СОДЕРЖАНИЕ

1. Введение

2 Описание особенностей и процессов в учета спецодежды

3. Выбор языка программирования

4. Структура приложения

5. Тестирование и мониторинг.

6. Приложение

1. ВВЕДЕНИЕ

В основу проекта положено WEB приложение позволяющее вести учет и контроль выдачи рабочей одежды на производственном предприятии.

Учет специальной одежды и СИЗ не является новой задачей. Все предприятия, так или иначе связанные с обеспечением работников средствами защиты уже поставили у себя процессы учета и выдачи специальной одежды. Обеспеченность СИЗ - один из важных показателей проверки соблюдения требований по охране труда. На производствах затраты на обеспечение СИЗ на 1 работника в среднем составляет порядка 7,4 тыс. рублей в год, в отдельных отраслях до 24 тыс. рублей. Суммы для предприятий значительные и важно эффективно организовать процесс обеспечения персонала спецодеждой и СИЗ.

Требование обеспечения СИЗ определяются ст. 212 и 219 Трудового кодекса Российской Федерации, существуют отраслевые нормы выдачи СИЗ, (Приказ Минздравсоцразвития РФ от 1 октября 2008 г. N 541н), есть правила обеспечения спецодеждой и СИЗ (Приказ Минздравсоцразвития РФ от 1 июня 2009 г. N 290н)

Какие потери от неэффективного учета наблюдаются на предприятиях?

* Необоснованные запасы, приобретенные без учета сроков возникновения потребности со временем превращаются в неликвиды.
* Отсутствует сквозной поразмерный учет, что приводит к нехватке обеспечения индивидуальных средств защиты нужного размера.
* Не учитывается срок использования спецодежды в эксплуатации, возврат неиспользованной спецодежды на склад.
* Высокая трудоемкость изменения расчетов при сокращении или изменении рабочих мест производства.
* Мониторинг фактической обеспеченности СИЗ не ведется оперативно

Эти проблемы ложатся на плечи специалиста по охране труда или складских работников им трудно без удобного инструмента организовать правильный и эффективный учет. Как можно реализовать необходимый учет специальной одежды? На практике есть 3 варианта программного решения:

1. Электронные таблицы - Настроить электронные таблицы и заполняемые бланки, занести в них и затем постоянно актуализировать информацию, вести сверку с бухгалтерией и учитывать в них параллельно факты выдачи и возврата. Вариант самый распространенный и не самый удобный. Мало того, что таблицы должны работать в многопользовательском режиме, для их разработки потребуется написание сложных запросов и даже макросов, они, как правило, имеют свойства теряться, ломаться и разобраться с ними без автора будет сложно.
2. Бухгалтерский учет - доработать бухгалтерский учет на предмет внесения в программу необходимых данных, реализовать в одной программе все разрезы учета спецодежды, от потребности, до учета эффективности. Это другая крайность реализации данной задачи. На крупных предприятиях процессы учета сложные и поддержание адаптированной программы регламентированного учета становится практически невозможным при увеличении реализуемых в них функциях.
3. Специальная программа оперативного учета - Создать отдельный модуль программы, который сможет собрать в себе данные из разных источников, существующих на предприятии информационных систем, и обеспечить выполнение требуемых функций и передачу результата в другие блоки. Минус данного решения в том, что создать такой программный продукт под силу не каждому отделу автоматизации и у них всегда есть "более важные" задачи.

На ряду с этим до сих пор сохраняется и ручное ведение личных карточек работника. Это связано с дороговизной и излишней сложностью существующих готовых продуктов, отсутствием специалистов для разработки своего продукта. Низкой квалификацией ответственных сотрудников Особенно актуален этот вопрос для предприятий с численностью от 100 до 250 человек, когда ручное ведение становится сложной задачей, а готовые продукты дороги и несут в себе излишний функционал. Что увеличивает не только стоимость продукта, но и время на внедрение, освоение и использование.

Целью проекта является создание WEB приложения по учету спецодежды включающее в себя минимально необходимый функционал. Что позволит вести учет сотрудников при том существенно упрощая и сокращая кол-во затрачиваемого времени, особенно по сравнению с ручным учетом. И при наличии мобильного интерфейса доступ к информации с любой точки предприятия, что особенно актуально в связи с запретом размещения со стороны требований пожарной безопасности в складских помещениях стационарных компьютеров. При этом минимализирует финансовую нагрузку на собственника по покупке внедрению и освоению приложения.

Приложение написано языком программирования Java. С использованием требуемых технологий предоставляемых фреймворком Spring таких как Spring Cloud, Spring JPA и другие, библиотек Lombok, Thymeleaf В основу приложения положена микросервисная архитектура включающая в себя модули ответственные за решение отдельных задач Приложение создано с учетом специфики работы конкретного предприятия. При этом в силу простоты каждого модуля позволяет легко адаптировать под запросы конкретного потребителя

Вся работа по проекту выполнена собственноручно с использованием полученных в ходе обучения и самостоятельной работы знаний и навыков

**2 Описание особенностей и процессов в учета спецодежды**

Учет — составная часть управления экономическими процессами и объектами, сущность которого состоит в фиксации их состояния и параметров, сборе и накоплении сведений об экономических объектах и процессах, отражения этих сведений в учетных ведомостях. В связи с этим на предприятиях должен вестись строгий учет и контроль за выдачей работникам спецодежды и требуемых СИЗ . Важной обязанностью работодателя является организация этого процесса. Сотрудники должны обеспечиваться всеми необходимыми СИЗ в установленные сроки. На предприятии требуется обеспечить строгий контроль за сроками применения средств защиты, а особенно — за соблюдением нормативных сроков носки.

Периодичность выдачи средств защиты устанавливается в зависимости от особенностей производства и требований к защите сотрудников от вредных воздействий. Под СИЗ понимается:

• спецодежда;

• спецобувь;

• головные уборы;

• перчатки или рукавицы;

• прочее

Задачи учета специальной одежды на предприятии

: • установка и учет размеров спецодежды и антропометрических размеров работников;

• установка норм и потребностей в спецодежде, спецобуви и СИЗ;

• формирование внутреннего заказа с учетом складских остатков и выявленных потребностей;

• ведение учета на складах спецодежды и сиз по размерам

• учет выданной спецодежды и СИЗ с учетом размеров и срока износа;

• формирование отчетности о потребностях, обеспеченности, запасах спецодежды и СИЗ.

Как видно из этого анализа задачи довольно обширны и это еще не все. Надо учесть, что каждое предприятие особенное и везде свои особенности учета и требования к организованным процессам. Есть разделение контуров учета на оперативный, регламентированный и управленческий учет. Есть стандартные процессы закупок, оценки и движения товарно-материальных ценностей.

При разработке необходимо учесть все вышеперечисленные требования.

**3. Выбор языка программирования**

Достоинства и недостатки трех лидирующих языков программирования

**Java:**

Преимущества:

* Объектно-ориентированное программирование
* Высокая производительность
* Кросс-платформенность
* Большое сообщество разработчиков
* Открытый исходный код

Недостатки:

* Сложность изучения для начинающих
* Громоздкий и сложный синтаксис

**Python:**

Преимущества:

* Простой и понятный синтаксис
* Интерпретируемый язык
* Большое количество готовых библиотек
* Подходит для быстрого прототипирования и автоматизации задач

Недостатки:

* Низкая производительность по сравнению с компилируемыми языками
* Не подходит для разработки сложных систем и приложений

**C#:**

Преимущества:

* Компилируемый язык, обеспечивающий высокую производительность
* Входит в состав платформы .NET, что предоставляет большое количество готовых библиотек и инструментов
* Подходит для разработки разнообразных приложений, в том числе игр и мобильных приложений

Недостатки:

* Требует установки специальной среды разработки (Visual Studio)
* Меньше возможностей для кастомизации по сравнению с другими языками .NET

Java выбран для разработки данного приложения, потому что он является одним из наиболее популярных и широко используемых языков программирования. Он обладает высокой производительностью, эффективностью и надежностью, что делает его идеальным для создания систем учета. Java также имеет большое и активное сообщество разработчиков, доступ к которому облегчает процесс разработки. Кроме того, является кросс-платформенным языком, что позволяет разработанному на нем приложению работать на различных типах устройств и операционных систем.

Java — это объектно-ориентированный язык программирования, который был разработан компанией Oracle. Он был создан в 1995 году и с тех пор стал одним из самых популярных языков программирования в мире. Java используется для создания различных типов приложений, включая веб-приложения, мобильные приложения, игры и многое другое. Поддерживает множество современных технологий и фреймворков, таких как Spring, Hibernate и др., что упрощает разработку и позволяет использовать готовые решения.

**4. Структура приложения**

Разрабатываемое приложение обладает следующими функциональными возможностями.

Хранение в БД (PostgreSQL) информации о трех базовых сущностях и их взаимодействии:

1. Сотрудники предприятиях

- Фамилия Имя Отчество,

- подразделение в котором работает сотрудника

- специальность сотрудника

2. Рабочая одежда

- Модель

- Тип

- Размер и рост

- Дату выдачи и замены

3. Рабочая обувь

- Модель

- Тип

- Размер

- Дату выдачи и замены

Формирует отчет о потребности с учетом окончания сроков эксплуатации отдельно по каждому типу рабочей одежды и обуви.

Формирует отчет по запасам

Приложение позволяет

Добавлять, удалять и редактировать данные сотрудников

Отслеживать выданную сотруднику спецодежду по каждой полученной единице.

Пополнять склад по мере поступления спецодежды

Осуществлять выдачу спецодежды сотруднику

Списание по мере окончания срока эксплуатации.

Контролировать складские запасы

Формировать заказ на спецодежду на основании окончания срока эксплуатации либо отсутствия размеров на складе.

Приложение имеет web интерфейс, что позволяет использовать приложение удаленно без привязки к рабочему месту.

Интерфейс был спроектирован в соответствии с принципом минимализма, обеспечивая максимальный доступ к функционалу и интуитивное понимание. Это позволяет использовать приложение сотрудникам с любой квалификацией.

Модульная архитектура приложения позволяет легко подключать новые модули для реализации дополнительных функций, например:

* Учет и контроль выдачи средств индивидуальной защиты (СИЗ), таких как перчатки, респираторы, каски и т.д.
* Формирование личных карточек сотрудников и журнала учета выдачи спецодежды, обуви и других СИЗ.
* Другие функции по требованию пользователей

В основе проекта лежит микросервисная архитектура. За взаимодействие между сервисами и интерфейсом отвечает клиентский модуль управляющий всей логикой приложения.

В проекте использованы следующие библиотеки и технологии добавленные в качестве зависимостей в файл pom.xml В качестве базовой используется 17 версия Java

Описание основных зависимостей и используемых в проекте технологий

Spring Boot - это инструмент для быстрой разработки Spring-приложений. Он упрощает процесс создания приложений и сокращает время разработки за счет автоматической конфигурации и минимизации конфигурации. Spring Boot позволяет создавать масштабируемые, высокопроизводительные приложения с минимальными усилиями.

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-actuator</artifactId>*

*</dependency>*

Spring Boot Starter Web - это стартовый набор для создания веб-приложений с использованием Spring Boot. Он содержит все необходимые зависимости для создания и запуска веб-приложения на основе Spring

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-web</artifactId>*

*</dependency>*

Spring Boot JPA Starter - это начальная зависимость для работы с JPA в Spring Boot приложениях. Она предоставляет все необходимые компоненты для работы с базами данных, такие как EntityManager, JPA Repositories и другие.

