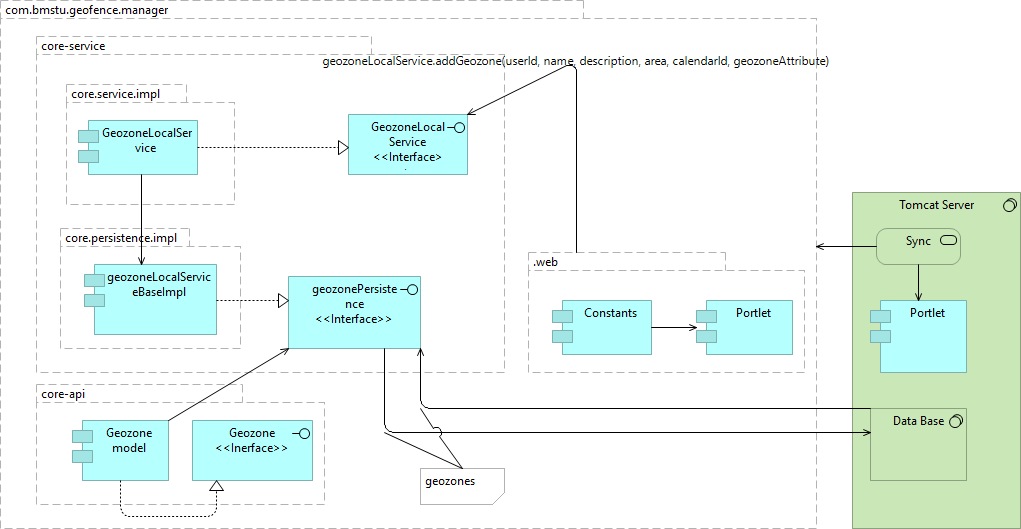
*Рисунок 2 - Диаграмма компонентов*

### Модульная декомпозиция системы

Портированный GeofenceManager можно разделить на три основных группы модулей:

* Внешние классы и интерфейсы Liferay (com.liferay.portal.kernel): MVCPortlet, PersistedModelLocalService, LocalServiceBase, LocalServiceBaseImpl, 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: GeofenceManagerPortlet, GeozoneLocalService, GeozoneLocalServiceImpl, GeozoneLocalServiceBaseImpl, GeozoneLocalServiceUtil, GeozonePersistance, GeozonePersistanceImpl.
* 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*

Детальная архитектура системы

### Файловая структура репозитория системы в GitHub

Разработанный проект расположен в репозитории GitHub [3].

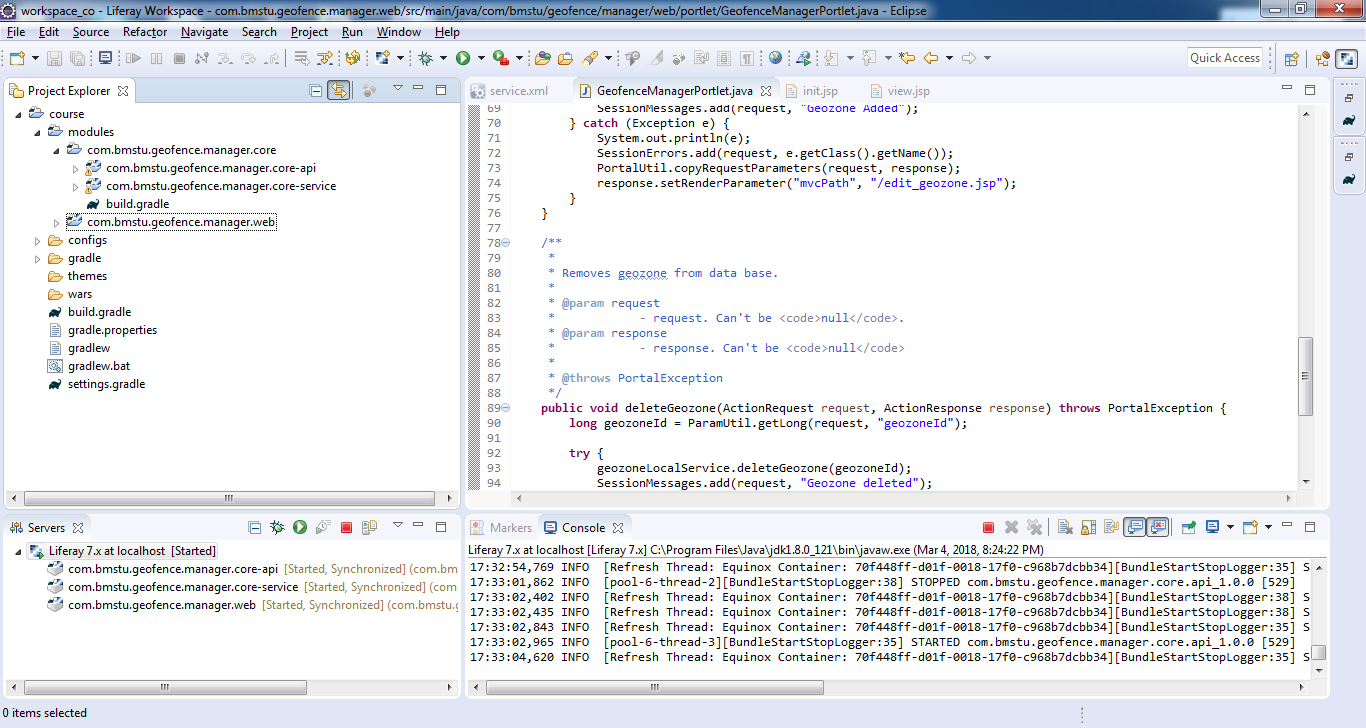
Структура проекта следующая:

1. Configs – папка конфигурации Liferay, с помощью которой Liferay IDE настраивает запуск системы;
2. Gradle/wrappew – 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;
3. 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 запускать сборку с Gradle без необходимости установки самого Gradle в систему.

