Разработка имитационно-моделирующего комплекса обеспечения безопасности работы персонала в промышленной среде

Специальность: разработчик-программист, Geekbrains

Опанасенко Андрей Владимирович

2023 г.

**Содержание**

Содержание

Введение

Глава 1. Обоснование необходимости

Глава 2. Выбор программного стека

Глава 3. Реализация основных функциональных требований

Глава 4. Описание интерфейса web-приложения

Заключение

Список используемой литературы

Приложения

**Введение**

**Тема проекта:** разработка веб-приложения, представляющего собой имитационно-моделирующих комплекс взаимодействия тяжелой техники и пешего персонала в промышленной среде, обеспечивающий подготовку синтетических данных, их визуализацию и построение аналитических отчётов.

**Цель:** демонстрация возможностей реального программного обеспечения при его внедрении в промышленную среду.

**Решаемые проблемы:**

**-** предотвращение наезда тяжелой техники на пеший персонал путем оповещения водителя о нахождении пешехода в опасной зоне;

- предотвращение столкновения техники;

- сбор статистики, выявления условий возникновения потенциально опасных ситуаций (распределение с нанесением на карту концентрации опасных случаев, выявление частоты возникновения опасных ситуаций в разрезе пар id техники - id пешеход).

**Задачи:**

- изучение нормативно-правовой базы в части требований к системам обеспечения безопасности в промышленной среде при взаимодействии тяжелая техника – человек, техника – техника;

- определение состава синтетических данных и написание программного кода для их генерации;

- выбор системы управления базой данных для хранения полученных данных;

- выбор фреймворка для разработки веб-приложения;

- подготовка аналитических отчетов;

- разработка плана дальнейшего развития приложения.

**Программный стек:** Python, Dash Plotly, Flask, PosgreSQL, SQLAlchemy, Numpy, Pandas, Git, Linux.

**Состав команды:** Опанасенко Андрей Владимирович (разработчик)

**Глава 1. Обоснование необходимости**

В настоящее время производители отечественных систем предотвращения опасных ситуаций в промышленной среде предполагающей совместную работу тяжелой техники и пешего персонала (шахты, карьеры, логистические хабы, строительные площадки) делают акцент на аппаратной составляющей таких систем и жестко детерминированных моделях реагирования техники в случае возникновения потенциально опасной ситуации, руководствуясь, например в горнодобывающей отрасли приказом Ростехнадзора № 505 от 8.12.2020 года, п. 362:

транспортные машины, эксплуатируемые на шахтах по добыче полезных ископаемых, должны быть оборудованы системами предотвращения столкновений. Система предотвращения столкновений должна обеспечивать своевременное оповещение машиниста о наличии людей и транспортных средств в радиусе траектории движения машины.

Обеспечение выполнения указанных требований как правило достигается установкой на технике центрального блока и одной или нескольких всенаправленных антенн, излучающих в диапазонах 120 кГц и 866 Мгц. Каждый сотрудник обязан при себе иметь RFID-метку. При движении техники в промышленной зоне установленная антенна производит непрерывное сканирование окружающего пространства, при возникновении в опасной близости пешего персонала активируется его RFID-метка (типовой состав оборудования приведен на рисунке 1, 2) и в зависимости от расстояния, скорости и прочих параметров с центральным блоком реализуется одна из нескольких жестко запрограммированных моделей:

* светозвуковое оповещение водителя с индикацией направления на пешехода
* снижение скорости транспортного средства
* блокировка возможности движения, остановка транспортного средства.





Рисунок 1 – Основные аппаратные компоненты системы предотвращения наезда и столкновений: антенна, RFID-метка, центральный блок с возможностью светозвукового оповещения



Рисунок 2 – типовой состав оборудования систем предотвращения наезда

Описание основных компонентов системы:

* антенна формирует вокруг спецтехники зону обнаружения датчиков пешеходов радиусом от 3 до 8 метров. Монтаж антенны производится на переднюю часть транспортного средства. Радиус зоны действия зависит от места монтажа антенны, посторонних предметов. Радиус измеряется от центра антенны.
* транспондер водителя представляет собой карточку желтого цвета, размер которой сопоставим с обычной банковской картой, пластиковой картой доступа.

Карточки водителя выдаются всем водителям. Они вставляются в Блок управления, размещенный в кабине транспортного средства, непосредственно перед началом работы на машине. Если транспондер водителя не вставлен в картодержатель перед запуском машины - блок управления издает постоянный звуко-световой сигнал, напоминающий водителю о необходимости выполнить данное действие.

При нахождении вне транспортного средства, транспондер водителя обладает функциями пешеходного.

* транспондер пешехода представляет собой серую карту-передатчик. Данные карты выдаются всем работникам-пешеходам, чья деятельность осуществляется в непосредственной близости от спецтехники. Так же они выдаются посетителям площадок. Карточки пешеходов могут быть вставлены в рабочие жилеты.

При пересечении транспондера пешехода зоны детекции Антенны на спецтехнике, Блок управления издает предупреждающий звуко-световой сигнал водителю транспортного средства. Другими словами, при приближении рабочего менее чем на 3-8 метров к работающей машине, например, погрузчику - в салоне погрузчика раздается звуковой сигнал и появляется световая сигнализация, которая сообщает водителю, что рядом находится человек! Опасно! Каждый такой транспондер имеет свой уникальный номер, который сохраняется в памяти Блока управления при каждом приближении к нему.

* блок управления обеспечивает звуко-световое предупреждение водителю транспортного средства (погрузчика, трактора и пр.) о приближении к пешеходу. Монтируется непосредственно в водительскую кабину.

Водители спецтехники используют специальные карточки водителя, которые помещаются в картодержатели Блока управления. Данный Блок имеет часы реального времени и встроенную память для хранения журнала до 7 000 событий обнаружения пешеходов в радиусе действия системы.

Одновременно с этим, центральные блоки имеют возможность фиксации:

* координат инцидента
* мощности излучения антенного блока
* состояния релейных выходов
* времени
* уникального номера метки пешехода
* уникального номера метки водителя
* расстояния до пешехода
* прочее

Аналогичные системы импортного производства (например оборудование ушедшей из РФ компании ZoneSafe) зачастую используют собираемую информацию для:

* выявления наиболее опасных мест, организации в этих местах “безопасных зон”
* выявления нарушающих правила движения водителей
* организации контроля и учета доступа персонала на объект
* сбора статистических данных (например, о состоянии и длительности заряда батарей RFID-меток).
* формирования аналитических отчетов

Цитата из описания программного обеспечения ZoneSafe Insight:

ZoneSafe Insight — это облачная система, которая использует технологию идентификации и регистрацию данных для записи важной аналитической информации и повышения осведомленности о рисках столкновений, чтобы обеспечить безопасность работников и имущества на объекте.

Ведение журнала того, где и когда метки попадают в зоны обнаружения, поможет предприятиям лучше понять, как промышленное оборудование и пешеходы взаимодействуют на площадке. Затем эту информацию можно использовать для разработки более надежного плана обеспечения безопасности, улучшения управления дорожным движением и повышения безопасности на объекте.

Некоторые из функций и преимуществ ZoneSafe Insight включают в себя:

- GPS-отслеживание местоположения и обнаружение тегов;

- идентификацию и сравнение событий

- контроль состояние батареи всех тегов

- назначение тегов пешеходам и водителям

- создание настраиваемых отчетов о собранных данных

- управление показателями безопасности

- регистрацию попытки несанкционированного доступа

- ограничение доступа персонала

Кроме того, дооснащение транспортных средств дополнительными аппаратными компонентами позволит детектировать и анализировать данные о событиях ведущих к незначительным повреждениям технических средств, контролю технического состояния, расхода топлива и пр.