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-data-jpa</artifactId>*

*</dependency>*

Spring Boot Validation Starter - это начальная зависимость для валидации входных данных в Spring Boot приложениях. Она позволяет валидировать данные с помощью JSR-303 аннотаций и других валидаторов.

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-validation</artifactId>*

*</dependency>*

Spring Cloud OpenFeign Starter - это зависимость, которая позволяет использовать OpenFeign для создания RESTful API в Spring Cloud. OpenFeign - это легкий, декларативный REST клиент, который упрощает создание и использование RESTful API.

*<dependency>*

*<groupId>org.springframework.cloud</groupId>*

*<artifactId>spring-cloud-starter-openfeign</artifactId>*

*</dependency>*

Spring Boot Thymeleaf Starter - это стартовая зависимость для создания HTML-шаблонов с использованием Thymeleaf. Thymeleaf - это шаблонизатор XHTML, который позволяет создавать динамические HTML-страницы с использованием Spring.

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-thymeleaf</artifactId>*

*</dependency>*

Lombok - это библиотека, которая позволяет сократить объем кода за счет автоматического генерации getter, setter, constructor и других методов. Она упрощает процесс разработки и уменьшает количество повторяющегося кода.

*<dependency>*

*<groupId>org.projectlombok</groupId>*

*<artifactId>lombok</artifactId>*

*<optional>true</optional>*

*</dependency>*

Spring Cloud Netflix Eureka Client starter - это зависимость, которая позволяет вашему приложению Spring Boot регистрироваться и обнаруживать другие службы на сервере обнаружения служб Netflix OSS Eureka.

*<dependency>*

*<groupId>org.springframework.cloud</groupId>*

*<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>*

*</dependency>*

Micrometer Core - это библиотека для измерения и мониторинга различных метрик, таких как время обработки запросов, количество запросов, использование памяти и другие. Она позволяет собирать и обрабатывать эти данные, а также отправлять их в различные системы мониторинга.

*<dependency>*

*<groupId>io.micrometer</groupId>*

*<artifactId>micrometer-core</artifactId>*

*<version>1.12.2</version>*

*</dependency>*

Micrometer Prometheus Registry - это зависимость, которая позволяет регистрировать метрики в Prometheus. Она обеспечивает поддержку различных форматов вывода, таких как text, csv, json и другие, а также позволяет настраивать параметры вывода.

*<dependency>*

*<groupId>io.micrometer</groupId>*

*<artifactId>micrometer-registry-prometheus</artifactId>*

*<scope>runtime</scope>*

*</dependency>*

**База данных (БД)**

- PostgreSQL под управлением SpringJPA.

**Модуль** **Employee (сотрудник)**

CRUD модуль по учету сотрудников.

**сущность Employee (сотрудник)**

*Long id – уникальный идентификатор*

*String firstName — имя сотрудника*

*String lastName — фамилия сотрудника*

*String patronymic — отчество сотрудника*

*ProductionDivision productionDivision — подразделение организации*

*Company company - организация*

*String specialization - специальность*

**Сервисные функции**

*List<Employee> findAllEmployee() - все сотрудники*

*void saveEmployee(Employee employee)-сохранение сотрудника*

*Employee findById(Long id)*

*Employee updateEmployee(Employee employee);*

*void deleteEmployee(Long id) — поиск сотрудника по id*

*List<Employee> findAllEmployeeByProductionDivision(ProductionDivision productionDivision) — поиск сотрудников подразделения*

*List<Employee> findAllEmployeeByLastName(String lastName) - поиск сотрудников по фамилии*

**сущность Company**

Сущность enumerated содержащая список компаний

**Сервисные функции/**

*static Company getType(String value) — поиск типа по строковому представлению*

*static List<Company> getValues() - отдает коллекцию типов*

*static List<String> getValuesString() - отдает коллекцию строковых представлений типов*

**сущность ProductionDivision(производственный участок)**

Сущность enumerated содержащая список производственных участков

**Сервисные функции**

*static ProductionDivision getType(String value) — поиск типа по строковому представлению*

*static List<ProductionDivision> getValues() - отдает коллекцию типов*

*static List<String> getValuesString() - отдает коллекцию строковых представлений типов*

**Модуль WorkWear (Спецодежда)**

Модуль отвечающий за обработку операций со спецодеждой

**сущность WorkWear(Спецодежда)**

Каждая единица рабочей одежды является самостоятельной сущностью хранения с уникальным Id.

*Long id – уникальный идентификатор*

*String modelWorkWear – модель спецодежды*

*WorkWearType workWearType — тип спецодежды (Enumerated)*

*WorkWearSize workWearSize — размер спецодежды (Enumerated)*

*WorkWearHeight workWearHeight — рост спецодежды (Enumerated)*

*int WorkWearStatus - флаг указывающий на статус выдано/свободно*

**Сервисные функции**

*List<WorkWear> findAllWorkWear() -вся спецодежда*

*void saveWorkWear(WorkWear workWear) - сохранение*

*WorkWear findById(Long id) — поиск по id*

*WorkWear updateWorkWear(WorkWear workWear) — редактирование*

*void deleteWorkWearById(Long id) - удаление*

*public List<WorkWear> saveAllWorkWear(List<WorkWear> workWearList) — сохранение из списка*

*List<WorkWear> findAllWorkWearByModelWorkWear(String modelWorkWear) — поиск по модели*

*List<WorkWear> findAllWorkWearByWorkWearType(WorkWearType workWearType) — поиск по типу*

*List<WorkWear> findAllWorkWearByWorkWearSize(WorkWearSize workWearSize) - поиск по размеру*

**сущность WorkWearTotal (Спецодежда обобщенная)**

Сервисная сущность для передачи клиенту суммарного количества спецодежды одного типа и размера для упрощения отображения складских остатков

*String modelWorkWear*

*WorkWearType workWearType;*

*WorkWearSize workWearSize;*

*WorkWearHeight workWearHeight;*

*int number — количество*

**Сервисные функции**

*List<WorkWearTotal> findWorkWearByTypeSortedNumber(WorkWearType workWearType) - формирует складской остаток спецодежды полученного типа*

*List<WorkWearTotal> findWorkWearBySizeSortedNumber(WorkWearSize workWearSize) - формирует складской остаток спецодежды полученного размера*

*List<WorkWearTotal> findAllWorkWearSortedNumber() - формирует складской остаток всего ассортимента спецодежды.*

**Сущность WorkWearIssue ( Спецодежда выданная)**

Сущность привязывается к сотруднику в момент выдачи спецодежды содержит в себе характеристики эксплуатации.

*Long id — уникальный идентификатор сущности*

*Long employeeId - идентификатор сотрудника получившего спецодежду*

*Long workWearId - идентификатор выданной спецодежды*

*Integer monthPeriod; - расчетный период эксплуатации*

*LocalDate replacementDate — дата планируемой замены*

*LocalDate dateIssued — дата выдачи спецодежды*

**Сервисные функции**

*List<WorkWearIssued> findAllWorkWearIssued() - все сущности*

*WorkWearIssued findWorkWearIssuedById(Long id) - поиск сущности по id*

*String saveWorkWearIssued(WorkWearIssued workWearIssued) — сохранение сущности*

*String updateWorkWearIssued(WorkWearIssued workWearIssued) — редактирование сущности*

*void deleteWorkWearIssuedById(Long id) — удаление сущности*

*List<WorkWearIssued> findWorkWearIssuedByEmployeeId(Long id) - поиск сущности по id сотрудника*

*List<WorkWearIssued> findWorkWearIssuedToBeReplaced() - поиск сущности по условию приближения плановой даты замены*

*List<WorkWearIssuedView> findWorkWearIssuedEmployee(Long id) — формирования сущности для передачи в представление*

**Сущность WorkWearIssueView**

Вспомогательна сущность для передачи информации о выданной спецодежде в представление.

*String modelWorkWear – модель спецодежды*

*String workWearType — тип спецодежды (Enumerated)*

*String workWearSize — размер спецодежды (Enumerated)*

*String workWearHeight — рост спецодежды (Enumerated)*

*Long workWearId - идентификатор выданной спецодежды*

*Integer monthPeriod; - расчетный период эксплуатации*

*LocalDate replacementDate — дата планируемой замены*

*LocalDate dateIssued — дата выдачи спецодежды*

**Сущность WorkWearOrder**

Вспомогательная сущность для формирования заказа на спецодежду

*WorkWearType workWearType;*

*WorkWearSize workWearSize;*

*WorkWearHeight workWearHeight;*

**Сервисные функции**

*List<WorkWearTotal> findWorkWearReplacedByType(WorkWearType workWearType) - запрос спецодежды подлежащей замене по указанному типу*

*List<WorkWearTotal> findWorkWearAllReplaced() - запрос всей спецодежды подлежащей замене*

*List<WorkWearOrder> searchForMissingDimensionsByType(WorkWearType workWearType) — поиск размеров отсутствующих на складе по указанному типу спецодежды.*

*List<WorkWearOrder> searchForMissingDimensionsAll() - поиск всех отсутствующих на складе размеров*

**сущность *WorkWearType***

Сущность enumerated содержащая перечень типов спецодежды

**Сервисные функции**

*static WorkWearType getType(String value) — поиск типа по строковому представлению*

*static List<WorkWearType> getValues() - отдает коллекцию типов*

*static List<String> getValuesString() - отдает коллекцию строковых представлений типов*

**сущность *WorkWearSize*(**

Сущность enumerated содержащая перечень типовых размеров спецодежды

***Сервисные функции***

*static WorkWearSize getType(String value) — поиск типа по строковому представлению*

*static List<WorkWearSize> getValues() - отдает коллекцию типов*

*static List<String> getValuesString() - отдает коллекцию строковых представлений типов*

**сущность *WorkWearHeight***

Сущность enumerated содержащая перечень ростовок спецодежды

**Сервисные функции**

*static WorkWearHeight getType(String value) — поиск типа по строковому представлению*

*static List<WorkWearHeight> getValues() - отдает коллекцию типов*

*static List<String> getValuesString() - отдает коллекцию строковых представлений типов*

**Модуль WorkShoes (Рабочая обувь)**

Модуль отвечающий за обработку операций с рабочей обувью

**Сущность WorkShoes (Рабочая обувь)**

*Long id - уникальный идентификатор*

*String modelWorkShoes — модель рабочей обуви*

*int workShoesSize — размер рабочей обуви*

*WorkShoesType workShoesType - тип рабочей обуви (Enumerated)*

*int workShoesStatus - флаг указывающий на статус выдано/свободно*

**Сервисные функции**

*List<WorkShoes> findAllWorkShoes() - все сущности*

*void saveWorkShoes(WorkShoes workShoes) — сохранение сущности*

*WorkShoes findById(Long id) поиск сущности по id*

*WorkShoes updateWorkShoes(WorkShoes workShoes) — редактирование сущности*

*void deleteWorkShoes(Long id) — удаление сущности*

*List<WorkShoes> findAllWorkShoesByWorkShoesSize (Integer workShoesSize) — поиск обуви по размеру*

*List<WorkShoes> findAllWorkShoesByWorkShoesType (WorkShoesType workShoesType) — поиск обуви по типа*

*void saveAllWorkShoes (@RequestBody List<WorkShoes> workShoesList) — сохранение рабочей обуви из списка*