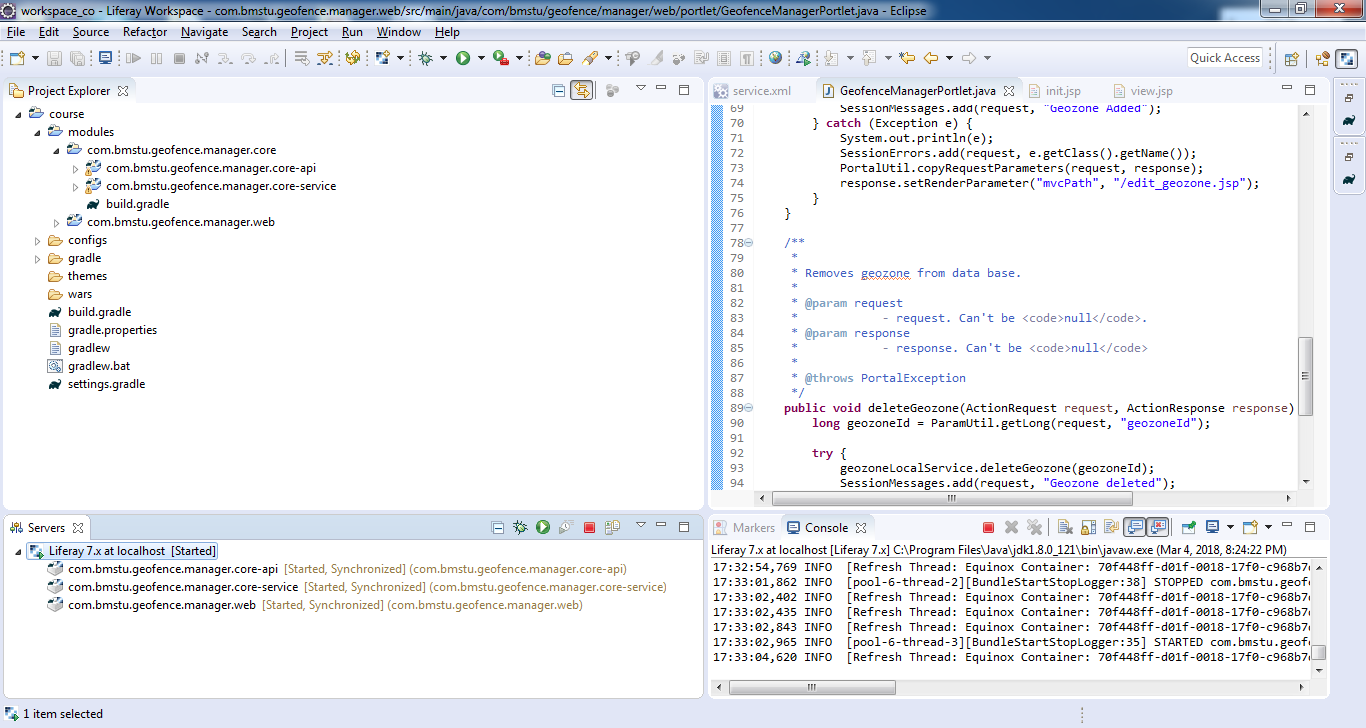
### Плагины и классы системы

Проект состоит из двух основных модулей: com.bmstu.geofence.manager.core – модуль, где описаны интерфейсы взаимодействия, сервисы, структуры для работы с базами данных и т.д., и com.bmstu.geofence.manager.web – где описан функционал портлета веб-приложения и структура пользовательского интерфейса. В свою очередь, модуль com.bmstu.geofence.manager.core разбивается на com.bmstu.geofence.manager.core-api, который содержит API для проекта, и com.bmstu.geofence.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.



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

ServiceBuilder 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. ServiceBuilder берет XML-файл в качестве входных данных и генерирует необходимую модель, уровни обслуживания для приложения. ServiceBuilder 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.

ServiceBuilder 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=«Geozone»
* Первичный ключ для таблицы – geozoneId
* 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 GeofenceManager):
* name
* area
* description
* calendarId
* geozoneAttributes
* Теги finder для генерации поисковых методов по заданным полям:
* по полю groupId
* по полю geozoneId

#### Модуль core-api

Модуль содержит API для разрабатываемого проекта. Все классы и интерфейсы а 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
* GeozonePersistence – интерфейс, который определяет методы CRUD, такие, как create, remove, countAll, find, findAll и т.д.
* GeozoneUtil – класс, который обертывает GeozonePersistenceImpl 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 GeozoneLocalServiceUtil или GeozoneServiceUtil
* Local Service
* GeozoneLocalService – локальный интерфейс обслуживания – главное звено в проекте.
* GeozoneLocalServiceUtil – локальный класс утилиты, который обертывает GeozoneLocalServiceImpl и служит в качестве основной локальной точки доступа к сервисному уровню.
* GeozoneLocalServiceWrapper – оболочка локального сервиса, которая реализует GeozoneLocalService. Этот класс предназначен для расширения и позволяет разработчикам настраивать локальные службы.
* Remote Service
* GeozoneService – интерфейс удаленного сервиса.
* GeozoneServiceUtil – утилита, которая обертывает GeozoneServiceImpl и служит основной точкой удаленного доступа к сервисному уровню.
* GeozoneServiceWrapper – оболочка удаленного сервиса, которая обертывает GeozoneService. Этот класс предназначен дял расширения и позволяет разработчикам настраивать службы удаленного объекта.
* GeozoneSoap – базовая реализация модели SOAP (SimpleObjectAccessProtocol — простой протокол доступа к объектам).
* Model
* GeozoneModel – интерфейс базовой модели. Этот интерфейс и его реализация (GeozoneModelImpl) служат только контейнером для объектов со свойствами по умолчанию.
* Geozone – интерфейс, который расширяется GeozoneModel.
* GeozoneWrapper – оболочка Geofence.

#### Модуль core-service

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

Рассмотрим структуру данного модуля:

* Persistence
* GeozonePersistenceImpl – класс реализации GeozonePersistence.
* Local Service
* GeozoneLocalServiceImpl – реализация локального сервиса – один из главных классов в проекте, определяющий функционал работы серверной части. В этом классе можно добавлять пользовательскую бизнес-логику.
* GeozoneLocalServiceBaseImpl – абстрактный класс, характеризующий реализацию базу сервиса.
* Remote Service
* GeozoneServiceImpl – реализация интерфейса удаленного сервиса. Здесь можно написать код, который добавляет дополнительные проверки безопасности и вызывает локальные сервисы. Для любых настраиваемых методов, добавленных здесь, ServiceBuilder добавляет соответствующие методы в GeozoneService интерфейс при следующем запуске
* GeozoneServiceBaseImpl – абстрактный класс удаленного сервиса.
* Model
* GeozoneModelImpl – реализация интерфейса базовой модели.
* GeozoneImpl – 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, ServiceBuilder добавляет соответствующие методы в Geofence интерфейс при следующем запуске.

