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. Основные функциональные возможности

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) 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

Одновременно с этим, центральные блоки имеют возможность фиксации:

* координат инцидента
* времени
* уникальных номеров пешехода и 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 Идентифицируйте, анализируйте и сравнивайте события Проверьте состояние батареи всех тегов Назначение тегов пешеходам и водителям Создавайте настраиваемые отчеты о собранных данных Управление показателями безопасности Регистрировать попытки несанкционированного доступа и ограничивать доступ персонала и транспортных средств.

**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:

* Apache Superset
* Redash
* Dash Plotly
* Streamlit

**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.

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.

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
* 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 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 - в упрощенном виде, это некая таблица с уникальным ключом и собственно связанным с ним значением, в котором может быть что угодно. 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 - это одна из наиболее популярных разновидностей NoSQL СУБД, где основной единицей логической модели данных является документ - структурированный текст, с определенным синтаксисом.

Графовые СУБД - специфичный тип, предназначены для работы с графами, с их узлами, свойствами, и произвольными отношениями между узлами.

Очень простой пример, это организация связей в различного типа социальных сетях, где нужно хранить связи между пользователями (узлами) по разным критериям (родственные связи, коллеги, общие интересы)

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.

**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

**Элементы управления имитационным моделированием**

**Глава 4. Основные функциональные возможности**

**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.

**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

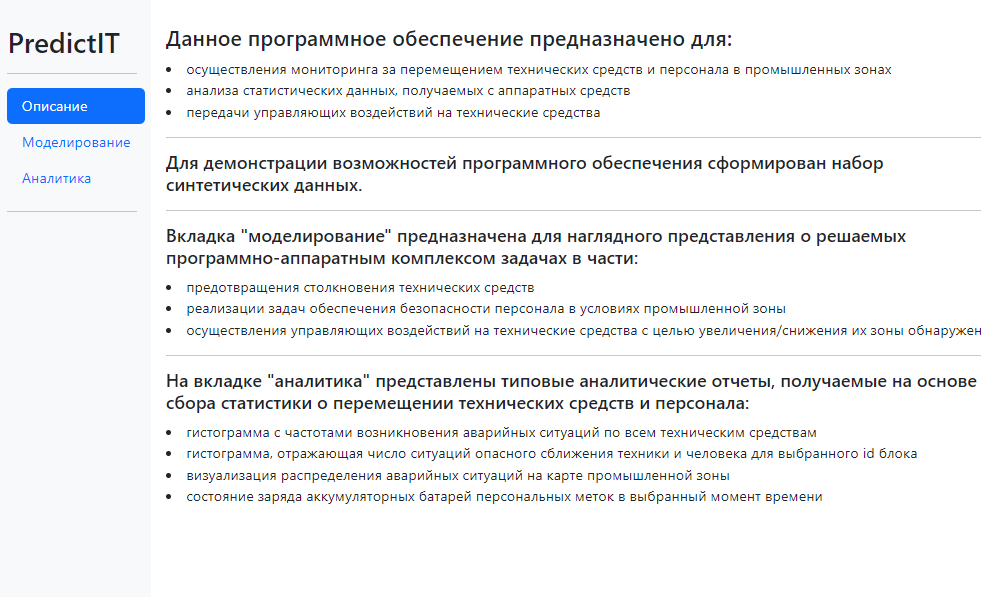


Рисунок 1 – вкладка с описанием приложения

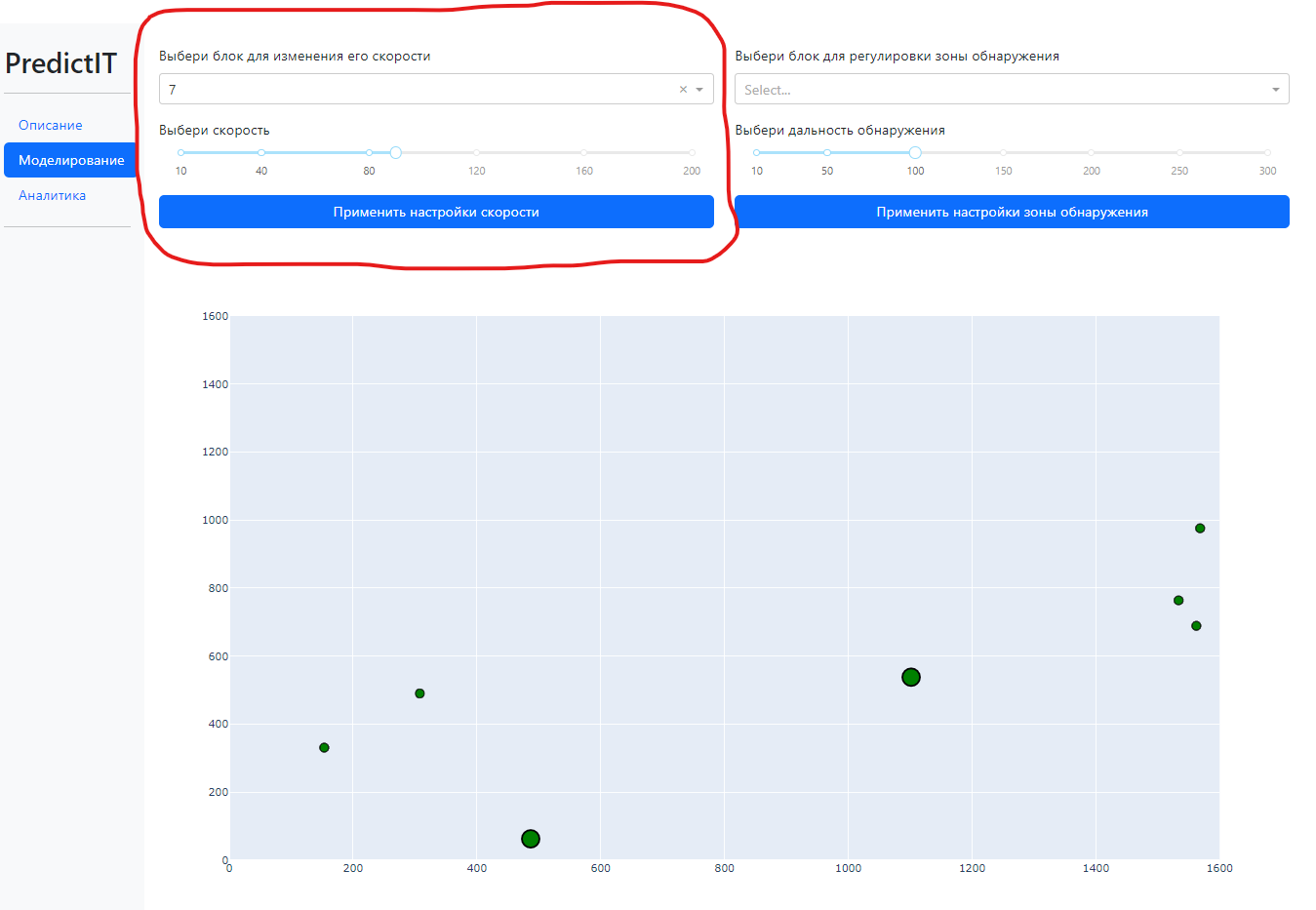


Рисунок 2 – блок моделирования, регулировка скорости

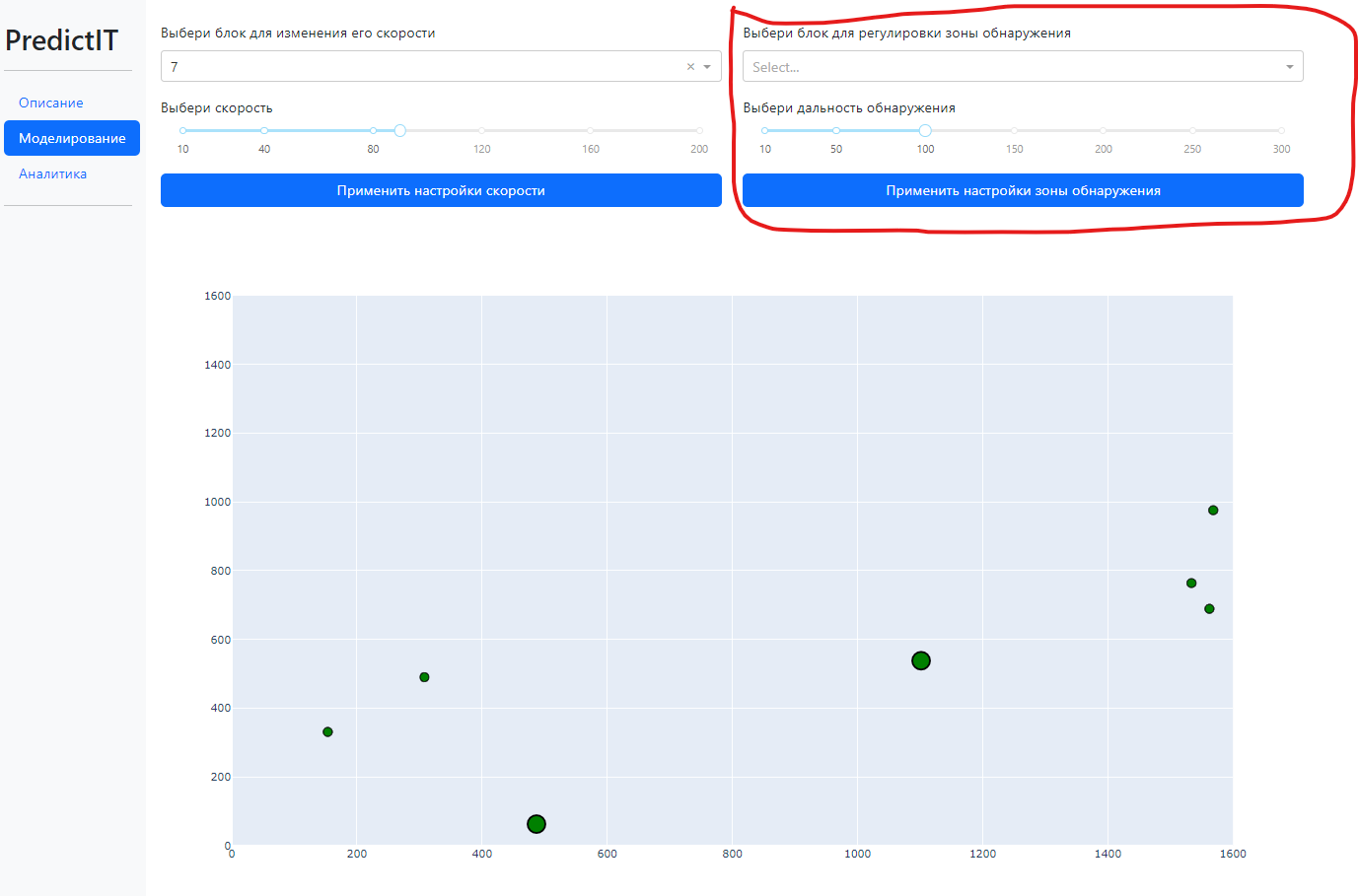


Рисунок 3 – блок моделирования, регулировка зоны обнаружения