**Сущность WorkShoesTotal(рабочая обувь обобщенная)**

*nt workShoesSize — размер рабочей обуви*

*WorkShoesType workShoesType — тип рабочей обуви (Enumerated)*

*Integer number — количество*

**Сервисные функции**

*List<WorkShoesTotal> findWorkShoesByTypeSortedNumber(WorkShoesType workShoesType) - формирует складской остаток рабочей обуви полученного типа*

*List<WorkShoesTotal> findWorkShoesBySizeSortedNumber(Integer size)- формирует складской остаток рабочей обуви полученного размера*

*List<WorkShoesTotal> findAllWorkShoesSortedNumber() - формирует складской остаток всего ассортимента рабочей обуви*

**Сущность WorkShoesIssued(рабочая обувь выданная)**

*Long id — уникальный идентификатор сущности*

*Long employeeId - идентификатор сотрудника получившего обувь*

*Long workShoesId - - идентификатор выданной обуви*

*Integer monthPeriod; - расчетный период эксплуатации*

*LocalDate replacementDate — дата планируемой замены*

*LocalDate dateIssued — дата выдачи обуви*

**Сервисные функции**

*List<WorkShoesIssued> findAllWorkShoesIssued() - все сущности*

*String saveWorkShoesIssued(WorkShoesIssued workShoesIssued)— сохранение сущности*

*WorkShoesIssued findById(Long id) - поиск сущности по id*

*WorkShoesIssued updateWorkShoesIssued(WorkShoesIssued workShoesIssued)— редактирование сущности*

*void deleteWorkShoesIssued(Long id)— удаление сущности*

*List<WorkShoesIssued> findWorkShoesIssuedByEmployeeId(Long id) - поиск сущности по id сотрудника*

*List<WorkShoesIssued> findWorkShoesToBeReplaced() - поиск сущности по условию приближения плановой даты замены*

*List<WorkShoesIssuedView> findWorkShoesIssuedEmployee(Long id)— формирования сущности для передачи в представление*

**Сущность WorkShoesIssuedView**

Вспомогательная сущность для передачи в представление информации о выданной обуви

*String modelWorkShoes - модель рабочей обуви*

*int workShoesSize— размер рабочей обуви*

*String workShoesType - тип рабочей обуви*

*Long workShoesIssuedId – идентификатор представляемой сущности*

*LocalDate replacementDate — дата планируемой замены*

*LocalDate dateIssued — дата выдачи обуви*

**Сущность WorkShoesOrder**

Вспомогательная сущность для формирования заказа обуви

*int workShoesSize — размер рабочей обуви*

*WorkShoesType workShoesType - тип рабочей обуви (Enumerated)*

**Сервисные функции**

*List<WorkShoesTotal> findWorShoesReplacedByType(WorkShoesType workShoesType) - запрос обуви согласно указанного типа подлежащей замене*

*List<WorkShoesTotal> findWorkShoesAllReplaced() - запрос обуви подлежащейзамене*

*List<WorkShoesOrder> searchForMissingDimensionsByType(WorkShoesType workShoesType) — поиск отсутствующих на складе размеров обуви согласно указанного типа*

*List<WorkShoesOrder> searchForMissingDimensionsAll() - поиск всех отсутствующих на складе размеров обуви*

**Сущность WorkShoesType**

Сущность enumerated содержащая перечень типов рабочей обуви

**Сервисные функции**

*static WorkShoesType getType(String value) — поиск типа по строковому представлению*

*static List<WorkShoesType> getValues() - отдает коллекцию типов*

*static List<String> getValuesString() - отдает коллекцию строковых представлений типов*

***Модуль W*orkWearClient  *(Рабочая обувь)***

Модуль осуществляющий связь между интерфейсом и сервисными модулями приложения.

В состав модуля входят следующие блоки:

***-* clientApi**

Блок использует технологию FeignClient для осуществления связи с контроллерами сервисных модулей.

*EmployeeApiClient.java*

*WorkShoesApiClient.java*

*WorkShoesIssuedApiClient.java*

*WorkShoesOrderApiClient.java*

*WorkShoesTotalApiClient.java*

*WorkWearApiClient.java*

*WorkWearIssuedApiClient.java*

*WorkWearOrderApiClient.java*

*WorkWearOrderApiClient.java*

**- clientControllerWeb**

Блок интерфейс-контроллеров модуля

*ClientControllerWeb.java — стартовый контроллер*

*EmployeeControllerWeb.java — контроллер обеспечивающий работу с персоналом*

*IssueControllerWeb.java — контроллер отвечающий за выдачу спецодежды и обуви*

*WorkShoesControllerWeb.java — контроллер обеспечивающий работу с рабочей обувью*

*WorkWearControllerWeb.java — контроллер обеспечивающий работу со спецодеждой*

*WorkWearOrderControllerWeb.java — контроллер отвечающий за блок заказа спецодежды и обуви*

*WorkWearReserveWeb.java — контроллер отвечающий за блок работы со складскими запасами*

**- controller**

Блок контроллеров сущностей модуля

*EmployeeController.java*

*WorkShoesController.java*

*WorkShoesIssuedController.java*

*WorkShoesTotalController.java*

*WorkWearController.java*

*WorkWearIssuedController.java*

*WorkWearTotalController.java*

**- model**

*Employee*

*WorkShoes.*

*WorkShoesIssued*

*WorkShoesOrder*

*WorkShoesTotal*

*WorkWear*

*WorkWearIssued*

*WorkWearOrder*

*WorkWearTotal*

Сущности аналогичны описанным в соответствующих модулях

**modelEnum**

*Company*

*ProductionDivision*

*WorkShoesType*

*WorkShoesType*

*WorkWearSize*

*WorkWearType*

*Сущности аналогичны описанным в соответствующих модулях*

**- modelView**

Вспомогательные сущности используемые в работе с web-представлением

*EmploeeView*

*ShoesIssuedView*

*WearIssuedView*

*WorkShoesArrival*

*WorkShoesIssuedView-*

*WorkShoesOrderView.*

*WorkShoesTotalView*

*WorkShoesView*

*WorkWearArrival*

*WorkWearIssuedView*

*WorkWearOrderView*

*WorkWearTotalView*

*WorkWearView*

**- service**

EmployeeService.java

WorkShoesIssuedService.java

WorkWearIssueService.java

WorkShoesService.java

WorkWearOrderService.java

WorkWearReserveService.java

WorkWearService.java

WorkwearclientApplication.java

**Модуль Gateway API**

Шлюз обеспечивающий контролируемый доступ к модулям приложения, обеспечивая стандартизацию, безопасность, мониторинг и кэширование вызовов между различными сервисами. Его назначение заключается в упрощении и улучшении доступа к внутренним и внешним API, а также в предоставляет дополнительные функции, такие как авторизация. балансировка нагрузки. В результате API Gateway повышает эффективность, масштабируемость и надежность системы, а также упрощает управление API**.**

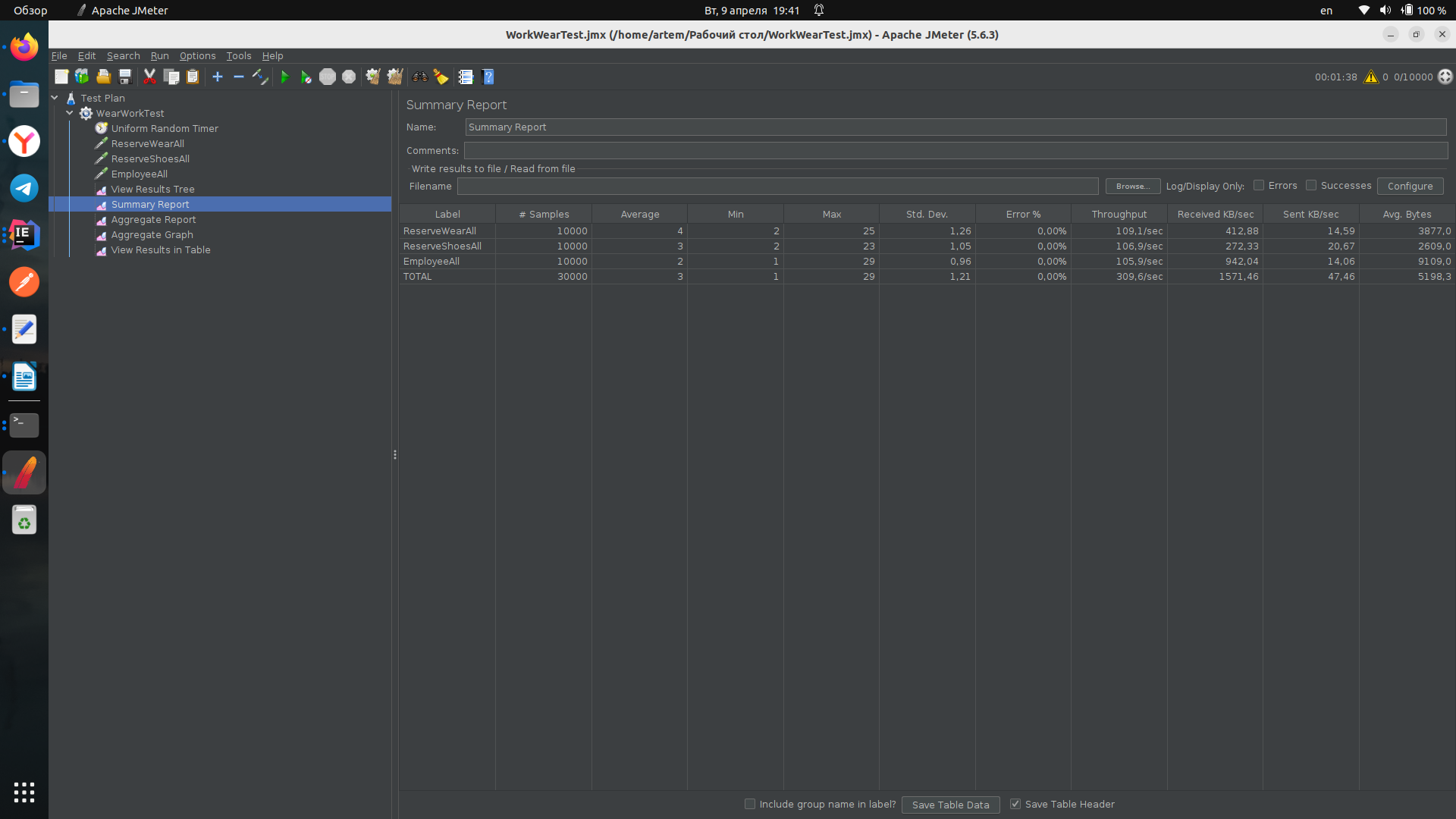
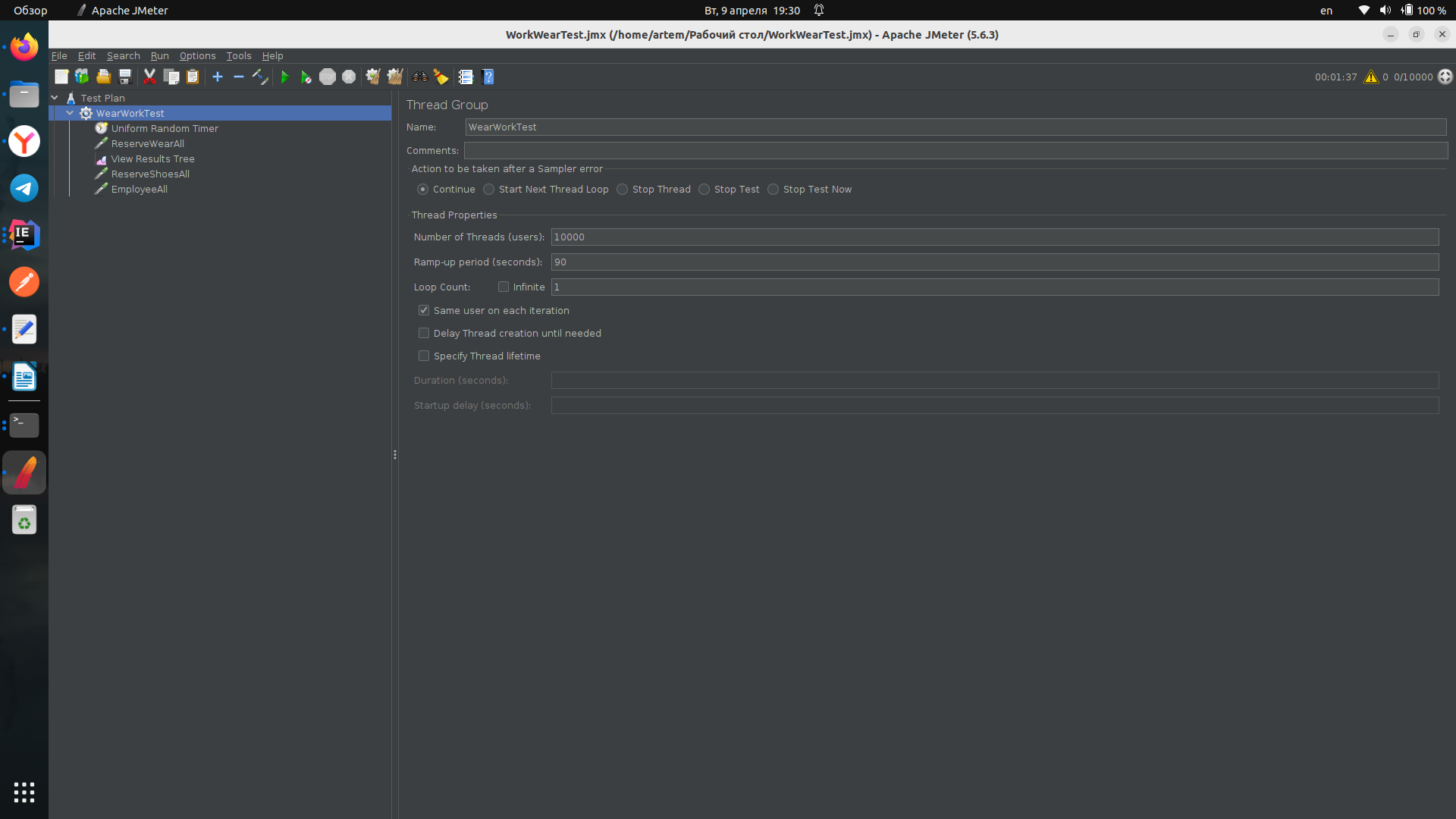
**Модуль Eureka Server**

