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

Студент группы ИУ3-72 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.Д. Мосин

(подпись, дата)

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

(подпись, дата)

*2018 г.*

Оглавление

[1](#_1fob9te) Исследовательская часть 5

[1.1](#_3znysh7) Обзор предметной области 5

[1.2](#_2et92p0) Разрабатываемая система как черный ящик в окружении внешних систем 5

[1.3](#_tyjcwt) Техническое задание 6

[1.4](#_3dy6vkm) Интересы заинтересованных сторон по отношению к системе 7

[2](#_1t3h5sf) Конструкторская часть 8

[2.1](#_4d34og8) Технические решения, позволяющие удовлетворить интересы заинтересованных сторон 8

[2.2](#_2s8eyo1) Высокоуровневая архитектура системы 9

[2.2.1](#_17dp8vu) Компонентная декомпозиция системы 9

[2.2.2](#_3rdcrjn) Модульная декомпозиция системы 10

[2.3](#_26in1rg) Детальная архитектура системы 11

[2.3.1](#_lnxbz9) Файловая структура репозитория системы в GitHub 11

[2.3.2](#_35nkun2) Плагины и классы системы 12

2.3.2.1 Core-api 12

2.3.2.1 Core-service 13

2.3.2.3 Web 13

[2.4](#_1ksv4uv) Особенности создания бинарной сборки системы 14

[2.5](#_44sinio) Особенности запуска разработанной системы 14

[2.6](#_2jxsxqh) Анализ реализации системы 15

[2.6.1](#_z337ya) Анализ исходного кода с помощью метрик качества 15

[2.6.2](#_3j2qqm3) Анализ зависимостей в коде системы 18

[2.7](#_1y810tw) Тестирование на корректность работы 19

[2.7.1](#_4i7ojhp) Корректный ввод данных и добавление новой геозоны 19

[2.7.2](#_2xcytpi) Корректный ввод данных и добавление второго пользователя 21

[2.7.3](#_1ci93xb) Корректный ввод удаляемых данных и удаление пользователя 22

[3](#_3whwml4) Выводы 24

[Список используемой литературы 25](#_2bn6wsx)

1. Исследовательская часть
   1. Обзор предметной области

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

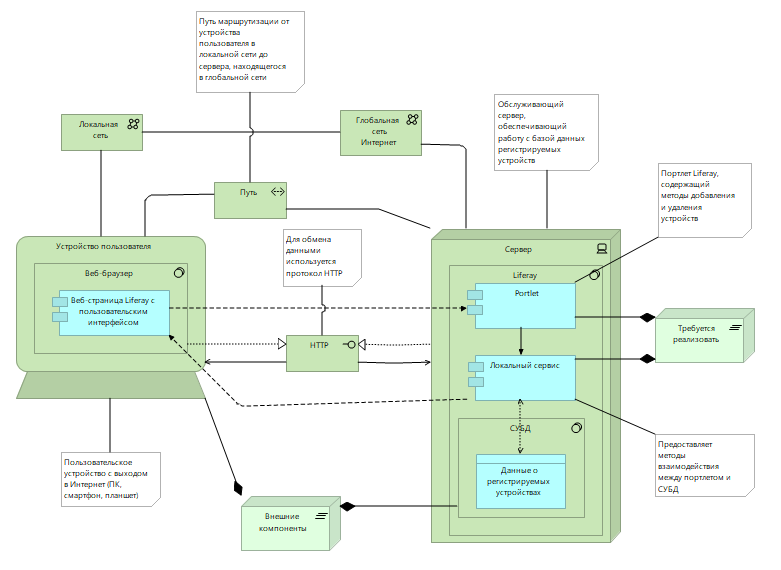
OSGi (Open Services Gateway Initiative) — спецификация динамической модульной системы и сервисной платформы для Java-приложений, разрабатываемая консорциумом OSGi Alliance.

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 - ведущее программное обеспечение для отслеживания GPS.

* 1. Разрабатываемая система как черный ящик в окружении внешних систем

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. Такая система отображена на рисунке 1.



1. принцип разрабатываемой системы
   1. Техническое задание

Портирование веб-сервиса UsersManager 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. 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.

1. Заинтересованные стороны и их интересы по отношению к системе.

|  |  |
| --- | --- |
| Заинтересованные стороны | Интересы заинтересованных сторон |
| Пользователь портлета | П1 Удобное отображение всех свойств добавленных пользователей Traccar.  П2 Удобное взаимодействие с системой добавления свойств пользователя системы Traccar.  П3 Валидация полей. |
| 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 свойств пользователей системы Traccar, добавленных в базу данных. |

1. Конструкторская часть
   1. Технические решения, позволяющие удовлетворить интересы заинтересованных сторон

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

1. Технические решения, удовлетворяющие интересам ЗС.