**Глава 2. Выбор программного стека**

**Выбор фреймворка для web-приложения**

Ввиду того, что основным назначением разрабатываемого приложения является сбор и представление собранной информации в виде аналитических отчетов и с учетом широкого выбора фреймворков для построения таковых отчетов с использованием языка программирования (далее – ЯП) Python, данный ЯП был выбран в качестве основного. Среди наиболее популярных фреймворков для построения аналитических отчетов с возможностью их публикации на web-сервере являются:

- платные решения (Power BI, Tableau)

* Apache Superset
* Redash
* Dash Plotly
* Streamlit

**Tableau**

Tableau — программное обеспечение для интерактивной бизнес-аналитики и визуализации данных. Оно помогает провести глубокий анализ большого количества информации и представить результаты в удобной и интуитивно понятной форме. Tableau может взаимодействовать с облачными решениями (Dropbox, Google Таблицы, AWS Redshift и пр.), а также современными инструментами анализа данных, например Python и R.

Аналитическую платформу разработала компания Tableau Software, Inc. в 2003 году. В 2019 году ее поглотила корпорация Salesforce, которая продолжила развитие продукта. Сегодня под брендом Tableau выпускается несколько версий программы, а также дополнительные продукты, расширяющие функционал. Вокруг Tableau образовалось обширное сообщество профессионалов и энтузиастов-любителей, обменивающихся опытом и способствующих совершенствованию сервиса.

Возможности Tableau:

* сбор информации из различных источников — от простых Excel-таблиц до многомерных массивов данных, облачных хранилищ и т.д.
* глубокая обработка полученных данных, позволяющая отслеживать события, тенденции, понимать их причины.
* визуализация результатов анализа в виде стандартных графиков, пузырьковых диаграмм, дерева событий, с привязкой к географическим картам и т.д.
* возможность объединить различные типы данных в сводных отчетах, например информацию о расходах на рекламу с данными о регистрациях, чтобы рассчитать их стоимость.
* создание отчетов любых типов — от простейших таблиц до глубоких исследований трендов и корреляций.
* обмен данными с другими пользователями, загрузка отчетов в облако, совместный просмотр и работа с результатами исследований.
* анализ эффективности бизнеса с помощью ключевых метрик из любой точки мира.

Преимущества Tableau:

* обработка файлов любых форматов, от электронных таблиц до многомерных массивов данных, реляционных БД и т.д.
* интуитивно понятный интерфейс и функция обработки естественных запросов, которые понижают порог вхождения и позволяют быстро освоить программу.
* высокая скорость обработки данных для анализа больших массивов информации.
* многоуровневая авторизация и защита пользовательских данных.
* быстрая установка, настройка и внедрение программного обеспечения, простое подключение дополнительных модулей, расширяющих основной функционал.
* отраслевые решения, позволяющие масштабировать платформу с учетом специфики компании.
* доступ к данным и отчетам из любой точки мира.
* Подробная техническая документация по каждому продукту из экосистемы Tableau, широкая поддержка официальных разработчиков и энтузиастов-любителей.

**Apache Superset**

Apache Superset - быстрый и гибкий BI инструмент в бизнес-аналитике, который разработан для высокой доступности и скалирования на крупных распределенных средах, он также отлично работает внутри контейнеров. Это мощная платформа для визуализации данных и аналитики бизнеса, которая позволяет пользователям создавать интерактивные дашборды и отчеты на основе различных источников данных. С помощью Apache Superset вы можете быстро и легко анализировать данные, отслеживать ключевые метрики и принимать важные решения на основе фактов. Эта платформа предлагает широкий набор инструментов для работы с данными, включая графики, таблицы, карты, фильтры и многое другое.

Superset предоставляет:

* Интуитивный интерфейс для визуализации наборов данных и создания интерактивных инфопанелей;
* Широкий набор визуализаций с возможностью подключения и создания собственных, используя JS;
* Простой и понятный инструмент для создания наборов данных (используя SQL);
* Встроенный редактор SQL (SQL Alchemy) для работы с данными;
* Удобный интерфейс для создания метрик и вычисляемых столбцов;
* Поддержка для большинства SQL баз данных (всего около 30 коннекторов, список постоянно пополняется);
* Встроенное асинхронное выполнение запросов и кэширование;
* Расширенная модель безопасности, предоставляет богатую кастомизацию для доступа к инфопанелям, наборам данных и визуальным элементам;
* Возможность интегрировать со многими инструментами аутентификации (БД, OpenID, LDAP, OAuth, REMOTE\_USER, и т.д.);
* API для программной кастомизации;
* Скалируемость, которая поддерживается «облачной» архитектурой.

Superset также позволяет выбрать инструменты для:

* web server (Gunicorn, Nginx, Apache);
* metadata database engine (MySQL, [Postgres](https://dbi.ru/uslugi-podderzhki-subd-postgresql/), MariaDB, etc);
* message queue (Redis, RabbitMQ, SQS, etc);
* results backend (S3, Redis, Memcached, etc);
* caching layer (Memcached, Redis, etc).

**Redash**

Redash - бизнес-аналитический сервис для работы с большим объёмом данных и его визуализации. Redash помогает работать с базами данных, подключив их к сервису пользователи смогут при помощи SQL-запросов обрабатывать имеющиеся данные и преобразовывать их в графики, и диаграммы. В Redash пользователь может работать как один, так и в команде с коллегами.

Для начала работы в Redash пользователю необходимо пройти цепочку из нескольких этапов. Сначала необходимо подключиться к источнику данных, это могут быть как локальные базы данных, так и базы, находящиеся на выделенных серверах. После подключения к источнику можно начинать работать с данными при помощи запросов. Чтобы правильно сформировать запрос можно воспользоваться детализированной инструкцией. Запустив запрос, данные сформируются в таблицу. Полученную таблицу можно превратить в визуальный элемент: графики, диаграммы, карты, облако слов, поворотные таблицы, диаграммы Sankey и другие.

Готовые графические данные размещаются на дашборде. Дашборд представляет из себя панель из нескольких графиков и текстовых данных, показывающих аналитическую информацию из запрашиваемых баз. Дашбордами можно делиться как с коллегами, так и открыть их для публичного доступа.

Особенности Redash:

* Создание запросов для работы с источниками данных
* Исследование схем и автозаполнение
* Создание сниппетов для часто используемых элементов
* Управление доступом сотрудников
* Визуализация полученных и запросов данных
* Подключение к нескольким источникам одновременно
* Автообновление данных на дашборде
* Фильтры и настройка данных
* Поддержка большинства источников данных

**Streamlit**

Streamlit — это [**фреймворк**](https://blog.skillfactory.ru/glossary/framework/) для языка программирования [**Python**](https://blog.skillfactory.ru/glossary/python/). Он содержит набор программных инструментов, которые помогают перенести модель машинного обучения в веб. Написанную «умную» программу с помощью этого фреймворка можно быстро сделать веб-приложением.

**Основные возможности Streamlit**

* Быстрое развертывание. ML-модель или обычную программу можно быстро превратить в одностраничное веб-приложение и управлять им. Не нужно долго верстать и загружать модель, пользоваться традиционными инструментами для создания веб-интерфейсов.
* Использование скриптов. Приложения можно делать интерактивными. Каждый раз, когда пользователь взаимодействует с получившимся веб-интерфейсом или разработчик меняет что-то в коде, Streamlit сам обновляет и перерисовывает нужные части страницы. Так интерфейс «откликается» на действия пользователя или на изменения модели в реальном времени. Поэтому с помощью фреймворка можно делать интерактивные визуализации, дашборды или простые пользовательские сервисы.
* Виджеты и визуализация. В Streamlit есть встроенные стандартные виджеты для частых действий, например ползунки или поля для ввода текста. Можно взять готовые виджеты и собрать из них работающий интерфейс. Еще можно отрисовать график или картинку, вывести результат работы программы в виде схемы или таблицы. Есть и функция для отрисовки карты, на которой можно указать с помощью кода какие-то координаты, маршруты и линии.

