АННОТАЦИЯ

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И СПЕЦИАЛЬНЫХ ТЕРМИНОВ

БД – База данных.

СУБД – Система управления базами данных.

ПП – Программный продукт.

CRUD – (Create Read Update Delete) – Операции с БД создание, чтение, модификация, удаление.

ERP – (Enterprise Resource Planning) – интегрированные программные решения, которые помогают организациям управлять и автоматизировать основные бизнес-процессы, например планирование или учет.

IDE – (Integrated Development Environment) – Интегрированная среда разработки.

MILP – (Mixed-Integer Linear Programming, смешанное целочисленное линейное программирование) – это математический метод оптимизации, который решает задачи линейного программирования с дополнительным условием.

ORM – (Object-relational mapping) – Объектно-реляционное отображение.

ВВЕДЕНИЕ

Актуальность темы. В современном мире компании по производству транспортного оборудования используют различные узлы и агрегаты, которые имеют свойство изнашиваться и терять работоспособность. Отсюда возникает необходимость проведения ремонтных работ или замены. При ремонте часто используется специализированное дорогостоящее оборудование, и для оптимизации ресурсов у предприятия есть необходимость в ведении учета о поврежденных деталях.

На начальном этапе ремонта выполняется разборка узлов и агрегатов на различные отдельные детали, для подготовке к процессу дефектации. При дефектации определяется тип и возможность восстановления детали, а также определяются необходимые работы по восстановлению формы или размера детали. В зависимости от типов деталей определяются способы устранения дефектов.

Соответственно, алгоритм системы хранения и ввода данных может быть актуален для записи и хранении данных о деталях, которые подлежат восстановлению, и может быть эффективен в последующих расчетах времени восстановления деталей для составления расписания времени выполнения ремонтных работ. При учете времени в начале необходимо упорядочить технологические процессы восстановления деталей, а после выполнять необходимые расчеты. Таким образом, при наличии данных о времени восстановлении система оптимизации расписаний восстановления узлов и агрегатов технологического и транспортного оборудования на специализированных ремонтных предприятиях будет полезна в обеспечении эффективной работы предприятий. Она предназначена для планирования, оптимизации и эффективном управлении процесса восстановления различных узлов и агрегатов техники и оборудования.

Цель работы. Целью текущей выпускной квалификационной работы является разработка алгоритма системы хранения и ввода данных о деталях, которые в последствии будут формироваться в партии в составлении и оптимизации расписания восстановления.

Для достижения поставленной цели необходимо выполнить **поставленные задачи**:

1) Разработать базу данных хранения информации о деталях;
2) Разработать формы ввода данных для деталей, заказов и расписаний;
3) Разработать методы формирования расчетов времени выполнения технологических процессов восстановления деталей цилиндрической формы;
4) Разработать методы формирования данных для оптимизации расписания на основе набора заданий;
5) Разработать систему интерпретации оптимизированного расписания.

Объект и предмет исследования. Объектом исследования в данной квалификационной работе является процесс восстановления детали на предприятии.

Предметом исследования является алгоритм хранения и ввода данных для оптимизации расписаний восстановления узлов и агрегатов.

Структура работы. Данная работа включает список сокращений, введение, три основных раздела, заключение, список использованных источников и одно приложение**.**

В первом разделе детально рассматривается предметная область, различные системы, которые решают схожие задачи по планированию, а также устанавливается цели и задачи разработки.

Второй раздел включает в себя информационное и функциональное проектирование, диаграммы нотаций IDEF0-IDEF1X, диаграмма в нотации Чена и BPMN. С помощью них можно эффективно определять функциональные требования.

Третий раздел включает в себя обоснование программных инструментов, структурно - функциональную схему и саму разработку. Завершается раздел тестированием и демонстрацией результатов.

Заключение содержит подведение итогов и описывает, какие цели в процессе выполнения работы были достигнуты.

1 АНАЛИЗ СИСТЕМ ОПТИМИЗАЦИЙ РАСПИСАНИЙ ДЛЯ ВОССТАНОВЛЕНИЯ ТРАНСПОРТНЫХ УЗЛОВ И АГРЕГАТОВ НА СПЕЦИАЛИЗИРОВАННОМ ПРЕДПРИЯТИИ

1.1 Анализ предметной области и основных ее характеристик

На специализированных предприятиях регулярно возникает необходимость минимизировать время получения результатов и максимизировать эффективность использования оборудования для получения хорошего качества. Создание математической модели для оптимизации расписания позволяет специализированным ремонтным предприятиям повысить производительность, сократить временные и ресурсные затраты, а также обеспечить высокое качество ремонтных работ, что в конечном итоге приводит к более эффективному и рентабельному производству.