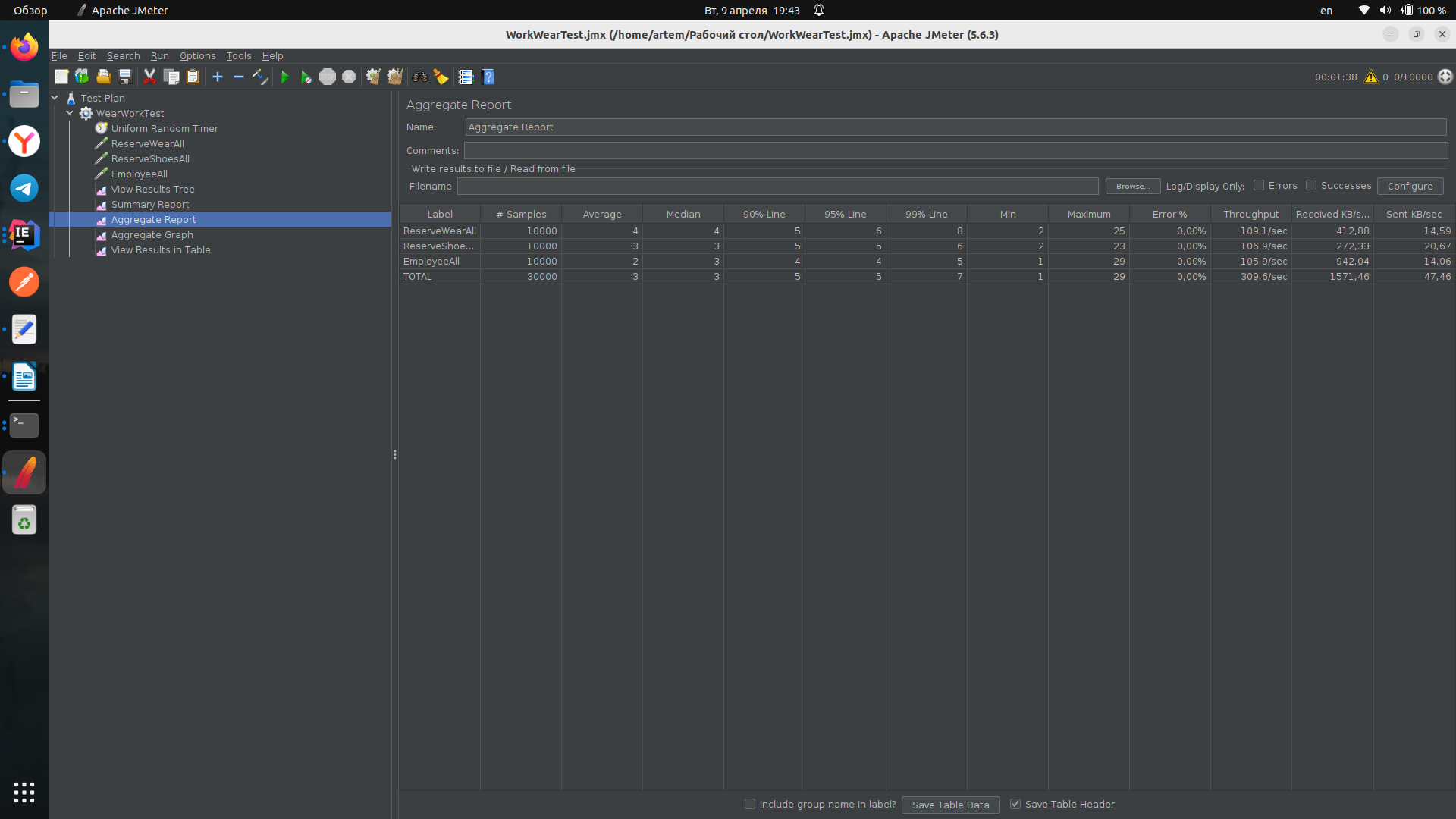
Cервер регистрации служб в микросервисной архитектуре, который используется для обнаружения сервисов, их масштабирования и обеспечения надежности системы. Он предоставляет информацию о доступных сервисах, их состоянии и метаданных, что позволяет другим компонентам системы автоматически обнаруживать и подключаться к ним. Назначение сервера Eureka состоит в автоматизации управления микросервисными архитектурами, улучшении масштабируемости, доступности и управляемости распределенных систем, а также упрощении разработки, развертывания и эксплуатации микросервисов.

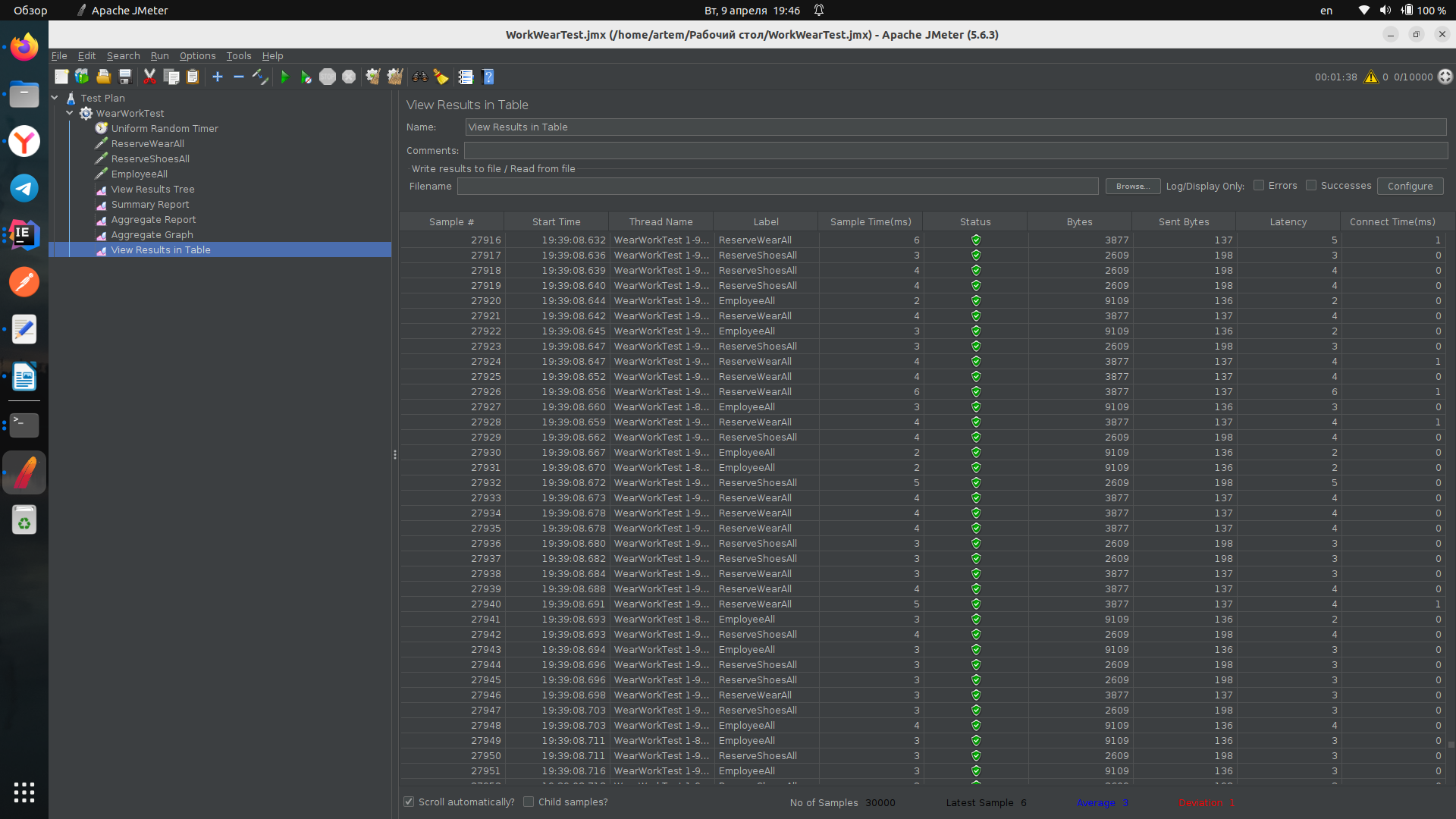
**4.Тестирование и мониторинг.**

Приложение протестировано в ручном режиме на соответствие принципа работы заявленному функционалу.

