МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт **математики, физики и информационных технологий**

Кафедра **«Прикладная математика и информатика»**

02.03.03 МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ И АДМИНИСТРИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

БАКАЛАВРСКАЯ РАБОТА

на тему
«Разработка серверной части системы мониторинга автотранспорта»

Студент А.С. Гридина _____

Руководитель А.И. Туищев _____

Консультант А.В. Кириллова _____

Допустить к защите
Заведующий кафедрой к.т.н. А.В. Очеповский _____

«_____» _____ 2017 г.

Тольятти 2017

АННОТАЦИЯ

В данной выпускной квалификационной работе подробно раскрывается актуальность системы мониторинга с использованием современных технологий глобальной навигации.

Ключевым вопросом выпускной квалификационной работы является особенность работы серверной части.

Мы начинаем с постановки задачи, а затем логически переходим к ее возможным решениям.

Мы исследуем с помощью каких технологий можно реализовать сервер, а также способы обработки (фильтрации) данных, полученных от транспортных средств.

Особое внимание уделяется математической части, в данной работе отработана технология фильтрации данных на основе фильтра Калмана.

Разработка системы велась на языке программирования Java. Система состоит из сервера приложений и СУБД MySQL.

Работа состоит из введения, трех глав и заключения. Объем работы составляет 57 страниц, на которых размещены 27 рисунков, 1 таблица и 7 приложений. При написании данной работы было использовано 20 литературных источников.

ABSTRACT

The title of the graduation work is Back-End Development for Monitoring the Movement of Vehicles.

This graduation work fully describes the necessity of making the tracking system using global navigation technologies.

Much attention is given to the server operation feature and possible problems appearing during the process of development.

We first discuss the definition of the main problem of the graduation work then logically pass over to its possible solutions. We perform the analysis of system requirements and server technologies that are able to help in the application creation in the best way.

Then we determine the problems of the navigation in urban area and countryside. These problems vary greatly depending on the type of the monitoring area, and we put full effort into finding the most elegant solution.

This graduation work represents mostly the mathematical part of the application development. The general technology used in our application was trajectory smoothing technology based on Kalman filtering also recognized as linear quadratic estimation.

The system was developed using the Java programming language. The system consists of application server Apache Tomcat and well-known database management system MySQL.

The work is of interest for a wide circle of readers.

The graduation work consists of an explanatory note on 57 pages, introduction, including 27 figures, 1 table, the list of 20 references including 5 foreign sources and 3 appendices.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Проблема определения своего местоположения актуальна для человечества на протяжении многих лет. Развитие космических технологий привело к созданию спутниковых систем позиционирования в пространстве, что привело к разработке американской системы GPS и российской системы ГЛОНАСС.

Благодаря использованию технологий спутниковой навигации ГЛОНАСС/GPS определение местоположения подвижного объекта, направления и скорости его передвижения, а также точного времени нашло широкое применение в системах мониторинга транспортных средств.

В настоящее время услуги систем GPS и ГЛОНАСС доступны любому человеку. Особой популярностью GPS системы (навигаторы) пользуются у автолюбителей, водителей такси. Трекеры со встроенным GPS-модулем широко используются на автотранспортных предприятиях, занимающихся перевозками грузов и пассажиров.

Федеральными нормативными документами, например, приказ Минтранса России №285 от 31.07.2012 «Об утверждении требований к средствам навигации», обязывает установку навигационного оборудования на транспортные средства, используемые для коммерческих перевозок пассажиров и на автомобили для перевозки опасных грузов.

Применение технологий автоматизированного спутникового слежения и контроля является необходимой составляющей бизнес-процесса. Автомобильный транспорт оказывает большое влияние на развитие социально-экономической сферы, широко применяется при перевозке на малые, средние и большие расстояния. Использование систем GPS-навигации обуславливается необходимостью эффективного мониторинга грузоперевозок и пассажиропотоков, качественного и своевременного предоставления автотранспортных услуг.

Автоматизированные системы мониторинга движения и других показателей автотранспорта способны обеспечить выполнение самых разных

задач в режиме реального времени. Эти системы предоставляют уникальную возможность всегда иметь точную и достоверную информацию о реальном местоположении и маршрутах движения транспорта. Появляется возможность проверить соответствие маршрутного задания с реальной траекторией движения автомобиля в рейсе, который отображается на географической карте, с полным списком пройденных адресов или координат пройденных точек. Внедрение современных технологий позволяет эффективно контролировать использование водителями транспортных средств, принадлежащих предприятию, а именно:

- позволяет отслеживать перевозку «левых» грузов;
- позволяет отслеживать отклонение от маршрута;
- помогает пресекать попытки угона автотранспорта;
- выявляет несанкционированные сливы топлива из топливного бака;
- позволяет контролировать соблюдение режима труда и отдыха водителей при выполнении рейсов на междугородних маршрутах;
- позволяет повысить трудовую дисциплину водителей.

Целью бакалаврской работы является разработка автоматизированной системы мониторинга за автотранспортом.

Объект бакалаврской работы: мониторинг транспортных средств.

Предмет бакалаврской работы: автоматизация мониторинга за автотранспортом.

Задачами бакалаврской работы, исходя из поставленной цели, являются:

- анализ системы мониторинга передвижения автотранспорта и проблем, возникающих при обработке данных, полученных от транспортных средств;
- построение математической модели для приближения траектории движения автотранспорта к действительной;
- определение и формулировка требований к серверной части мониторинга автотранспорта;

- разработка базы данных для серверной части мониторинга автотранспорта;
- разработка серверной части с применением фильтра Калмана в соответствии с построенной математической моделью.

Бакалаврская работа состоит из трех глав.

Первая глава работы посвящена анализу процесса мониторинга автотранспорта и проблем его позиционирования.

Вторая глава описывает математическую модель системы контроля передвижения автотранспорта.

В третьей главе описывается разработка серверной части системы мониторинга автотранспорта.

В заключении подводятся итоги разработанной серверной части, формируются окончательные выводы по рассматриваемой теме.

Глава 1 АНАЛИЗ ПРОЦЕССА МОНИТОРИНГА АВТОТРАНСПОРТА И ПРОБЛЕМ ЕГО ПОЗИЦИОНИРОВАНИЯ

1.1 Спутниковый мониторинг автотранспорта

GPS-мониторинг транспорта – это технология, которая широко применяется в диспетчерских службах на транспорте, а также для решения задач транспортной логистики в системах управления перевозками и автоматизированных системах управления автопарком для контроля реального маршрута транспортных средств при помощи спутников GPS.

Схема работы системы навигационного мониторинга изображена на рисунке 1.1.

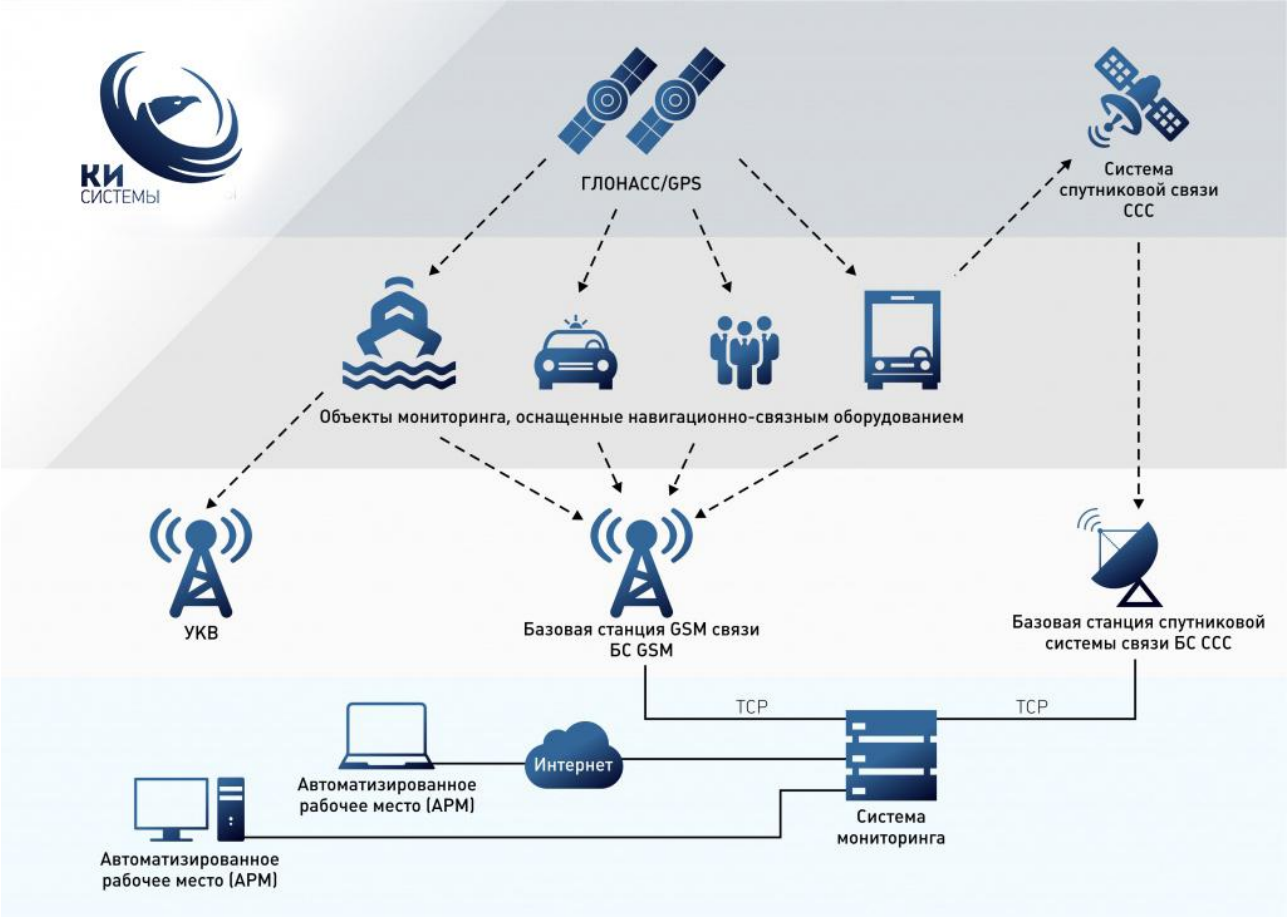

Рисунок 1.1 – Схема работы системы мониторинга автотранспорта

Общий принцип работы систем таков. На транспорт устанавливается специальное бортовое навигационное оборудование мониторинга транспорта, в

которое встроен ГЛОНАСС / GPS приемник, обрабатывающий сигналы со спутниковых навигационных систем. Так же эти устройства именуются как абонентские терминалы. В настоящее время производители изготовляют их многосистемными для увеличения качества приема сигнала. Так же в терминалы устанавливается SIM-карта сотового оператора, с помощью которой через Internet происходит передача данных в диспетчерский центр.

Терминал получает навигационные сигналы, собирает телеметрические данные с дополнительных и штатных устройств и далее передает их на телематический сервер через коммуникационную среду. На сервере вся информация обрабатывается и далее отправляется диспетчеру или оперативному дежурному на автоматизированное рабочее место (АРМ). АРМ – это компьютер с доступом в Интернет и с организацией работы через специальное клиентское приложение – программное обеспечение для автоматизации работы диспетчера или оперативного дежурного.

Системы спутникового мониторинга транспорта решают следующие задачи:

- мониторинг включает определение координат местоположения транспортного средства, его направления, скорости движения и других параметров: расход топлива, показания одометра и др. Системы спутникового мониторинга транспорта помогают водителю в навигации при передвижении в незнакомой местности;
- контроль соблюдения графика движения - учёт передвижения транспортных средств, автоматический учёт доставки грузов в заданные точки и др.;
- сбор статистки и оптимизация маршрутов - анализ пройденных маршрутов, скоростного режима, расхода топлива и другого автотранспорта с целью определения наиболее оптимальных маршрутов;
- обеспечение безопасности – возможность определения местоположения помогает обнаружить угнанный автомобиль. Современные системы спутникового мониторинга в случае аварии помогает передать сигнал

о бедствии в службы спасения. Также на основе спутникового мониторинга транспорта действуют некоторые системы автосигнализации [1].

Современные технологии позволяют использовать в системах мониторинга как спутниковую навигационную систему ГЛОНАСС, так и GPS. Дублирование друг другом чипов навигационных систем GPS и ГЛОНАСС мониторинга в системах управления автопарком, позволяет достичь наибольшей эффективности и надежной работы.