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

#### 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 GeofenceManagerPortletKeys с описанием констант, которые используются в портлете, в данном случае, только одной константы – public static final String GeofenceManager = “GeofenceManager”, используемой для задание пути.
* portlet – собственно, описание самого портлета: функциональное описание используемых методов. Данный модуль содержит класс GeofenceManagerPortlet, в котором определены два метода:
* public void addGeozone (ActionRequest request, ActionResponse response) – методдобавленияустройствавбазуданных;
* public void deleteGeozone (ActionRequest request, ActionResponse response) – методудаленияустройстваизбазуданных;

Также здесь используются OSGI-аннотации создания компонента портлета, т.к. класс GeofenceManagerPortlet 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, определен и сам метод привязки сервиса – protected void bindGeozoneService(GeozoneLocalService geozoneLocalService), который определен в классе GeofenceManagerPortlet.

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 GeofenceManagerPortlet.
* edit\_geozone.jsp – используется для описания UI на странице ввода данных для добавления нового устройства, которая вызывается после нажатия кнопки «Add Geozone».

# 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 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 Geozone.

Особенности запуска разработанной системы

После того, как была осуществлена сборка проекта, необходимо загрузить и установить сервер ApachiTomcat. После скачивания сервера необходимо указать полный путь до папки с установленным сервером в EclipceServers. 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 Liferay 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.

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:

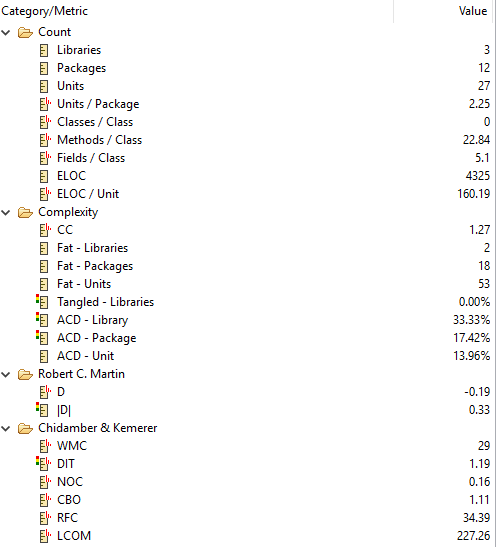
* количество классов верхнего уровня (Unit);
* среднее число внутренних классов на класс (Classes / Class);
* среднее число методов в классе (Methods / Class);
* среднее число полей в классе (Fields / Class);
* число строчек кода (ELOC);
* число строчек кода на модуль (ELOC / Unit).

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 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, 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 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.

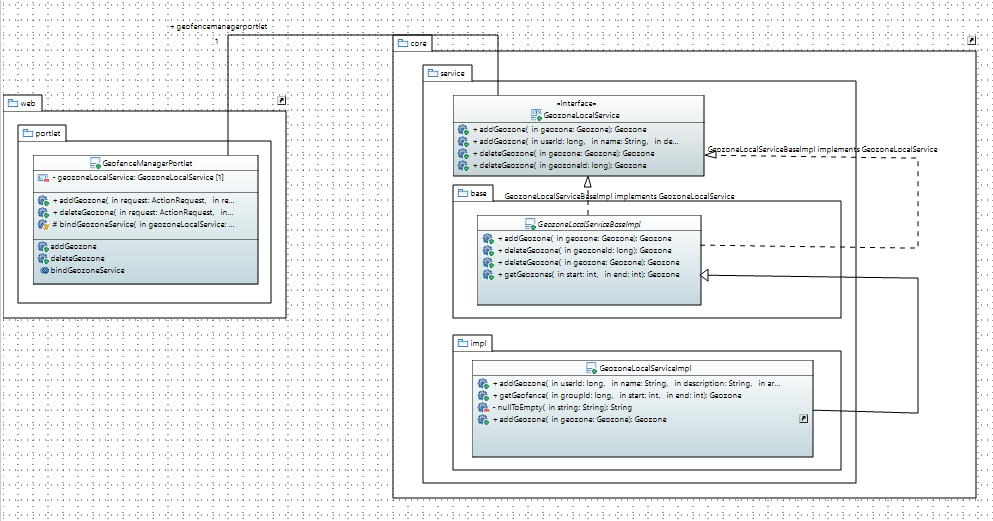
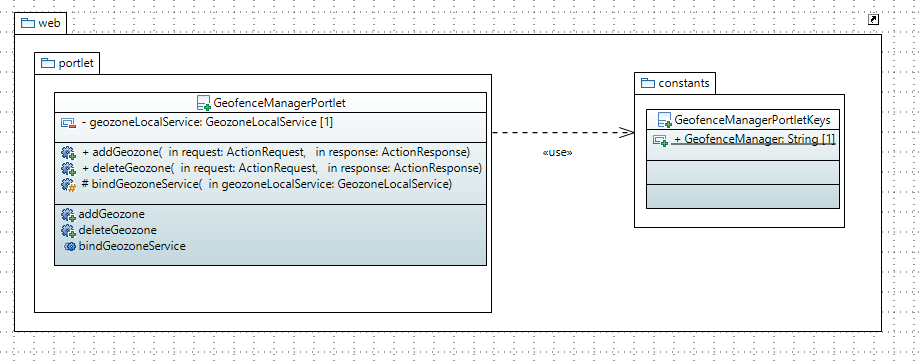
### Анализ зависимостей в коде системы

На рисунке 7 представлена диаграмма классов. Диаграмма состоит из двух основных компонентов: web(com.bmstu.geofence.manager.web) и core(com.bmstu.geofence.manager.core). Как и в описании плагинов и классов (п.2.3), модуль core делиться на два модуля: service и api, соответствующие модулям программы. В модуле service выделены два основных класса, описывающие реализацию функционала локального сервиса: GeozoneLocalServiceImpl и GeozoneLocalServiceBaseImpl, также в полях этих классов выделены основные методы и атрибуты, которые используются в проекте:

* Атрибуты:
* GeozoneLocalService
* GeozonePersistence
* counterLocalService
* resourceLocalService
* и др.
* Методы:
* аddGeozone(long userId, other field…,ServiceContextserviceContext)
* getGeozone(long geozoneId)
* createGeozone(long geozoneId)
* deleteGeozone(long geozoneId)
* updateGeozone(Command geozone)

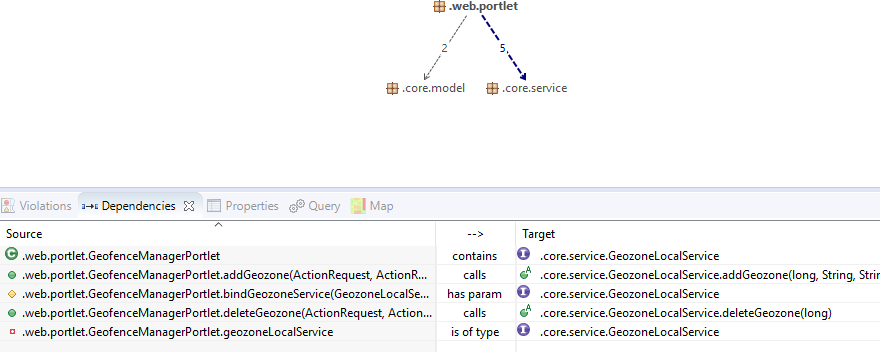
В модуле api на диаграмме показан только основной интерфейс GeozoneLocalService, который реализуется классом GeozoneLocalServiceBaseImpl, который, в свою очередь, наследуется классом GeozoneLocalServiceImpl.