Помимо ручного тестирования проведено нагрузочное тестирование приложения для проверки работы наиболее ресурсно-затратных методов в режиме повышенной нагрузки с применением Apache Jmeter. Ниже приведены снимки экрана с базовыми настройками и результатами некоторых тестов

****

****

****

Метод  searchForMissingDimensionsAll  выполняет поиск отсутствующих размеров рабочих комбинезонов для всех типов, размеров и роста:

1. Устанавливаем флаг  is  в  true.
2. Получаем список всех рабочих комбинезонов из сервисного слоя с помощью метода  findAllWorkWear()  и сохраняем его в  workWearListAll.
3. Создаем пустой список  workWearOrderListAll  для хранения всех возможных комбинаций типов, размеров и роста рабочих комбинезонов.
4. Создаем пустой список  workWearOrderList  для хранения отсутствующих комбинаций типов, размеров и роста рабочих комбинезонов.
5. Для каждого типа рабочего комбинезона из перечисления  WorkWearType:
   * Пропускаем тип  OTHER.
   * Для каждого размера из перечисления  WorkWearSize:
     + Пропускаем неизвестные размеры.
     + Для каждой роста из перечисления  WorkWearHeight:
       - Пропускаем неизвестный рост.
       - Добавляем комбинацию типа, размера и роста в список  workWearOrderListAll.
6. Для каждого объекта  WorkWearOrder  из списка  workWearOrderListAll:
   * Получаем тип, размер и рост рабочего комбинезона.
   * Для каждого объекта  WorkWear  из списка  workWearListAll:
     + Если комбинация типа, размера и роста совпадает с текущим объектом  WorkWear, устанавливаем  is  в  false  и прерываем цикл.
   * Если  is  остается  true, добавляем объект  WorkWearOrder в список  workWearOrderList.
   * Сбрасываем  is  в  true.
7. Возвращаем список  workWearOrderList.

Этот метод ищет комбинации типов, размеров и роста рабочих комбинезонов, которые отсутствуют в списке всех рабочих комбинезонов. Если комбинация не найдена, она добавляется в список для возврата.

*public List<WorkWearOrder> searchForMissingDimensionsAll() {*

*boolean is = true;*

*List<WorkWear> workWearListAll = workWearService.findAllWorkWear();*

*List<WorkWearOrder> workWearOrderListAll = new ArrayList<>();*

*List<WorkWearOrder> workWearOrderList = new ArrayList<>();*

*for (WorkWearType workWearType : WorkWearType.values()) {*

*if (workWearType.equals(WorkWearType.OTHER)) continue;*

*for (WorkWearSize workWearSize : WorkWearSize.values()) {*

*if (workWearSize.equals(WorkWearSize.UNKNOWN)) continue;*

*for (WorkWearHeight workWearHeight : WorkWearHeight.values()) {*

*if (workWearHeight.equals(WorkWearHeight.UNKNOWN)) continue;*

*workWearOrderListAll.add(new WorkWearOrder(workWearType, workWearSize, workWearHeight));*

*}*

*}*

*}*

*for (WorkWearOrder wwo : workWearOrderListAll) {*

*WorkWearType workWearType = wwo.getWorkWearType();*

*WorkWearSize workWearSize = wwo.getWorkWearSize();*

*WorkWearHeight workWearHeight = wwo.getWorkWearHeight();*

*for (WorkWear ww : workWearListAll) {*

*if (workWearType.equals(ww.getWorkWearType()) && workWearSize.equals(ww.getWorkWearSize())*

*&& workWearHeight.equals(ww.getWorkWearHeight())) {*

*is = false;*

*break;*

*}*

*}*

*if (is) workWearOrderList.add(wwo);*

*is = true;*

*}*

*return workWearOrderList;*

*}*

Метод searchForMissingDimensionsByType принимает в качестве аргумента тип рабочей одежды workWearType.

Устанавливаем переменную is в значение true и создаем пустой список workWearOrderListAll для хранения объектов WorkWearOrder.

Получаем список всех рабочих комбинезонов определенного типа с помощью метода findAllWorkWearByWorkWearType() из сервисного слоя и сохраняем его в переменной workWearListAll.

Проходим по всем возможным значениям размеров и роста рабочих комбинезонов, и если они не равны UNKNOWN, добавляем их в список workWearOrderListAll.

Для каждого объекта WorkWearOrder из списка workWearOrderListAll:

Получаем тип, размер и рост рабочего комбинезона.

Для каждого объекта WorkWear из списка workWearListAll:

Если тип, размер и высота совпадают, устанавливаем is в false и прерываем цикл.

Если is остается true, добавляем объект WorkWearOrder в список workWearOrderList.

Сбрасываем is в true.

Возвращаем список workWearOrderList.

Этот метод ищет размеры, которые не используются в списке рабочих комбинезонов определенного типа. Он сравнивает каждый возможный размер с реально существующими размерами в списке всех рабочих комбинезонов определенного типа и добавляет неиспользуемые размеры в список для возврата.

Переменные, используемые в методе:

workWearListAll: список всех рабочих комбинезонов определенного типа.

workWearOrderListAll: список всех возможных размеров рабочих комбинезонов определенного типа.

workWearOrderList: список размеров, неиспользуемых в рабочих комбинезонах определенного типа.

workWearType, workWearSize и workWearHeight: переменные для хранения текущего типа, размера и высоты рабочего комбинезона.

is: булева переменная для отслеживания наличия размера в списке всех рабочих комбинезонов определенного типа.

*public List<WorkWearOrder> searchForMissingDimensionsByType(WorkWearType workWearType){*

*boolean is = true;*

*List<WorkWear> workWearListAll = workWearService.findAllWorkWearByWorkWearType(workWearType);*

*List<WorkWearOrder> workWearOrderListAll = new ArrayList<>();*

*List<WorkWearOrder> workWearOrderList = new ArrayList<>();*

*for (WorkWearSize workWearSize : WorkWearSize.values()) {*

*if (workWearSize.equals(WorkWearSize.UNKNOWN)) continue;*

*for (WorkWearHeight workWearHeight : WorkWearHeight.values()) {*

*if (workWearHeight.equals(WorkWearHeight.UNKNOWN)) continue;*

*workWearOrderListAll.add(new WorkWearOrder(workWearType, workWearSize, workWearHeight));*

*}*

*}*

*for (WorkWearOrder wwo : workWearOrderListAll) {*

*WorkWearSize workWearSize = wwo.getWorkWearSize();*

*WorkWearHeight workWearHeight = wwo.getWorkWearHeight();*

*for (WorkWear ww : workWearListAll) {*

*if (workWearSize.equals(ww.getWorkWearSize())*

*&& workWearHeight.equals(ww.getWorkWearHeight())) {*

*is = false;*

*break;*

*}*

*}*

*if (is) workWearOrderList.add(wwo);*

*is = true;*

*}*

*return workWearOrderList;*

*}*

Метод  typeSortedNumber  принимает в качестве аргументов список workWearList  рабочей одежды и тип рабочей одежды  workWearType.

1. Обнуляем счетчик  number  и создаем пустой список workWearTotalList  для хранения объектов  WorkWearTotal.
2. Создаем множество  workWearSizeSet  для хранения уникальных размеров рабочей одежды в списке  workWearList.
3. Для каждого уникального размера  wearSize  в множестве w orkWearSizeSet:
   * Фильтруем список  workWearList и оставляем только те элементы, у которых размер совпадает с текущим  wearSize.
   * Сортируем отфильтрованный список по размеру  WorkWearSize  и росту  WorkWearHeight.
   * Определяем первый рост  workWearHeight  в отсортированном списке.
   * Для каждого элемента  workWear  в отсортированном списке:
     + Если рост текущего элемента совпадает с предыдущим, то увеличиваем счетчик  number.
     + В противном случае:
       - Добавляем объект  WorkWearTotal  в список  workWearTotalList  с текущим типом, размером, ростом и количеством  number.
       - Обновляем текущий рост  workWearHeight  и обнуляем счетчик  number.
   * После прохода по всем элементам с текущим размером добавляем последний объект  WorkWearTotal  в список  workWearTotalList.
4. Возвращаем список  workWearTotalList  с объектами  WorkWearTotal, отсортированными по количеству.

Метод  typeSortedNumber  позволяет отсортировать рабочую одежду по количеству экземпляров для каждого уникального размера и роста внутри заданного типа рабочей одежды.

*public List<WorkWearTotal> typeSortedNumber(List<WorkWear> workWearList, WorkWearType workWearType) {*

*int number = 0;*

*List<WorkWearTotal> workWearTotalList = new ArrayList<>();*

*Set<WorkWearSize> workWearSizeSet = new HashSet<>();*

*List<WorkWear> workWearListFilterSize;*

*for (WorkWear wear : workWearList)*

*workWearSizeSet.add(wear.getWorkWearSize());*

*for (WorkWearSize wearSize : workWearSizeSet) {*

*workWearListFilterSize = workWearList.stream()*

*.filter(x -> wearSize.equals(x.getWorkWearSize()))*

*.sorted(Comparator.comparing(WorkWear::getWorkWearSize))*

*.sorted(Comparator.comparing(WorkWear::getWorkWearHeight))*

*.toList();*

*WorkWearHeight workWearHeight = workWearListFilterSize.get(0).getWorkWearHeight();*

*for (WorkWear workWear : workWearListFilterSize) {*

*if (workWearHeight.equals(workWear.getWorkWearHeight())) {*

*number++;*

*} else {*

*workWearTotalList.add(new WorkWearTotal(workWearType, wearSize, workWearHeight, number));*

*workWearHeight = workWear.getWorkWearHeight();*

*number = 1;*

*}*

*}*

*workWearTotalList.add(new WorkWearTotal(workWearType, wearSize, workWearHeight, number));*

*number = 0;*

*}*

*return workWearTotalList;*

*}*

Метод  sizeSortedNumber  принимает в качестве аргументов список  workWearList  рабочей одежды и размер рабочей одежды  workWearSize.

1. Обнуляем счетчик  number  и создаем пустой список  workWearTotalList  для хранения объектов  WorkWearTotal.
2. Создаем множество  workWearTypeSet  для хранения уникальных типов рабочей одежды в списке  workWearList.
3. Для каждого уникального типа  wearType  в множестве  workWearTypeSet:
   * Фильтруем список  workWearList  и оставляем только те элементы, у которых тип совпадает с текущим  wearType.
   * Сортируем отфильтрованный список по росту WorkWearHeight.
   * Определяем первый рост  workWearHeight в отсортированном списке.
   * Для каждого элемента  workWear  в отсортированном списке:
     + Если рост текущего элемента совпадает с предыдущим, то увеличиваем счетчик  number.
     + В противном случае:
       - Добавляем объект  WorkWearTotal  в список  workWearTotalList  с текущим типом, размером, ростом и количеством  number.
       - Обновляем текущий рост  workWearHeight  и обнуляем счетчик  number.
   * После прохода по всем элементам с текущим типом добавляем последний объект  WorkWearTotal  в список  workWearTotalList.
4. Возвращаем список  workWearTotalList  с объектами  WorkWearTotal, отсортированными по количеству.

Метод  sizeSortedNumber  позволяет отсортировать рабочую одежду по количеству экземпляров для каждого уникального типа и роста внутри заданного размера рабочей одежды.