**Dash Plotly**

Dash Ploly ― это платформа с открытым исходным кодом для создания интерфейсов визуализации данных. Это достаточно нетривиальное решение для создания дашбордов и аналитических витрин. В отличие от остальных продуктов, данное решение предлагает самый высокий уровень гибкости для представления данных, так как весь дашборд от начала и до конца пишется с помощью Python кода как полноценное веб-приложение. Библиотека создана и поддерживается канадской компанией [Plotly](https://plotly.com/) и является логичным развитием одноименной популярной среди аналитиков данных и дата-сайентистов Python-библиотеки для построения разнообразных и красивых визуализаций.

В основе библиотеки Dash помимо Plotly, который отвечает за отрисовку графики, также используется знакомый многим Python-разработчикам фреймворк Flask, а также фреймворк React. Flask предоставляет библиотеке функциональность веб-сервера для простого и быстрого создания наших дэшбордов как небольших сайтов, которым не нужен сложный бэкенд. React же отвечает за отображение веб-интерфейса нашего приложения.

В данной работе сделан выбор в пользу фреймворка Dash Plotly ввиду следующих его характеристик:

* распространяется по лицензии MIT;
* предоставляет функциональность веб-сервера благодаря интеграции c Flask;
* благодаря интеграции с Flask поддерживает возможность работы с объектами базы данных в объектно-ориентированной парадигме;
* отображение веб-интерфейса благодаря интеграции с React;
* включает в себя один из наиболее известных пакетов инструментов по визуализации данных Plotly;
* возможность создавать многостраничные приложения;
* широкий набор встроенных элементов управления (слайдеры, выпадающие списки, чекбоксы, кнопки);
* интеграция с bootstrap;
* нетребовательность к аппаратным ресурсам

**Выбор системы управления базой данных**

Наиболее часто используемые в настоящее время СУБД подразделяются на:

* реляционные
* ключ-значение
* документные
* графовые
* колоночные

**Реляционные СУБД**

Реляционные СУБД чаще всего используются для построения решений OLTP (Online Transaction Processing). В таких решениях СУБД работает с небольшими по размерам транзакциями, но идущими большим потоком, и при этом от системы требуется минимальное время отклика, а так же возможность, при определенных условиях, отменить любые изменения выполняемых в рамках транзакции. Если вы строите систему, в рамках которой требуется хранить значительное количество сущностей (таблиц), с различными типами связей между ними (один-к-одному, один-к-многим, многие-ко-многим), то это скорее всего про реляционные СУБД.

Один из основных признаков, который говорит о том что нужно выбирать реляционную СУБД – это высокая нормализация данных. Дополнительными признаками будет необходимость обработки большого кол-ва коротких транзакций, с большей долей операций на вставку.

**СУБД типа ключ-значение**

База данных на основе пар «ключ‑значение» – это тип нереляционных баз данных, в котором для хранения данных используется простой метод «ключ‑значение». База данных на основе пар «ключ‑значение» хранит данные как совокупность пар «ключ‑значение», в которых ключ служит уникальным идентификатором. Как ключи, так и значения могут представлять собой что угодно: от простых до сложных составных объектов. Базы данных с использованием пар «ключ‑значение» поддерживают высокую разделяемость и обеспечивают беспрецедентное горизонтальное масштабирование, недостижимое при использовании других типов баз данных. Например, [Amazon DynamoDB](https://aws.amazon.com/dynamodb/) выделяет дополнительные разделы на таблицу, если существующий раздел заполняется до предела и требуется больше пространства для хранения.

Чаще всего такие СУБД используют для кэширования, т.к. они очень быстро работают, а это и не сложно, когда есть уникальный ключ, и запрос возвращает только одно значение. У некоторых представителей данных СУБД есть возможность работать полностью в памяти, а так же есть возможность задавать срок жизни записи, после истечения которого, записи будут автоматически удаляться.

**Документные или документно-ориентированные СУБД**

Документные СУБД - одна из наиболее популярных разновидностей NoSQL СУБД, где основной единицей логической модели данных является документ - структурированный текст, с определенным синтаксисом.

Документная СУБД - специально предназначенная для хранения иерархических структур данных (документов) и обычно реализуемая с помощью подхода NoSQL. В основе документоориентированных СУБД лежат документные хранилища, имеющие структуру дерева. Структура дерева начинается с корневого узла и может содержать несколько внутренних и листовых узлов. Листовые узлы содержат данные, которые при добавлении документа заносятся в индексы, что позволяет даже при достаточно сложной структуре находить место (путь) искомых данных. API для поиска позволяет находить по запросу документы и части документов. В отличие от хранилищ типа ключ-значение, выборка по запросу к документному хранилищу может содержать части большого количества документов без полной загрузки этих документов в оперативную память.

Документы могут быть организованы (сгруппированы) в *коллекции*. Их можно считать отдалённым аналогом таблиц реляционных СУБД, но коллекции могут содержать другие коллекции. Хотя документы коллекции могут быть произвольными, для более эффективного индексирования лучше объединять в коллекцию документы с похожей структурой.

**Графовые СУБД**

Графовую модель данных обычно рассматривают как обобщение RDF-модели или сетевой модели данных. Основными элементами модели являются узлы и связи. В зависимости от реализации узлов и рёбер графовую модель данных разделяют на несколько подтипов.

В графовых СУБД, как правило, разделяют подсистему хранения и механизм обработки.

Для аналитической работы с большими объёмами данных в глобальных графах применяются специализированные механизмы графовых вычислений. В отличие от графовых СУБД, ориентированных в основном на OLTP-приложения, в системах графовых вычислений используются подходы и методы оптимизации, свойственные OLAP. Существуют различные реализации механизмов для графовых вычислений, как резидентные (i*n-memory*), так и использующие энергонезависимые устройства хранения, как работающие на одном узле, так и распределённые (работающие на нескольких узлах одновременно).

Графовые базы данных применяются для моделирования социальных графов (социальных сетей)[[3]](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85#cite_note-3), в биоинформатике, а также для семантической паутины. Для задач с естественной графовой структурой данных графовые СУБД могут существенно превосходить реляционные по производительности, а также иметь преимущества в наглядности представления и простоте внесения изменений в схему базы данных.

**Колоночные СУБД**

Колоночные СУБД очень похожи на реляционные. Они так же состоят из строк, которые имеют атрибуты, а строки группируются в таблицах. Различия в логических моделях несущественные, а вот на уровне физического хранения данных различия значительные.

В реляционных СУБД данные хранятся "построчно", это означает что для считывания значения определенной колонки, придется прочитать практически всю строку, как минимум от первой до нужной колонки. В колоночной СУБД данные хранятся "поколоночно", т.е. колонка - это как отдельная таблица. Соответственно чтение будет происходить из конкретного столбца сразу. На практике это реально работает очень быстро (проверено мной на нескольких реализованных хранилищах данных).

Основные преимущества колоночных СУБД – эффективное выполнения сложных аналитических запросов на больших объемах, и легкое, практически мгновенное, изменение структуры таблиц с данными, плюс существенная компрессия и сжатие, которое позволяет значительно экономить место.

В таблице 1 приведено краткое сравнение описанных выше СУБД.

**Таблица 1 – краткое сравнение типов СУБД**

|  |  |  |
| --- | --- | --- |
| **Тип СУБД** | **Когда выбирать** | **Примеры популярных СУБД** |
| Реляционные | Нужна транзакционность; высокая нормализация; большая доля операций на вставку | Oracle, MySQL, Microsoft SQL Server, PostgreSQL |
| Ключ-значение | Задачи кэширования и брокеры сообщений | Redis, Memcached |
| Документные | Для хранения объектов в одной сущности, но с разной структурой; хранение структур на основе JSON | CouchDB, MongoDB, Amazon DocumentDB |
| Графовые | Задачи подобные социальным сетям; системы оценок и рекомендаций | Neo4j, Amazon Neptune, InfiniteGraph, InfoGrid |
| Колоночные | Хранилища данных; выборки со сложными аналитическими вычислениями; количество строк в таблице превышает сотни миллионов | Vertica, ClickHouse, Google BigTable, Sybase \ SAP IQ, InfoBright, Cassandra |

Среди приведенных выше типов СУБД наиболее подходящей для хранения информации, используемой в разрабатываемом приложении представляется реляционная СУБД. Среди наиболее популярных РСУБД: SQLite, MySQL и PostgreSQL. В разрабатываемом приложении выбор остановлен на последней в силу следующих её преимуществ:

* Полная SQL-совместимость.
* Сообщество: PostgreSQL поддерживается опытным сообществом 24/7.
* Поддержка сторонними организациями: несмотря на очень продвинутые функции, PostgreSQL используется в многих инструментах, связанных с РСУБД.
* Расширяемость: PostgreSQL можно программно расширить за счёт хранимых процедур.
* Объектно-ориентированность: PostgreSQL — не только реляционная, но и объектно-ориентированная СУБД.