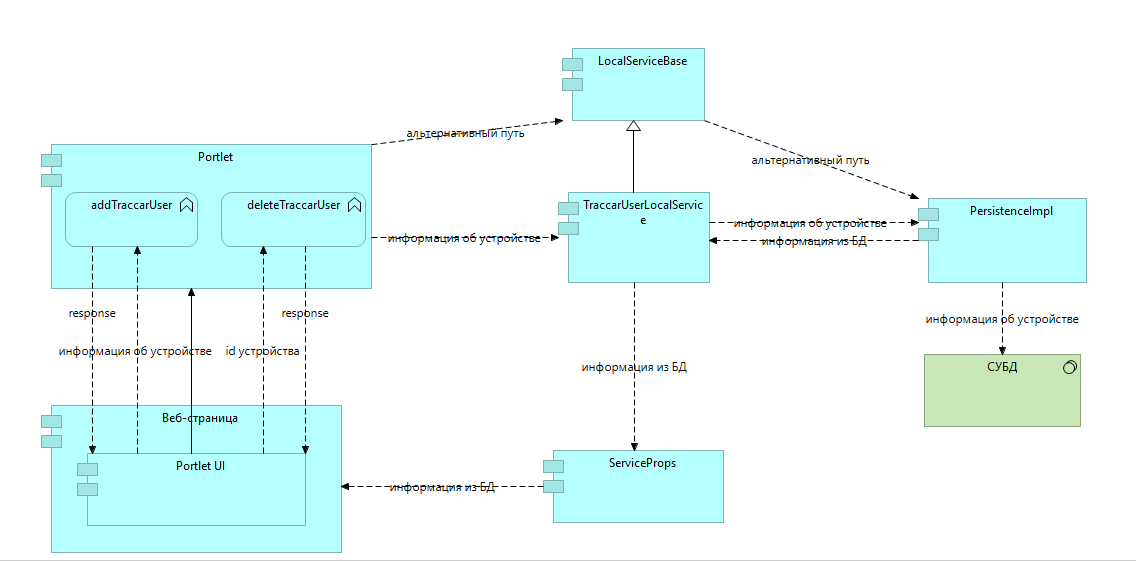
|  |  |
| --- | --- |
| Интересы заинтересованных сторон | Технические решения |
| П1 Удобное отображение всех свойств добавленных пользователей Traccar.  П2 Удобное взаимодействие с системой добавления свойств пользователя системы Traccar.  П3 Валидация полей. | Визуализатор данных из БД.  Эргономичный интерфейс реализованных функций взаимодействия с базой данных пользователей.  Жесткая типизация вводимых данных. |
| Р1 Использование распространенных средств разработки. | Хорошо документированный код, наличие обучений и файлов конфигураций. |
| В1 Быстрая и полная передача исходного кода, настроек, документов.  В2 Возможность в дальнейшем расширять систему, например, добавляя методы изменения полей свойств пользователей системы Traccar, добавленных в базу данных. | Код и настройки разрабатываемой системы будут находиться в системе GitHub. Контроль версий будет производится с использованием системы Git.  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. |

* 1. Высокоуровневая архитектура системы
     1. Компонентная декомпозиция системы

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

* name типа String;
* password типа String;
* email типа String;

Таким образом, 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. Тогда компонентная декомпозиция системы будет выглядеть как на рисунке 2.

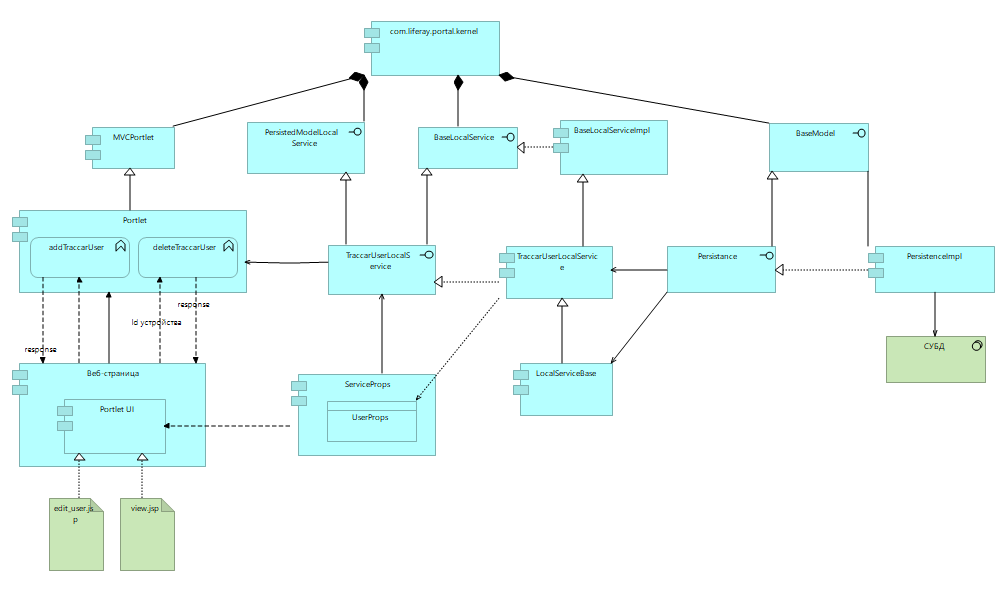


1. Компонентная декомпозиция системы

На сервере реализован портлет, который использует протокол передачи данных в виде набора данных протокола передачи. С помощью функции добавление 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 UsersProps и добавить это значение в базу данных.

* + 1. Модульная декомпозиция системы

В составе данного проекта будет использоваться 3 модуля, 2 из которых – core-api и core-service, создаются автоматически компонентом, который называется Service Builder. Общая структура проекта отображена на рисунке 3.



1. Модульная диаграмма проекта Users Manager-а
   1. Детальная архитектура системы
      1. Файловая структура репозитория системы в GitHub

Ссылка на репозиторий GitHub приведена в списках литературы под пунктом 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 вам запускать сборку с Gradle без необходимости установки самого Gradle в систему.

* + 1. Плагины и классы системы

Проект состоит из 2-х модулей, один из которых из 2-х подмодулей:

* com.bmstu.users.manager.core;
* com.bmstu.users.manager.core-api
* com.bmstu.users.manager.core-service
* com.bmstu.users.manager.web.