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

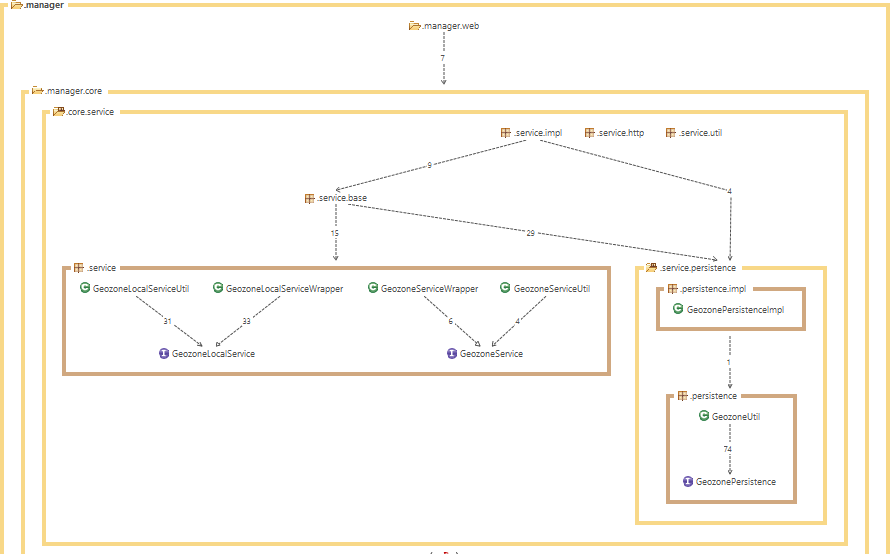


*Рисунок 7 - Диаграммы классов*

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



*Рисунок 8 - Зависимости между основными модулями(\*.web от \*.core.service)*



*Рисунок 9 - Зависимости между основными модулями(\*.web и подмодулей*

*\*.core)*

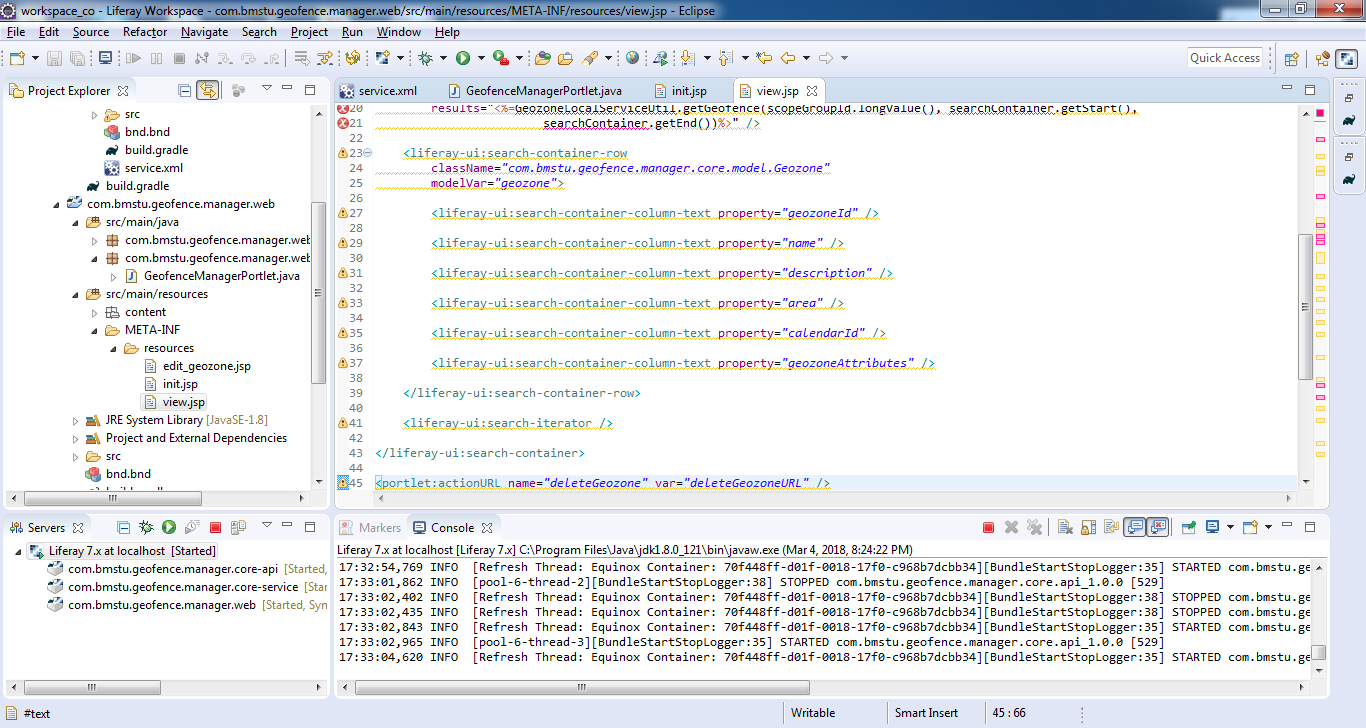
Тестирование на корректность работы

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 устройства;
* Проверка уникальности первичного ключа geozoneId;
* Корректный ввод данных и удаление устройства;