Взаимодействие с СУБД осуществляется с использованием объектно-ориентированного подхода посредством ORM (Object-Relational Mapping) SQLAlchemy. Благодаря использованию ORM:

* взаимодействие с СУБД происходит в ООП-стиле;
* код приложения будет оставаться тем же вне зависимости от используемой базы данных. Это позволяет с легкостью мигрировать с одной базы данных на другую, не переписывая код.

**Инструменты анализа данных и построения отчетов**

Для анализа и визуализации данных в разработанном приложении использованы библиотеки Python: pandas и plotly.

Pandas — программная библиотека на языке Python для обработки и анализа данных. Работа pandas с данными строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня. Предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временны́ми рядами.

Главный элемент pandas - DataFrame, с которым можно производить необходимые преобразования. представляющий собой таблицу, состоящую из строк и столбцов. По умолчанию, строчки таблицы - это объекты, а столбцы - признаки (фичи) объектов.

Pandas предоставляет широкие возможности для анализа данных. С ее помощью очень удобно загружать, обрабатывать и анализировать табличные данные с помощью SQL-подобных запросов. В связке с графическими библиотеками Matplotlib, Seaborn, Plotly появляется возможность удобного визуального анализа табличных данных.

Plotly - это библиотека с открытым исходным кодом для Python и R, которая отлично подходит для создания красивых и интерактивных визуализаций. Это хороший инструмент для выявления закономерностей в наборах данных и предоставления информации пользователям.

Plotly имеет несколько особенностей, которые выделяют ее на фоне других инструментов визуализации:

* интерактивность позволяет увеличивать и уменьшать масштаб  частей карт и графиков, чтобы проанализировать график более детально.
* простота создания сложных визуализаций с помощью высокоуровневого API  Plotly Express);
* простой синтаксис – каждый график использует одни и те же параметры;
* много возможностей персонализации и дизайна

Самое главное – это интерактивность, которая позволяет пользователям взаимодействовать с графиками. Изменение масштаба графиков и карт и отображение значений точек при наведении курсора позволяют обнаружить аномалии в огромных выборках.

**Глава 3.** **Реализация основных функциональных требований**

Приложение реализовано в соответствии с принципами разделения логики хранения и извлечения данных, бизнес-логики и представления информации. В разработке применён объектно-ориентированный подход. В веб-приложении реализован функционал:

* имитационного моделирования движения в виртуальном пространстве технических средств, пешего персонала, событий, возникающих при их опасном сближении: остановка транспортного средства и запись информации о событии в БД.
* генерация синтетических данных для наполнения базы данных
* вывод аналитических отчётов на основе хранящейся в БД информации о произошедших событиях:
  + статистика возникновения опасных ситуаций по всем техническим средствам
  + статистика возникновения опасных ситуаций по каждому отдельному техническому средству
  + распределение потенциально опасным мест виртуального пространства с отображением на карте
  + контроль состояния заряда аккумулятора RFID меток пешего персонала

**Модель данных**

Взаимодействие с БД осуществляется с помощью ORM SQLAlchemy.

Модель данных реализована посредством класса Events:

URL = URL.create(

drivername="postgresql",

username=config\_sql\_alchemy.db.db\_user,

host=config\_sql\_alchemy.db.db\_host,

port=config\_sql\_alchemy.db.db\_port,

database=config\_sql\_alchemy.db.database,

password=config\_sql\_alchemy.db.db\_pass  
)  
  
engine = orm.create\_engine(URL)  
Base = declarative\_base()  
  
  
class Events(Base):  
 \_\_tablename\_\_ = 'Events'  
  
 id = Column(Integer(), primary\_key=True)

component\_id = Column(Integer())

component\_type = Column(Text())

battery\_charge\_status = Column(Numeric())

latitude = Column(Numeric())

longitude = Column(Numeric())

is\_danger = Column(Integer())

nearest\_danger\_object\_id = Column(Integer(), default=10E6)

nearest\_danger\_object\_meters = Column(Numeric(), default=10E6)

event\_dt = Column(DateTime())  
  
  
# try:  
# Events.\_\_table\_\_.drop(engine)  
# except:  
# pass

Base.metadata.create\_all(engine)  
Session = sessionmaker(bind=engine)  
session = Session()

**Имитационная модель**

С целью имитационного моделирования взаимодействия технических средств и пешего персонала определены классы:

- Map (карта)

- Block (центральный блок, установленный на техническом средстве)

- Tag (RFID метка персонала)

Реализация классов осуществлена с использованием принципа наследования ООП-парадигмы. Использование Data classes при определении указанных классов позволяет избежать избыточности кода, автоматическую генерацию стандартных методов класса. Кроме того, использование Data classes требует обязательной аннотации типов.

Класс Map – базовый класс, содержащий в себе данные обо всех объектах, представленных на карте и определяющий методы, контролирующие их передвижение, обрабатывающие события при возникновении опасных ситуаций и внесении информации об этих событиях в базу данных.

@dataclass  
class Map:  
 tags: [Tag] = field(default\_factory=list)  
 blocks: [Block] = field(default\_factory=list)  
 longitude\_start: int = LONGITUDE\_START  
 longitude\_end: int = LONGITUDE\_END  
 latitude\_start: int = LATITUDE\_START  
 latitude\_end: int = LATITUDE\_END  
 time\_delta\_seconds: int = 0  
  
 def move\_block(self, block: Block, time\_step=1):  
 block.speed\_before\_stop = block.speed if block.speed else block.speed\_before\_stop

block.is\_danger\_area = 0

min\_distance\_between\_this\_block\_and\_other\_component = min([calculate\_distance(block, el) for el in self.tags + self.blocks if el != block])  
 min\_distance\_other\_id = block.id

for el in self.tags + self.blocks:  
  
 if block != el:  
 distance\_between\_this\_block\_and\_other\_el = calculate\_distance(block, el)

if distance\_between\_this\_block\_and\_other\_el ==

min\_distance\_between\_this\_block\_and\_other\_component:  
 min\_distance\_other\_id = el.id

if distance\_between\_this\_block\_and\_other\_el < block.danger\_area:

if block.speed:

block.speed\_before\_stop = block.speed

block.speed = 0  
 block.is\_danger\_area = 1  
 el.is\_danger\_area = 1  
  
 if block.is\_danger\_area and block.speed == 0:

if all([calculate\_distance(block, tag) > block.danger\_area for tag in self.tags]):

block.speed = block.speed\_before\_stop  
 block.move(time\_step=time\_step)  
  
 if not block.is\_danger\_area and block.speed == 0:  
 block.speed = block.speed\_before\_stop  
  
 block.move(time\_step=time\_step)

block.nearest\_danger\_object\_meters =

min\_distance\_between\_this\_block\_and\_other\_component  
 block.nearest\_danger\_object\_id = min\_distance\_other\_id

def move\_elements(self, block\_id=None, block\_speed=None, add\_into\_db=False):

self.time\_delta\_seconds += random.randint(20, 1000)

if block\_id and block\_speed:

for block in self.blocks:

if block.id == block\_id:  
 block.speed = block\_speed

for tag in self.tags:  
 tag.is\_danger\_area = 0  
 tag.move(time\_step=1)

for block in self.blocks:  
 self.move\_block(block=block, time\_step=1)

if add\_into\_db:  
 with server.app\_context():  
 for element in self.tags + self.blocks:  
 db.session.add(Events(component\_id=element.id,

component\_type=element.type, latitude=round(element.latitude, 2),

longitude=round(element.longitude, 2),  
battery\_charge\_status=round(element.battery\_charge\_status, 2),  
event\_dt=round\_seconds(datetime.datetime.now() +  
datetime.timedelta(seconds=self.time\_delta\_seconds)),  
 is\_danger=element.is\_danger\_area,  
nearest\_danger\_object\_id=element.nearest\_danger\_object\_id,  
nearest\_danger\_object\_meters=element.nearest\_danger\_object\_meters  
 ))  
 db.session.commit()

Классы Block и Tag определяют основные атрибуты объектов транспортное средство и пеший персонал, а также метод, определяющий их перемещение на карте.