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 указываются поля заданной модели. Необходимые поля взаимодействия Traccar и сервера указаны на сайте описания API [5]. Таким образом было необходимо внести поля, отображены на рисунке 2.

* + - 1. Core-api

Модуль содержит API для проекта UsersManager. Все классы и интерфейсы в core-api модуле упакованы в .jar файл, UsersManager-api.jar, указанный в build/libs папке модуля. Этот .jar файл генерируется всякий раз, когда компилируется и развертывается сервисный модуль. При развертывании этого JAR в Liferay доступны необходимые интерфейсы для определения API сервиса.

* + - 1. Core-service

Модуль содержит реализацию интерфейсов, определенных в модуле UsersManager–api. Эти интерфейсы предоставляют службы OSGi для экземпляра портала, на котором развертывается приложение. Service Builder генерирует классы и интерфейсы, относящиеся к уровню сохранения, уровню обслуживания и уровню модели.

Ключевым классом данного модуля является TraccarUsersLocalServiceImpl.java, который обрабатывает полученный от портлета запрос и заносит новые данные в базу данных. Данный класс наследуется от TraccarUsersServiceBaseImpl.java и переопределяет ранее определенные методы наследуемого класса. Это позволило вносить нужные нам изменения в интерфейсе сайта, который далее будут передаваться в качестве данных в базу данных. Одним из таких переписанных методов является метод addTraccarUser, который позволяет добавлять пользователя в базу данных в формате отображения на сайте, тогда как стандартный метод добавления пользователя предполагал уже добавление сформированного объекта типа UsersProps.

* + - 1. Web

Модуль содержит реализацию MVC-портлета. Портлеты являются одним из основных базовых блоков Liferay Portal. Liferay Portal предоставляет платформу, которая содержит общие функции, необходимые для современных приложений, включая управление пользователями, безопасность, пользовательские интерфейсы, службы и многое другое.

В классе UsersManagerPortlet.java описываются методы обработки запроса на добавление и удаление группы addTraccarUser и deleteTraccarUser. Эти методы посредством запроса с сайта (response) преобразуют вводимые пользователем данные в удобный формат для взаимодействия с ними. Так удаление пользователя производится с помощью уникального идентификатора “traccarUserId”, который вводится пользователем на основе отображаемых данных портлета.

Два файла view.jsp и edit\_user.jsp описывают 2 состояния портлета – основное и состояние добавления нового пользователя. Технологическая часть

* 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 traccarUser.

* 1. 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 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.

* 1. Анализ реализации системы
     1. Анализ исходного кода с помощью метрик качества

На рисунке 5 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:

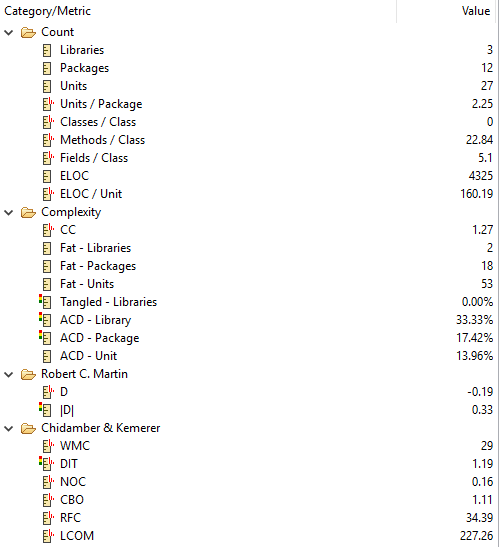
* количество классов верхнего уровня (Unit);
* среднее число внутренних классов на класс (Classes / Class);
* среднее число методов в классе (Methods / Class);
* среднее число полей в классе (Fields / Class);
* число строчек кода (ELOC);
* число строчек кода на модуль (ELOC / Unit).

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).



1. 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 5, 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 моем 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

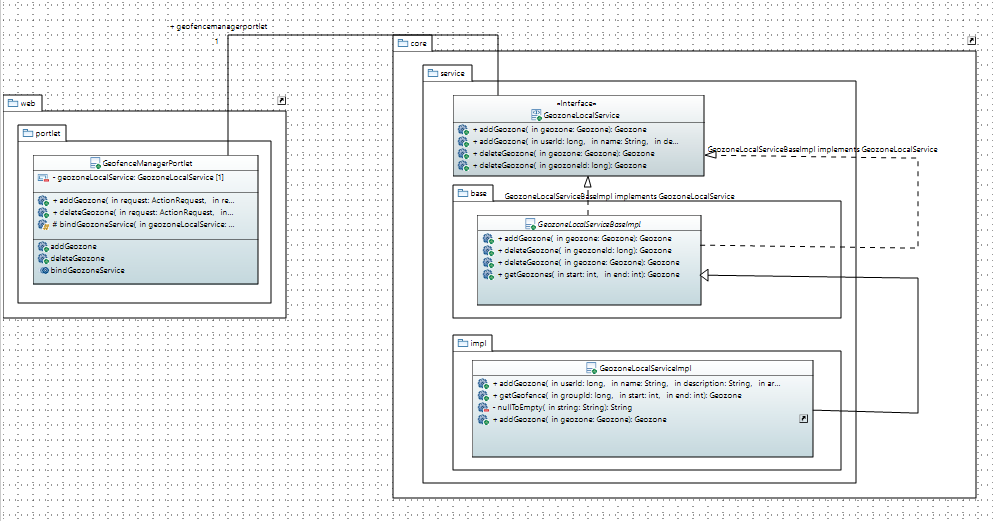
стандарту, должен быть не больше 20% [7, 8]. Согласно рисунку 5, данное требование удовлетворено, а значит, удовлетворен интерес разработчика к легкой поддержке и расширяемости проекта.