1.2 Анализ навигационных систем ГЛОНАСС и GPS

Системы GPS и ГЛОНАСС во многом подобны, но имеют и различия. Они разрабатывались с учетом наиболее вероятных областей применения. Поэтому ГЛОНАСС имеет преимущества на высоких широтах, а GPS - на средних [1].

В таблице 1 приведена сравнительная характеристика основных навигационных систем.

Таблица 1 - Основные характеристики навигационных систем ГЛОНАСС и GPS

Характеристики	**ГЛОНАСС**	**GPS**
Количество спутников (резерв)	24	24
Количество орбитальных плоскостей	3	6
Количество спутников в каждой плоскости	8	4
Тип орбиты	Круговая (S=0+-0,01)	Круговая
Высота орбиты	19100 км	20200 км
Наклонение орбиты, град	64,8+-0,3	55 (63)
Период обращения	11 ч 15,7 мин.	11 ч 56,9 мин.
Способ разделения сигналов	Частотный	Кодовый
Навигационные частоты, МГц: L1 L2	1602,56 - 1615,5 1246,44 - 1256,5	1575,42 1227,6
Период повторения ПСП	1 мс	1 мс (С/А-код) 7 дней (Р-код)

Продолжение таблицы 1

Характеристики	ГЛОНАСС	GPS
Тактовая частота ПСП, МГц	0,511	1,023 (С/А-код) 10,23 (Р,Y-код)
Скорость передачи цифровой информации, бит/с	50	50
Длительность суперкадра, мин	2,5	12,5
Число кадров в суперкадре	5	25
Число строк в кадре	15	5
Погрешность* определения координат в режиме ограниченного доступа: горизонтальных, м вертикальных, м	не указана	18 (Р,Y-код) 28 (Р,Y-код)
Погрешности* определения проекций линейной скорости, см/с	15 (СТ-код)	<200 (С/А-код) 20 (Р,Y-код)
Погрешность* определения времени в режиме свободного доступа, нс в режиме ограниченного доступа, нс	1000 (СТ-код) -	340 (С/А-код) 180 (Р,Y-код)
Система отсчета пространственных координат	ПЗ-90	WGS-84
* Погрешности в определении координат, скорости и времени для системы ГЛОНАСС - 0,997, для GPS - 0,95.		

Погрешности и неточности установленного автомобильного трекера в транспортном средстве могут быть вызваны целым рядом причин. Среди них можно выделить:

- потерю сигнала от спутника;
- изменения в геометрии расположения спутников;
- отражение сигналов от зданий и других объектов;
- вычислительные ошибки и ошибки при округлении.

Таким образом, информация, которую получает система, может не соответствовать маршруту транспортного средства с необходимой для анализа данных точностью.

Чтобы проанализировать неточности измерений координат местоположения автотранспортного средства, которые получаем благодаря установленному в автотранспорт GPS-трекеру, было принято решение отобразить передвижение объектов на географической карте.

Для решения этой задачи использовался фреймворк MapBox Studio. Он позволяет создавать пользовательские карты в соответствии с потребностью пользователя. Созданные функции с помощью встроенных инструментов можно импортировать в файлы CSV или GeoJSON.

GeoJSON - это открытый формат, который предназначен для хранения географических структур данных, основан на JSON. GeoJSON позволяет хранить примитивные типы для описания географических объектов, например: точки (адреса и местоположения), линии (улицы, шоссе, границы), полигоны (страны, штаты, участки земли). Также обладает возможностью хранить мультитипы, представляющие собой объединение нескольких примитивных типов.

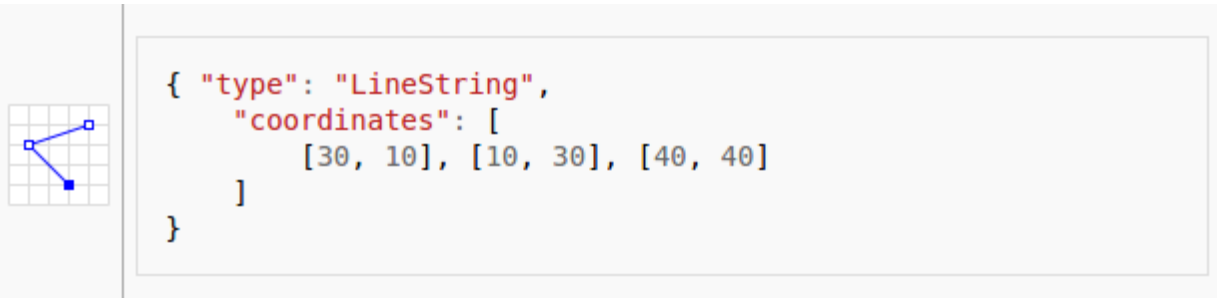

Рисунок 1.2 – Пример GeoJSON файла

Проведем анализ данных, записанных GPS-трекером. Данные получаемые GPS-трекером записываются в CSV файл, поэтому они были предварительно переведены в формат GeoJSON.

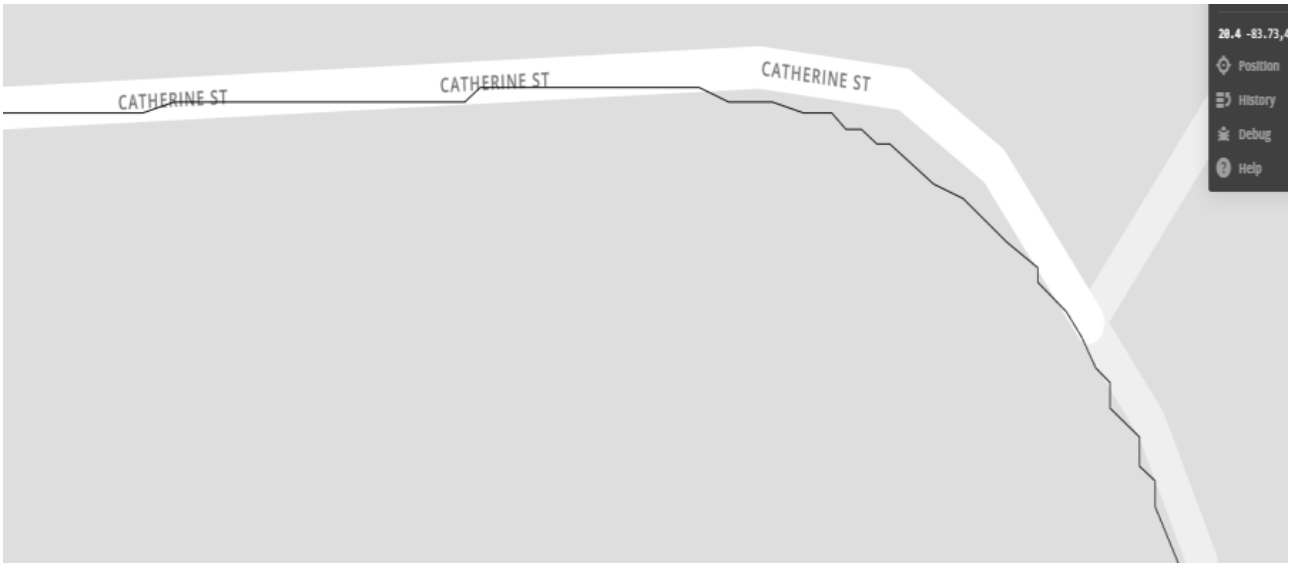

Рисунок 1.3 – Полученные данные

Из рисунка 1.3 видно, что полученные от GPS-трекера данные необходимо отфильтровать, перед тем, как «передавать» их на более высокий front-end уровень, т.е на сайт или на мобильное приложение, это будет являться следующим этапов обработки данных о местоположении посредством выбора колеблющихся точек направления движения.

На следующих графиках (рисунок 1.4 и рисунок 1.5) показаны зависимости направления (синий цвет) и скользящего среднего значения для направления (красный цвет):

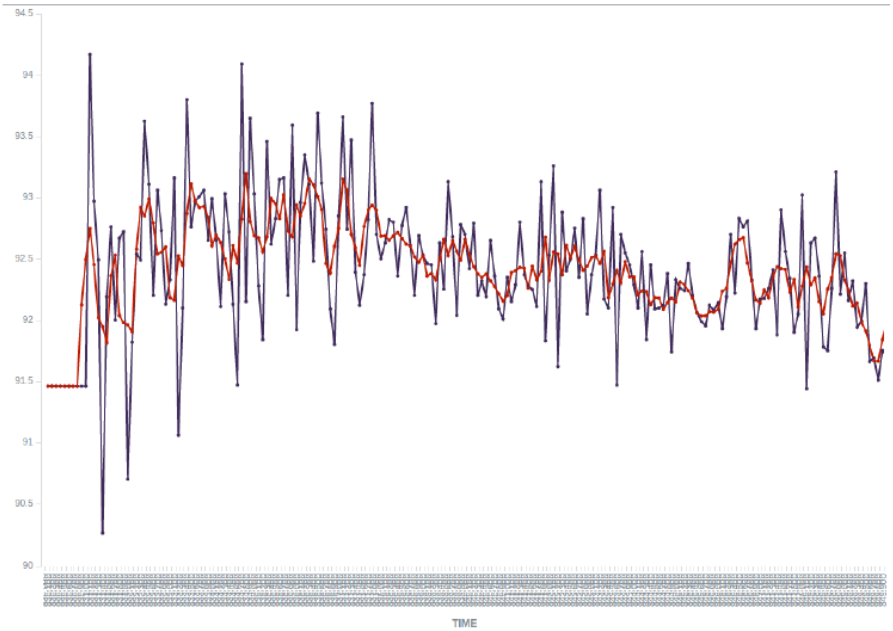

Рисунок 1.4 – Зависимость направления и средней скользящей

Интервал для среднего скользящего значения в данном случае равен 0,4 с. Очевидно, что выбранный интервал недостаточно сглаживает колебания значений направления, из чего можно сделать вывод, что по этому значению не будет эффективно выявление точек.

Увеличив интервал, мы может видеть, что колебания сглаживаются лучше, но при этом сами точки изменения направления сдвигаются (рисунок 1.5):

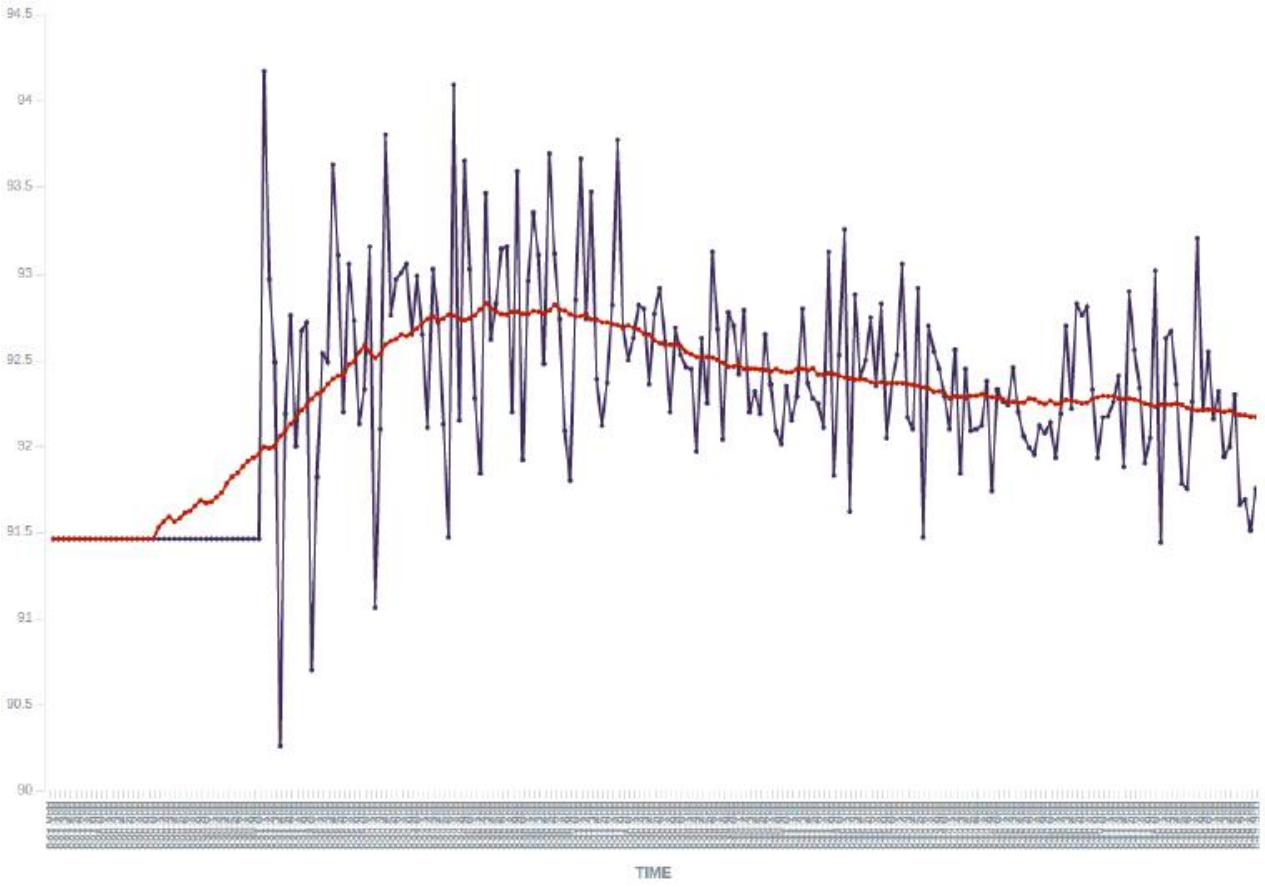

Рисунок 1.5 – Изменения зависимости после увеличения интервала

Вывод: Погрешности в исходных данных могут сильно исказить результат. Поэтому для повышения точности исходных данных необходимо произвести их коррекцию. В качестве основы алгоритма коррекции возьмем фильтр Калмана, математическая модель которого описывается в следующей главе.