@dataclass  
class SystemComponent:

id: int

longitude: float

latitude: float

speed: float

type: str

battery\_charge\_status: float = 100.0

steps: int = 0

longitude\_direction: int = random.choice([-1, 1])

latitude\_direction: int = random.choice([-1, 1])

nearest\_danger\_object\_id: int = 10E6

nearest\_danger\_object\_meters: float = 10E6

is\_danger\_area: int = 0  
  
 def move(self, time\_step):  
 self.battery\_charge\_status = 100.0 if (self.battery\_charge\_status - random.random()) <= 0 else (  
 self.battery\_charge\_status - random.random())

self.steps += 1

distance = self.speed \* time\_step

delta\_x = random.random() \* distance

delta\_y = (distance \*\* 2 - delta\_x \*\* 2) \*\* 0.5  
  
 if not LONGITUDE\_START < self.longitude + delta\_x \* self.longitude\_direction < LONGITUDE\_END:

self.longitude\_direction \*= -1

if not LATITUDE\_START < self.latitude + delta\_y \* self.latitude\_direction < LATITUDE\_END:

self.latitude\_direction \*= -1

self.latitude += delta\_y \* self.latitude\_direction

self.longitude += delta\_x \* self.longitude\_direction

@dataclass  
class TagData(SystemComponent):

type: str = 'tag'  
  
  
@dataclass  
class BlockData(SystemComponent):

type: str = 'block'  
 danger\_area: float = DANGER\_DISTANCE  
  
 def \_\_post\_init\_\_(self):  
 self.speed\_before\_stop = self.speed

class Tag(TagData):  
 pass  
  
  
class Block(BlockData):  
 pass

**Глава 4. Описание интерфейса web-приложения**

Разработанное программное обеспечение представляет собой веб-приложение.

Навигация осуществляется посредством бокового меню.

Представлено три раздела:

* описание программного обеспечения
* моделирование
* аналитика

В разделе “описание” приведены основные сведения о приложении, назначении, основных функциональных возможностях и предоставляемых данных.

Раздел “моделирование” (см. приложение Б) разработанного web-приложения содержит интерактивную карту, на которой имитируется перемещение транспортных средств и пешего персонала. При этом обрабатываются события возникновения потенциально опасных ситуаций – при таковом сближении транспортного средства и пешего персонала цвет соответствующих объектов изменяется, транспортное средство останавливается до тех пор, пока пешеход не переместится на достаточное расстояние. При этом реализована возможность:

* управлять скоростью перемещения транспортных средств
* изменять параметр, определяющий размер потенциально опасной зоны сближения

В разделе, посвященном аналитике, приведены примеры типовых аналитических отчетов (см. приложение Б):

* статистика возникновения опасных ситуаций по всем блокам на технике;
* статистика возникновения опасных ситуаций по отдельному блоку – с помощью выпадающего списка можно выбрать id интересующего блока и получить гистограмму, отражающую количество опасных сближений данного транспортного средства с каждым из пешеходов;
* распределение потенциально опасных мест с визуализацией на карте;
* контроль состояния заряда аккумуляторов RFID-меток с привязкой ко времени.

**Заключение**

В работе проанализировано текущее состояние дел в сфере обеспечения безопасности пешего персонала в зонах работы промышленного оборудования и связанных с этим ситуаций повышенной опасности. Приведен состав аппаратной составляющей комплексов обеспечения безопасности, обеспечивающий выполнение требований регулирующих органов. Одновременно с тем указаны предпосылки и возможности для повышения эффективности данных систем путем сбора статистических данных, анализа этих данных и использования их в системах поддержки принятия решений. Проанализирован опыт иностранных компаний в этой сфере.

В результате проведенного анализа первоочередных потребностей в части обеспечения безопасности в промышленной среде становится очевидной необходимость разработки программной части системы с целью повышения её эффективности и функциональных возможностей. Были определены основные требования предъявляемые к программному обеспечению:

* выявление наиболее опасных мест с целью организации в этих местах “безопасных зон”
* выявление нарушающих правила движения водителей
* организация контроля и учета доступа персонала на объект
* сбора статистических данных (например, о состоянии и длительности заряда батарей RFID-меток).
* формирования аналитических отчетов

В теоретической части также рассмотрен возможный стек программных компонентов для реализации web-приложения, осуществлен выбор языка программирования для разработки ПО, состав используемых фреймворков, системы управления базой данных для хранения информации о возникающих событиях.

Разработанное приложение обеспечивает:

* имитационное моделирование движения в виртуальном пространстве технических средств, пешего персонала, событий, возникающих при их опасном сближении: остановка транспортного средства и запись информации о событии в БД.
* генерацию синтетических данных для наполнения базы данных
* вывод аналитических отчётов на основе хранящейся в БД информации о произошедших событиях:
  + статистика возникновения опасных ситуаций по всем техническим средствам
  + статистика возникновения опасных ситуаций по каждому отдельному техническому средству
  + распределение потенциально опасным мест виртуального пространства с отображением на карте
  + контроль состояния заряда аккумулятора RFID меток пешего персонала

Приложение реализовано в соответствии с принципами разделения логики хранения и извлечения данных, бизнес-логики и представления информации. В разработке применён объектно-ориентированный подход. Развернутая на сервере версия приложения доступна по адресу <http://45.141.78.116:3300/>. Приложение работает на синтетических данных, однако может быть внедрено в промышленную среду, обрабатывать данные поступающие с датчиков в режиме реального времени. Типы аналитических отчетов могут быть расширены в соответствии с потребностями конкретной области применения.

Направлениями совершенствования разработанного приложения могут быть:

* адаптация под различные носимые и стационарные устройства с различным разрешением экрана;
* разработка мобильной и десктопной версии приложения
* расширение номенклатуры аналитических отчетов.
* регистрация попытки несанкционированного доступа
* интеграция с системой контроля и учета доступа

Кроме того, дооснащение транспортных средств дополнительными аппаратными компонентами позволит детектировать и анализировать данные о событиях ведущих к незначительным повреждениям технических средств, контролю технического состояния, расхода топлива и пр.

**Список используемой литературы**

**-** [**https://dash.plotly.com/layout**](https://dash.plotly.com/layout)

**-** [**https://www.postgresql.org/docs/**](https://www.postgresql.org/docs/)

**-** приказ Ростехнадзора 505 от 8.12.2020г.

- https://zonesafe.com/

- Даббас Элиас. Интерактивные дашборды и приложения с Plotly и Dash. Используем полноценный веб-фреймворк в Python на всю мощь - без JavaScript. 2023