Метрика D показывает, насколько сборка проекта сбалансирована относительно её абстрактности и стабильности. Согласно рисунку 5, D = -0.19, а значит, сборка сбалансирована относительно её абстрактности и стабильности. Таким образом, удовлетворен интерес разработчика к расширяемости проекта.

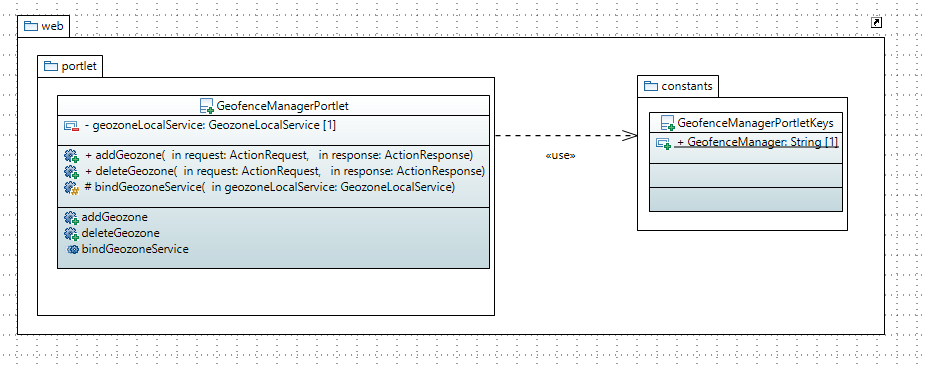
Метрика CBO дает возможность определить количество классов, с которыми связан данный класс. Низкое значение данной метрики (рисунок 5) улучшает модульность и содействует инкапсуляции проекта, а значит, удовлетворен интерес разработчика к расширяемости проекта.

* + 1. Анализ зависимостей в коде системы

На рисунке 6-7 представлена диаграмма классов, которая отражает взаимодействие модулей \*.core и \*.web.



1. Упрощенная диаграмма классов



1. Диаграмма классов \*.web

На диаграмме классов описана общая структура проекта, которая соответствует диаграмме компонентов. Вынесено два модуля:

1. Взаимодействие \*.web и \*.core (рис. 6);
2. Взаимодействие внутри \*.web.
   1. Тестирование на корректность работы

В качестве тестирования было принято применить метод ручного тестирования. Для этого будет проверяться выполнение всех требований технического задания (в упрощенном варианте, объединяя схожие компоненты в одну графу), а именно:

1. корректный ввод данных и добавление нового пользователя;
2. корректный ввод данных и добавление второго пользователя (проверка создания уникального первичного ключа traccarUserId);
3. корректный ввод удаляемых данных и удаление пользователя;
   * 1. Корректный ввод данных и добавление нового пользователя

Изначально было сконфигурирован view.jsp файл, который должен выводить добавленные в базу данных данные в виде таблице с параметрами, указанными на рисунке 13.

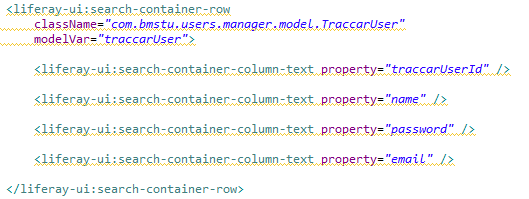


Рисунок 13 – Поля выводимой на web-страницу таблицы файла view.jsp

Таким образом, на этапе добавления портлета на web-страницу для отображения будут доступны только кнопки Add Traccar User и Delete, так как в базе данных портлета еще нет записей (рис. 14).