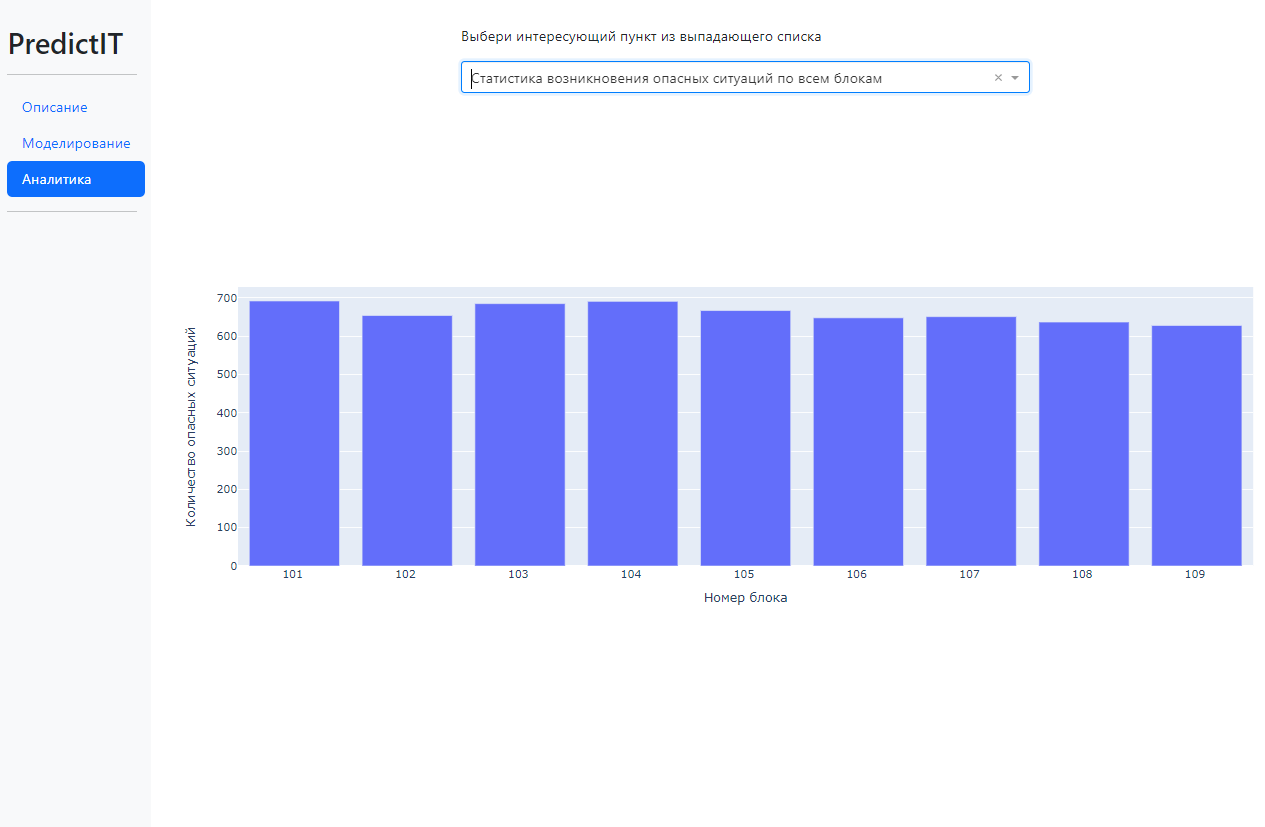


Рисунок 4 – блок аналитики

статистика возникновения опасных ситуаций по всем блокам

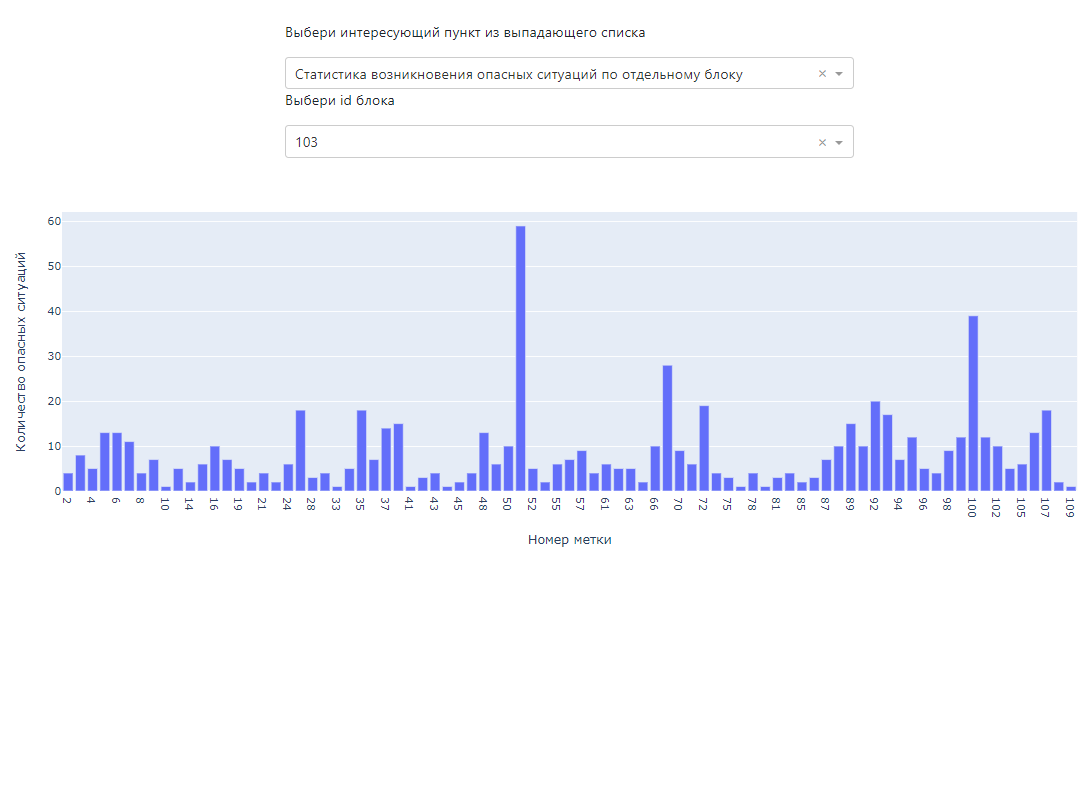


Рисунок 5 – блок аналитики

статистика возникновения опасных ситуаций по выбранному блоку

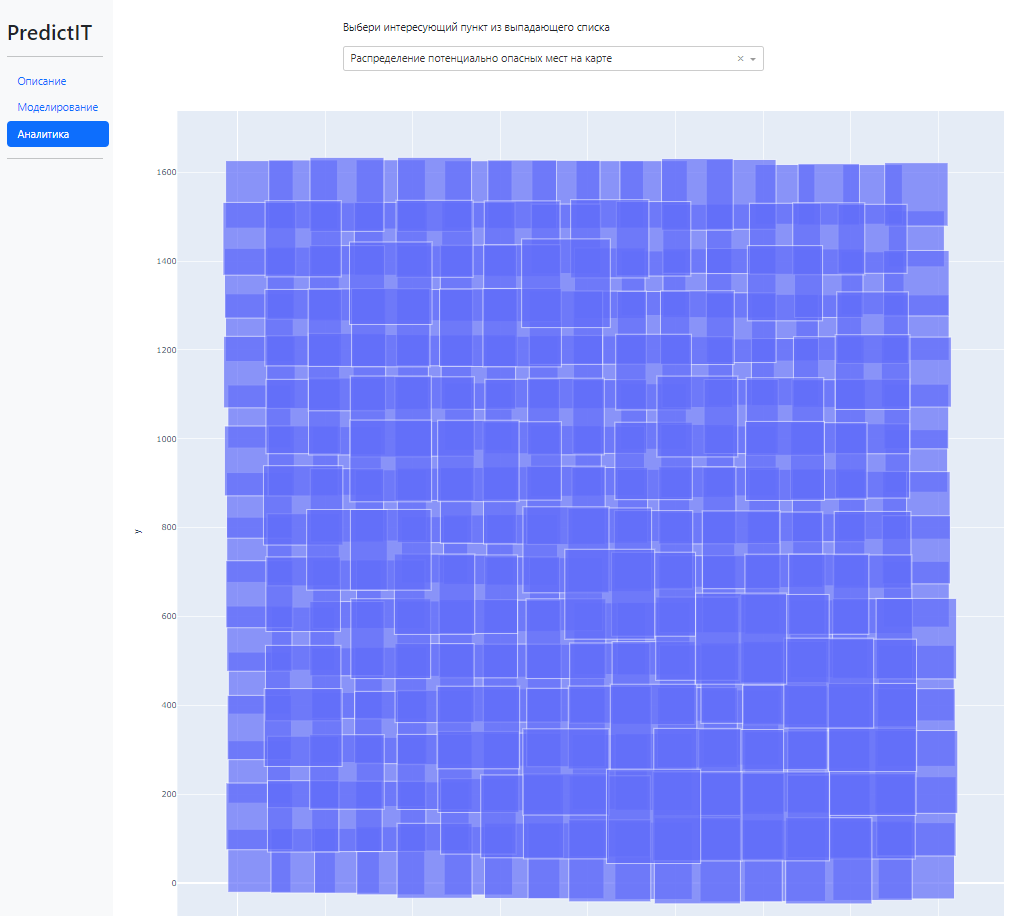


Рисунок 6 – блок аналитики

распределение потенциально опасных мест на карте

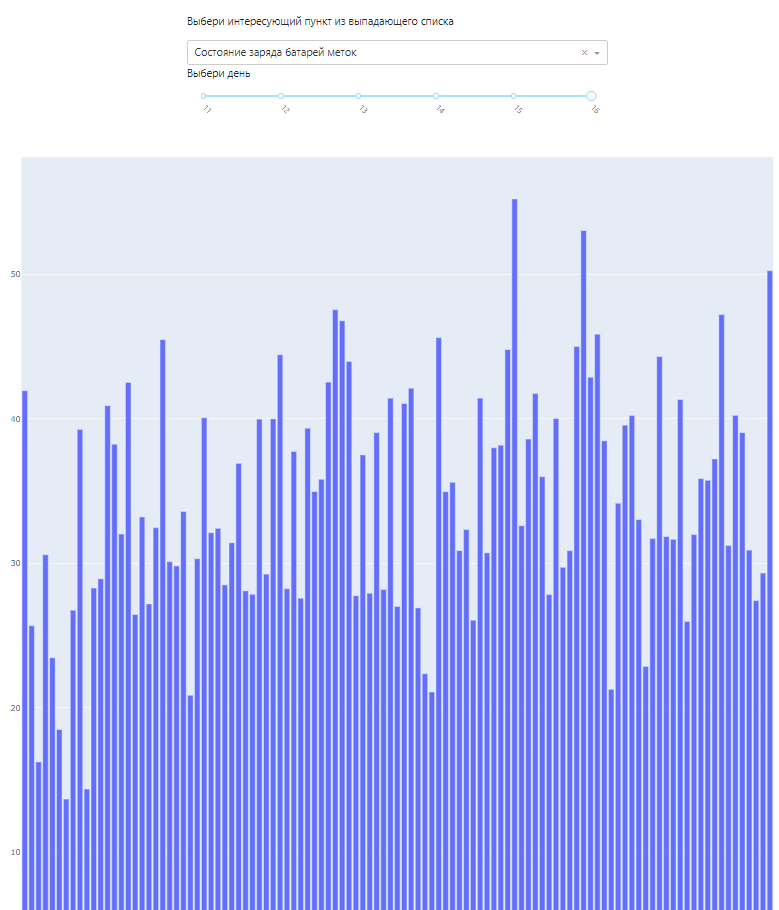


Рисунок 7 – блок аналитики

состояние заряда аккумуляторов RFID-меток