**Приложение А**

Инструкция по развертыванию локальной версии приложения

* cоздать виртуальное окружение, установить следующие зависимости:
* altair==5.1.1  
  ansi2html==1.8.0  
  attrs==23.1.0  
  blinker==1.6.2  
  cachetools==5.3.1  
  certifi==2023.7.22  
  charset-normalizer==3.2.0  
  click==8.1.7  
  colorama==0.4.6  
  contourpy==1.1.0  
  cycler==0.11.0  
  dash==2.12.1  
  dash-bootstrap-components==1.5.0  
  dash-core-components==2.0.0  
  dash-html-components==2.0.0  
  dash-table==5.0.0  
  Flask==2.2.5  
  Flask-SQLAlchemy==3.0.5  
  fonttools==4.42.1  
  gitdb==4.0.10  
  GitPython==3.1.35  
  greenlet==2.0.2  
  idna==3.4  
  importlib-metadata==6.8.0  
  itsdangerous==2.1.2  
  Jinja2==3.1.2  
  jsonschema==4.19.0  
  jsonschema-specifications==2023.7.1  
  kiwisolver==1.4.5  
  markdown-it-py==3.0.0  
  MarkupSafe==2.1.3  
  matplotlib==3.7.2  
  mdurl==0.1.2  
  nest-asyncio==1.5.7  
  numpy==1.25.2  
  packaging==23.1  
  pandas==2.1.0  
  Pillow==9.5.0  
  plotly==5.16.1  
  protobuf==4.24.3  
  psycopg2-binary==2.9.7  
  pyarrow==13.0.0  
  pydeck==0.8.0  
  Pygments==2.16.1  
  Pympler==1.0.1  
  pyparsing==3.0.9  
  python-dateutil==2.8.2  
  python-dotenv==1.0.0  
  pytz==2023.3.post1  
  pytz-deprecation-shim==0.1.0.post0  
  referencing==0.30.2  
  requests==2.31.0  
  retrying==1.3.4  
  rich==13.5.2  
  rpds-py==0.10.2  
  six==1.16.0  
  smmap==5.0.0  
  SQLAlchemy==2.0.20  
  tenacity==8.2.3  
  toml==0.10.2  
  toolz==0.12.0  
  tornado==6.3.3  
  typing\_extensions==4.7.1  
  tzdata==2023.3  
  tzlocal==4.3.1  
  urllib3==2.0.4  
  validators==0.22.0  
  watchdog==3.0.0  
  Werkzeug==2.2.3  
  zipp==3.16.2

- развернуть сервер БД под управлением СУБД postgres

- прописать параметры соединения с БД, указанные в .env

- запустить на несколько секунд скрипт generate\_events\_in\_database.py. В зависимости от длительности выполнения скрипта получаем различный объем данных в БД (~ 1000 записей/сек)