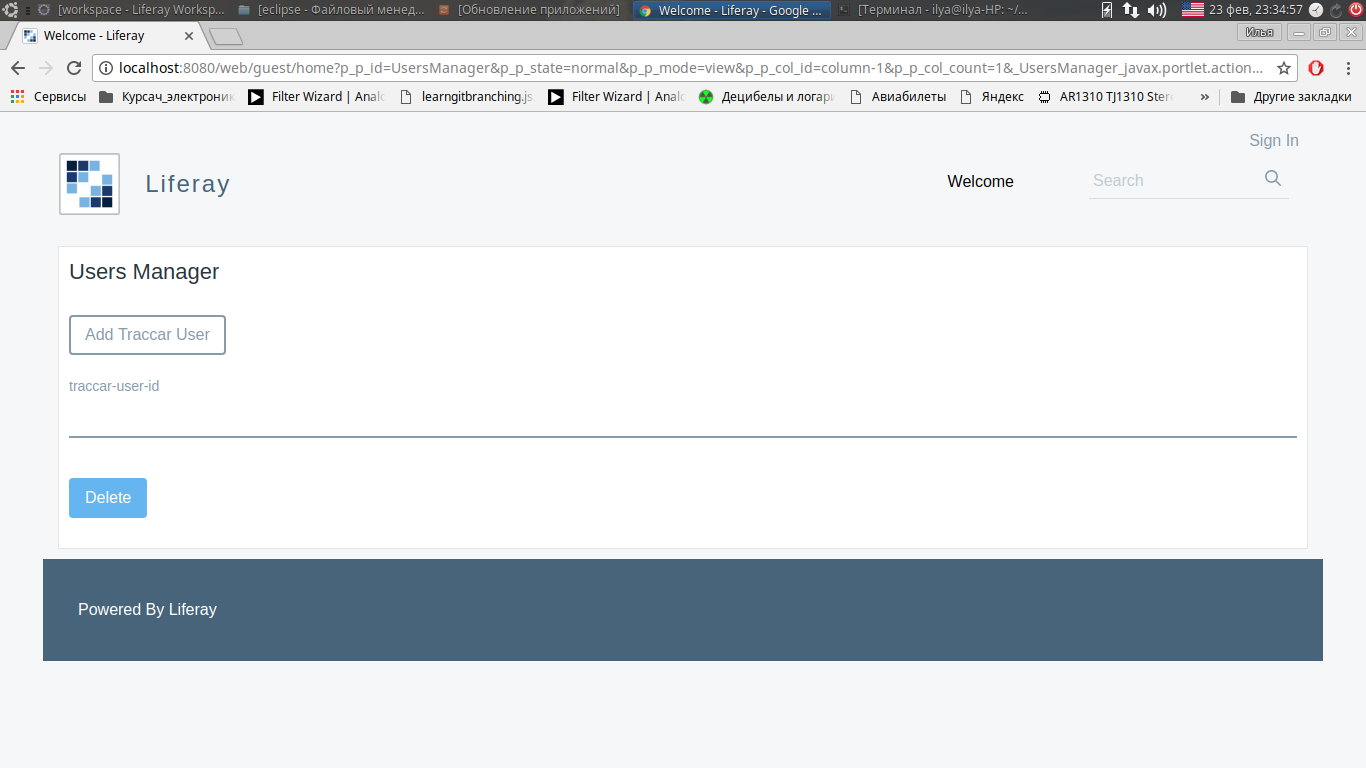


Рисунок 14 – Добавление созданного портлета в Liferay

Попробуем создать пользователя. При нажатии кнопки “Add Traccar User” должна появиться новая web-страница со следующими заполняемыми формами (после знака равно значения, которыми мы заполним первую запись в данной таблице) (рис. 15).

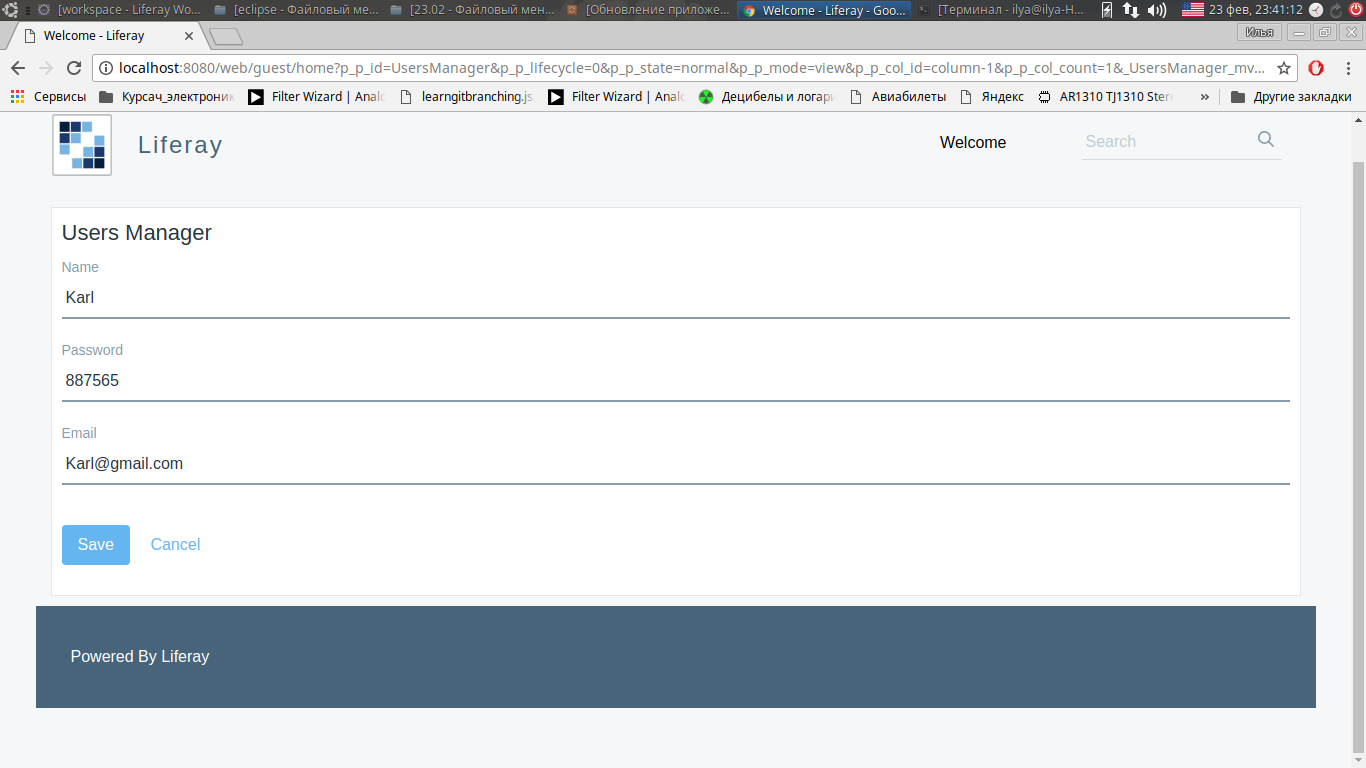


Рисунок 15 – Форма, появившаяся после нажатия на кнопку “Add Traccar User”. Заполнение формы значениями

Должна обновиться таблица и добавиться новые данные, что и произошло (рис 16), значит функционал добавления работает верно.