На данный момент, транспортное оборудование регулярно требует обслуживания и ремонта, так как детали имеют срок эксплуатации и подвергаются износу. Современные ремонтные специализированные предприятия производят ремонтно-восстановительные работы узлов и агрегатов транспортного оборудования. Для проведения ремонта, в начале производится разбор деталей транспортного оборудования на составляющие компоненты.

Порядок восстановления детали начинается с определения вида ремонтных работ. После разбора узлов, требующих ремонта, проводится начальный этап восстановительных работ - дефектация. Данный этап помогает определить возможность восстановления детали путем оценки остаточного ресурса самой детали, который можно использовать при восстановлении. Затраты на восстановление составляют 10 – 50 % от стоимости детали [8]. Таким образом, при дефектации определяется, какие детали подлежат замене, какие детали можно восстановить, и какие детали могут быть использованы дальше, без необходимости проведения восстановительных работ. Для последующих работ

детали, которые можно восстановить, поступают на склады, а после на цеха восстановительного ремонта.

Износ цилиндрической поверхности составляет 52% [8]. По этой причине, на специализированных ремонтных предприятиях выделяется особое внимание к восстановлению деталей цилиндрической формы.

К деталям цилиндрической формы больших размеров относятся различные валы. К деталям цилиндрической формы малых размеров относятся пальцы, валики, оси. Детали этих типов не восстанавливаются и подлежат замене [11].

Технологический процесс - это совокупность методов, направленных на восстановление изношенного состояния детали и доведения до нормативных рекомендаций. Процесс включает в себя использование специализированной техники, оборудования и инструментов, которые способствуют достижению необходимого результата и качества с минимальными затратами [7].

Методы устранения дефектов входят в состав технологических процессов ремонтно-восстановительных предприятий, и зависят от видов дефектов деталей. Процесс восстановления детали цилиндрической формы включает в себя:

– удаление следов износа.. Поверхность детали для начала подвергается механической обработке, так как износ на поверхности чаще всего неравномерен, и важно обеспечить равномерное покрытие при восстановлении формы и размеров деталей;

– восстановительные операции (наплавка, сварка, металлизация и др.);

– после восстановления детали проходят токарную механическую обработку;

– заключительным этапом является шлифование.

Проведение ремонтно-восстановительных работ на любого рода предприятиях требует наличия специальной техники, которая могла бы сделать процесс восстановления некоторых параметров детали (например, размер, форма) автоматическим. Наиболее часто применимы такие технологические операции, как:

– автоматическая электродуговая наплавка под слоем флюса;

– вибродуговая наплавка;

– металлизация (напыление расплавленного металла на подготовленную поверхность деталей).

Операция наплавки применяется в 50% случаев восстановления деталей [9]. Наиболее распространенными способами восстановления деталей цилиндрической формы являются автоматическая электродуговая наплавка под слоем флюса и вибродуговая наплавка.

Ремонт деталей цилиндрической формы осуществляется путем объединения их в партии [13]. Партии - это группы или наборы деталей, которые объединяются для выполнения определённых технологических операций, таких как ремонт, обработка или сборка. Объединение деталей в партии позволяет оптимизировать производственный процесс, улучшить управление качеством и повысить эффективность выполнения задач.

При включении деталей в партии, учитываются такие аспекты:

– Единый технологический способ устранения дефектов (различные дефекты устраняются одинаковым методом);

– Единые дефекты восстанавливаемых деталей (одинаковые дефекты цилиндрических деталей устраняются с применением одного технологического процесса - маршрутной технологии);

Для группирования деталей в партии используется маршрутная технология восстановления деталей с однотипными дефектами. Восстанавливаемые детали относятся к одному типу, если они имеют одинаковую форму и размеры, а также одинаковые виды дефектов, которые могут быть устранены одним технологическим процессом (время устранения дефектов этих деталей одинаковое). Соответственно, по маршрутной технологии осуществляется восстановление деталей разных типов, но с одинаковыми видами дефектов [11].

Количество типов восстанавливаемых деталей и число деталей каждого типа является достаточно большим. Это затрудняет использование методов частично целочисленного линейного программирования для планирования восстановления партий деталей разных типов, а также для оптимизации состава

этих партий и расписания их восстановления. При высокой размерности задачи, её решение невозможно получить с помощью известных методов. Поэтому возникает необходимость применения математических моделей выполнения пакетов заданий в многостадийных системах, а также численных методов оптимизации состава партий и расписания операций для планирования восстановления деталей разных типов [11].

Таким образом, при большом количестве типов деталей и деталей каждого типа задача планирования их восстановления в поточных системах специализированных ремонтных предприятий остается нерешенной. Применение математических моделей и численных методов оптимизации позволяет сократить общее время восстановления деталей разных типов и повысить эффективность работы оборудования за счет уменьшения его простоев.