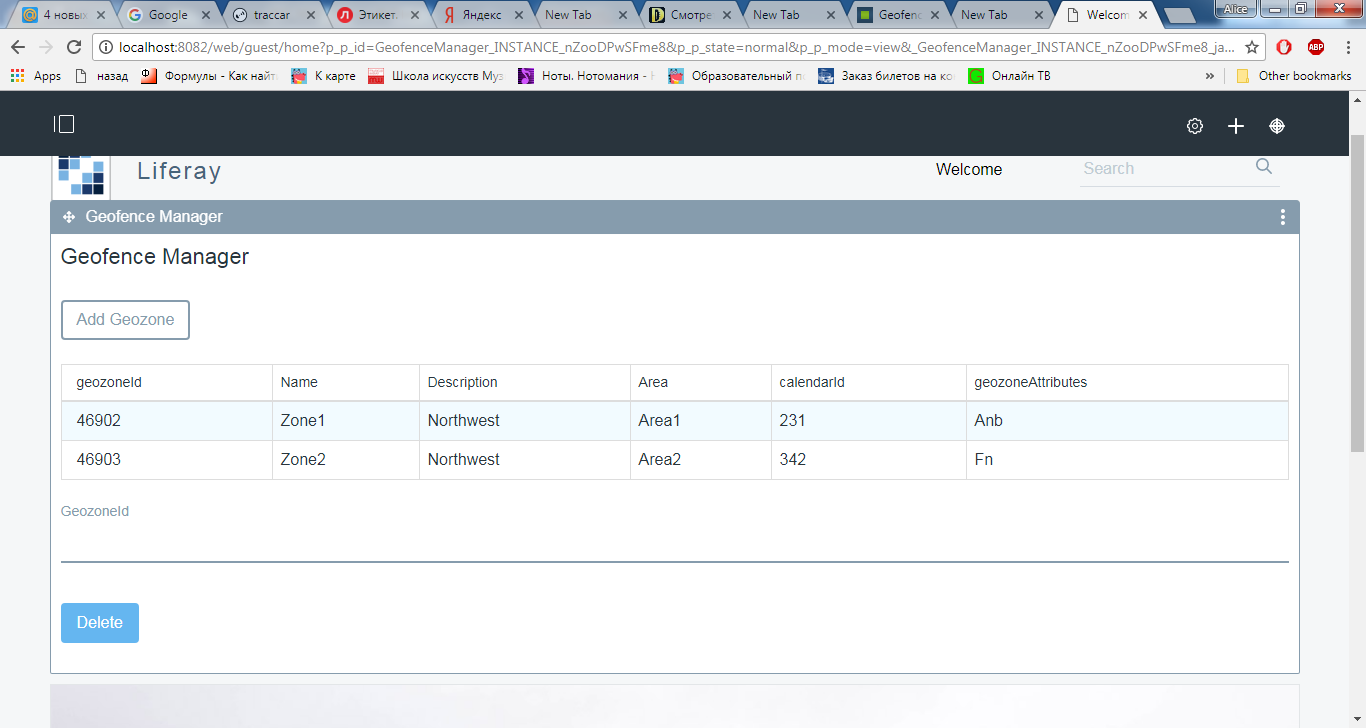
### Проверка GeofenceManager

Изначально ServiceBuilder 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 устройства по полю geozoneId. Структура контейнера поиска представлена на рисунке 10.