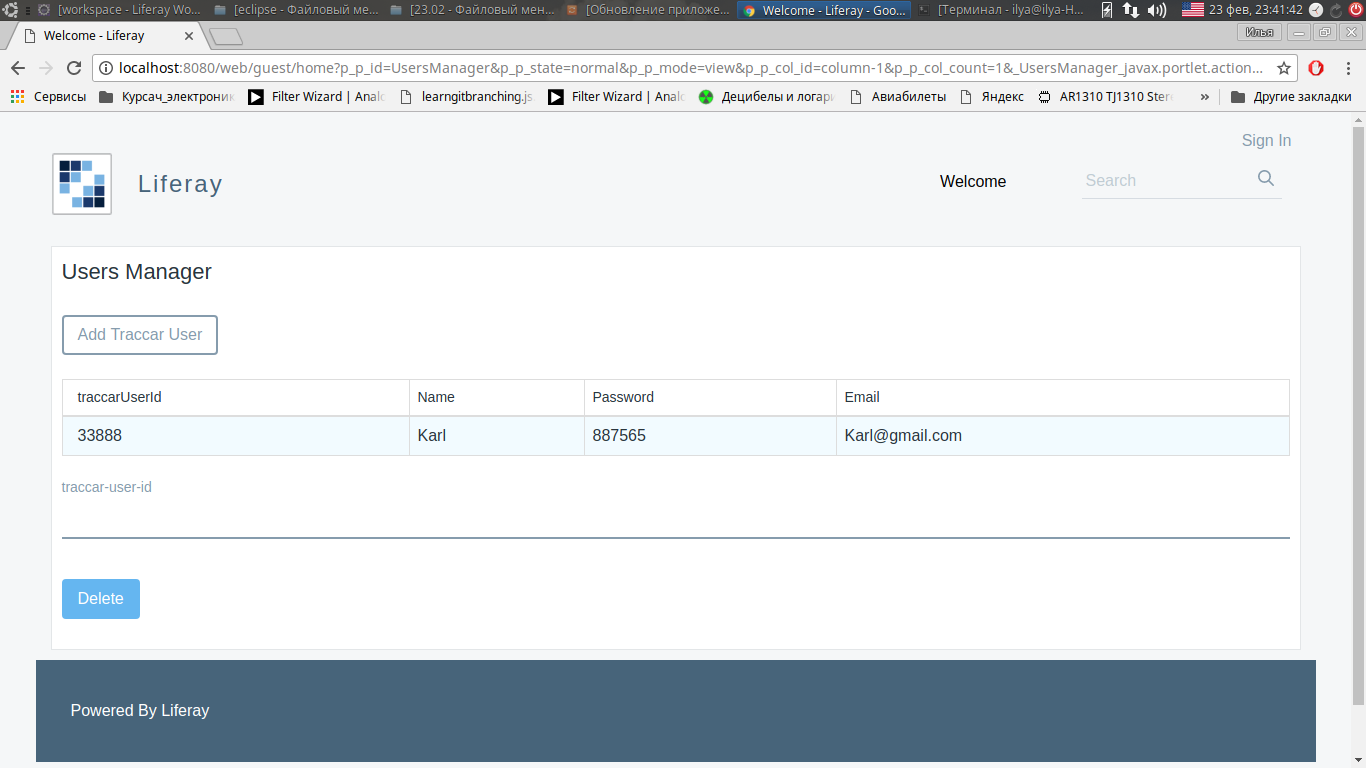


Рисунок 16 – Обновленный вид web-страницы портлета с добавленной записью в базу данных

* + 1. Корректный ввод данных и добавление второго пользователя

Для проверки уникальности генерируемого ключа создадим строку данных, как и в п.4.4.1. Ключ должен быть уникальным.