Для планирования процессов на производствах часто используют различные ERP-системы. ERP (Enterprise Resource Planning) системы – это комплексные программные решения, которые интегрируют и автоматизируют ключевые бизнес-процессы компании. Данные системы эффективны в планировании и управлении ресурсами, координации деятельности, интегрировании и управлении различных отделов производства. [22]

Так как система, разрабатываемая в данной выпускной работе имеет непосредственное назначение в планировании и оптимизации, то она может относиться к ERP-системам. Для анализа рассмотрим некоторые функциональные аналоги систем данного вида на рынке.

Общей задачей ERP-систем является автоматизировать процессы, и вести и планировать полный учет бизнес-процессов компании.

Рассмотрим некоторые примеры ERP-систем:

1. “1С:ERP Управление предприятием”;
2. SAP ERP
3. Галактика ERP

1.2. Обзор решения “1С:ERP Управление предприятием”

1С:ERP Управление предприятием - система управления организацией, объединяющий учет, менеджмент операциями, анализ и планирование деятельности фирмы. Система обеспечивает автоматизацию ключевых функций компании, таких как учет товаров или услуг, управление закупками и продажами, менеджмент производства, финансовый учет и отчетность.

Главным преимуществом системы является умение унифицировано управлять всеми бизнес-процессами производства или компании. Данная система поможет также сократить расходы производства, так как может эффективно рассчитывать и планировать, вести учет и помогать в осуществлении контроля и управлении компанией. Система может легко интегрировать другие модули 1С. Также данная система может повысить эффективность и качество работы предприятия благодаря учету и планированию.

Из минусов у данной системы могут быть:

- Сложность внедрения. Система требует тщательной настройки и внедрения. Это может затратить большое количество усилий и времени, а также уйдет время на обучение персонала;
- Высокая стоимость;
- Сложность пользовательского интерфейса. Это также может увеличить затраты времени, денег и ресурсов на обучение персонала а также влияет на сложность внедрения;
- Недостаточная глубина аналитики. Этот фактор может сыграть важную роль в для более сложных и специфических потребностей предприятий. Дополнительные инструменты и настройки могут потребоваться для более детального анализа данных.
- Сложность в адаптировании. Специализированные предприятия имеют потребности в более адаптивных решениях, и данный фактор также влияет на количество затрат на разработку адаптивного решения.

Таким образом, рассмотрев плюсы и минусы системы, сделаем вывод, что

данная система не сможет удовлетворить потребности специализированного ремонтного предприятия на составление оптимизированного расписания, так как требуется большое количество затрат и ресурсов на работу с системой, а также система может иметь недостаточное количество функционала.

1.3. Обзор решения “SAP ERP”

SAP ERP (Enterprise Resource Planning) – это программное решение для управления бизнесом. Данная система предназначена для автоматизации, планирования и интеграции основных бизнес-процессов, что позволяет компаниям повышать эффективность, улучшать координацию и обеспечивать прозрачность всех операций.

Плюсами данной системы являются большой функционал поддерживаемых процессов в системе. Она может не только вести учет персонала, но и помогать планировать затраты ресурсов и улучшать эффективность производства. Система объединяет все основные бизнес-процессы компании. Система также имеет возможность планировать производственную мощность, помогая получить максимальную эффективность и выгоду при использовании оборудования и рабочей силы.

Минусы данной системы:

- Высокая стоимость. Локальные предприятия чаще всего не могут позволить себе дорогостоящее ПО.
- Сложность внедрения. Для внедрения скорее всего понадобится больше количество времени на анализ бизнес-процессов предприятия, а также большое количество ресурсов на настройку и внедрение системы в предприятия.
- Кроссплатформенность. Вероятнее всего, в предприятии могут возникнуть сложности с внедрением и взаимодействием данной системы с другими модулями.

Проанализировав данную систему, можно также сделать вывод, что она может не подойти под требования ремонтных специализированных предприятий,

для составления оптимизированного расписания восстановления. SAP мощная система, но она не учитывает особенности и детали планирования на ремонтных предприятиях.

1.4. Обзор решения “Галактика ERP”

Система управления предприятием "Галактика ERP" - это комплексное решение, разработанное отечественной компанией "Галактика". Ее целью является автоматизация разнообразных деловых процессов и поддержка разнообразных функциональных возможностей для различных сфер промышленности, коммерции и услуг. Система эффективна в планировании, так как поддерживает большой выбор возможностей. Например, она собирает аналитику, решает задачи оптимизации запасов и планирование производственных мощностей.

Плюсы системы:

- Широкий спектр поддерживаемых услуг. Система может помочь не только в автоматизации, но и в планировании и управлении;
- Система адаптивна под отечественные условия. Как и 1С, система разрабатывается на российском рынке и может учитывать особенности стандартов и норм.
- Гибкость. Система достаточно гибкая и адаптивная, что будет преимуществом при выборе системы на специализированном предприятии.