Глава 2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ПРИБЛИЖЕНИЯ ТРАЕКТОРИИ ДВИЖЕНИЯ АВТОТРАНСПОРТА К ДЕЙСТВИТЕЛЬНОЙ

2.1 Постановка задачи для реализации серверной части

Пусть x_k – величина, которую будем измерять, а после фильтровать. За эту величину можно взять ускорение, скорость, координату, влажность, температуру, давление и т.д.

Рассмотрим пример, которым приведет нас к формулировке общей задачи. Предположим у нас есть некий движущийся объект, который может двигаться только вперед и назад. Нам известен вес объекта, форма, покрытые дороги и т.д.

В этом случае координаты объекта будут изменяться по следующему закону:

$$x_{k+1} = x_k + v_k dt$$

В реальной жизни не всегда можно учесть в расчетах маленькие возмущения, действующие на какой-либо объект. Это может быть ветер, ухабы, камни на дороге и т.д., поэтому действительная скорость объекта будет отличаться от расчетной. В этом случае к правой части уравнения будет добавлена случайная величина ξ_k:

$$x_{k+1} = x_k + v_k dt + \xi_k$$

Допустим наш объект – транспортное средство, на котором установлен GPS-сенсор, измеряющий истинную координату x_k автотранспорта. GPS-сенсор не может точно измерить координаты, а измеряет с ошибкой η_k, которая также является случайной величиной. В итоге от сенсора поступают данные с ошибкой:

$$z_k = x_k + \eta_k$$

Смысл задачи заключается в том, что когда известны неверные показания сенсора z_k, необходимо найти приближение для истинной координаты машины x_k. Обозначим это хорошее приближение как x_k^{opt}. В определении общей

задачи за координату x_k может отвечать любая величина (температура, влажность), а член, отвечающий за контроль системы извне, обозначим как u_k.

Уравнение для координаты и показания сенсора будут выглядеть следующим образом:

$$\begin{matrix} x_{k+1} = x_k + u_k + \xi_k \\ z_k = x_k + \eta_{k+1} \end{matrix} \quad (1)$$, где:

- u_k – это известная величина, которая характеризует управляющее воздействие (например, нажатие педали или поворот руля);
- ошибка модели ξ_k и ошибка сенсора η_k – случайные величины, законы распределения которых, не зависят от времени (от номера итерации k);
- средние значения ошибок равны нулю: $E\xi_k = E\eta_k = 0$;
- нам известны дисперсии закона распределения случайных величин σ_ξ^2 и σ_η^2. Отметим, что дисперсии не зависят от k, так как законы распределения не зависят от него;
- подразумевается, что все случайные ошибки независимы: не зависит какая ошибка будет в момент времени k от ошибки в другой момент времени k'.

В решаемой нами задаче нет зависимости между координатами. Поэтому мы можем использовать описанный ниже алгоритм отдельно для каждой координаты и обойтись без перехода к уравнениям в матричном виде.

В нашем случае данных о нажатии педалей и поворота руля нет, но есть данные о скорости и направлении движения транспортного средства. Эти данные мы можем использовать в качестве управляющего воздействия. Пусть v – скорость движения транспортного, φ – ориентация транспортного средства относительно северного направления по часовой стрелке, x – широта, y – долгота. Тогда система (1) для широты перепишется в виде:

$$\begin{matrix} x_{k+1} = x_k + \dfrac{v_k \cos\ \varphi_k \quad t_k - t_{k-1}}{111.1} + \xi_k \\ z_k = x_k + \eta_{k+1} \end{matrix} \quad (2)$$

Для долготы:

$$y_{k+1} = y_k - \frac{v_k \sin\ \varphi_k \quad t_k - t_{k-1}}{111.1 \cdot \cos\ x_k} + \xi_k \quad (3)$$
$$z_k = x_k + \eta_{k+1}$$

Численное значение в знаменателе – число километров в одном градусе широты. Рассчитано исходя из длины экватора $\frac{40000}{360} \approx 111.1$. Число километров в одном градусе долготы зависит от широты, поэтому в знаменателе появился $\cos\ x_k$.

Для решения такого рода задач на практике используется фильтр Калмана – лучший среди всех линейных фильтров.

Фильтр Калмана один из самых популярных способов фильтрации, который использует во многих областях науки и техники. По причине своей эффективности и простоты его можно встретить в GPS-приёмниках, при реализации систем управления, в обработчиках датчиков показаний.

Фильтр – это алгоритм обработки данных, который позволяет устранять шумы и лишнюю информацию. В фильтре Калмана есть возможность задать начальную информацию о характере системы, связи переменных и на основании этого построить более точную оценку, но даже в простейшем случае (без ввода предварительной информации) он дает отличные результаты.

Отметим, что задача фильтрации – это не задача сглаживания. Мы не преследуем цель сгладить данные с сенсора, мы стремимся получить наиболее близкое значение к действительной координате x_k.

2.2 Получение итерационных формул из алгоритма Калмана

Допустим, что на k-ом шаге нашли уже отфильтрованное значение сенсора x_k^{opt}, которое хорошо приближает истинную координату системы x_k.

Нам известно уравнение, которое контролирует изменение неизвестной нам координаты: $x_{k+1} = x_k + u_k + \xi_k$, поэтому если мы еще не получили значение с сенсора, то можно предположить, что на шаге $k+1$ система эволюционирует согласно этому закону и сенсор покажет значение, близкое к $x_k^{opt} + u_k$. Более тучную информацию, полученную с сенсора, пока не можем

выдать. С другой стороны на шаге $k+1$ мы будем иметь неточное показание сенсора z_{k+1}.

Идея Калмана заключается в том, чтобы получить наилучшее приближение к истинной координате x_{k+1}, мы должны выбрать среднее значение между показателем неточного сенсора z_{k+1} и предположение того, что ожидается увидеть $x_k^{opt} + u_k$. Показанию сенсора дадим вес K, а на предположительное значение вес $1-K$. Вследствие чего получается следующая формула: $x_{k+1}^{opt} = K \cdot z_{k+1} + \ 1-K \ \cdot (x_k^{opt} + u_k)$ (4).

K является коэффициентом Калмана, который зависит от шага итерации, который записывается как K_{k+1}, но чтобы не переполнять расчетные формулы, опустим его индекс. Необходимо выбрать коэффициент Калмана K таким, чтобы получившееся оптимальное значение координаты x_{k+1}^{opt} было бы наиболее приближено к истинной координате x_{k+1}. Например, нам известно, что сенсор очень точный, то мы доверимся его показаниям и дадим значению z_{k+1} больше весу (K близко к единице). В случае если сенсор наоборот не точен, тогда необходимо больше ориентироваться на теоретически предполагаемое значение $x_k^{opt} + u_k$.

Таким образом, чтобы найти точное значение коэффициента Калмана K, необходимо минимизировать ошибку: $e_{k+1} = x_{k+1} - x_{k+1}^{opt}$.

Для того чтобы переписать выражение для ошибки используем уравнения (1):

$$e_{k+1} = x_{k+1} - x_{k+1}^{opt} = x_{k+1} - Kz_{k+1} - \ 1-K \ \ x_k^{opt} + u_k \ =$$
$$= x_k + u_k + \xi_k - K \ x_k + u_k + \xi_k + n_{k+1} \ - (1-K)(x_k^{opt} + u_k) =$$
$$= \ 1-K \ \ x_k - x_k^{opt} + \xi_k \ - K\eta_{k+1} = \ 1-K \ \ e^2 + \xi_k \ - K\eta_{k+1}$$

Ошибка является случайной величиной, значения которой каждый раз различны. Найдем среднее значение от квадрата ошибки:

$$E \ e_{k+1}^2 \ = E \quad 1-K \ \ e_k + \ \xi_k \ - K\eta_{k+1} \ ^2 \ =$$
$$= E \ \ 1-K \ ^2 \ e_k + \ \xi_k \ ^2 - 2 \ 1-K \ \ e_k + \ \xi_k \ K\eta_{k+1} + K^2\eta_{k+1}^2 \ =$$

$$= E\ \ 1-K\ ^2\ e_k+\xi_k\ ^2\ -E\ 2\ 1-K\ \ e_k+\xi_k\ K\eta_{k+1}\ +E\ K^2\eta_{k+1}^2\ =$$
$$=\ 1-K\ ^2E\ e_k^2+2e_k\xi_k+\xi_k^2\ -2\ 1-K\ K\cdot E\ e_k+\xi_k\ +K^2E\ \eta_{k+1}^2\ =$$
$$=\ 1-K\ ^2\ Ee_k^2+2Ee_k\xi_k+E\xi_k^2\ -2\ K-K^2\ \ Ee_k+E\xi_k\ +K^2E\ \eta_{k+1}^2$$

Будем считать, что средние значения ошибок сенсора равны нулю, т.е. $E\eta_{k+1}=E\xi_k=Ee_k=0$. Все входящие в выражение случайные величины независимы, т.е. $E\ \eta_{k+1}\xi_k\ =E\ \xi_k e_k\ =E\ \eta_{k+1}e_k\ =0$. Поэтому:

$$E\ e_{k+1}^2\ =\ 1-K\ ^2\ Ee_k^2+E\xi_k^2\ +K^2E\ \eta_{k+1}^2\ =$$
$$=\ 1-K\ ^2\ Ee_k^2+\sigma_\xi^2\ +K^2\sigma_\eta^2$$

Мы будем минимизировать среднее значение от квадрата ошибки:

$$E\ e_{k+1}^2\ =\ 1-K\ ^2\ Ee_k^2+\sigma_\xi^2\ +K^2\sigma_\eta^2\to min$$

Для этого возьмем производную по K и приравняем выражение к нулю:

$$-2(1-K)\ Ee_k^2+\sigma_\xi^2\ +2K\sigma_\eta^2=0$$

Выразим K:

$$K-1\ \ Ee_k^2+\sigma_\xi^2\ +K\sigma_\eta^2=0$$
$$K\ Ee_k^2+\sigma_\xi^2\ +K\sigma_\eta^2=Ee_k^2+\sigma_\xi^2$$
$$K(Ee_k^2+\sigma_\xi^2+\sigma_\eta^2)=Ee_k^2+\sigma_\xi^2$$
$$K_{k+1}=\frac{Ee_k^2+\sigma_\xi^2}{Ee_k^2+\sigma_\xi^2+\sigma_\eta^2}$$

Это выражение пишем для коэффициента Калмана с индексом шага $k+1$, чем мы подчеркиваем, что он зависит от шага итерации.

Подставляем в выражение для среднеквадратической ошибки $E(e_{k+1}^2)$ минимизирующее значение коэффициента Калмана K_{k+1}.