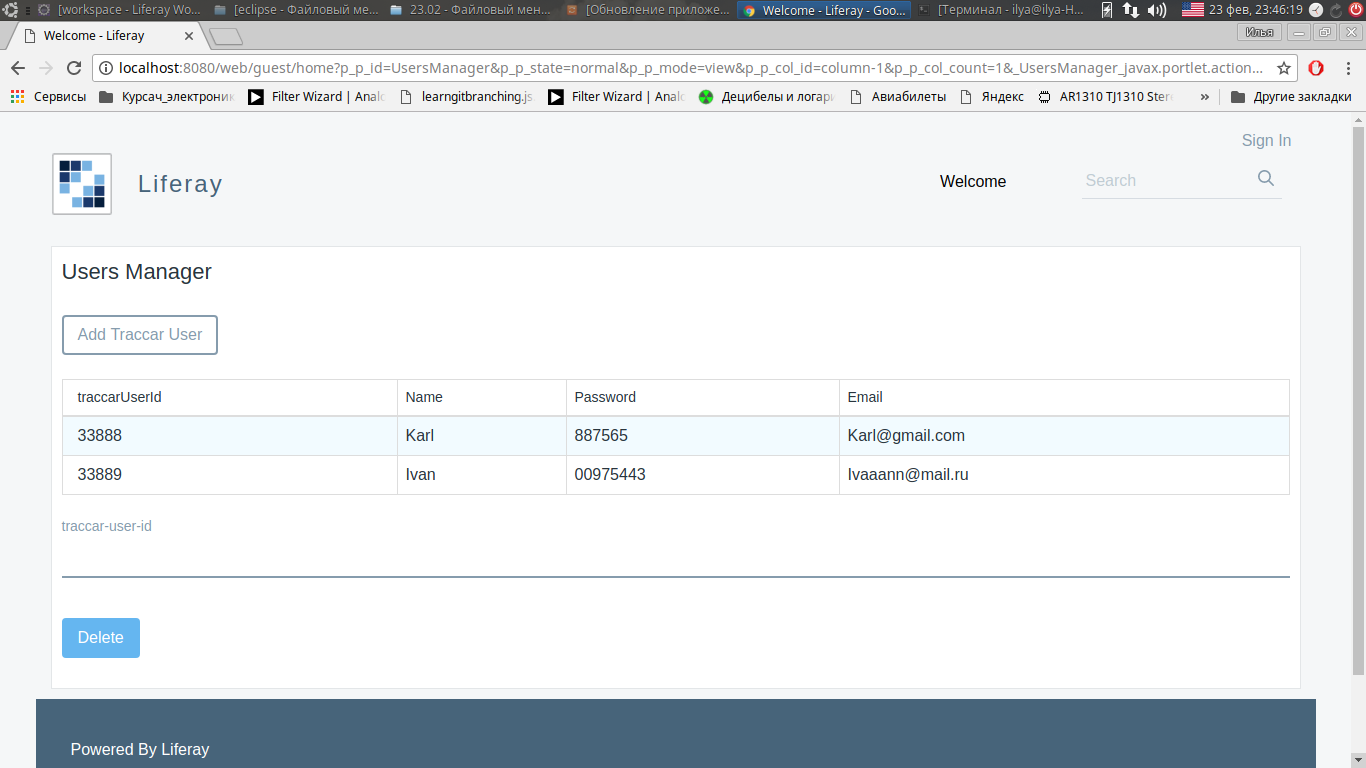


Рисунок 17 - Обновленный вид web-страницы портлета с добавлением новой строки записи.

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

* + 1. Корректный ввод удаляемых данных и удаление пользователя

При нажатии на кнопку “Delete” должна производиться операция удаления строки с уникальным ключом traccar-user-Id из базы данных. Так как реализованный метод удаления deleteTraccarUser(long traccarUserId) является стандартным сгенерированным методом класса UsersLocalServiceBaseImpl, то удаление записи должно произвестись по уникальному ключу записи. Проверим это, введя в поле traccar-user-Id номер 33888, соответствующий последней записи.

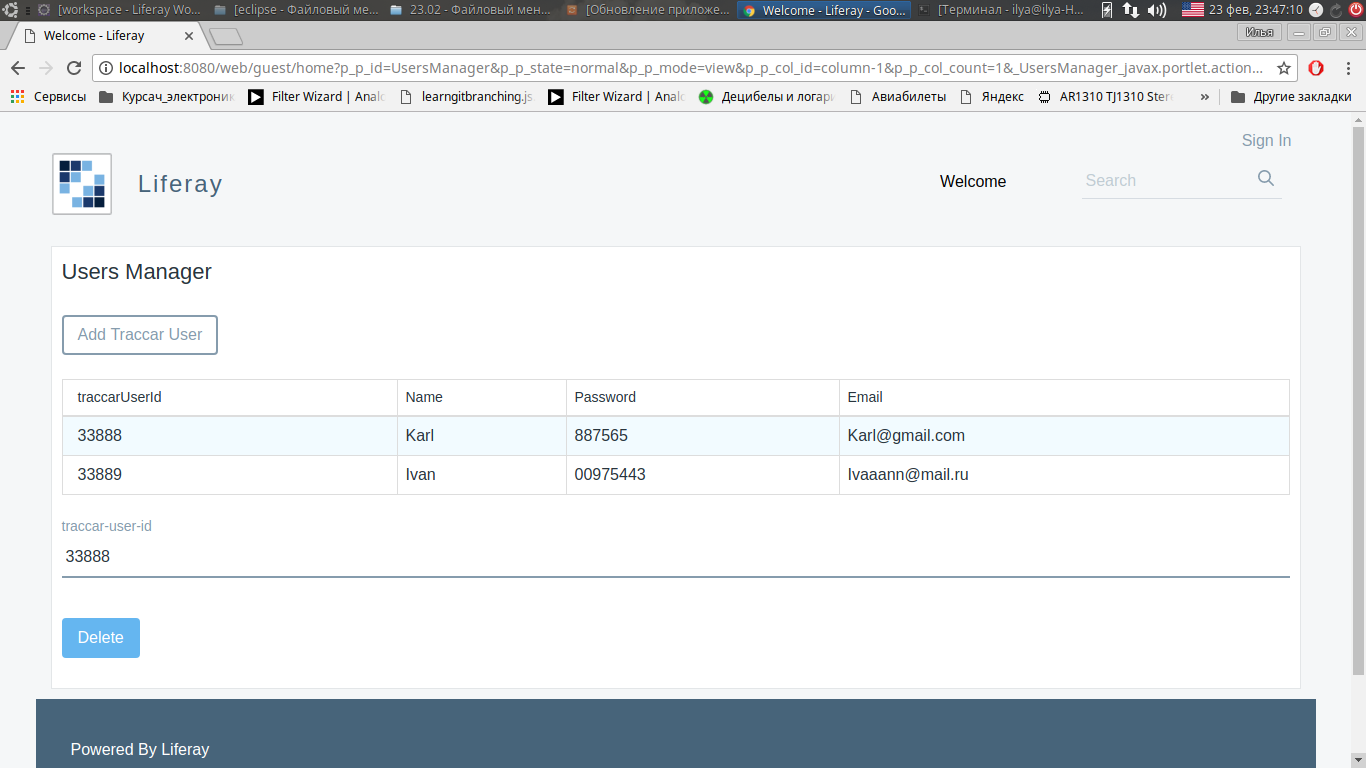


Рисунок 18 – Корректное открытие web-страницы с удалением после нажатия кнопки “Delete”