- запустить run.py

- веб-приложение доступно по адресу: http://localhost:3300/

**Приложение Б**

Основные элементы приложения

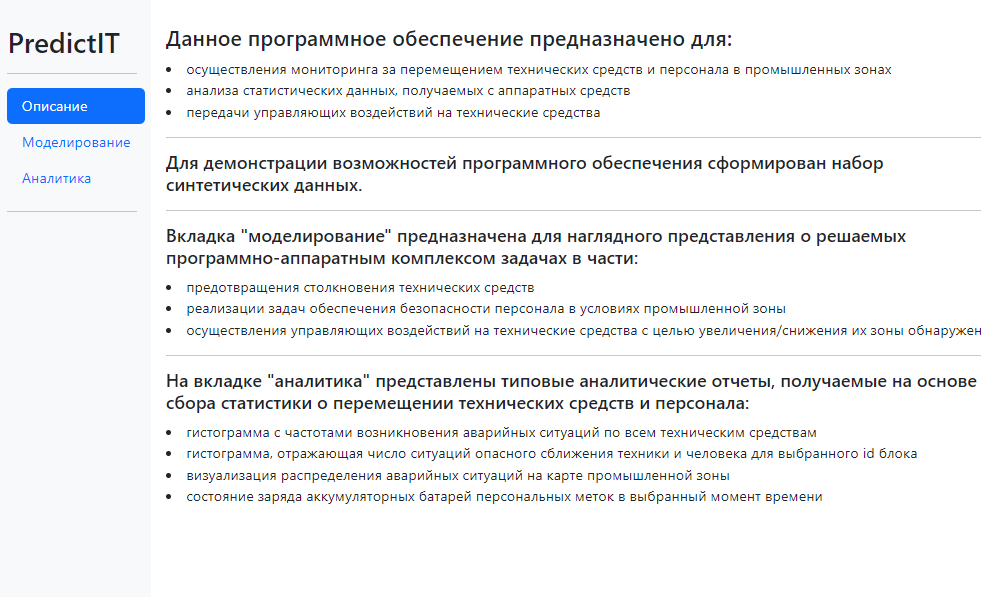


Рисунок 1 – вкладка с описанием приложения

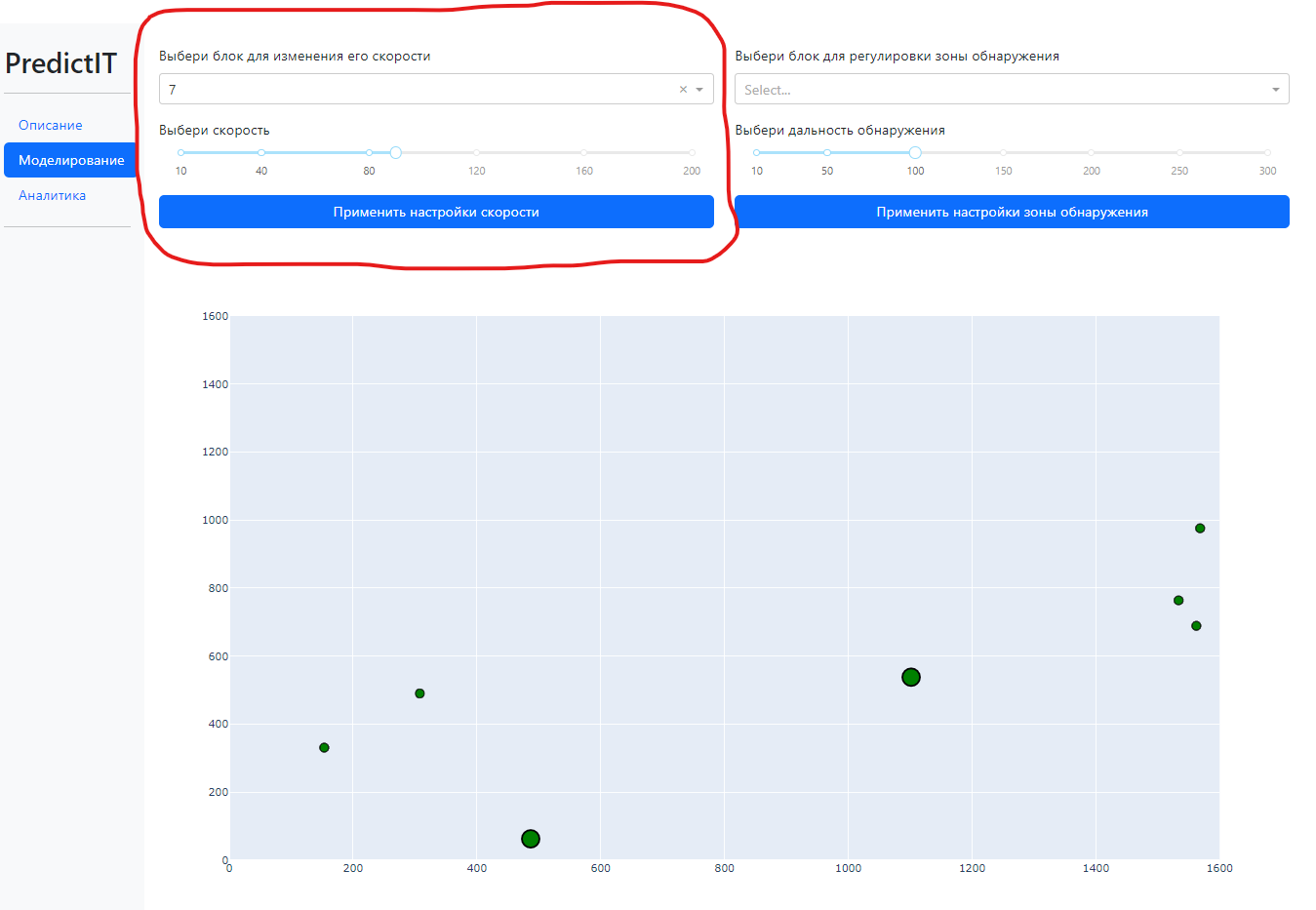


Рисунок 2 – блок моделирования, регулировка скорости

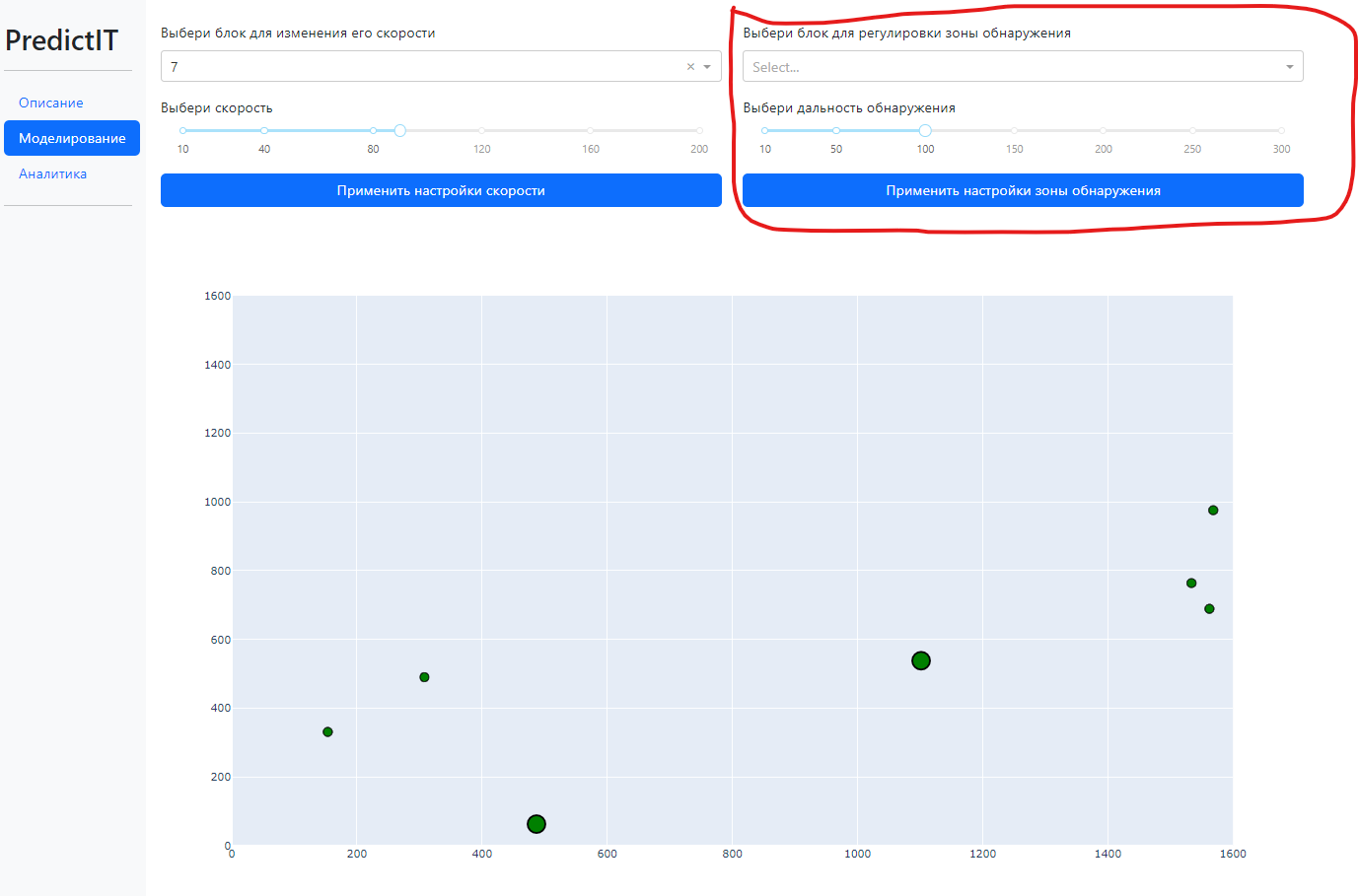


Рисунок 3 – блок моделирования, регулировка зоны обнаружения

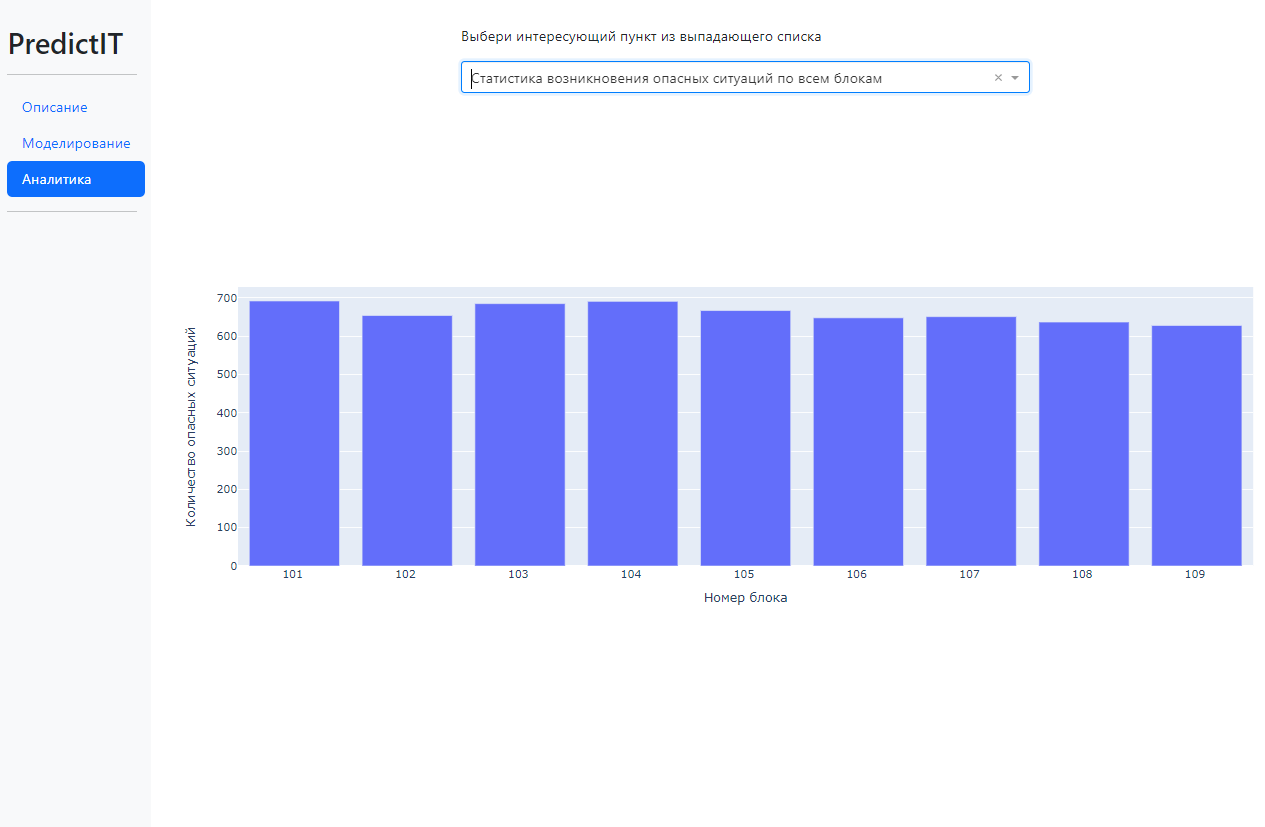