Минусами данной системы можно назвать также высокую стоимость продукта, так как она поддерживает большой функционал. Также как и предыдущие системы, данная система сложно внедряемая, и, следовательно, много затратна в ресурсах. Также система при обработке больших объемов данных работает медленно, что может влиять на скорость принятия решения.

1.5 Сравнительная характеристика проанализированных вариантов и постановка задачи проектирования

В процессе подробного рассмотрения трех существующих систем ERP, были выявлены их основные особенности, а также плюсы и минусы систем.

Особенностью системы 1С:ERP является хорошая совместимость с другими модулями 1С. Также система имеет широкий спектр услуг.

Особенностью системы SAP ERP являются мощные аналитические и алгоритмические способности, что непосредственно влияет на качество работы бизнес-процессов.

Галактика ERP также имеет свои особенности: гибкость, адаптивность под государственные нормы и большое количество инструментов.

Таблица 1.1. демонстрирует сравнительную характеристику рассмотренных систем ERP.

Таблица 1.1 - Сравнительная характеристика систем

Параметр	**1С: ERP**	**SAP ERP**	**Галактика ERP**
Основные возможности	Учет, управление бизнес-процессами, анализ и планирование	Всеобъемлющее управление бизнес-процессами, включая финансы, производство, планирование	Учет, управление производством, планирование цепочек поставок, финансов и продаж
Гибкость настройки	Средняя, есть возможность кастомизации под конкретные бизнес-процессы	Высокая, но требует значительных затрат на настройку и кастомизацию	Высокая, возможность адаптации под специфические требования
Сложность внедрения	Средняя, требует консультации специалистов	Высокая, потребуется большое количество времени и затрат	Средняя, могут потребоваться дополнительные модули

Параметр	**1С: ERP**	**SAP ERP**	**Галактика ERP**
Интеграция системы	Хорошая совместимость с другими продуктами 1С	Сложность интеграции с местными системами	Возможны сложности с внешней интеграцией, требует дополнительных настроек

Таким образом, проведенный анализ сравнительных характеристик ERP-систем (1С Управление предприятием, SAP ERP и Галактика ERP) выявил ряд дополнительных особенностей, которые могут повлиять на выбор подходящего решения компанией.

1С Управление предприятием предоставляет высокую гибкость в настройке и кастомизации под специфические бизнес-процессы, что особенно важно для отечественных компаний. Возможность использования различных конфигураций и модулей позволяет эффективно учитывать изменения в законодательстве и требованиях рынка.

SAP ERP известен своей широкой масштабируемостью и поддержкой международных стандартов, что делает его привлекательным выбором для крупных глобальных корпораций. Система обеспечивает интеграцию с различными внешними системами, что особенно важно для компаний с многофункциональной и мультинациональной структурой.

Галактика ERP, помимо полной адаптации под российское законодательство, обеспечивает удобство в использовании и низкую стоимость владения. Эта система ориентирована на средний и крупный бизнес, предлагая комплексные решения для управления производственными процессами, цепочками поставок и финансовыми операциями.

Так как мы рассматриваем создание системы для специализированного ремонтно-восстановительного предприятия, важно учесть особенности.

Особенностями требуемой системы является то, что предприятие

обрабатывает цилиндрические формы валов, и требует учета ресурсов отталкиваясь от вида детали. Также стоит учесть, что после восстановления детали складываются в партии, которые удобно обрабатывать на многостадийных предприятиях. Для построения пакетов в составлении расписания также необходимо знать время выполнения операций, рассмотренные системы не могут предоставить данный функционал расчетов времени восстановления и переналадок.

Исходя из сравнительной характеристики, можем сделать вывод, что ни одна из рассмотренных систем не учитывает особенности предприятия - не строит пакеты при создании расписания, не учитывают цилиндрическую форму агрегатов и узлов, не может учитывать время выполнения восстановительных работ. Следовательно, необходимо разработать такую ERP - систему, которая будет учитывать все особенности специализированного ремонтного предприятия.

Выводы по разделу 1

В данной главе был проведен анализ предметной области, в котором был выявлен основной минимальный функционал системы.

Далее был проведен сравнительный анализ особенностей и характеристик ERP-систем на рынке - 1С: ERP, SAP ERP, Галактика ERP. В ходе проведения анализа были выявлены основные особенности, характеристики и свойства систем в планировании, и с помощью сравнения и анализа было выявлено, что системы не имеют необходимого функционала для специализированных ремонтных предприятий. Рассмотрение плюсов и минусов систем окончательно выявило необходимость в разработке системы оптимизации расписаний восстановления узлов и агрегатов технологического и транспортного оборудования на специализированных ремонтных предприятиях.

2 СИСТЕМОТЕХНИЧЕСКИЙ АНАЛИЗ И ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ ХРАНЕНИЯ И ВВОДА ДАННЫХ ОПТИМИЗИРОВАННОГО РАСПИСАНИЯ

2.1 Построение диаграммы потоков данных для описания процессов взаимодействия пользователя с системой

Подсистема ввода и хранения данных является частью общей системы оптимизации расписания процессов восстановления узлов и агрегатов технологического и транспортного оборудования на специализированных ремонтных предприятиях.