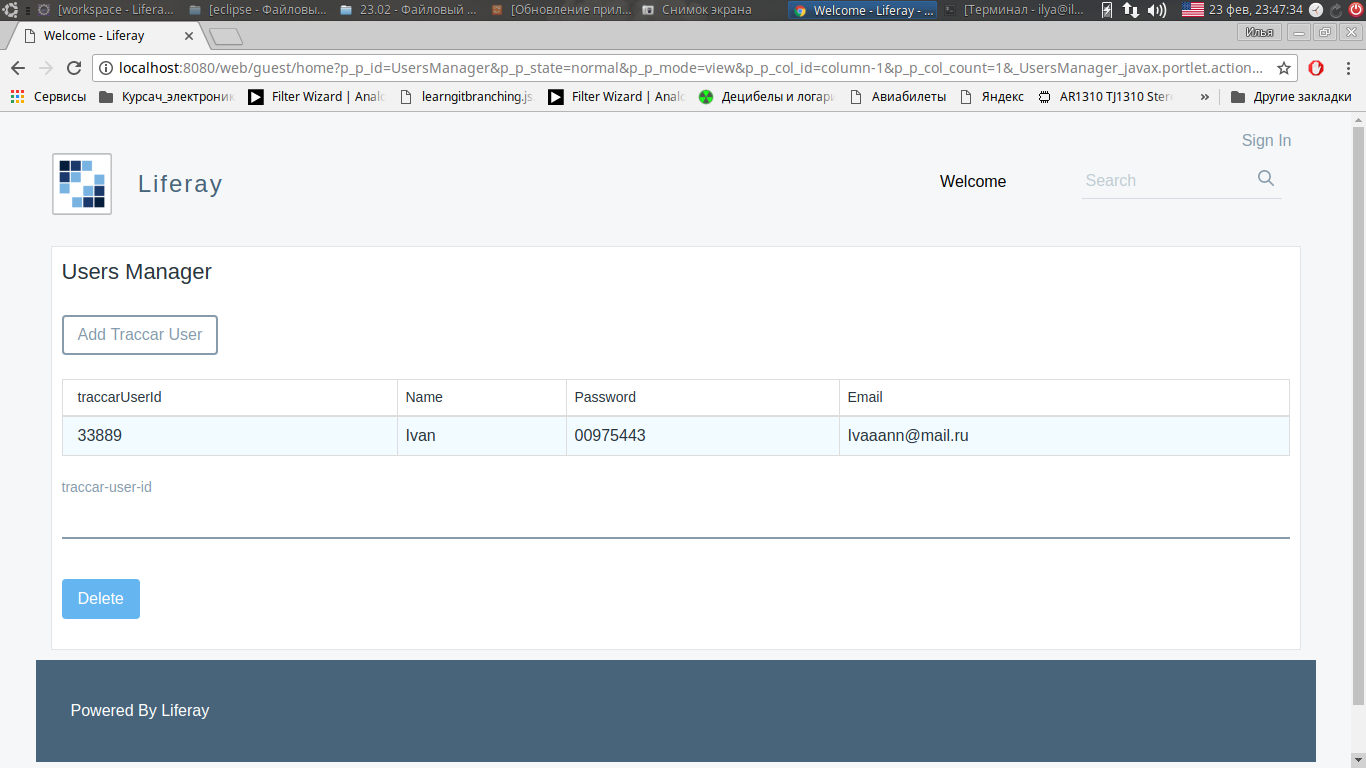


Рисунок 19 – Корректное удаление записи с уникальным ключом 33888

Как видно из рисунка 19, web страница с полем ввода уникального идентификационного номера открылась корректно, а после ввода данных и нажатии кнопки “Delete” произведено удаление записи с уникальным ключом traccar-user-Id равным 33888. Таким образом, функция удаления пользователя работает.

1. Выводы

Техническим заданием данной курсовой работы являлось портирование веб-сервиса UsersManager и компонента пользовательского интерфейса системы Traccar на OSGi сервис и портлет платформы Liferay с сохранением протокола взаимодействия клиента с сервером. Веб-сервис успешно портирован с сохранением протокола взаимодействия.

В результате выполнения данной курсовой работы веб-сервис UsersManager и компонент пользовательского интерфейса системы Traccar портированы на платформу Liferay с сохранением протокола взаимодействия клиента с сервером. Пользовательский интерфейс реализован в виде портлета платформы, а веб-сервис в виде OSGi DS сервиса. С помощью тестов была проверена корректность работы функций портированного сервиса, а именно добавление новых пользователей, удаление пользователей, а так же проверка валидации полей. Данные тесты были проведены с помощью метода ручного тестирования, позволяющего смоделировать взаимодействие пользователя и портированного веб-сервиса.

Данный проект можно использовать для более удобного отображения пользователей, взаимодействия и управления таблицей пользователей, а так расширить его возможности с помощью добавления новых функций корректировки введенных данных, а также отбора нужных данных таблицы пользователей.

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 – UsersManager [Электронный ресурс]. Режим доступа: https://github.com/mosin96/com.bmstu.liferay.users.manager

[4] Liferay 7 Features [Электронный ресурс]. Режим доступа: <http://www.liferaysavvy.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