Получаем:

$$E\ e_{k+1}^2\ =\ 1-\frac{Ee_k^2+\sigma_\xi^2}{Ee_k^2+\sigma_\xi^2+\sigma_\eta^2}\ ^2\ Ee_k^2+\sigma_\xi^2\ +\ \frac{Ee_k^2+\sigma_\xi^2}{Ee_k^2+\sigma_\xi^2+\sigma_\eta^2}\ ^2\sigma_\eta^2=$$

$$=\ 1-\frac{2\ Ee_k^2+\sigma_\xi^2}{Ee_k^2+\sigma_\xi^2+\sigma_\eta^2}+\ \frac{Ee_k^2+\sigma_\xi^2}{Ee_k^2+\sigma_\xi^2+\sigma_\eta^2}\ ^2\ Ee_k^2+\sigma_\xi^2\ +\frac{Ee_k^2+\sigma_\xi^2\ ^2\sigma_\eta^2}{Ee_k^2+\sigma_\xi^2+\sigma_\eta^2\ ^2}=$$

$$= \left(Ee_k^2 + \sigma_\xi^2\right) - \frac{\left(Ee_k^2 + \sigma_\xi^2\right)^2 \left(Ee_k^2 + \sigma_\xi^2 + 2\sigma_\eta^2\right)}{\left(Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2\right)^2} + \frac{\left(Ee_k^2 + \sigma_\xi^2\right)^2 \sigma_\eta^2}{\left(Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2\right)^2} =$$

$$= \left(Ee_k^2 + \sigma_\xi^2\right) - \frac{\left(Ee_k^2 + \sigma_\xi^2\right)^2 \left(Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2\right)}{\left(Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2\right)^2} =$$

$$= \left(Ee_k^2 + \sigma_\xi^2\right) - \frac{\left(Ee_k^2 + \sigma_\xi^2\right)^2}{Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2} = \frac{\left(Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2\right)\left(Ee_k^2 + \sigma_\xi^2\right) - \left(Ee_k^2 + \sigma_\xi^2\right)^2}{Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2} =$$

$$= \frac{\left(Ee_k^2 + \sigma_\xi^2\right)\left(Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2 - Ee_k^2 - \sigma_\xi^2\right)}{Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2} = \frac{\sigma_\eta^2 \left(Ee_k^2 + \sigma_\xi^2\right)}{Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2}$$

Итак,

$$E\left(e_{k+1}^2\right) = \frac{\sigma_\eta^2 \left(Ee_k^2 + \sigma_\xi^2\right)}{Ee_k^2 + \sigma_\xi^2 + \sigma_\eta^2} \quad (5).$$

Вывод: Итак, мы получили итерационную формулу для вычисления коэффициента Калмана (4), формулу вычисления среднеквадратичной ошибки (5) и можем использовать их для решения систем (2) и (3). Эти формулы были использованы при разработке программного кода сервера, описание которого представлено в следующей главе.

Глава 3 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ МОНИТОРИНГА АВТОТРАНСПОРТА

1.1 Предмет разработки и требования к серверной части мониторинга автотранспорта

Предметом разработки является серверная часть системы мониторинга автотранспорта.

Данные, получаемые от транспортных средств, представляют собой набор записей:

- идентификатор транспортного средства;
- идентификатор поездки;
- время измерений;
- географические данные (широта, долгота);
- скорость движения;
- направление движения.

В результате система должна принимать данные от транспортных средств, сохранять их в базу данных, обрабатывать (фильтровать) и возвращать отфильтрованные данные для отображения на сайте.

Разработанная система должна удовлетворять следующему набору требований:

1. данные транспортных средств должны поступать в систему посредством обработки GET-запросов и POST-запросов;
2. для хранения данных должна быть использована СУБД;
3. система должна фильтровать полученные данные от транспортных средств и устранять шумы;
4. система должна уметь обрабатывать данные независимо от погрешности измерений;
5. система должна отдавать отфильтрованные данные посредством обработки GET-запросов;

6. система должна быть многопоточной, иметь возможность обрабатывать несколько запросов одновременно;

7. система должна иметь возможность аутентификации пользователей;

8. главный администратор системы должен иметь возможность управления пользователями;

1.2 Архитектура серверной части мониторинга автотранспорта

Разрабатываемая система была реализована на языке программирования Java. В качестве сервера приложений использовался фреймворк Jersey по следующим причинам:

- подробная документация;
- поддержка асинхронной обработки запросов;
- встроенная поддержка JSON;
- простота организации маршрутизации.

Работа системы состоит из следующих этапов:

1. Добавление администратором водителей и транспортных средств.
2. Привязка администратором определенных водителей к определенным транспортным средствам.
3. Получение данных о поездках от транспортных средств по следующему циклу:
    а. признак начала поездки;
    б. данные, получаемые от транспортных средств, перечисленные в разделе “предмет разработки”;
    в. признак конца поездки.
4. Отправка данных на сайт или на мобильное приложение по запросу пользователя.

При получении от транспортного средства признака конца поездки полученные данные о координатах обрабатываются фильтром Калмана.

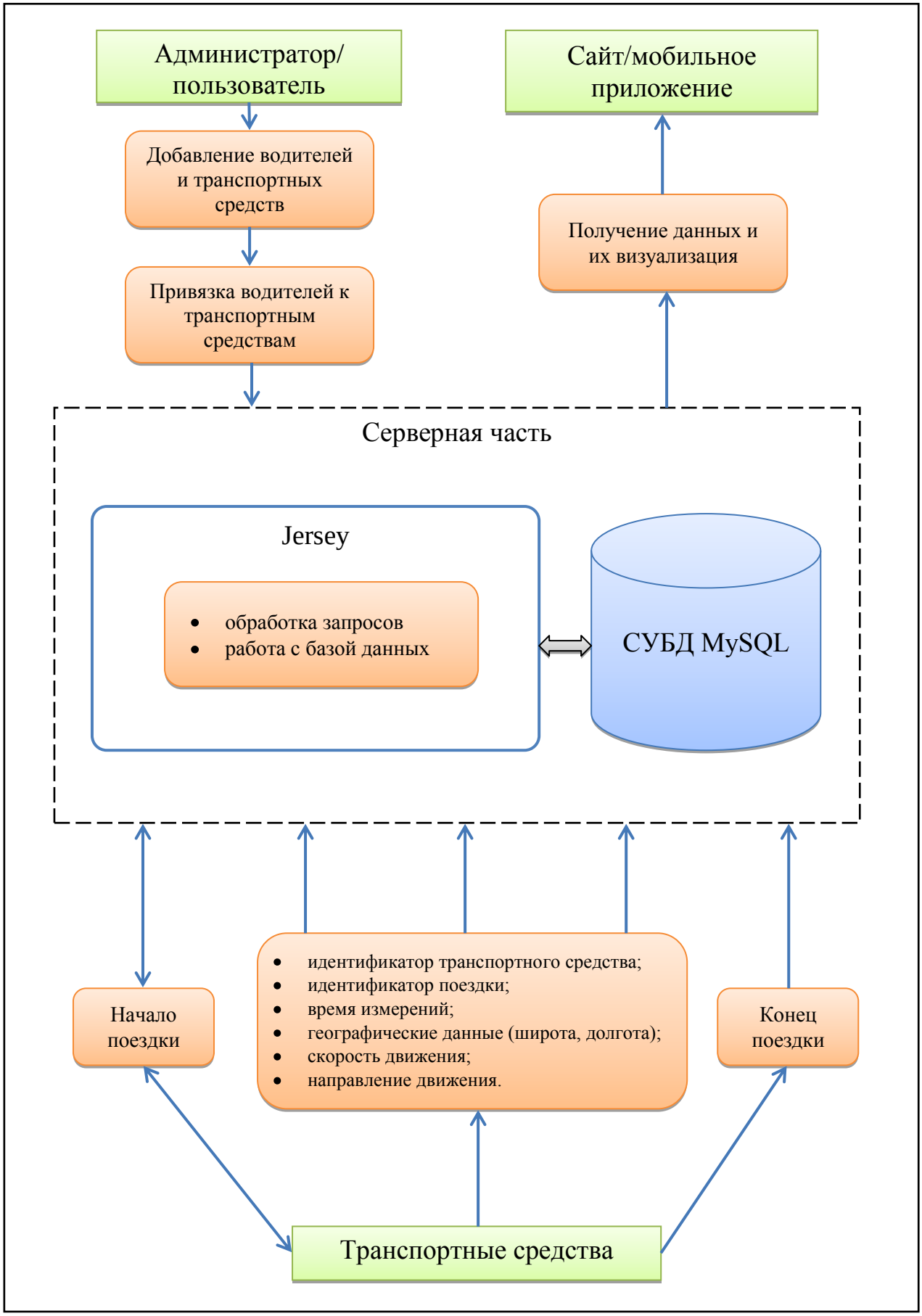

Рисунок 3.1 – Архитектура серверной части системы

Архитектура разрабатываемой системы представлена на рисунке 3.1. Имея общее представление об архитектуре серверной части системы мы можем приступить к разработке самого сервера.

1.3 Разработка базы данных для серверной части мониторинга автотранспорта

MySQL – это реляционная система управления базами данных с открытым исходным кодом.

Основными критериями для выбора данной СУБД стали:

- поддержка большого количества платформ, в том числе Linux и Windows;
- имеет API и библиотеки для множества языков программирования (в том числе для Java) для получения доступа к базам данных;
- бесплатна;
- поддержка Unicode (UTF-8);
- поддержка кеширования запросов;
- возможность написания вложенных запросов;
- простота развертывания;
- обладает высокой производительностью благодаря уникальной архитектуре движка хранения данных.

СУБД использована для хранения данных, полученных от транспортных средств, а так же данных о самих транспортных средствах и водителях.

В системе должны быть пользователи (администраторы), которые могут управлять водителями и транспортными средствами, а именно:

- добавлять водителей и транспортные средства;
- удалять водителей и транспортные средства;
- связывать водителей и транспортные средства между собой.

Код для создания таблиц пользователей, транспортных средств и водителей соответственно приведен на рисунке 3.2.

В таблице user имеется поле hash. В нем мы будем хранить MD5-hash строки, получаемой соединением логина и пароля. Это сделано в целях безопасности. В этом случае даже если злоумышленник сможет заполучить базу данных, то он не сможет узнать пароли пользователей, так как MD5-hash не имеет обратного алгоритма, алгоритма дешифрования.

Для установки связей между водителями и транспортными средствами нужна отдельная таблица, так как возможна ситуация, когда либо у водителя временно нет транспортного средства, либо он пользуется разными транспортными средствами.

```
CREATE TABLE user
(
  id    int           PRIMARY KEY,
  login nvarchar(40)  NOT NULL  UNIQUE,
  hash  BINARY(32)    NOT NULL
);

CREATE TABLE truck
(
  id      int           PRIMARY KEY,
  mark    int           NOT NULL,
  model   int           NOT NULL,
  color   nvarchar(20)  NOT NULL,
  number  nvarchar(20)  NOT NULL
);

CREATE TABLE driver
(
  id          int           PRIMARY KEY,
  firstName   nvarchar(32),
  secondName  nvarchar(32),
  patronymic  nvarchar(32)
);
```

Рисунок 3.2 – Таблицы пользователей, ТС и водителей

Код для создания таблицы связей водителей и транспортных средств представлен на рисунке 3.3.

```
CREATE TABLE ownership
(
  id        int PRIMARY KEY,
  driverId  int NOT NULL,
  truckId   int NOT NULL,
  CONSTRAINT FOREIGN KEY (driverId) REFERENCES driver(id),
  CONSTRAINT FOREIGN KEY (truckId)  REFERENCES truck(id)
);
```

Рисунок 3.3 – Таблица связей

В таблице связей ownership были использованы внешние ключи на таблицы driver и truck. Это было сделано по ряду причин:

- внешние ключи позволяют исключить появление в базе некорректных данных, например, связи несуществующего водителя с каким-либо транспортным средством и т.п.;
- внешние ключи помогают другим администраторам и программистам понять взаимосвязи между таблицами, просто взглянув на схему базы данных;
- внешние ключи позволяют ускорить выборку данных при использовании запроса с JOIN.

Для хранения информации о поездках были созданы еще две таблицы, схема которых представлена на рисунке 3.4.

```
41  CREATE TABLE trip
42  (
43    tripId      int PRIMARY KEY AUTO_INCREMENT,
44    ownershipId int NOT NULL,
45    kalman      bit NOT NULL    DEFAULT 0,
46    CONSTRAINT FOREIGN KEY (ownershipId) REFERENCES ownership (id)
47  );
48
49  CREATE TABLE track
50  (
51    tripId      int             NOT NULL,
52    lat         decimal(10, 8)  NOT NULL,
53    lng         decimal(11, 8)  NOT NULL,
54    orientation float           NOT NULL,
55    fuel        float           NOT NULL,
56    speed       int             NOT NULL,
57    time        timestamp       NOT NULL,
58    CONSTRAINT FOREIGN KEY (tripId) REFERENCES trip(tripId)
59  );
```

Рисунок 3.4 – Таблицы информации о поездках

Эти две таблицы позволяют хранить наборы данных, о которых было сказано в разделе «предмет разработки». Помимо описанных ранее данных, в таблице trip появилось поле kalman. Оно свидетельствует о том, завершена ли поездка и обработаны ли данные фильтром Калмана.