Рисунок 4 – блок аналитики

статистика возникновения опасных ситуаций по всем блокам

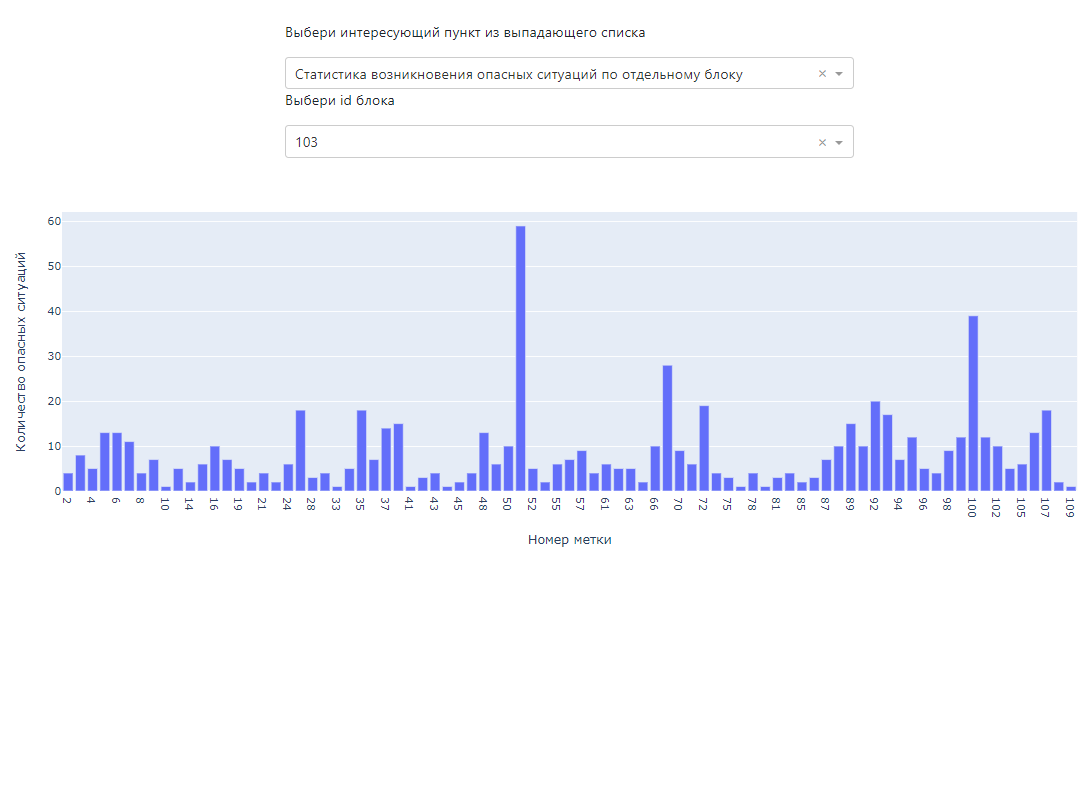


Рисунок 5 – блок аналитики

статистика возникновения опасных ситуаций по выбранному блоку

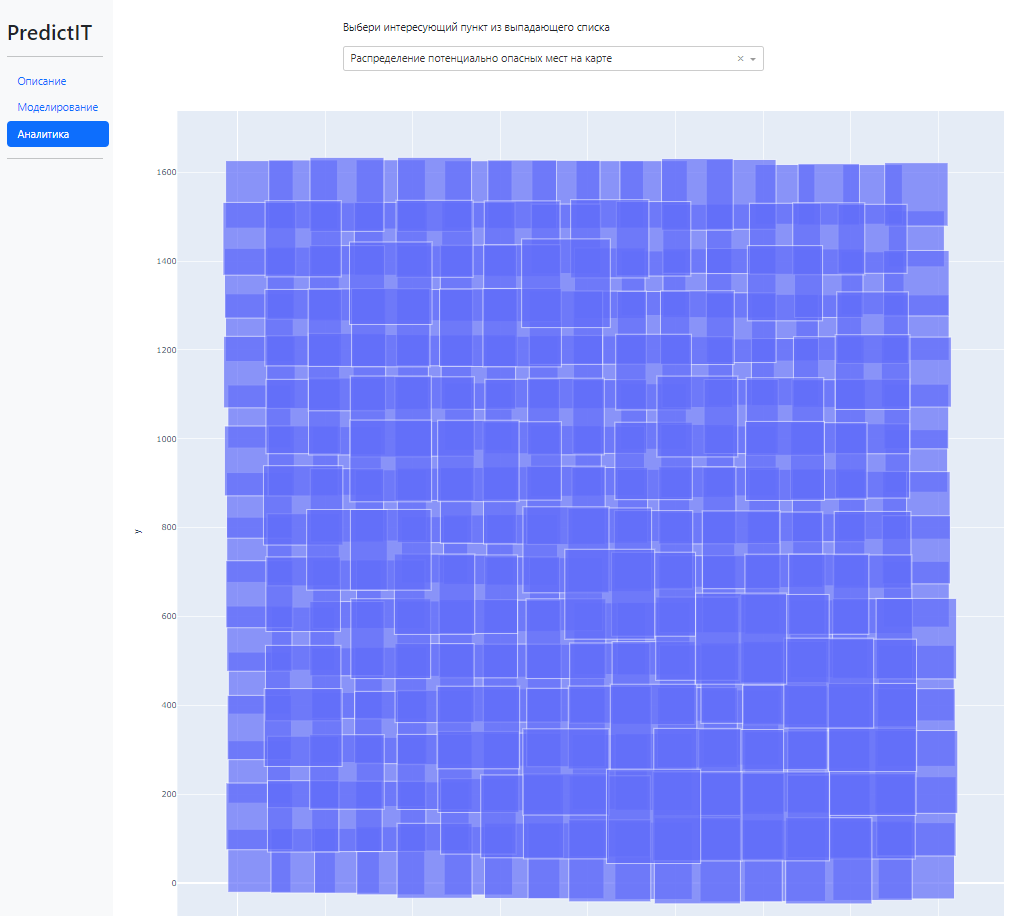


Рисунок 6 – блок аналитики

распределение потенциально опасных мест на карте

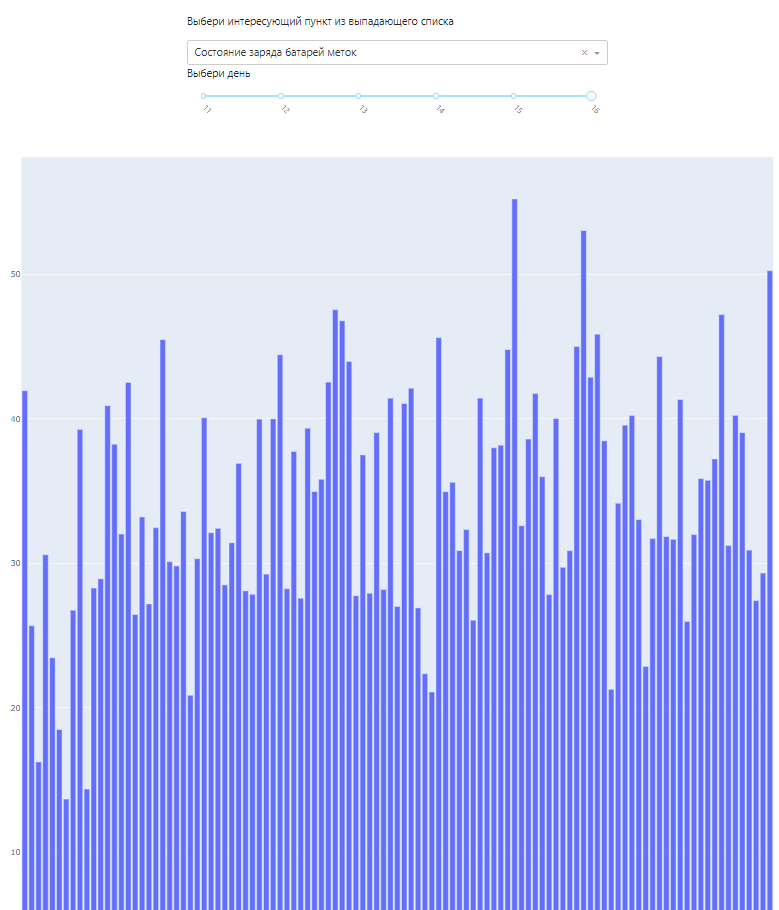


Рисунок 7 – блок аналитики

состояние заряда аккумуляторов RFID-меток