*public List<WorkWearTotal> sizeSortedNumber(List<WorkWear> workWearList, WorkWearSize workWearSize) {*

*int number = 0;*

*List<WorkWearTotal> workWearTotalList = new ArrayList<>();*

*Set<WorkWearType> workWearTypeSet = new HashSet<>();*

*List<WorkWear> workWearListFilterType;*

*for (WorkWear workWear : workWearList) {*

*workWearTypeSet.add(workWear.getWorkWearType());*

*}*

*for (WorkWearType wearType : workWearTypeSet) {*

*workWearListFilterType = workWearList.stream()*

*.filter(x -> wearType.equals(x.getWorkWearType()))*

*.sorted(Comparator.comparing(WorkWear::getWorkWearHeight))*

*.toList();*

*WorkWearHeight workWearHeight = workWearListFilterType.get(0).getWorkWearHeight();*

*for (WorkWear workWear : workWearListFilterType) {*

*if (workWearHeight.equals(workWear.getWorkWearHeight())) {*

*number++;*

*} else {*

*workWearTotalList.add(new WorkWearTotal(wearType, workWearSize, workWearHeight, number));*

*workWearHeight = workWear.getWorkWearHeight();*

*number = 1;*

*}*

*}*

*workWearTotalList.add(new WorkWearTotal(wearType, workWearSize, workWearHeight, number));*

*number = 0;*

*}*

*return workWearTotalList;*

*}*

Метод findWorkShoesBySizeSortedNumber принимает в качестве аргумента размер рабочей обуви workShoesSize.

Обнуляем счетчик number и создаем пустой список workShoesTotals для хранения объектов WorkShoesTotal.

Получаем список всех рабочих обуви определенного размера с помощью метода findAllWorkShoesByWorkShoesSize() из репозитория рабочей обуви и сортируем его по типу рабочей обуви с помощью метода sorted() и компаратора Comparator.comparing(WorkShoes::getWorkShoesType). Сохраняем отсортированный список в переменную workShoesFilterType.

Получаем текущий тип рабочих обуви из первого элемента отсортированного списка и устанавливаем его в переменную workShoesType.

Проходим по отсортированному списку и подсчитываем количество рабочих обуви каждого типа. Если текущий тип рабочей обуви равен предыдущему, увеличиваем счетчик number. В противном случае, добавляем текущий тип и количество в список workShoesTotals с помощью метода add(). Затем, обновляем текущий тип рабочих обуви и обнуляем счетчик number.

Добавляем последний тип и количество в список workShoesTotals с помощью метода add().

Возвращаем список workShoesTotals.

В методе используются следующие переменные:

workShoesList - список всех рабочих обуви определенного размера.

workShoesTotals - список, содержащий суммарное количество рабочих обуви каждого типа.

workShoesFilterType - отсортированный список рабочих обуви по типу.

workShoesType, workShoesSize и number - переменные для хранения текущего типа, размера и количества рабочих обуви.

Метод состоит из трех частей. В первой части он получает список рабочих обуви определенного размера и сортирует его по типу. Во второй части он проходит по отсортированному списку и подсчитывает количество рабочих обуви каждого типа. В третьей части он добавляет каждый тип и количество в список для возврата.

*public List<WorkShoesTotal> findWorkShoesBySizeSortedNumber(Integer workShoesSize) {*

*int number = 0;*

*List<WorkShoes> workShoesList = sortedWorkShoesNotIssued(workShoesRepository.findAllWorkShoesByWorkShoesSize(workShoesSize));*

*List<WorkShoesTotal> workShoesTotals = new ArrayList<>();*

*List<WorkShoes> workShoesFilterType = workShoesList.stream()*

*.sorted(Comparator.comparing(WorkShoes::getWorkShoesType))*

*.toList();*

*try {*

*WorkShoesType workShoesType = workShoesFilterType.get(0).getWorkShoesType();*

*for (WorkShoes workShoes : workShoesFilterType) {*

*if (workShoesType.equals(workShoes.getWorkShoesType())) {*

*number++;*

*} else {*

*workShoesTotals.add(new WorkShoesTotal(workShoesType, workShoesSize, number));*

*workShoesType = workShoes.getWorkShoesType();*

*number = 1;*

*}*

*}*

*workShoesTotals.add(new WorkShoesTotal(workShoesType, workShoesSize, number));*

*} catch (Exception e) {*

*e.printStackTrace();*

*}*

*return workShoesTotals;*

*}*

Метод  typeSortedNumber  принимает список рабочей обуви  workShoesList  и тип рабочей обуви  workShoesType.

1. Проверяем, пуст ли список  workShoesList. Если да, возвращаем пустой список.
2. Инициализируем счетчик  number  равным 0 и создаем пустой список  workShoesTotals  для хранения объектов  WorkShoesTotal.
3. Сортируем список  workShoesList  по размеру рабочей обуви и сохраняем отсортированный список в  workShoesFilterSize.
4. Получаем размер рабочей обуви из первого элемента отсортированного списка и сохраняем его в переменную  workShoesSize.
5. Для каждого объекта  WorkShoes  из отсортированного списка:
   * Если размер рабочей обуви равен текущему размеру, увеличиваем счетчик  number.
   * Если размер меняется, добавляем объект  WorkShoesTotal с предыдущим размером и количеством в список  workShoesTotals, обновляем размер и счетчик, и продолжаем.
6. Добавляем последний тип и количество в список  workShoesTotals.
7. Возвращаем список  worэkShoesTotals.

Этот метод считает количество рабочей обуви каждого размера для определенного типа, сортируя их по размеру и создавая объекты  WorkShoesTota l с информацией о типе, размере и количестве рабочей обуви. Переменные, используемые в методе:

* workShoesList: список рабочей обуви.
* workShoesType: тип рабочей обуви.
* number: переменная для подсчета количества рабочей обуви каждого размера.
* workShoesTotals: список для хранения объектов WorkShoesTotal.
* workShoesFilterSize: отсортированный список рабочей обуви по размеру.
* workShoesSize: текущий размер рабочей обуви в процессе итерации.

*public List<WorkShoesTotal> typeSortedNumber(List<WorkShoes> workShoesList, WorkShoesType workShoesType) {*

*if (workShoesList.isEmpty()) return new ArrayList<>();*

*int number = 0;*

*List<WorkShoesTotal> workShoesTotals = new ArrayList<>();*

*List<WorkShoes> workShoesFilterSize = workShoesList.stream()*

*.sorted(Comparator.comparing(WorkShoes::getWorkShoesSize))*

*.toList();*

*int workShoesSize = workShoesFilterSize.get(0).getWorkShoesSize();*

*for (WorkShoes workShoes : workShoesFilterSize) {*

*if (workShoesSize == workShoes.getWorkShoesSize()) {*

*number++;*

*} else {*

*workShoesTotals.add(new WorkShoesTotal(workShoesType, workShoesSize, number));*

*workShoesSize = workShoes.getWorkShoesSize();*

*number = 1;*

*}*

*}*

*workShoesTotals.add(new WorkShoesTotal(workShoesType, workShoesSize, number));*

*return workShoesTotals;*

*}*

Метод  searchForMissingDimensionsAll  выполняет поиск отсутствующих размеров рабочей обуви для всех типов:

1. Устанавливаем флаг  is  в  true.
2. Получаем список всех рабочей обуви из сервисного слоя с помощью метода  findAllWorkShoes()  и сохраняем его в  workShoesListAll.
3. Создаем пустой список  workShoesOrderListAll  для хранения всех возможных комбинаций размеров и типов рабочей обуви.
4. Создаем пустой список  workShoesOrderList  для хранения отсутствующих размеров рабочей обуви.
5. Для каждого типа рабочей обуви из перечисления  WorkShoesType:
   * Пропускаем тип  OTHER.
   * Для каждого размера от минимального до максимального добавляем комбинацию размера и типа в список  workShoesOrderListAll.
6. Для каждого объекта  WorkShoesOrder  из списка  workShoesOrderListAll:
   * Получаем тип и размер рабочей обуви.
   * Для каждого объекта  WorkShoes  из списка  workShoesListAll:
     + Если тип и размер совпадают с текущим объектом  WorkShoesOrder, устанавливаем  is  в  false  и прерываем цикл.
   * Если  is  остается  true, добавляем объект  WorkShoesOrder в список  workShoesOrderList.
   * Сбрасываем  is  в  true.
7. Возвращаем список  workShoesOrderList.

Этот метод ищет размеры рабочей обуви, которые отсутствуют в списке всех рабочих обуви для каждого типа. Если размер не найден для определенного типа, он добавляется в список для возврата. Переменные, используемые в методе:

* workShoesListAll: список всех рабочей обуви.
* workShoesOrderListAll: список всех возможных комбинаций размеров и типов рабочей обуви.
* workShoesOrderList: список отсутствующих размеров рабочей обуви.
* workShoesType, workShoesSize: переменные для хранения текущего типа и размера рабочей обуви.
* is: булева переменная для отслеживания наличия размера рабочей обуви в списке всех рабочей обуви.

*public List<WorkShoesOrder> searchForMissingDimensionsAll() {*

*boolean is = true;*

*List<WorkShoes> workShoesListAll = workShoesService.findAllWorkShoes();*

*List<WorkShoesOrder> workShoesOrderListAll = new ArrayList<>();*

*List<WorkShoesOrder> workShoesOrderList = new ArrayList<>();*

*for (WorkShoesType workShoesType : WorkShoesType.values()) {*

*if (workShoesType.equals(WorkShoesType.OTHER)) continue;*

*for (int i = WorkShoes.MIN\_SIZE; i <= WorkShoes.MAX\_SIZE; i++) {*

*workShoesOrderListAll.add(new WorkShoesOrder(i, workShoesType));*

*}*

*}*

*for (WorkShoesOrder wso : workShoesOrderListAll) {*

*WorkShoesType workShoesType = wso.getWorkShoesType();*

*int workShoesSize = wso.getWorkShoesSize();*

*for (WorkShoes ws : workShoesListAll) {*

*if (workShoesType.equals(ws.getWorkShoesType()) && workShoesSize == ws.getWorkShoesSize()){*

*is = false;*

*break;*

*}*

*}*

*if (is) workShoesOrderList.add(wso);*

*is = true;*

*}*

*return workShoesOrderList;*

*}*

*Метод  searchForMissingDimensionsByType  ищет размеры рабочей обуви, которые отсутствуют на складе для определенного типа рабочей обуви.*

1. Вначале метод получает список всей рабочей обуви определенного типа с помощью метода  findAllWorkShoesByWorkShoesType  из сервисного слоя.
2. Затем он создает список всех возможных размеров рабочей обуви определенного типа с помощью перечисления  WorkShoes  и цикла по всем размерам от  MIN\_SIZE  до  MAX\_SIZE.
3. Далее он проходит по каждому размеру рабочей обуви в списке  workShoesOrderListAll  и проверяет, есть ли он в списке всей рабочей обуви определенного типа.
4. Если размер не найден в списке всей рабочей обуви определенного типа, он добавляется в список  workShoesOrderList  для возврата.
5. В конце метод возвращает список  workShoesOrderList, содержащий размеры рабочей обуви, которые не используются для определенного типа рабочей обуви.

Этот метод полезен для проверки, какие размеры рабочей обуви *отсутствуют на складе* для определенного типа рабочей обуви, и может быть использован для управления запасами.