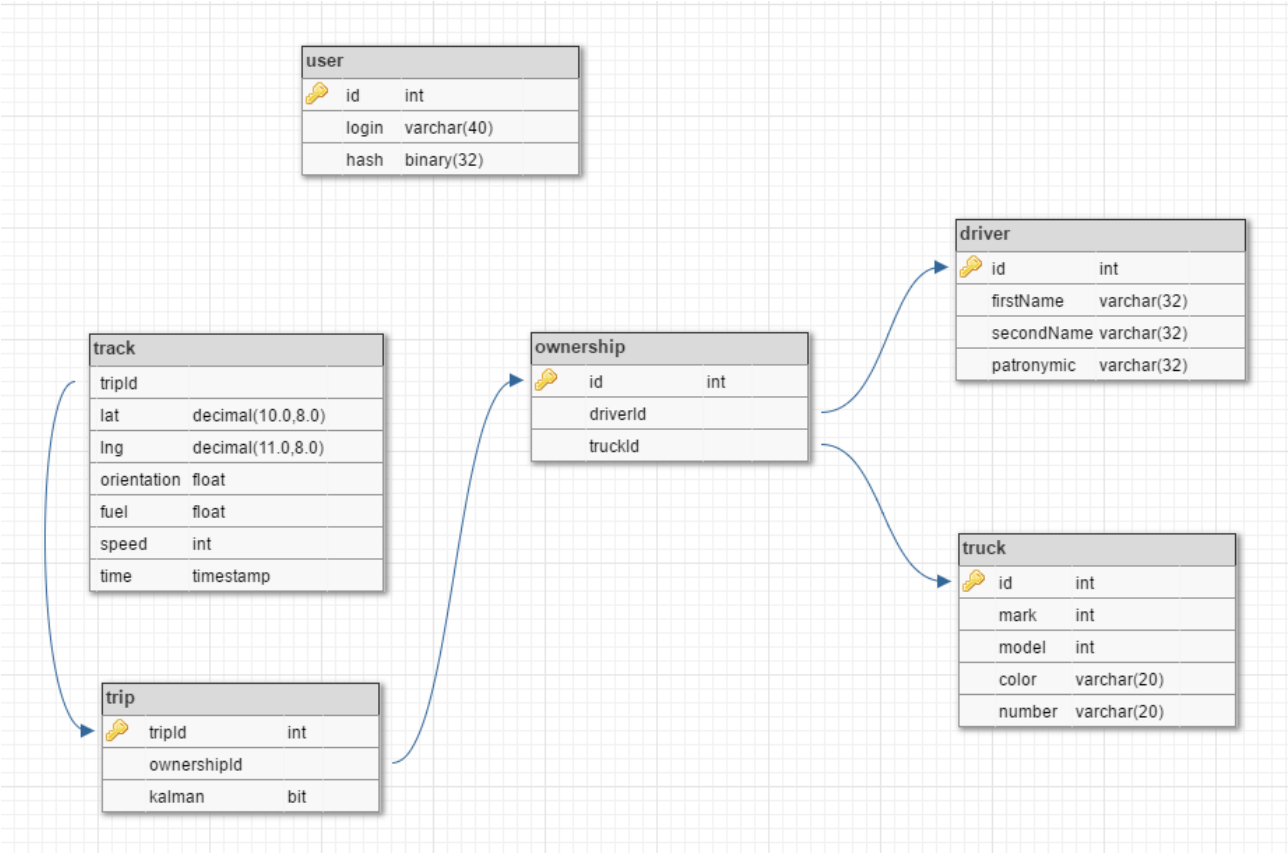

Рисунок 3.5. – Диаграмма базы данных

Целиком схема базы данных представлена в приложении А, а диаграмма, включающая в себя описание структуры, ограничения целостности и содержание необходимых данных базы данных представлена на рисунке 3.5.

1.4 Разработка программного кода серверной части мониторинга автотранспорта

Прежде всего, на начальном этапе разработки программного кода сервера необходимо определиться с протоколом, по которому будут общаться клиент и сервер. Под клиентом подразумевается либо сайт, либо мобильное приложение. Выбор Jersey предполагает RESTful архитектуру и использование протокола HTTP. Но речь идет не о нем, а о протоколе, который будет использоваться поверх него. Все данные между клиентам и сервером было принято передавать в формате JSON.

JSON – это формат хранения данных, который может воспринимать как человек, так и машина. Это текстовый формат, который не относится ни к

каким языкам программирования, но использует соглашения, которые близки к соглашениям семейства языков C, включая C, C++, C#, Java, JavaScript, Perl, Python и многие другие. Эти свойства делают JSON идеальным форматом обмена данными.

Протокол передачи данных между клиентом и сервером основан на формате JSON. В рамках формата JSON были определены два шаблона, с помощью которых можно передавать все данные. При обработке запросов сервером важным признаком является успешность обработки. Клиенту, как правило, ничего обрабатывать не нужно. Поэтому во всех ответах от сервера в JSON объекте должно присутствовать поле «success». Так клиент сможет определить, успешность обработки запроса и, соответственно, отобразить ошибку конечному пользователю. Помимо поля успешности иногда серверу нужно передать клиенту какую-либо информацию, а в случае неуспешной обработки запроса передать строку с ошибкой.

С учетом вышесказанного были написаны два класса (рисунок 3.6 и 3.7):

```
3   public class AnswerBoolean {
4       private boolean success;
5       private String error;
6
7       public AnswerBoolean(boolean success) { this.success = success; }
11
12      public AnswerBoolean(boolean success, String error)
13      {
14          this.success = success;
15          this.error = error;
16      }
17  }
```

Рисунок 3.6 – Класс AnswerBoolean

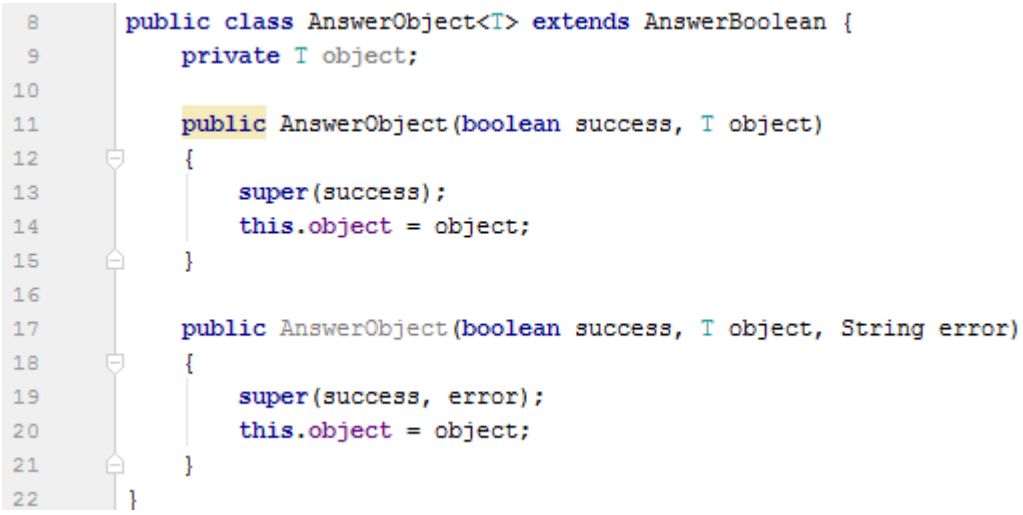

Рисунок 3.7 – Класс AnswerObject

Класс AnswerBoolean используется, когда клиенту важно знать только результат обработки запроса, но никакой информации от сервера он не ждет. Например, при входе пользователю можно отправить сообщение о том, что он успешно аутентифицирован, либо ошибку и сообщение об ошибке (либо неверный пароль, либо нет доступа к базе данных).

Класс AnswerObject используется, как правило, когда клиент запрашивает у сервера информацию, которую нельзя описать одним параметром «success». Например, клиент может запросить данные о водителе с определенным id. Тогда в случае успешной обработки запроса в JSON объекте в поле object будут находиться данные о водителе. В случае ошибки при обработке запроса, опять же, будет использован класс AnswerBoolean, так как нет возможности вернуть какой-либо объект.

Таким образом, весь протокол обмена данными строится на двух классах – AnswerBoolean и AnswerObject.

Для преобразования объекта класса в формат JSON использовалась библиотека GSON. Эта библиотека позволяет также делать и обратное – преобразовывать JSON строку в объект класса. Данная библиотека была выбрана по ряду причин:

- имеет возможность обработки коллекций, шаблонов классов и вложенных классов;

- в процессе преобразования JSON-строки к объекту класса проход идет по элементам объекта класса, что позволяет игнорировать дополнительные поля в JSON строке;
- имеет возможность преобразования объекта класса в «удобочитаемый» JSON;
- имеет возможность задать способ обработки null-объектов (по умолчанию них нет на выходе в JSON-строке).

Преобразование JSON-строки в объект класса нужно тогда, когда клиенту нужно передать на сервер более сложный объект, чем, например, идентификатор, который можно передавать через GET-запрос. Для передачи более сложного объекта нужно использовать POST-запрос, а в его теле как раз передавать JSON объект. Например, POST-запрос можно использовать для передачи логина и пароля при аутентификации пользователя.

Помимо использования POST-запроса для аутентификации пользователей на серверной стороне необходимо использовать механизм сессий. Рассмотрим работу POST-запроса и механизма сессий на примере функции аутентификации пользователя login (рисунок 3.8), которая была написана в процессе разработки программного кода сервера.

```
@POST
@Path("/login")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public Response login(User user, @Context HttpServletRequest request)
```

Рисунок 3.8 – Объявление функции login

Аннотация @Path("/login") указывает часть URL, по которому надо проходить при аутентификации. Аннотации @Produces и @Consumes указывают на то, что на входе и на выходе будет JSON-объект. Входной параметр функции – объект класса User (рисунок 3.9).

```
6   public class User {
7       private String login;
8       private String password;
9
10      public String getLogin() { return login; }
13
14      public String getPassword() { return password; }
17
18      public void setLogin(String login) { this.login = login; }
21
22      public void setPassword(String password) { this.password = password; }
25  }
```

Рисунок 3.9 – Класс User

Благодаря указанным выше аннотациям JSON-строка в POST-запросе автоматически будет преобразована в объект класса User. Второй параметр – объект класса HttpServletRequest, который содержит в себе информацию об HTTP-запросе.

При аутентификации в первую очередь надо проверить, не аутентифицирован ли клиент (рисунок 3.10).

```
// Проверим, не залогинены ли мы
if (request.getSession( create: false) != null)
{
    return Response.status(200).entity(new AnswerBoolean( success: false,
        error: "You are already signed in.")).build();
}
```

Рисунок 3.10 – Проверка прежней аутентификации

Для этого запрашивается текущая сессия. Аргумент false функции запроса сессии указывает на то, сессию не надо создавать в случае ее отсутствия. Если сессия существует, то пользователю отправляется сообщение о том, что он уже аутентифицирован. Пример такого ответа в формате JSON представлен на рисунке 3.11.

```
1 {
2     "success": false,
3     "error": "You are already signed in."
4 }
```

Рисунок 3.11 – Пример ответа на повторную аутентификацию

Если сессии нет, то нужно взять логин и пароль, посчитать их MD5-hash, сравнить его с тем, что лежит в базе данных для полученного login в классе

User. Для подсчета MD5-hash была написана отдельная функция, которая представлена на рисунке 3.12

```
82  @   private String getHash(String string) throws NoSuchAlgorithmException {
83          // Задаем алгоритм хеширования
84          MessageDigest md = MessageDigest.getInstance("MD5");
85          // Переводим строку в массив байт и получаем его hash
86          byte[] thedigest = md.digest(string.getBytes());
87
88          // Преобразуем полученный массив байт (hash) в строку
89          StringBuffer sb = new StringBuffer();
90          for (byte b : thedigest) {
91              sb.append(String.format("%02x", b & 0xff));
92          }
93
94          return sb.toString();
95      }
```

Рисунок 3.12 – Функция хеширования строки

Функция getHash принимает произвольную строку и возвращает ее MD5-hash опять же в виде строки.