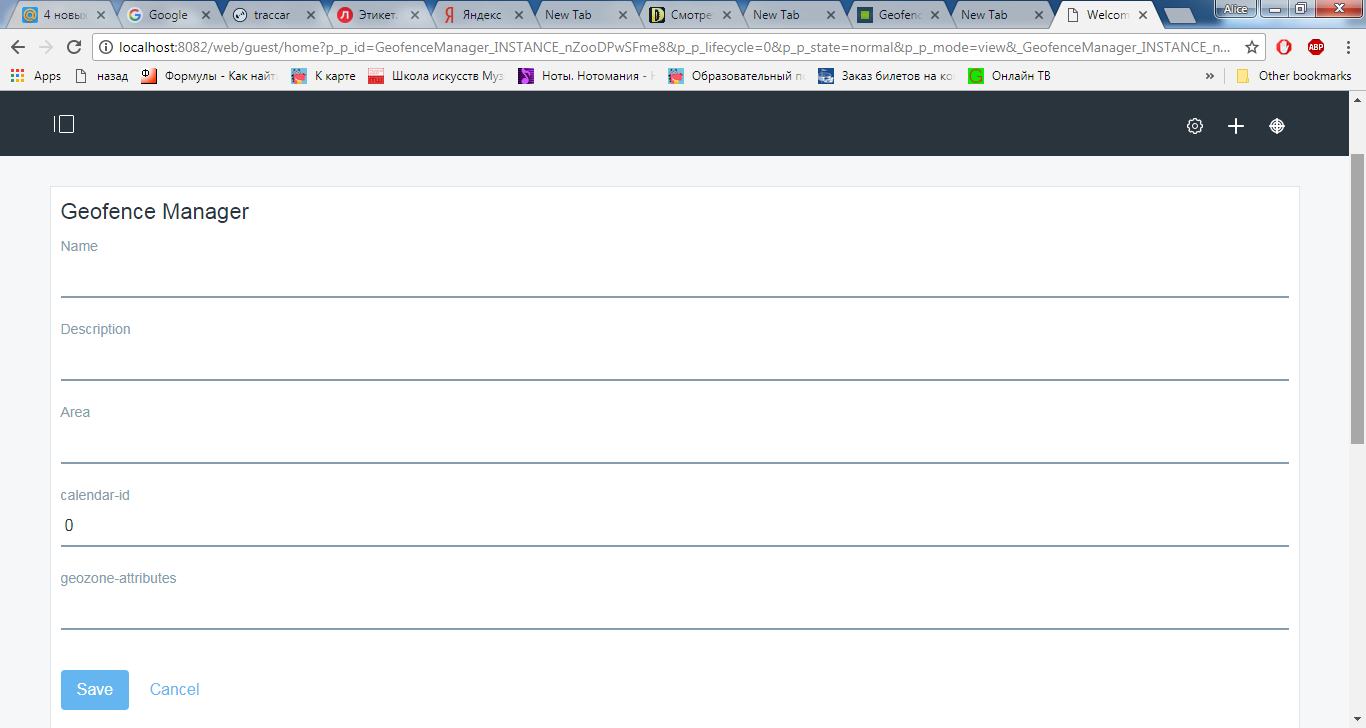
*Рисунок 10 - Структура контейнера*

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



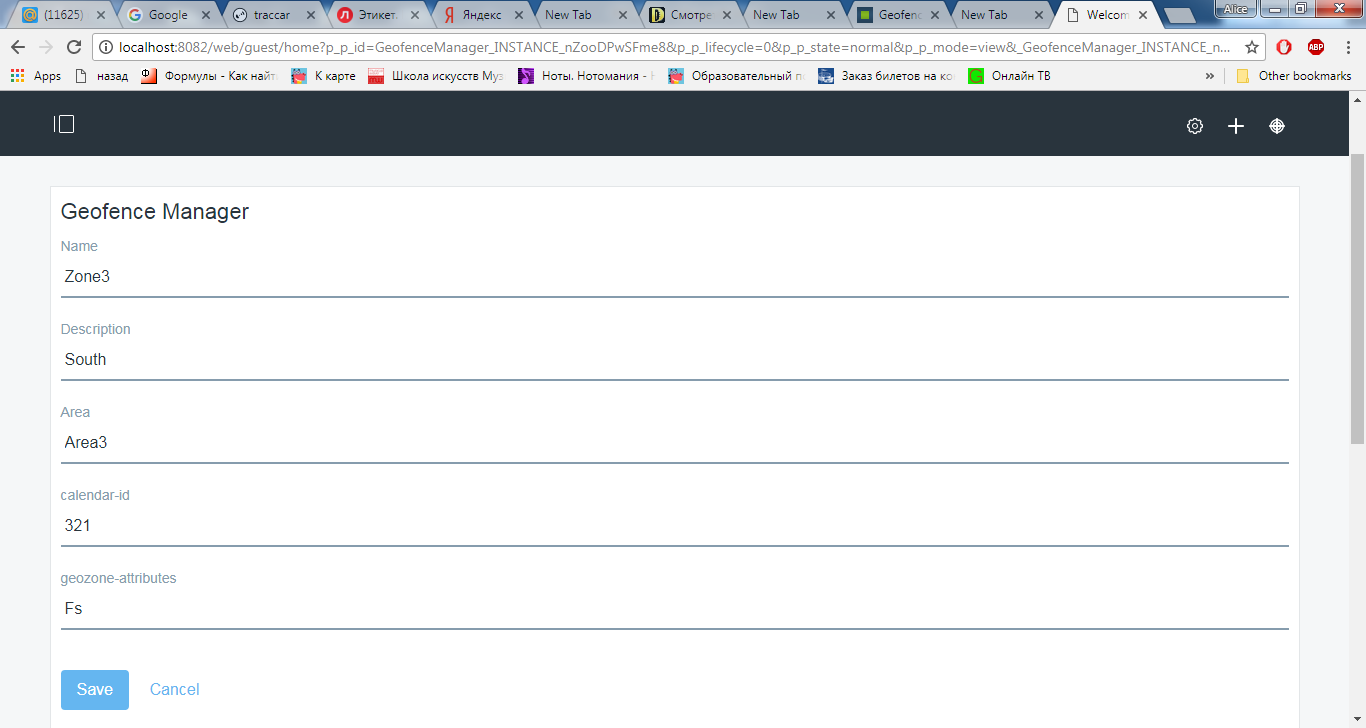
*Рисунок 11 - Пользовательский интерфейс*

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



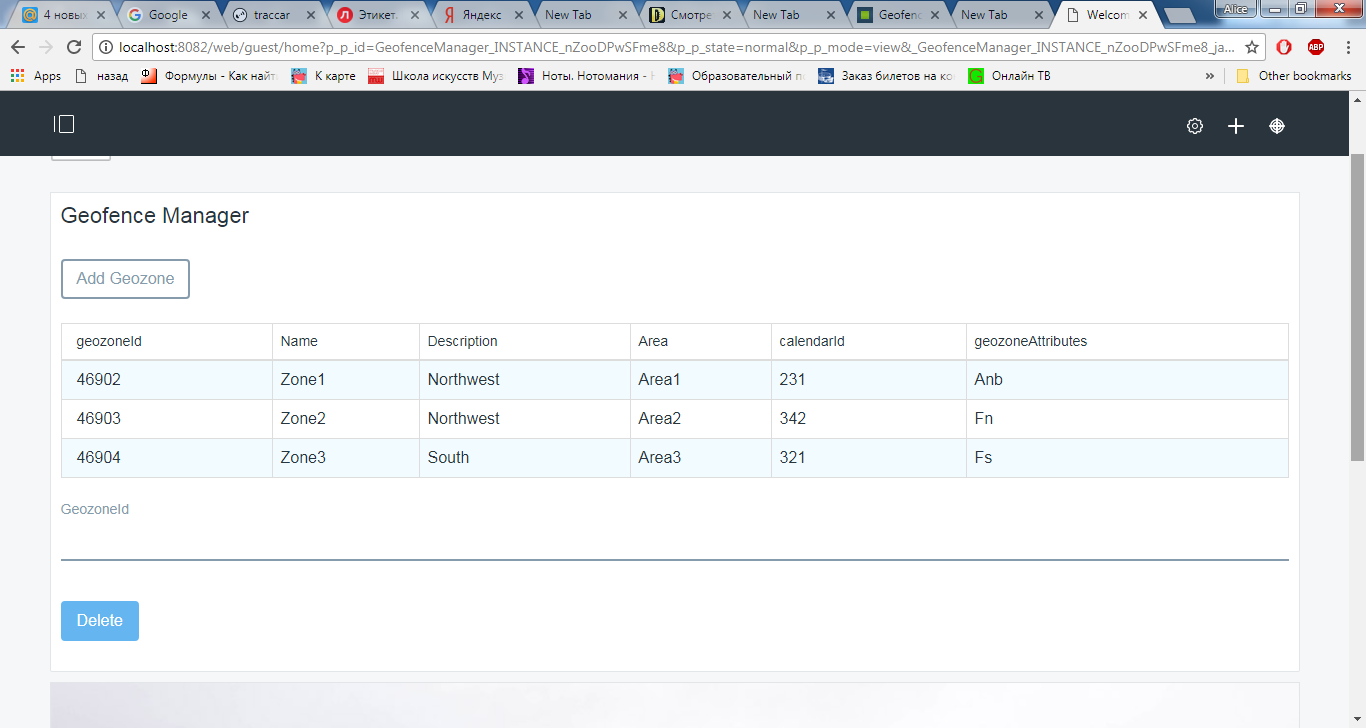
*Рисунок 12 - Интерфейс ввода данных для добавления нового геозона*

Заполняем поля ввода данных (рисунок 13).