Работа с подсистемой осуществляется непосредственно пользователем, путем записи данных о деталях, а также составлением наборов деталей для передачи данных о технологических процессах подсистеме рещения MILP, оптимизирующей расписание.

Подсистема решения MILP в свою очередь также взаимодействует с данной подсистемой, предоставляя оптимизированное расписание восстановления транспортных узлов и агрегатов, которое в дальнейшем будет отформатировано подсистемой ввода и хранения данных и отображено пользователю.

Таким образом, можно выделить две внешние сущности, взаимодействующие с подсистемой – «Пользователь» и «Подсистема MILP».

Рассмотрим потоки данных. Важно выделить, что входными данными от пользователя будут информация о деталях: тип детали, изначальный диаметр, износ и размеры повреждений. Далее для дальнейшей работы с подсистемой от пользователя поступает запрос на составление расписания восстановления набора деталей. В данном запросе пользователем формируется необходимый набор деталей, и далее система рассчитывает необходимые данные для его составления.

Выходными потоками данных из подсистемы являются результаты: данная подсистема передает подсистеме решения MILP результаты проведенных ею

вычислений времени выполнения восстановления детали для набора, определенного пользователем. После того, как подсистема смешанного целочисленного линейного программирования предоставит результаты оптимизированного расписания, подсистема формирует и отображает результаты пользователю.

На рисунке 2.1 продемонстрирована описанная выше диаграмма потоков данных.

Рисунок 2.1 - Основная DFD-диаграмма

Процесс «система ввода и хранения данных» можно декомпозировать. Данная система непосредственно работает с чтением и записью в базе данных (БД). Опишем процессы обмена информации с базой данных в подсистеме.

Анализ входных данных подразумевает ввод данных пользователем и запрос на выполнение записи в БД. При поступлении от пользователя запроса на составление оптимизированного расписания, выполняется процесс формирования набора деталей для расписания. Соответственно, для составления набора выполняется выборка из существующих данных в БД и запрос в базу данных на чтение. Далее, из базы данных поступает ответ о свойствах деталей в составляемом наборе, и выполняется расчет общего и вспомогательного времени выполнения восстановления узлов и агрегатов транспортного оборудования.

От подсистемы решения MILP посредством файла поступают данные об оптимизированном расписании восстановления, которое подвергается интерпретации для дальнейшей записи в БД и последующем форматировании и отображении данных пользователю. Рисунок 2.2 демонстрирует декомпозированный процесс «система ввода и хранения данных».

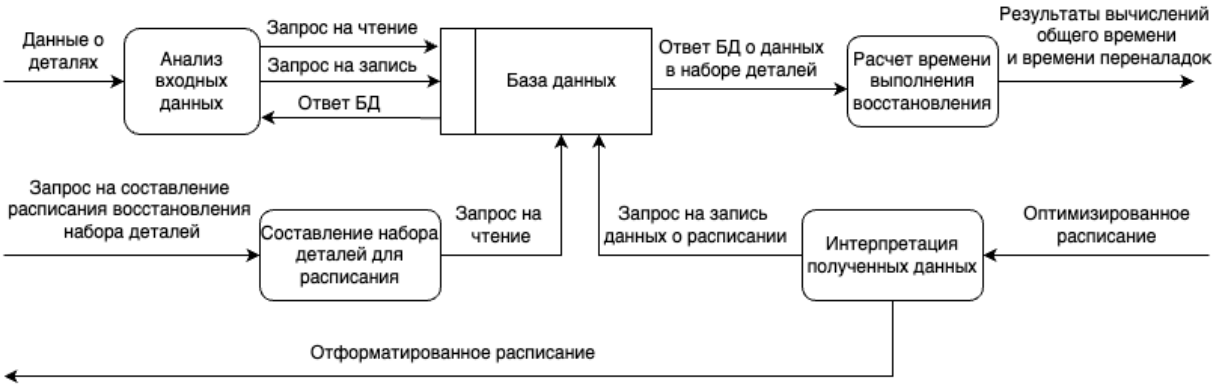


Рисунок 2.2 - Декомпозиция процесса «Система ввода и хранения данных»

2.2 Разработка функциональных и информационных моделей IDEF0-IDEF1 системы ввода и хранения данных

Для построения функциональной модели используется IDEF0. С помощью нее можно продемонстрировать бизнес-процессы разрабатываемой системы.

Рисунок 2.3 демонстрирует основную функцию - «Система оптимизации расписания восстановления узлов и агрегатов транспортного оборудования на специализированных предприятиях».