Вызов этой функции выглядит следующим образом:

```
// Складываем hash из логина и пароля
String hash = getHash( string: user.getLogin() + user.getPassword());
```

Рисунок 3.13 – Вызов функции getHash

Далее проверяем hash с тем, что лежит в базе данных.

```
PreparedStatement stmt = conn.prepareStatement( sql: "SELECT id FROM user WHERE login=? AND hash=?");

stmt.setString( parameterIndex: 1, user.getLogin());
stmt.setString( parameterIndex: 2, hash);

ResultSet rs = stmt.executeQuery();

AnswerBoolean ans;

// Есть в базе запись с таким же логином и хешем, "аутентификация пройдена"
if (rs.next()) {
    int id = rs.getInt( columnLabel: "id");

    HttpSession session = request.getSession( create: true);
    session.setAttribute( name: "userId", id);

    ans = new AnswerBoolean( success: true);
} else {
    ans = new AnswerBoolean( success: false,  error: "Wrong login or password.");
}
```

Рисунок 3.14 – Поиск логина и hash в БД

Для отправляем в базу запрос с выборкой id пользователя, у которого логин и hash совпадают с полученными от клиента.

Если такая строка в базе нашлась, и мы получили id пользователя, то создаем новую сессию, присваиваем переменной сессии userId id текущего пользователя и формируем «успешный» ответ для клиента. Переменные сессии безопасны с точки зрения инъекций. Они хранятся только на сервере и скрыты от клиента, поэтому он никак не может на них повлиять с целью взлома системы.

Если такой строки нет, то мы формируем клиенту «неуспешный» ответ о том, что допущена ошибка в логине или пароле.

Далее мы закрываем соединение в базой и отправляем сформированный ответ клиенту (рисунок 3.15).

```
rs.close();
stmt.close();
conn.close();

return Response.status(200).entity(ans).build();
```

Рисунок 3.15 – Закрытие соединения с БД и отправка ответа

Все остальные запросы на сервер, связанные с добавлением пользователей, удалением, сменой пароля построены аналогичным образом (приложение В).

Теперь рассмотрим ту часть сервера, которая связана с предыдущей главой, а именно с применением фильтра Калмана. Архитектура сервера построена так, что сначала клиент (теперь под клиентом подразумевается транспортное средство) посылает GET-запрос с идентификатором водителя и транспортного средства на сервер, после чего клиент получает идентификатор поездки. Далее клиент через определенные интервалы времени посылает POST-запрос с данными о поездке, которые были описаны в разделе «предмет разработки». Когда поездка завершена, клиент отправляет на сервер GET-запрос с идентификатором поездки, после чего на сервере происходит обработка всего пути, применяется фильтр Калмана.

Программный код обработки запроса конца поездки и обработки данных фильтром Калмана представлен в приложениях В и Г соответственно.

Для проверки работы обработки запросов в систему были загружены тестовые данные. Исходные данные поездки изображены на рисунке 3.16.

Приведенные ниже рисунки показывают, что применения фильтра Калмана позволяет снизить «шумы» траектории движения транспортного средства и получить более реалистичную модель его перемещения.

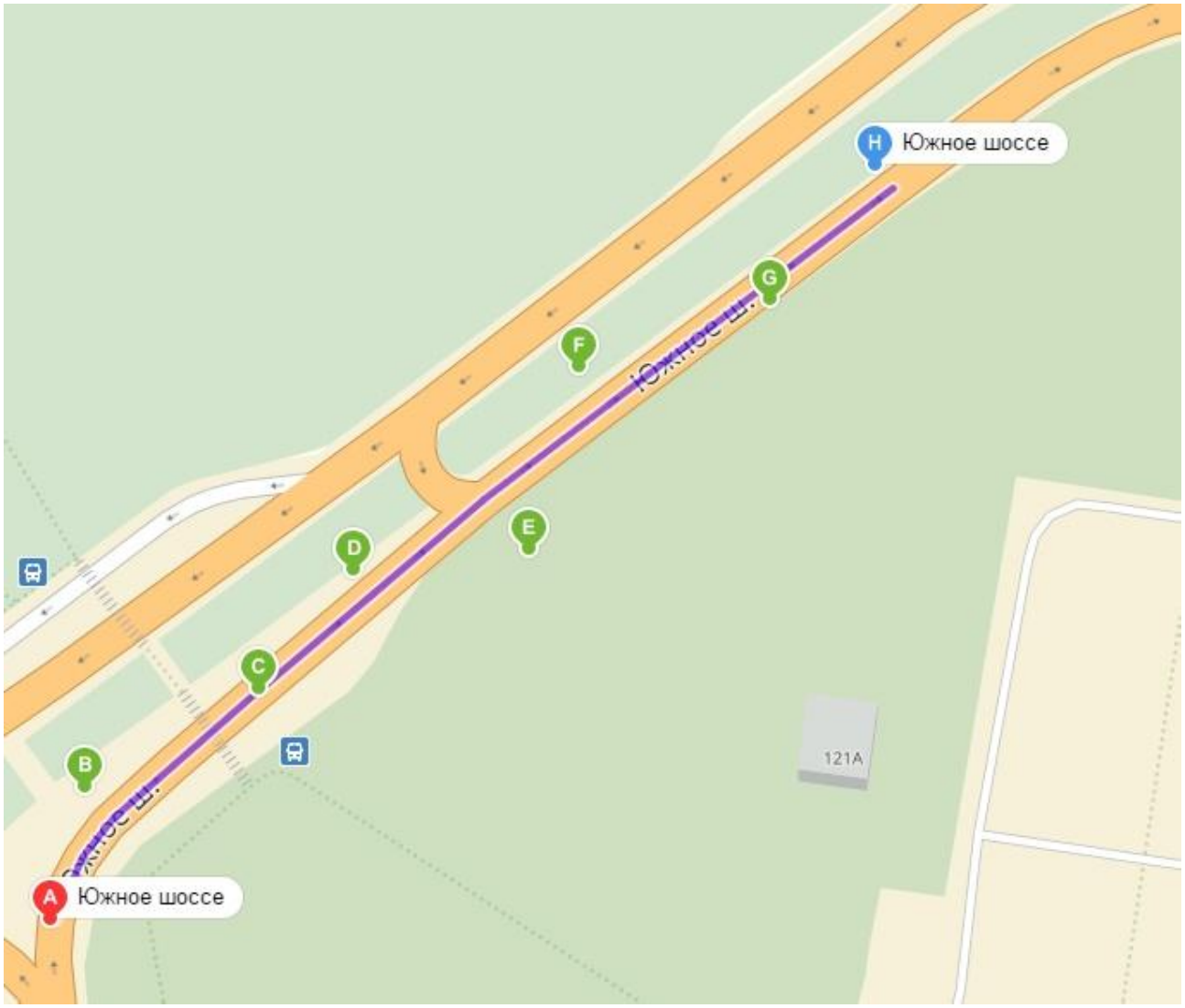

Рисунок 3.16 – Визуализация необработанных тестовых данных

Результат обработки запроса конца поездки представлен на рисунке 3.17

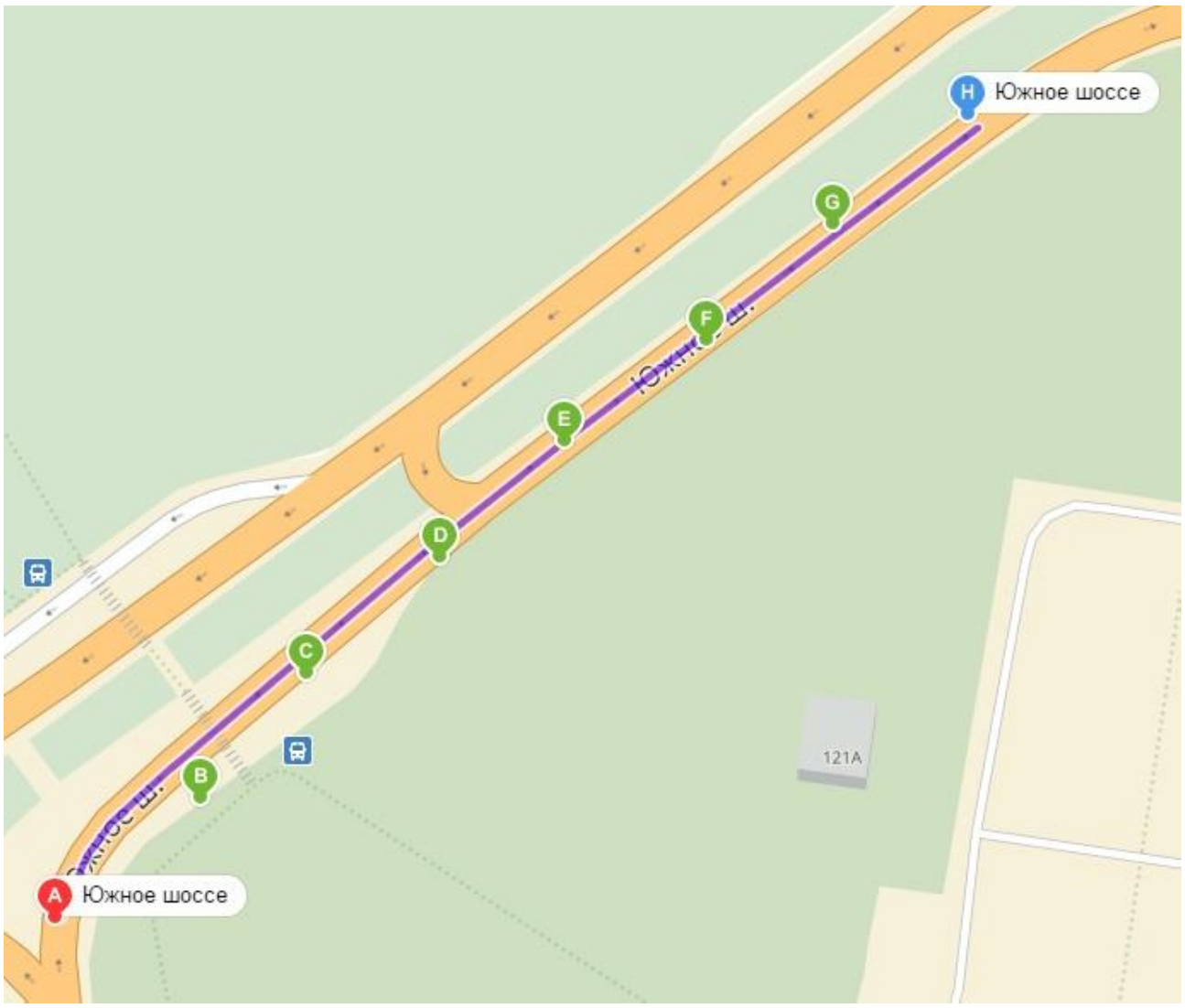

Рисунок 3.17 – Визуализация данных поездки после обработки фильтром Калмана

Мы рассмотрели части сервера, которые отвечают за обработку данных, полученных от транспортных средств данных и работу с пользователями. Теперь рассмотрим на примере функции обработки GET-запроса getTrucksByDriver (приложение Б) ту часть сервера, которая позволяет сайту или мобильному приложению получить данные для отображения.

Функция getTrucksByDriver принимает на вход параметр driverId и объект класса HttpServletRequest, который содержит в себе информацию об HTTP-запросе. Функция обрабатывает запросы по пути /getdata/trucksByDriver/{driverId}, где {driverId} и есть входной параметр функции. В случае успешной обработки запроса на выходе функции должен

быть список автомобилей, с которыми связан водитель с идентификатором driverId.

Работа функции начинается с аутентификации клиента (рисунок 3.18).

```
HttpSession session = request.getSession( create: false);

if (session == null)
    return Response.status(200).entity(new AnswerBoolean( success: false,  error: "Please, sign in.")).build();

int userId = (Integer) session.getAttribute( name: "userId");

// Если залогинены
if (userId != 0) {
```

Рисунок 3.18. – Авторизация клиента

Сначала проверяется, создана ли для клиента сессия. Если нет, то значит клиент не аутентифицирован и ему высылается «неуспешный» ответ с сообщением о том, что он должен пройти аутентификацию. Далее проверяется переменная сессии userId и если она не 0, то пользователь считается аутентифицированным.

Далее необходимо проверить, есть ли в базе водитель, для которого запрашивается список автомобилей (рисунок 3.19).

```
PreparedStatement checkDriver = conn.prepareStatement( sql: "SELECT 1 FROM driver WHERE id = ?");
checkDriver.setInt( parameterIndex: 1, driverId);

checkDriver.executeQuery();
```

Рисунок 3.19. – Проверка существования водителя

Если такого водителя не существует, то клиенту отправляется «неуспешный» ответ с сообщением о том, что водитель не найден.