*public List<WorkShoesOrder> searchForMissingDimensionsByType(WorkShoesType workShoesType) {*

*boolean is = true;*

*List<WorkShoes> workShoesListAll = workShoesService.findAllWorkShoesByWorkShoesType(workShoesType);*

*List<WorkShoesOrder> workShoesOrderListAll = new ArrayList<>();*

*List<WorkShoesOrder> workShoesOrderList = new ArrayList<>();*

*for (int i = WorkShoes.MIN\_SIZE; i <= WorkShoes.MAX\_SIZE; i++) {*

*workShoesOrderListAll.add(new WorkShoesOrder(i, workShoesType));*

*}*

*for (WorkShoesOrder wso : workShoesOrderListAll) {*

*int workShoesSize = wso.getWorkShoesSize();*

*for (WorkShoes ws : workShoesListAll) {*

*if (workShoesSize == ws.getWorkShoesSize()){*

*is = false;*

*break;*

*}*

*}*

*if (is) workShoesOrderList.add(wso);*

*is = true;*

*}*

*return workShoesOrderList;*

*}*

index.html

Код HTML-страницы представляет собой простой шаблон для веб-страницы, связанной с управлением рабочей одеждой. Вот краткое описание каждой части:

## Заголовок и Метаинформация

* <!DOCTYPE html>: Объявление типа документа.
* <html lang="en" xmlns:th="http://www.thymeleaf.org">: Открывающий тег HTML с указанием языка и пространства имен Thymeleaf.
* <head>: Содержит метаинформацию о документе.
  + <meta charset="UTF-8">: Устанавливает кодировку документа.
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">: Задает параметры отображения на мобильных устройствах.
  + <title>Work Wear</title>: Заголовок страницы.
  + <link th:href="@{/styles/index.css}" rel="stylesheet" />: Подключение внешнего CSS-файла с использованием Thymeleaf.

## Тело страницы

* <body>: Основное содержимое страницы.
  + <div class="header">: Блок с заголовком.
    - <p>Рабочая одежда</p>: Заголовок раздела.
    - <p><span th:text="${message}"></span></p>: Вывод переменной message с использованием Thymeleaf.
  + <div class="content">: Блок с контентом.
    - <div class="result">: Блок с результатами.
      * Форма для перехода на страницу "Сотрудники".
      * Форма для перехода на страницу "Рабочая одежда".
      * Форма для перехода на страницу "Рабочая обувь".
      * Форма для перехода на страницу "Запасы".

## Закрытие тегов

* Закрывающие теги </div>, </body>, </html>: Закрытие блоков и документа.

Этот HTML-шаблон представляет собой простую страницу с заголовком, блоками контента и кнопками для перехода на различные страницы, связанные с управлением рабочей одеждой.

employee\_create.html

Этот HTML-код представляет собой форму для добавления сотрудников с использованием Thymeleaf.

* <!DOCTYPE html>: Объявление типа документа.
* <html lang="en" xmlns:th="http://www.thymeleaf.org">: Открывающий тег HTML с указанием языка и пространства имен Thymeleaf.
* <head>: Содержит метаинформацию о документе.
  + <meta charset="UTF-8">: Устанавливает кодировку документа.
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">: Задает параметры отображения на мобильных устройствах.
  + <link th:href="@{/styles/employee\_create.css}" rel="stylesheet" />: Подключение внешнего CSS-файла с использованием Thymeleaf.
  + <title>Work Wear</title>: Заголовок страницы.
* <body>: Основное содержимое страницы.
  + <div class="header">: Блок с заголовком.
    - <p>Сотрудники ввод</p>: Заголовок раздела.
    - <p><span th:text="${message}"></span></p>: Вывод переменной message с использованием Thymeleaf.
  + <div class="content">: Блок с контентом.
    - <form class="form" action="#" th:action="@{/employee/create}" th:object="${employee}" method="post">: Форма для добавления сотрудников.
      * <div class="input">: Блок с полем ввода.
        + <label for="lastName">Фамилия</label>: Поле для ввода фамилии сотрудника.
        + <input type="text" th:field="\*{lastName}" id="lastName" placeholder="Фамилия">: Ввод фамилии сотрудника.
      * <div class="input">: Блок с полем ввода.
        + <label for="firstName">Имя</label>: Поле для ввода имени сотрудника.
        + <input type="text" th:field="\*{firstName}" id="firstName" placeholder="Имя">: Ввод имени сотрудника.
      * <div class="input">: Блок с полем ввода.
        + <label for="patronymic">Отчество</label>: Поле для ввода отчества сотрудника.
        + <input type="text" th:field="\*{patronymic}" id="patronymic" placeholder="Отчество">: Ввод отчества сотрудника.
      * <div class="input">: Блок с полем выбора.
        + <label for="productionDivision">Участок</label>: Поле для выбора участка сотрудника.
        + <select class="form-control" id="productionDivision" name="productionDivision">: Выбор участка сотрудника.

<option value="0">выберите участок</option>: Вариант выбора "выберите участок".

<option th:each="productionDivision : ${productionDivisions}" th:value="${productionDivision}" th:text="${productionDivision}"></option>: Вывод списка участков с использованием Thymeleaf.

* + - * <div class="input">: Блок с полем выбора.
        + <label for="company">Компания</label>: Поле для выбора компании сотрудника.
        + <select class="form-control" th:field="\*{company}" id="company">: Выбор компании сотрудника.

<option value="0">выберите компанию</option>: Вариант выбора "выберите компанию".

<option th:each="company : ${companies}" th:value="${company}" th:text="${company}"></option>: Вывод списка компаний с использованием Thymeleaf.

* + - * <div class="input">: Блок с полем ввода.
        + <label for="specialization">Специальность</label>: Поле для ввода специальности сотрудника.
        + <input type="text" th:field="\*{specialization}" id="specialization" placeholder="Специальность">: Ввод специальности сотрудника.
      * <div class="input">: Блок с кнопкой.
        + <button class="button" type="submit" value="Create">Записать</button>: Кнопка для отправки формы.
* <form class="button\_form" action="/employee" method="get">: Форма для возврата на предыдущую страницу.
  + <button class="button" type="submit">Вернуться</button>: Кнопка для отправки формы.

Закрывающие теги </div>, </body>, </html>: Закрытие блоков и документа.Этот HTML-код позволяет пользователю ввести данные о сотруднике и отправить их на сервер для сохранения.

employee\_issue.html

Этот HTML-код представляет собой страницу для отображения списка выданной рабочей одежды и обуви с использованием Thymeleaf.

* <!DOCTYPE html>: Объявление типа документа.
* <html lang="en" xmlns:th="http://www.thymeleaf.org">: Открывающий тег HTML с указанием языка и пространства имен Thymeleaf.
* <head>: Содержит метаинформацию о документе.
  + <meta charset="UTF-8">: Устанавливает кодировку документа.
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">: Задает параметры отображения на мобильных устройствах.
  + <title>Work Wear</title>: Заголовок страницы.
  + <link th:href="@{/styles/employee\_list.css}" rel="stylesheet" />: Подключение внешнего CSS-файла с использованием Thymeleaf.
* <body>: Основное содержимое страницы.
  + <div class="header">: Блок с заголовком.
    - <p>Выдано</p>: Заголовок раздела.
    - <p><span th:text="${message}"></span></p>: Вывод переменной message с использованием Thymeleaf.
  + <h1>Рабочая одежда</h1>: Заголовок раздела с рабочей одеждой.
  + <table>: Таблица для отображения данных о рабочей одежде.
    - <thead>: Заголовок таблицы.
      * Заголовки столбцов: Модель, Тип спецодежды, Размер, Рост, Дата выдачи, Дата замены, Списать, Вернуть.
    - <tbody>: Тело таблицы.
      * <tr th:each="wear : ${wears}">: Повторяющийся блок для каждой записи о рабочей одежде.
        + Данные о рабочей одежде: Модель, Тип спецодежды, Размер, Рост, Дата выдачи, Дата замены.
        + Ссылки для действий: Списать, Вернуть.
  + <h1>Рабочая обувь</h1>: Заголовок раздела с рабочей обувью.
  + <table>: Таблица для отображения данных о рабочей обуви.
    - <thead>: Заголовок таблицы.
      * Заголовки столбцов: Модель, Тип обуви, Размер, Дата выдачи, Дата замены, Списать, Вернуть.
    - <tbody>: Тело таблицы.
      * <tr th:each="shoes : ${shoesList}">: Повторяющийся блок для каждой записи о рабочей обуви.
        + Данные о рабочей обуви: Модель, Тип обуви, Размер, Дата выдачи, Дата замены.
        + Ссылки для действий: Списать, Вернуть.
  + <form class="button\_form" action="/employee/search" method="get">: Форма для возврата к поиску.
    - <button class="button" type="submit">Вернуться</button>: Кнопка для возврата.

Этот код представляет информацию о выданной рабочей одежде и обуви, а также предоставляет возможность выполнить действия по списанию и возврату.

employee\_list.html

Этот HTML-код представляет собой страницу для отображения списка сотрудников с возможностью редактирования, удаления, просмотра выданной рабочей одежды и выдачи рабочей одежды.

* <!DOCTYPE html>: Объявление типа документа.
* <html lang="en" xmlns:th="http://www.thymeleaf.org">: Открывающий тег HTML с указанием языка и пространства имен Thymeleaf.
* <head>: Содержит метаинформацию о документе.
  + <meta charset="UTF-8">: Устанавливает кодировку документа.
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">: Задает параметры отображения на мобильных устройствах.
  + <title>Work Wear</title>: Заголовок страницы.
  + <link th:href="@{/styles/employee\_list.css}" rel="stylesheet" />: Подключение внешнего CSS-файла с использованием Thymeleaf.
* <body>: Основное содержимое страницы.
  + <div class="header">: Блок с заголовком.
    - <p>Сотрудники</p>: Заголовок раздела.
    - <p><span th:text="${message}"></span></p>: Вывод переменной message с использованием Thymeleaf.
  + <table>: Таблица для отображения данных о сотрудниках.
    - <thead>: Заголовок таблицы.
      * Заголовки столбцов: id, Фамилия, Имя, Отчество, Участок, Компания, Специальность, Редактировать, Удалить, Выдано, Выдать.
    - <tbody>: Тело таблицы.
      * <tr th:each="employee : ${employees}">: Повторяющийся блок для каждого сотрудника.
        + Данные о сотруднике: id, Фамилия, Имя, Отчество, Участок, Компания, Специальность.
        + Ссылки для действий: Редактировать, Удалить, Просмотр выданной рабочей одежды, Выдача рабочей одежды.
  + <form class="button\_form" action="/employee/search" method="get">: Форма для возврата к поиску.
    - <button class="button" type="submit">Вернуться</button>: Кнопка для возврата.

Этот код представляет информацию о сотрудниках, а также предоставляет возможность выполнить действия по редактированию, удалению, просмотру выданной рабочей одежды и выдаче рабочей одежды.

employee\_search\_param.html

Этот HTML-код представляет собой страницу для поиска сотрудников по фамилии или по подразделению с использованием Thymeleaf.