На вход данной системы поступают запросы: на создание оптимизированного расписания, ввод и хранения данных о деталях, запрос на интерпретацию и отображение оптимизированного расписания. Инструментами выполнения запроса являются различные формы, формулы и шаблоны: форма ввода данных и создания заявки, формулы расчетов данных времени выполнения, шаблон интерпретации оптимизированного расписания. Взаимодействовать с данным процессом могут как система так и пользователь. В итоге, на выходе результатом будут данные о времени восстановления узлов и агрегатов и содержании ПЗ, а также интерпретированное оптимизированное расписания выполнения ремонтных работ транспортных узлов и агрегатов.

Рисунок 2.3 - Диаграмма основного процесса системы в нотации IDEF0

Основной процесс можно декомпозировать. На рисунке 2.4 продемонстрирована декомпозиция основного процесса A0 разрабатываемой системы

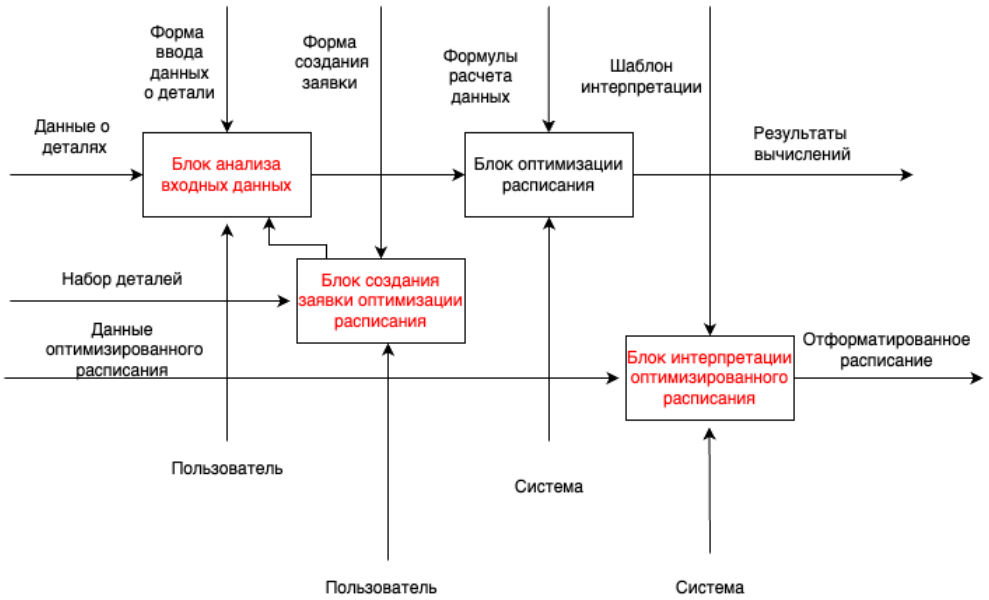


Рисунок 2.4 - Декомпозиции основного бизнес-процесса A0 в нотации IDEF0

В данной выпускной работе будут рассматриваться только три блока - блок анализа ввода данных, блок создания заявки оптимизации расписания и блок интерпретации оптимизированного расписания. Блок оптимизации расписания относится к подсистеме решения задач целочисленного программирования MILP.

Декомпозиция включает в себя:

1. Функциональный блок «A1» - Блок анализа входных данных. Данный блок подразумевает ввод пользователям входных данных о детали: тип детали, износ, изначальный диаметр и размер участков повреждения детали. Декомпозиция данного процесса представлена на рисунке 2.5.