Если водитель существует, то нужно выбрать из базы список автомобилей, с которыми он связан. Для этого необходимо использовать сложный вложенный запрос:

```
SELECT * FROM truck WHERE id IN (SELECT truckid FROM ownership WHERE driverid = ?)
```

Внутренний запрос выбирает идентификаторы всех автомобилей, которыми владеет водитель. Внешний выбирает всю информацию об

автомобилях, идентификатор которых совпадает с теми, что были выбраны во внутреннем запросе.

Отправка в базу данных описанного запроса представлена на рисунке 3.20.

```
PreparedStatement getTrucks = conn.prepareStatement
        ( sql: "SELECT * FROM truck WHERE id IN (SELECT truckid FROM ownership WHERE driverid = ?)");
getTrucks.setInt( parameterIndex: 1, driverId);

ResultSet getTrucksRS = getTrucks.executeQuery();
```

Рисунок .3.20 – Отправка сложного запроса в базу данных

Далее нужно получить от базы результат запроса и отправить ответ клиенту(рисунок 3.21).

```
boolean hasRows = false;
ArrayList<Truck> trucks = new ArrayList<Truck>();

while (getTrucksRS.next()) {
    hasRows = true;

    int id = getTrucksRS.getInt( columnLabel: "id");
    String mark = getTrucksRS.getString( columnLabel: "mark");
    String model = getTrucksRS.getString( columnLabel: "model");
    String color = getTrucksRS.getString( columnLabel: "color");
    String number = getTrucksRS.getString( columnLabel: "number");

    trucks.add(new Truck(id, mark, model, color, number));
}

if (hasRows == false)
{
    return Response.status(200).entity(
            new AnswerBoolean( success: false,  error: "Driver has no trucks.")).build();
}

return Response.status(200).entity(
        new AnswerObject<ArrayList<Truck>>( success: true, trucks)).build();
```

Рисунок 3.21. – Получение результата запроса и отправка ответа.

Для этого создадим массив объектов класса Truck. Далее в цикле проходим построчно по результатам запроса. На каждой итерации создаем объект класса Truck с параметрами из выбранной строки и помещаем этот объект в массив.

На случай, если водитель не связан ни с какими транспортными средствами, предусмотрен флаг hasRows. Если выборка из базы была пустой, то обработки в цикле не будет и флаг будет установлен в значение false. При этом клиенту будет отправлен «неуспешный» ответ с сообщением о том, что водитель не связан ни с какими транспортными средствами.

Если же выборка из базы не пустая, то флаг hasRows принимает значение true. В этом случае клиенту в качестве объекта отправляется массив автомобилей. Пример результата успешно обработанного запроса представлен на рисунке 3.22.


```
{
    "object": [
        {
            "id": 1,
            "mark": "2",
            "model": "1",
            "color": "синий",
            "number": "а321вг123"
        },
        {
            "id": 2,
            "mark": "1",
            "model": "1",
            "color": "красный",
            "number": "а123вг123"
        }
    ],
    "success": true,
    "error": null
}
```


Рисунок 3.22. – Пример результата обработки запроса

На рисунке 3.23 представлен пример программного кода класса SmallTrack

```
public class SmallTrack {
    private double lat;
    private double lng;
    private float orientation;
    private int speed;
    private Timestamp time;

```

```
 8     public double getLat() {
 9         return lat;
10     }
11
12     public double getLng() {
13         return lng;
14     }
15
16     public int getSpeed() {
17         return speed;
18     }
19
20     public float getOrientation() {
21         return orientation;
22     }
23
24     public Timestamp getTime() {
25         return time;
26     }
27
28     public void setLat(double lat) {
29         this.lat = lat;
30     }
31
32     public void setLng(double lng) {
33         this.lng = lng;
34     }
35
36     public SmallTrack(double lat, double lng, float orientation,
37                       int speed, Timestamp time) {
38         this.lat = lat;
39         this.lng = lng;
40         this.orientation = orientation;
41         this.speed = speed;
42         this.time = time;
```

```
43      }
44 }
```

Вывод: Итак, мы рассмотрели, каким образом сервер общается с клиентами, как он обрабатывает запросы и взаимодействует с базой данных на трех примерах, каждый из которых относится к своей части (выделены зеленым цветом) архитектуры сервера (рисунок 3.1).

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была описана актуальность рассматриваемой темы, определены объект и предмет выпускной квалификационной работы, поставлена цель и выявлены задачи. Так же был изучен принцип работы систем мониторинга передвижения автотранспорта и проблемы, возникающие при обработке данных, полученных от транспортных средств.

В соответствии с поставленными задачами в работе описана математическая модель фильтра Калмана с соответствующими математическими выкладками и описан способ применения этой модели для решения проблемы фильтрации, полученных от GPS-трекера данных.

По построенной архитектурой серверной части, была разработана схема базы данных для СУБД MySQL и сам сервер для мониторинга автотранспорта с применением фильтра Калмана.

Разработанная система полностью удовлетворяет заявленным требованиям:

- данные транспортных средств поступают в систему посредством обработки GET-запросов и POST-запросов;
- для хранения данных использована СУБД MySQL;
- система фильтрует полученные данные от транспортных средств и устраняет шумы;
- система обрабатывает данные независимо от погрешности измерений;
- система отдает отфильтрованные данные посредством обработки GET-запросов;
- система имеет возможность обрабатывать несколько запросов одновременно;
- система иметь механизм аутентификации пользователей;
- главный администратор системы имеет возможность управления пользователями.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Нормативно-правовые акты

1. ГОСТ 7.1-2003. Библиографическая запись. Библиографическое описание документа.

2. ГОСТ 7.82-2001. Библиографическая запись. Библиографическое описание электронных ресурсов.

3. ГОСТ 2.105 – 95. Общие требования к текстовым документам [Текст]. – М.: Изд-во стандартов, 1996. – 29 с. – (Единая система конструкторской документации).

4. ГОСТ 7.32-2001. Отчет о научно-исследовательской работе. Структура и правила оформления.

5. ГОСТ 19.701 – 90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения (ИСО 5807–85) [Текст]. Введен 1992–01–01. – М.: Изд-во стандартов, 1992. – 14 с. – (Единая система программной документации).

Научная и методическая литература

6. Богданов М.Р. Применение GPS/ГЛОНАСС. Издательство: ИД Интеллект, 2012 г. – 137 стр.

7. Грег Риккарди. Системы баз данных. Теория и практика использования в Internet и среде Java. Издательство: Вильямс, 2001 г. – 480 стр.

8. Егоров, А.Г. Правила оформления выпускных квалификационных работ по программам подготовки бакалавра и специалиста: учебно-методическое пособие / А.Г. Егоров, В.Г. Виткалов, Г.Н. Уполовникова, И.А. Живоглядова – Тольятти: ТГУ, 2012. – 135 с.

9. Леонтьев Б.К. GPS: Все, что Вы хотели знать, но боялись спросить. Издательство: Бук-Пресс и К, 2005 г. – 352 стр.

10. Молчанов А.Ю. Системное программное обеспечение. Учебник для вузов. 3-е изд. Издательство: СПб.:Питер, 2010 г. – 400 стр.

11. Положение о выпускной квалификационной работе. – Тольятти: ТГУ. – 2014.

12. Рудинский И. Д. Технология проектирования автоматизированных систем обработки информации и управления [Электронный ресурс] : учеб. пособие / И. Д. Рудинский. - Москва : Горячая линия - Телеком, 2011. - 304 с

13. Федеральный государственный образовательный стандарт высшего образования направления подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем (уровень бакалавриата), утвержденный приказом Минобрнауки России от 12.03.2015 N222.

14. Энтони Гонсалвес. Изучаем Java EE 7. Издательство: СПб.:Питер, 2014 г. – 640 стр.

15. Яценков В.С. Основы спутниковой навигации. Системы GPS, NAVSTAR и ГЛОНАСС. Издательство: Москва, 2005 г. – 272 стр.

Литература на иностранном языке

16. Cyprian M. Wronka, Matthew W. Dunnigan. Internet remote control interface for a multipurpose robotic arm, Electronic & Computer Engineering, 2009.

17. Keqiang Zhang, Bingjie Qu. Design and Implementation of Browser based GPS/GPRS Vehicle Positioning and Tracking System, MATEC Web of Conferences, 2015.

18. Maged N Kamel Boulos, Steve Wheeler. How smartphones are changing the face of mobile and participatory healthcare: an overview, with example from eCAALYX, BioMedical Engineering OnLine, 2011.

19. Matthew Rutishauser, Vladislav Petkov. CARNIVORE: A Disruption-Tolerant System for Studying Wildlife, Hindawi Publishing Corporation, 2010.

20. Paul P. Gardner. A RESTful API for Accessing Microbial Community Data for MG-RAST, MG-RAST. PLoS Comput Bio, 2015.

ПРИЛОЖЕНИЕ А

Схема базы данных

```
1 CREATE TABLE user
2 (
3   id    int           PRIMARY KEY AUTO_INCREMENT,
4   login nvarchar(40)  NOT NULL UNIQUE,
5   hash  BINARY(32)    NOT NULL
6 );
7
8 CREATE TABLE truck
9 (
10  id      int           PRIMARY KEY AUTO_INCREMENT,
11  mark    int           NOT NULL,
12  model   int           NOT NULL,
13  color   nvarchar(20)  NOT NULL,
14  number  nvarchar(20)  NOT NULL
15 );
16
17 CREATE TABLE driver
18 (
19  id          int           PRIMARY KEY AUTO_INCREMENT,
20  firstName   nvarchar(32),
21  secondName  nvarchar(32),
22  patronymic  nvarchar(32)
23 );
24
25 CREATE TABLE ownership
26 (
27  id        int PRIMARY KEY AUTO_INCREMENT,
28  driverId  int NOT NULL,
29  truckId   int NOT NULL,
30  CONSTRAINT FOREIGN KEY (driverId) REFERENCES driver(id),
31  CONSTRAINT FOREIGN KEY (truckId)  REFERENCES truck(id)
32 );
```

```
33
34 CREATE TABLE trip
35 (
36   tripId      int PRIMARY KEY AUTO_INCREMENT,
37   ownershipId int NOT NULL,
38   kalman      bit NOT NULL    DEFAULT 0,
39   CONSTRAINT FOREIGN KEY (ownershipId) REFERENCES ownership (id)
40 );
41
42 CREATE TABLE track
43 (
44   tripId      int             NOT NULL,
45   lat         decimal(10, 8)  NOT NULL,
46   lng         decimal(11, 8)  NOT NULL,
47   orientation float           NOT NULL,
48   fuel        float           NOT NULL,
49   speed       int             NOT NULL,
50   time        timestamp       NOT NULL,
51   CONSTRAINT FOREIGN KEY (tripId) REFERENCES trip(tripId)
52 );
```

ПРИЛОЖЕНИЕ Б

Программный код функции getTrucksByDriver

```
@GET
@Path("/trucksByDriver/{driverId}")
@Produces(MediaType.APPLICATION_JSON)
public Response getTrucksByDriver(@PathParam("driverId") int
driverId, @Context HttpServletRequest request) throws
SQLException, ClassNotFoundException {

    HttpSession session = request.getSession(false);

    if (session == null)
        return Response.status(200).entity(new
AnswerBoolean(false, "Please, sign in.")).build();

    int userId = (Integer) session.getAttribute("userId");

    // Если залогинены
    if (userId != 0) {
        Class.forName("com.mysql.jdbc.Driver");
        Connection conn =
DriverManager.getConnection("jdbc:mysql://212.109.221.253/trans
port", "gridya", "2486258");

        PreparedStatement checkDriver =
conn.prepareStatement("SELECT 1 FROM driver WHERE id = ?");
        checkDriver.setInt(1, driverId);

        checkDriver.executeQuery();

        ResultSet checkDriverRS = checkDriver.executeQuery();
        if (!checkDriverRS.next())
        {
            checkDriverRS.close();
```

```
33              checkDriver.close();
34              conn.close();
35
36              return Response.status(200).entity(new
37  AnswerBoolean(false, "Driver not found.")).build();
38          }
39
40          PreparedStatement getTrucks = conn.prepareStatement
41                  ("SELECT * FROM truck WHERE id IN (SELECT
42  truckid FROM ownership WHERE driverid = ?)");
43          getTrucks.setInt(1, driverId);
44
45          ResultSet getTrucksRS = getTrucks.executeQuery();
46
47          boolean hasRows = false;
48          ArrayList<Truck> trucks = new ArrayList<Truck>();
49
50          while (getTrucksRS.next()) {
51              hasRows = true;
52
53              int id = getTrucksRS.getInt("id");
54              String mark = getTrucksRS.getString("mark");
55              String model = getTrucksRS.getString("model");
56              String color = getTrucksRS.getString("color");
57              String number = getTrucksRS.getString("number");
58
59              trucks.add(new Truck(id, mark, model, color,
60  number));
61          }
62
63          if (hasRows == false)
64          {
65              return Response.status(200).entity(new
66  AnswerBoolean(false, "Driver has no trucks.")).build();
67          }
```

```
68
69          return Response.status(200).entity(new
70  AnswerObject<ArrayList<Truck>>(true, trucks)).build();
71      }
72      else // Не залогинены
73      {
74          return Response.status(200).entity(new
75  AnswerBoolean(false, "Please, sign in.")).build();
76      }
77  }
```