*Рисунок 13 - Заполненная форма ввода данных*

Нажимаем кнопку «Save» (рисунок 14):

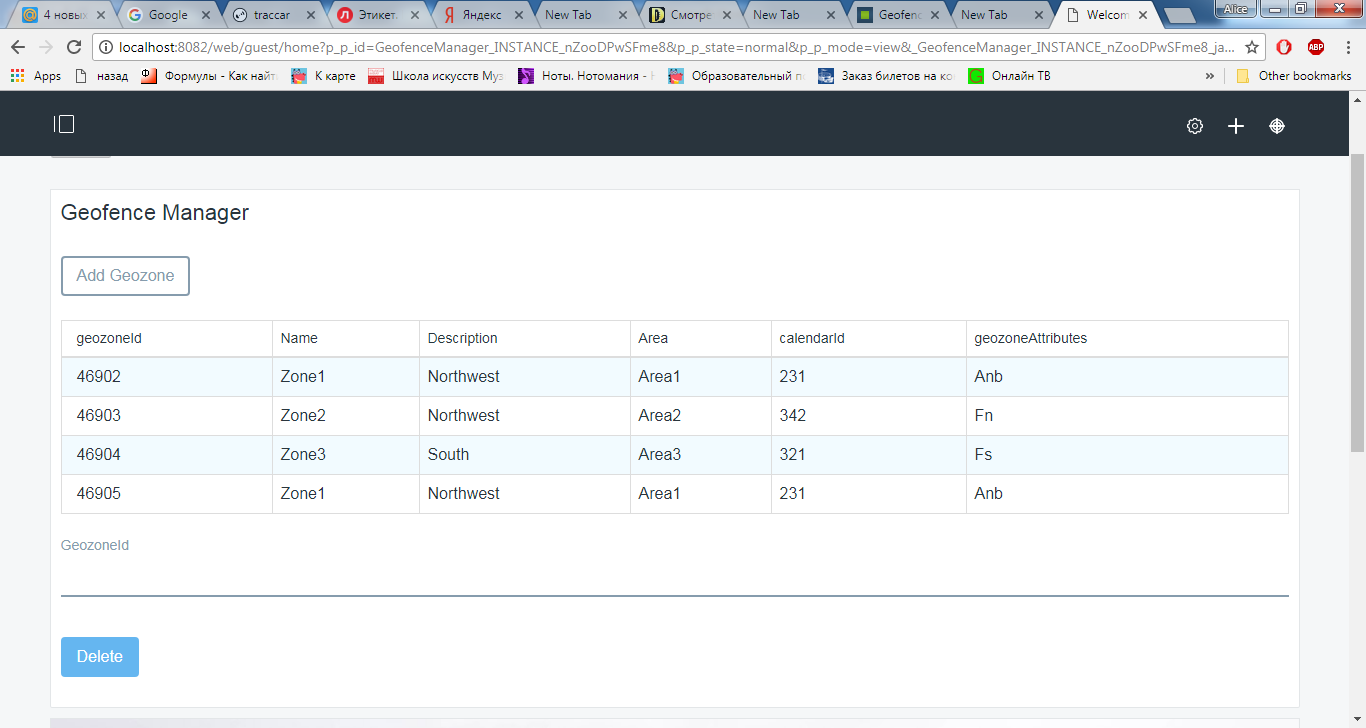


*Рисунок 14 - Результат добавления geozone*

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

### Проверка уникальности первичного ключа geozoneId

Для проверки уникальности генерируемого ключа создадим абсолютно такую же строку данных, как и в п.3.4.1. Ключ должен быть уникальным, несмотря на полное совпадение всех полей.

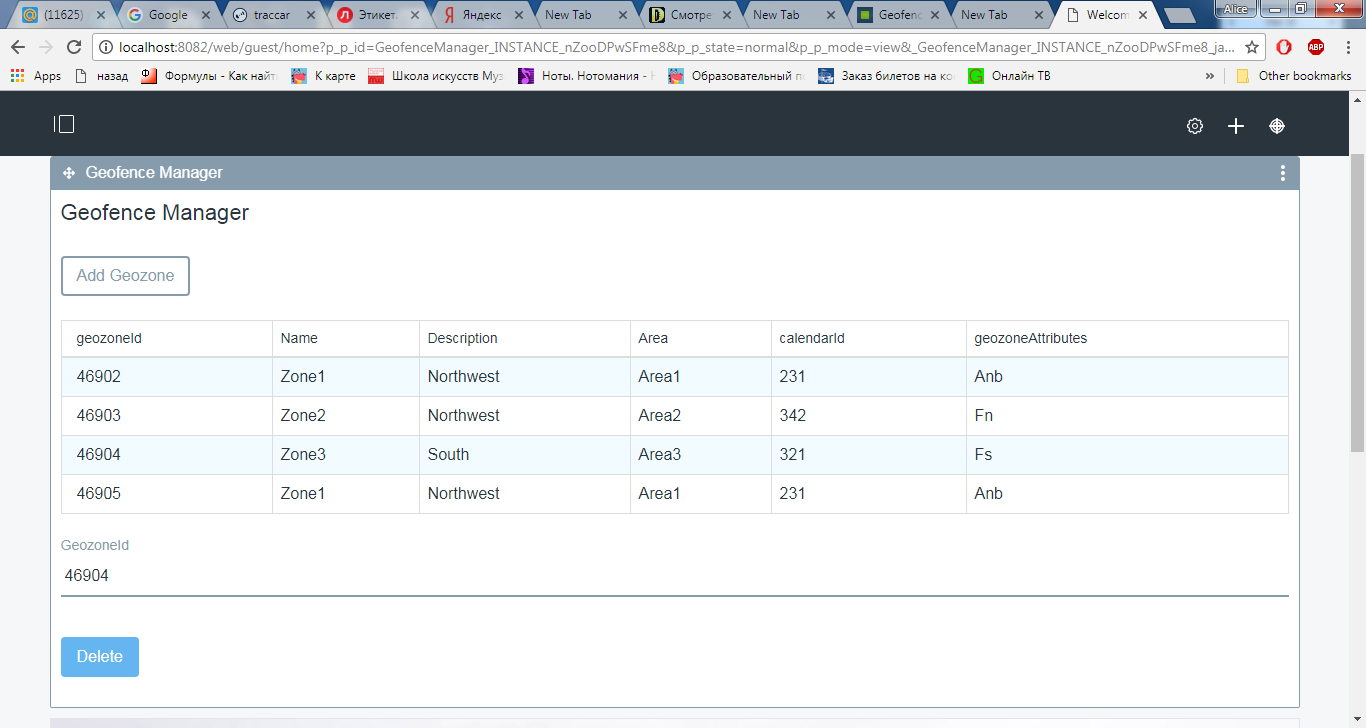


*Рисунок 15 - Результат тестирования на уникальность первичного ключа*

Как видно из рисунка 17, данные добавились с новым идентификационным ключом, значит уникальный ключ генерируется корректно.