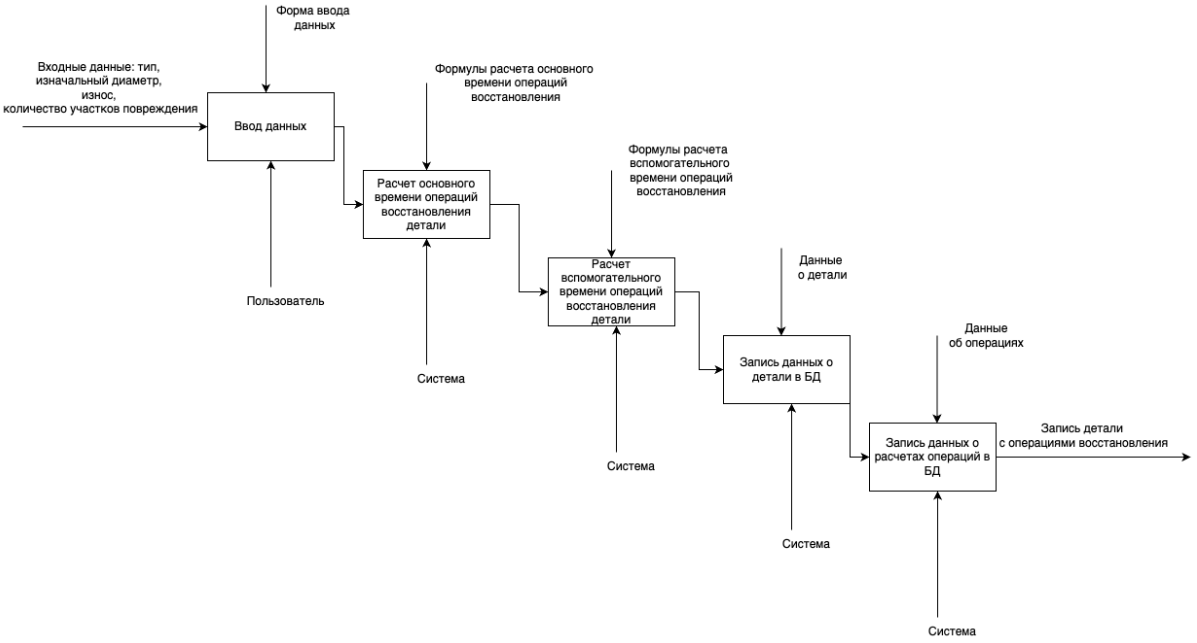


Рисунок 2.5 - Декомпозиция блока анализа входных данных

2. Функциональный блок «A2» - Блок создания заявки системе оптимизации расписания набора деталей. В этом блоке подозревается, что пользователь формирует набор деталей, подлежащих восстановлению. Декомпозиция данного процесса представлена на рисунке 2.6.

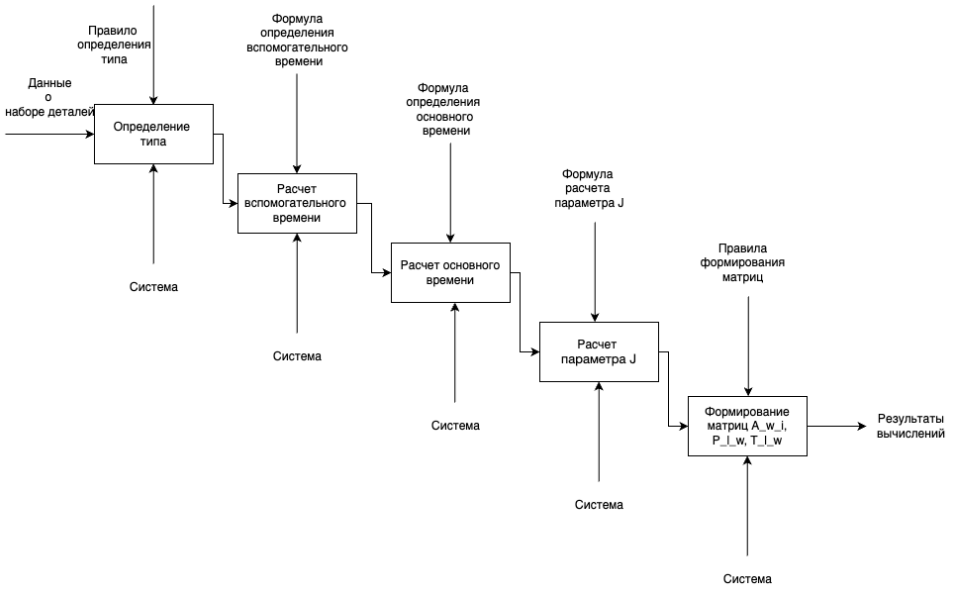


Рисунок 2.6 - Декомпозиция блока создания заявки

3. Функциональный блок «А3» - «Блок интерпретации оптимизированного расписания». Входными данными для данного блока является оптимизированное системой расписание в виде файла. Декомпозиция данного процесса представлена на рисунке 2.7.

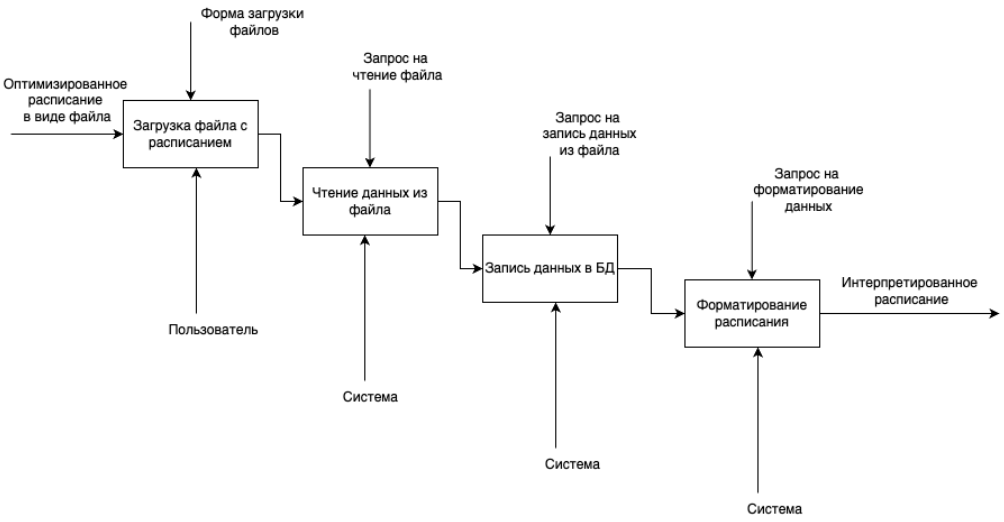


Рисунок 2.7 - Декомпозиция блока интерпретирования оптимизированного расписания.