ПРИЛОЖЕНИЕ В

Обработка POST-запроса смены пароля

```
1 @POST
2 @Path("/changepass")
3 @Produces(MediaType.APPLICATION_JSON)
4 @Consumes(MediaType.APPLICATION_JSON)
5 public Response changePass(User user, @Context
6 HttpServletRequest request)
7    throws NoSuchAlgorithmException, ClassNotFoundException,
8 SQLException {
9     HttpSession session = request.getSession(false);
10
11    if (session != null)
12    {
13
14        if((Integer)session.getAttribute("userId") == 1)
15        {
16            String hash = getHash(user.getLogin() +
17 user.getPassword());
18
19            Class.forName("com.mysql.jdbc.Driver");
20            Connection conn = DriverManager.getConnection
21                    ("jdbc:mysql://127.0.0.1/transport",
22                            "login", "******");
23            PreparedStatement stmt = conn.prepareStatement
24                    ("SELECT id FROM user WHERE login=?");
25            stmt.setString(1, user.getLogin());
26
27            ResultSet rs = stmt.executeQuery();
28
29            if (rs.next()) {
30                PreparedStatement changePass =
31 conn.prepareStatement
32                        ("UPDATE user SET hash = ? WHERE id =
```

```
33 ?");
34                 changePass.setString(1, hash);
35                 changePass.setInt(2, rs.getInt("id"));
36
37                 try {
38                     changePass.executeUpdate();
39                 }
40                 catch(SQLException se) {
41                     return Response.status(200).entity(new
42 AnswerBoolean
43                             (false, se.getMessage())).build();
44                 }
45
46                 return Response.status(200).entity(new
47 AnswerBoolean
48                         (true)).build();
49             }
50             else
51                 return Response.status(200).entity(new
52 AnswerBoolean
53                         (false, "User not found.")).build();
54         }
           else
               return Response.status(200).entity(new
   AnswerBoolean(false,
                       "You must be logged in as
   administrator.")).build();
       }
       else
           return Response.status(200).entity(new
   AnswerBoolean(false,
                   "You must be logged in as
   administrator.")).build();
   }
```

ПРИЛОЖЕНИЕ Г

Обработка запроса конца поездки

```
1  @GET
2 @Path("/endTrip/{tripId}")
3 @Produces(MediaType.APPLICATION_JSON)
4 public Response finishTrip(@PathParam("tripId") int tripId,
5                            @Context HttpServletRequest request)
6         throws SQLException, ClassNotFoundException {
7
8     HttpSession session = request.getSession(false);
9
10    if (session == null)
11        return Response.status(200).entity(new
12 AnswerBoolean(false,
13                "Please, sign in.")).build();
14
15    int userId = (Integer) session.getAttribute("userId");
16
17    // Если залогинены
18    if (userId != 0) {
19        Class.forName("com.mysql.jdbc.Driver");
20        Connection conn = DriverManager.getConnection
21                ("jdbc:mysql://127.0.0.1/transport",
22                        "login", "******");
23
24        PreparedStatement checkKalman = conn.prepareStatement(
25                "SELECT kalman FROM trip WHERE tripId = ?");
26        checkKalman.setInt(1, tripId);
27
28        ResultSet rs1 = checkKalman.executeQuery();
29
30        if (rs1.next()) {
31            if (rs1.getBoolean("kalman"))
32                return Response.status(200).entity(new
```

```
33 AnswerBoolean
34                         (false, "Trip already ended."))
35                         .build();
36         }
37
38         PreparedStatement getTrack =
39 conn.prepareStatement("SELECT " +
40                 "lat, lng, orientation, speed, time FROM track
41 WHERE " +
42                 "tripId = ? ORDER BY time");
43         getTrack.setInt(1, tripId);
44
45         List<SmallTrack> tracks = new ArrayList<SmallTrack>();
46
47         ResultSet rs = getTrack.executeQuery();
48
49         while (rs.next()) {
50             SmallTrack track = new SmallTrack(rs.getDouble
51                     ("lat"), rs.getDouble("lng"),
52                     rs.getFloat("orientation"),
53                     rs.getInt("speed"), rs.getTimestamp
54                     ("time"));
55             tracks.add(track);
56         }
57
58         Kalman(tracks);
59
60         for (int i=0; i<tracks.size(); i++) {
61             PreparedStatement updTrack =
62 conn.prepareStatement("UPDATE " +
63                     "track SET lat = ?, lng = ? WHERE tripId = ?
64 AND time = ?");
65             updTrack.setDouble(1, tracks.get(i).getLat());
66             updTrack.setDouble(2, tracks.get(i).getLng());
67             updTrack.setInt(3, tripId);
```

```
68              updTrack.setTimestamp(4, tracks.get(i).getTime());
69
70              try {
71                  updTrack.executeUpdate();
72              }
73              catch(SQLException se) {
74                  return Response.status(200).entity(new
75 AnswerBoolean
76                          (false, se.getMessage())).build();
77              }
78          }
79
80          PreparedStatement updTripKalman = conn.prepareStatement(
81                  "UPDATE trip SET kalman = 1 WHERE tripId = ?");
82          updTripKalman.setInt(1, tripId);
83          try {
84              updTripKalman.executeUpdate();
85          }
86          catch(SQLException se) {
87              return Response.status(200).entity(new AnswerBoolean
88                      (false, se.getMessage())).build();
89          }
90
91          return Response.status(200).entity(new
92 AnswerBoolean(true))
93                  .build();
      }
      else // Не залогинены
      {
          return Response.status(200).entity(new
  AnswerBoolean(false,
                  "Please, sign in.")).build();
      }
  }
```

ПРИЛОЖЕНИЕ Д

Обработка координат поездки фильтром Калмана

```
1  void Kalman(List<SmallTrack> tracks)
 2 {
 3     double sigmaPsi = 0.0000001;
 4     double sigmaEta = 0.000001;
 5
 6     List<SmallTrack> trackExpected = new
 7 ArrayList<SmallTrack>();
 8
 9     trackExpected.add(tracks.get(0));
10
11     for(int i=1; i<tracks.size(); i++) {
12         float orientation = tracks.get(i-1).getOrientation();
13         int speed = tracks.get(i-1).getSpeed();
14         double timeDiff = (tracks.get(i).getTime().getTime() -
15 tracks.get
16                 (i-1).getTime().getTime())/1000.0f;
17
18         double latExp = tracks.get(i-1).getLat() +
19 (Math.cos(orientation)
20                 * speed / 3600 * timeDiff) / 111.1f;
21         double lngExp = tracks.get(i-1).getLng() + (-
22 Math.sin(orientation)
23                 * speed / 3600 * timeDiff) / (111.f *
24 Math.cos(latExp));
25
26         trackExpected.add(new SmallTrack(latExp, lngExp,
27 tracks.get(i)
28                 .getOrientation(), tracks.get(i).getSpeed(),
29 tracks.get(i)
30                 .getTime()));
31     }
32
```

```
33      double eOptLat = sigmaEta;
34      double eOptLng = sigmaEta;
35
36      for(int i=1; i<tracks.size(); i++) {
37          float orientation = tracks.get(i-1).getOrientation();
38          int speed = tracks.get(i-1).getSpeed();
39          double timeDiff = (tracks.get(i).getTime().getTime() -
40  tracks.get
41                  (i-1).getTime().getTime())/1000.0f;
42
43          eOptLat = Math.sqrt(sigmaEta*sigmaEta * (eOptLat*eOptLat
44  +
45                  sigmaPsi*sigmaPsi) / (sigmaEta*sigmaEta +
46  eOptLat*eOptLat
47                  + sigmaPsi*sigmaPsi));
48          eOptLng = Math.sqrt(sigmaEta*sigmaEta * (eOptLng*eOptLng
49  +
50                  sigmaPsi*sigmaPsi) / (sigmaEta*sigmaEta +
51  eOptLng*eOptLng
52                  + sigmaPsi*sigmaPsi));
53
54          double KLat = eOptLat*eOptLat / (sigmaEta*sigmaEta);
55          double KLng = eOptLng*eOptLng / (sigmaEta*sigmaEta);
56
57          double latOpt = (tracks.get(i-1).getLat() +
58  (Math.cos(orientation)
59                  * speed / 3600 * timeDiff) / 111.1f)*(1 - KLat)
60  +
61                  KLat*trackExpected.get(i).getLat();
62          double lngOpt = (tracks.get(i-1).getLng() + (-
63  Math.sin(orientation)
64                  * speed / 3600 * timeDiff) / (111.f *
65  Math.cos(latOpt)))*(1
66                  - KLng) + KLng*trackExpected.get(i).getLng();
67
```

```
68          tracks.get(i).setLat(latOpt);
69          tracks.get(i).setLng(lngOpt);
70      }
71 }
```

ПРИЛОЖЕНИЕ Е

Программный код обработки POST-запроса получения данных о поездке

```
@POST
@Path("/bytrackid")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public Response setByTrackId(Track track,
                             @Context HttpServletRequest
request)
  throws SQLException, ClassNotFoundException,
NoSuchAlgorithmException {
    HttpSession session = request.getSession(false);

    if (session == null)
        return Response.status(200).entity(new
AnswerBoolean(false,
                "Please, sign in.")).build();

    int userId = (Integer) session.getAttribute("userId");

    // Если залогинены
    if (userId != 0) {
        Class.forName("com.mysql.jdbc.Driver");
        Connection conn = DriverManager.getConnection
                ("jdbc:mysql://127.0.0.1/transport",
                        "login", "******");

        PreparedStatement setTracks = conn.prepareStatement(
                "INSERT INTO track VALUES(?, ?, ?, ?, ?, ?,
?)");
        setTracks.setInt(1, track.getTripId());
        setTracks.setDouble(2, track.getLat());
        setTracks.setDouble(3, track.getLng());
```

```
32          setTracks.setDouble(3, track.getOrientation());
33          setTracks.setFloat(4, (float)track.getFuel());
34          setTracks.setInt(5, track.getSpeed());
35          setTracks.setTimestamp(6, track.getTime());
36
37          try {
38              setTracks.executeUpdate();
39          }
40          catch(SQLException se) {
41              return Response.status(200).entity(new
42 AnswerBoolean(false,
43                      se.getMessage())).build();
44          }
45
46          return Response.status(200).entity(new
47 AnswerBoolean(true))
48                  .build();
49      }
50      else // Не залогинены
51      {
52          return Response.status(200).entity(new
53 AnswerBoolean(false,
54                  "Please, sign in.")).build();
55      }
56 }
```

ПРИЛОЖЕНИЕ Ж

Программный код класса SmallTrack

```
1 public class SmallTrack {
2     private double lat;
3     private double lng;
4     private float orientation;
5     private int speed;
6     private Timestamp time;
7
8     public double getLat() {
9         return lat;
10    }
11
12    public double getLng() {
13        return lng;
14    }
15
16    public int getSpeed() {
17        return speed;
18    }
19
20    public float getOrientation() {
21        return orientation;
22    }
23
24    public Timestamp getTime() {
25        return time;
26    }
27
28    public void setLat(double lat) {
29        this.lat = lat;
30    }
31
32    public void setLng(double lng) {
```

```
33          this.lng = lng;
34      }
35
36      public SmallTrack(double lat, double lng, float orientation,
37                        int speed, Timestamp time) {
38          this.lat = lat;
39          this.lng = lng;
40          this.orientation = orientation;
41          this.speed = speed;
42          this.time = time;
43      }
44  }
```