* <!DOCTYPE html>: Объявление типа документа.
* <html lang="en" xmlns:th="http://www.thymeleaf.org">: Открывающий тег HTML с указанием языка и пространства имен Thymeleaf.
* <head>: Содержит метаинформацию о документе.
  + <meta charset="UTF-8">: Устанавливает кодировку документа.
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">: Задает параметры отображения на мобильных устройствах.
  + <link th:href="@{/styles/employee\_create.css}" rel="stylesheet" />: Подключение внешнего CSS-файла с использованием Thymeleaf.
  + <title>Work Wear</title>: Заголовок страницы.
* <body>: Основное содержимое страницы.
  + <div class="header">: Блок с заголовком.
    - <p>Сотрудники поиск</p>: Заголовок раздела.
    - <p class="p"><span th:text="${message}"></span></p>: Вывод переменной message с использованием Thymeleaf.
  + <div class="content">: Блок с контентом.
    - Форма для поиска сотрудников по фамилии.
      * Поле ввода фамилии.
      * Кнопка для отправки формы.
    - Форма для поиска сотрудников по подразделению.
      * Выпадающий список с подразделениями.
      * Кнопка для отправки формы.
  + <form class="button\_form" action="/employee/search" method="get">: Форма для возврата к поиску.
    - <button class="button" type="submit">Вернуться</button>: Кнопка для возврата.

Этот код позволяет пользователю искать сотрудников по фамилии или по подразделению.

issue\_shoes\_list

Этот HTML-код представляет собой страницу для отображения списка рабочей обуви с возможностью выдачи обуви.

* <!DOCTYPE html>: Объявление типа документа.
* <html lang="en" xmlns:th="http://www.thymeleaf.org">: Открывающий тег HTML с указанием языка и пространства имен Thymeleaf.
* <head>: Содержит метаинформацию о документе.
  + <meta charset="UTF-8">: Устанавливает кодировку документа.
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">: Задает параметры отображения на мобильных устройствах.
  + <title>Work Wear</title>: Заголовок страницы.
  + <link th:href="@{/styles/employee\_list.css}" rel="stylesheet" />: Подключение внешнего CSS-файла с использованием Thymeleaf.
* <body>: Основное содержимое страницы.
  + <div class="header">: Блок с заголовком.
    - <p>Рабочая обувь</p>: Заголовок раздела.
    - <p><span th:text="${message}"></span></p>: Вывод переменной message с использованием Thymeleaf.
  + <table>: Таблица для отображения данных о рабочей обуви.
    - <thead>: Заголовок таблицы.
      * Заголовки столбцов: id, Модель, Тип, Размер, Выдать.
    - <tbody>: Тело таблицы.
      * <tr th:each="workshoes : ${workShoesViewList}">: Повторяющийся блок для каждой пары рабочей обуви.
        + Данные о рабочей обуви: id, Модель, Тип, Размер.
        + Ссылка для действия: Выдать.
  + <form class="button\_form" action="/issue" method="get">: Форма для возврата к выдаче обуви.
    - <button class="button" type="submit">Вернуться</button>: Кнопка для возврата.

Этот код представляет информацию о рабочей обуви, а также предоставляет возможность выдачи обуви.

reserve\_wear\_list.html

Этот HTML-код представляет собой страницу для отображения запасов рабочей одежды.

* <!DOCTYPE html>: Объявление типа документа.
* <html lang="en" xmlns:th="http://www.thymeleaf.org">: Открывающий тег HTML с указанием языка и пространства имен Thymeleaf.
* <head>: Содержит метаинформацию о документе.
  + <meta charset="UTF-8">: Устанавливает кодировку документа.
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">: Задает параметры отображения на мобильных устройствах.
  + <title>Work Wear</title>: Заголовок страницы.
  + <link th:href="@{/styles/employee\_list.css}" rel="stylesheet" />: Подключение внешнего CSS-файла с использованием Thymeleaf.
* <body>: Основное содержимое страницы.
  + <div class="header">: Блок с заголовком.
    - <p>Запасы одежда</p>: Заголовок раздела.
    - <p><span th:text="${message}"></span></p>: Вывод переменной message с использованием Thymeleaf.
  + <table>: Таблица для отображения данных о запасах одежды.
    - <thead>: Заголовок таблицы.
      * Заголовки столбцов: Тип, Размер, Рост, Количество.
    - <tbody>: Тело таблицы.
      * <tr th:each="workwear : ${workWearTotalViewList}">: Повторяющийся блок для каждой записи о запасах одежды.
        + Данные о запасах одежды: Тип, Размер, Рост, Количество.
  + <form class="button\_form" action="/reserve/wear\_param" method="get">: Форма для возврата к параметрам резерва одежды.
    - <button class="button" type="submit">Вернуться</button>: Кнопка для возврата.

Этот код представляет информацию о запасах рабочей одежды.

workshoes\_create.html

Этот HTML-код представляет собой страницу для ввода данных о поступлении рабочей обуви.

* <!DOCTYPE html>: Объявление типа документа.
* <html lang="en" xmlns:th="http://www.thymeleaf.org">: Открывающий тег HTML с указанием языка и пространства имен Thymeleaf.
* <head>: Содержит метаинформацию о документе.
  + <meta charset="UTF-8">: Устанавливает кодировку документа.
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">: Задает параметры отображения на мобильных устройствах.
  + <link th:href="@{/styles/employee\_create.css}" rel="stylesheet" />: Подключение внешнего CSS-файла с использованием Thymeleaf.
  + <title>Work Wear</title>: Заголовок страницы.
* <body>: Основное содержимое страницы.
  + <div class="header">: Блок с заголовком.
    - <p>Рабочая обувь ввод</p>: Заголовок раздела.
    - <p class="p"><span th:text="${message}"></span></p>: Вывод переменной message с использованием Thymeleaf.
  + <div class="content">: Блок с формой для ввода данных о поступлении рабочей обуви.
    - Форма для ввода данных о поступлении рабочей обуви.
      * Поле для ввода модели обуви.
      * Выпадающий список с типами обуви.
      * Выпадающий список с размерами обуви.
      * Поле для ввода количества обуви.
      * Кнопка для отправки формы.
  + <form class="button\_form" action="/workshoes" method="get">: Форма для возврата к списку рабочей обуви.
    - <button class="button" type="submit">Вернуться</button>: Кнопка для возврата.

Этот код позволяет пользователю вводить данные о поступлении рабочей обуви в систему.

workwear\_create.html

Этот HTML-код представляет собой страницу для ввода данных о поступлении рабочей одежды.

* <!DOCTYPE html>: Объявление типа документа.
* <html lang="en" xmlns:th="http://www.thymeleaf.org">: Открывающий тег HTML с указанием языка и пространства имен Thymeleaf.
* <head>: Содержит метаинформацию о документе.
  + <meta charset="UTF-8">: Устанавливает кодировку документа.
  + <meta name="viewport" content="width=device-width, initial-scale=1.0">: Задает параметры отображения на мобильных устройствах.
  + <link th:href="@{/styles/employee\_create.css}" rel="stylesheet" />: Подключение внешнего CSS-файла с использованием Thymeleaf.
  + <title>Work Wear</title>: Заголовок страницы.
* <body>: Основное содержимое страницы.
  + <div class="header">: Блок с заголовком.
    - <p>Рабочая одежда ввод</p>: Заголовок раздела.
    - <p class="p"><span th:text="${message}"></span></p>: Вывод переменной message с использованием Thymeleaf.
  + <div class="content">: Блок с формой для ввода данных о поступлении рабочей одежды.
    - Форма для ввода данных о поступлении рабочей одежды.
      * Поле для ввода модели одежды.
      * Выпадающий список с типами спецодежды.
      * Выпадающий список с размерами спецодежды.
      * Выпадающий список с ростом спецодежды.
      * Поле для ввода количества одежды.
      * Кнопка для отправки формы.
  + <form class="button\_form" action="/workwear" method="get">: Форма для возврата к списку рабочей одежды.
    - <button class="button" type="submit">Вернуться</button>: Кнопка для возврата.

Этот код позволяет пользователю вводить данные о поступлении рабочей одежды в систему.

employee\_create.css

Это CSS-код определяет стили для HTML-элементов.

* .header: Определяет стили для элементов с классом header.
  + text-align: center;: Выравнивает текст по центру.
  + height: 200px;: Задает высоту блока.
  + background-color: rgb(69, 102, 247);: Задает цвет фона.
  + font-weight: bolder;: Устанавливает жирный шрифт.
  + color: aliceblue;: Задает цвет текста.
* .content: Определяет стили для элементов с классом content.
  + display: flex;: Определяет блок как flex-контейнер.
  + flex-wrap: wrap;: Разрешает перенос элементов на новую строку.
  + justify-content: center;: Выравнивает содержимое по центру.
  + align-content: space-around;: Равномерно распределяет пространство вокруг элементов.
  + width: 350px;: Задает ширину блока.
  + margin-top: 30px;: Задает отступ сверху.
* .button: Определяет стили для элементов с классом button.
  + background-color: rgb(55, 65, 199);: Задает цвет фона.
  + font-size: 22px;: Задает размер шрифта.
  + text-align: center;: Выравнивает текст по центру.
  + cursor: pointer;: Устанавливает курсор как у указателя.
  + border-radius: 10px;: Округляет углы.
  + width: 200px;: Задает ширину элемента.
  + height: 50px;: Задает высоту элемента.
  + color: bisque;: Задает цвет текста.
  + margin: 25px;: Задает отступы вокруг элемента.
* .input: Определяет стили для элементов с классом input.
  + display: block;: Определяет элемент как блок.
  + margin-left: 50px;: Задает отступ слева.
  + height: 80px;: Задает высоту элемента.
  + width: 350px;: Задает ширину элемента.
  + color: #16033b;: Задает цвет текста.
  + background-color: rgb(175, 213, 245);: Задает цвет фона.
  + font-size: 20px;: Задает размер шрифта.
  + align-items: center;: Выравнивает содержимое по центру.
* input, select: Определяет стили для элементов input и select.
  + width: 250px;: Задает ширину элемента.
  + height: 50px;: Задает высоту элемента.
  + font-size: 20px;: Задает размер шрифта.
  + color: #1c0447;: Задает цвет текста.
  + margin: 10px;: Задает отступы вокруг элемента.
  + margin-left: 50px;: Задает отступ слева.
* label: Определяет стили для элементов label.
  + margin-left: 60px;: Задает отступ слева.
  + margin-bottom: 5px;: Задает отступ снизу.
  + display: block;: Определяет элемент как блок.
  + font-size: 20px;: Задает размер шрифта.
* p, a: Определяет стили для элементов  p  и  a.
  + font-size: 80px;: Задает размер шрифта.
  + margin: 6px;: Задает отступы вокруг элемента.
  + color: aliceblue;: Задает цвет текста.
* .p: Определяет стили для элементов с классом p.
  + font-size: 35px;: Задает размер шрифта.
  + margin: 6px;: Задает отступы вокруг элемента.
  + color: aliceblue;: Задает цвет текста.
* .long: Определяет стили для элементов с классом long.
  + background-color: rgb(55, 65, 199);: Задает цвет фона.
  + font-size: 22px;: Задает размер шрифта.
  + text-align: center;: Выравнивает текст по центру.
  + cursor: pointer;: Устанавливает курсор как у указателя.
  + border-radius: 10px;: Округляет углы.
  + justify-content: center;: Выравнивает содержимое по центру.
  + padding-block: calc(0.375em - 1px);: Задает отступы сверху и снизу.
  + padding-inline: 0.75em;: Задает отступы слева и справа.
  + width: 250px;: Задает ширину элемента.
  + height: 50px;: Задает высоту элемента.
  + color: bisque;: Задает цвет текста.
  + margin: 25px;: Задает отступы вокруг элемента.