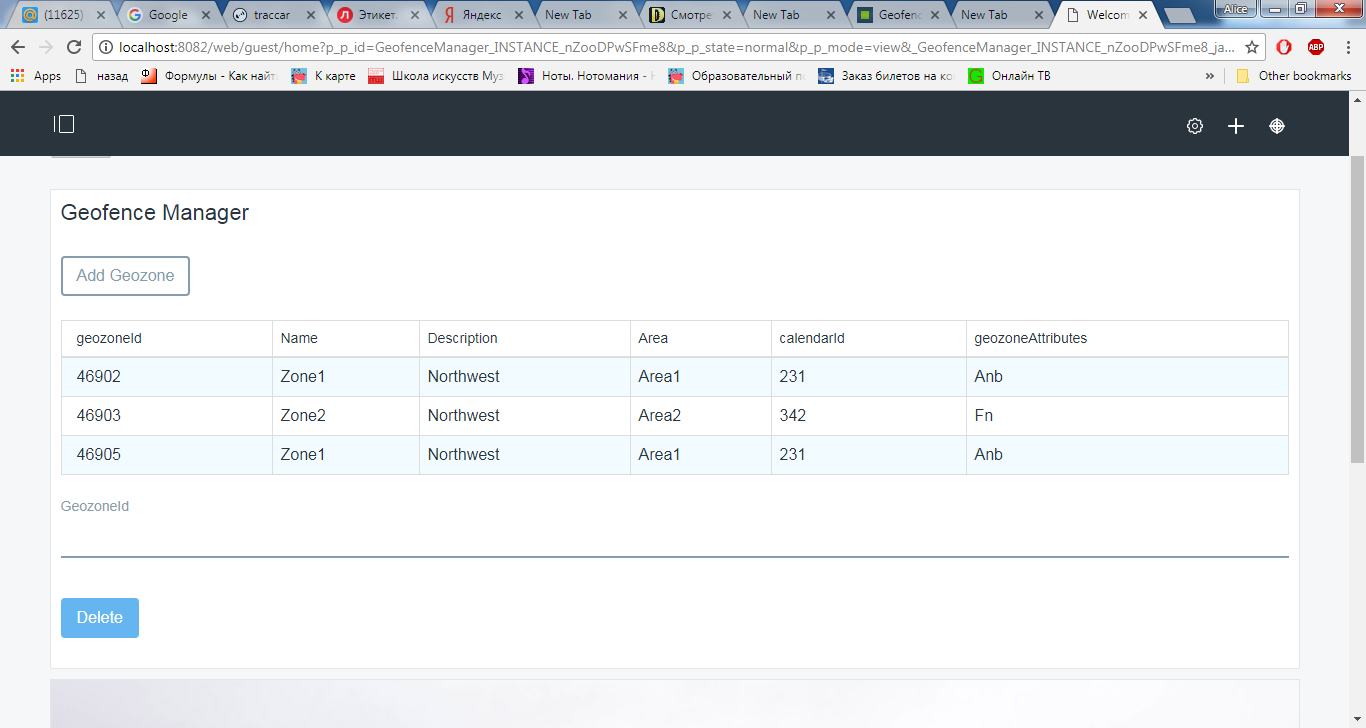
### Корректный ввод данных и добавление нового геозона.

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



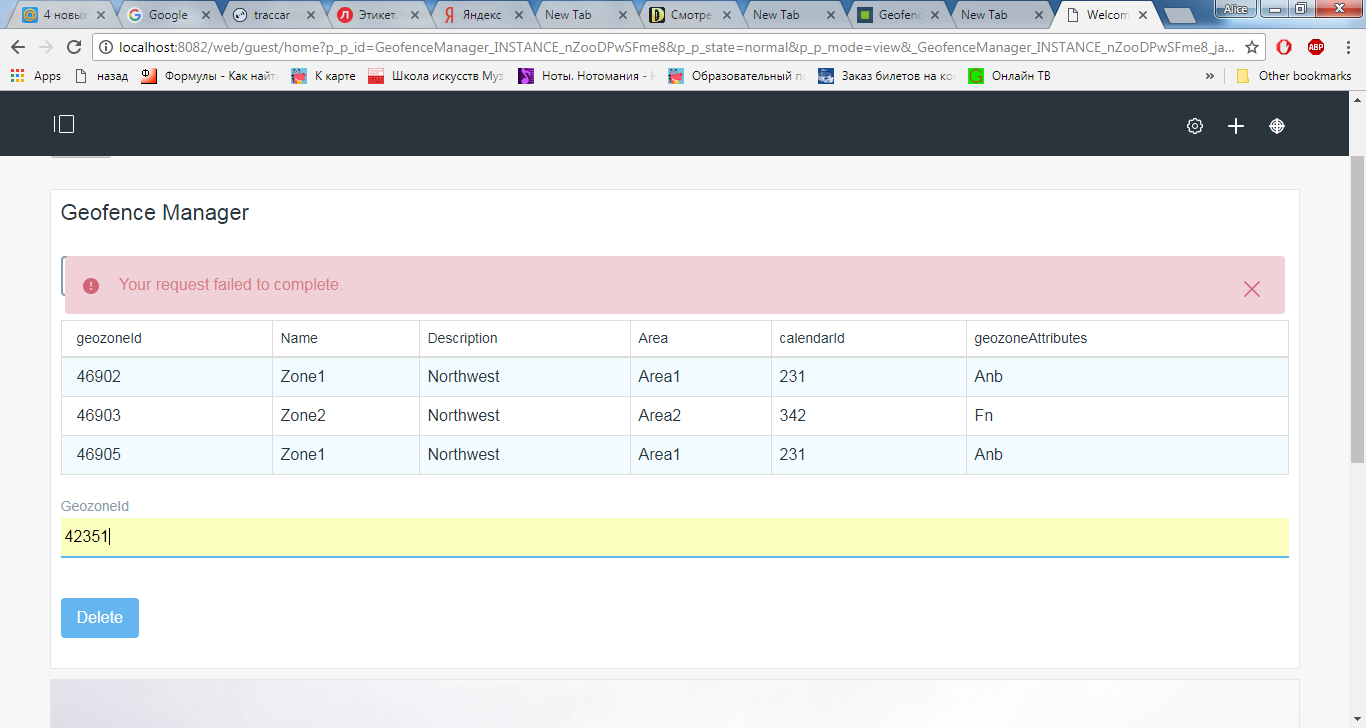
*Рисунок 16 - Удаление геозона из БД*

Введем в поле GeozoneId значение того геозона, которое нужно удалить (рисунок 16).



*Рисунок 17 - Результат удаления геозона*

Теперь попробуем ввести значение GeozoneId, которого нет в БД:



*Рисунок 18 - Результат удаления несуществующего геозона*

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

# Выводы

В ходе данной курсовой работы веб-сервис GeofenceManager и компонент пользовательского интерфейса системы Traccar 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 – GeofeManager [Электронныйресурс]. Режим доступа:

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

[5] TraccarAPI-reference [Электронныйресурс]. 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