В итоге была построена и декомпозирована диаграмма IDEF0, которая в дальнейшем будет использоваться при проектировании подсистемы.

Для определения ключевых сущностей системы, для дальнейшего проектирования. Таблица 2.1 содержит перечень потенциальных сущностей системы.

Таблица 2.1 - Потенциальные сущности системы

№	Название сущности	Описание
1	Деталь	Содержит основные данные о детали в системе
2	Заявка	Содержит основную информацию о наборе деталей, требуемых к восстановлению
3	Операция	Содержит основную информацию о расчетах времени восстановления на различных этапах
4	Технологический процесс	Содержит информацию о времени процесса восстановления для одной детали на различных этапах
5	Расписание	Содержит данные оптимизированного расписания, получаемого от подсистемы решения MILP

Список, представленный в таблице 2.1 помог выявить основные сущности и их атрибуты. Таблица 2.2 демонстрирует разделение сущностей и их атрибутов.

Таблица 2.2 - Атрибуты сущностей

№	Название сущности	Описание
1	Деталь	id_детали, тип, название, износ, участки_износа, начальный_диаметр
2	Заявка	id_заявки, id_деталей, количество_деталей, количество_приборов, матрица_основного_времени_восстановления, матрица_переналадок, матрица_соответствия_задания_типу_задания, J_parameter
3	Операция	id_детали, id_операции, тип_операции, основное время, вспомогательное_время
4	Технологический процесс	id_процесса, id_деталей, массив_операций
5	Расписание	id_расписания, данные_расписания

Также было составлено описание предметной области на естественном языке

1. Каждая заявка (сущность 2) <может> <иметь> <много> деталей (сущность 1).
2. Каждая деталь (сущность 1) <может> <содержать> <много> операций (сущность 3).
3. Каждый технологический процесс (сущность 4) может <может> <содержать> <много> операций (сущность 3).
4. Каждое расписание (сущность 5) может состоять из множества технологических процессов (сущность 6).
5. Каждый технологический процесс (сущность 4) может <может> <содержать> <одну> деталь (сущность 1).

Далее были определены имена и связи сущностей. Также заданы мощности связей между сущностями. Результаты представлены в таблице 2.3.

Таблица 2.3 - Матрица отношений между сущностями

	Деталь	Заказ	Операция	Тех. процесс	Расписание
Деталь		M:N	1:N	1:1	M:1
Заказ	N:M		N:M	1:M	1:1
Операция	N:1	M:N		M:1	M:1
Тех. процесс	1:1	M:1	1:N		M:1
Расписание	M:1	1:1	M:1	M:1	

Таблица 2.3 демонстрирует основные сущности и их ключевые атрибуты.

Таблица 2.3 - Ключевые атрибуты сущностей

№	Название сущности	Описание
1	Деталь	id_детали, тип, название, износ, участки_износа, начальный_диаметр
2	Заявка	id_заявки, id_деталей, количество_деталей, количество_приборов, матрица_основного_времени_восстановления, матрица_переналадок, матрица_соответствия_задания_типу_задания, J_parameter
3	Операция	id_детали, id_операции, тип_операции, основное время, вспомогательное_время
4	Технологический процесс	id_процесса, id_деталей, массив_операций
5	Расписание	id_расписания, данные_расписания

Исходя из всех данных выше требуется создать информационную модель уровня «сущность-связь» – ER-диаграмму в нотации П.Чена. Рисунок 2.8 демонстрирует диаграмму.

Опишем, какие связи имеют описанные сущности.

Сущность “Заявка” состоит из деталей и имеет связь многие-ко-многим с сущностью “Детали”. Деталь же содержит операции восстановления, соответственно сущность “Деталь” имеет связь один-ко-многим с сущностью “Операции”. Технологический процесс содержит множество операций соответствующей детали, соответственно сущность “Технологический процесс” имеет связь один-ко-многим с сущностью “Операции” и связь один-к-одному. Далее рассмотрим сущность “Расписание”. Данная сущность может хранить несколько технологических процессов, следовательно, она имеет связь один-ко-многим с сущностью “Технологический процесс”.

Далее, необходимо построить логическую модель данных, основанную на ключах. Исходя из составленных таблиц выше, можем проанализировать, что первичным ключом для каждой таблицы был выбран id - уникальный идентификатор записи в таблице.

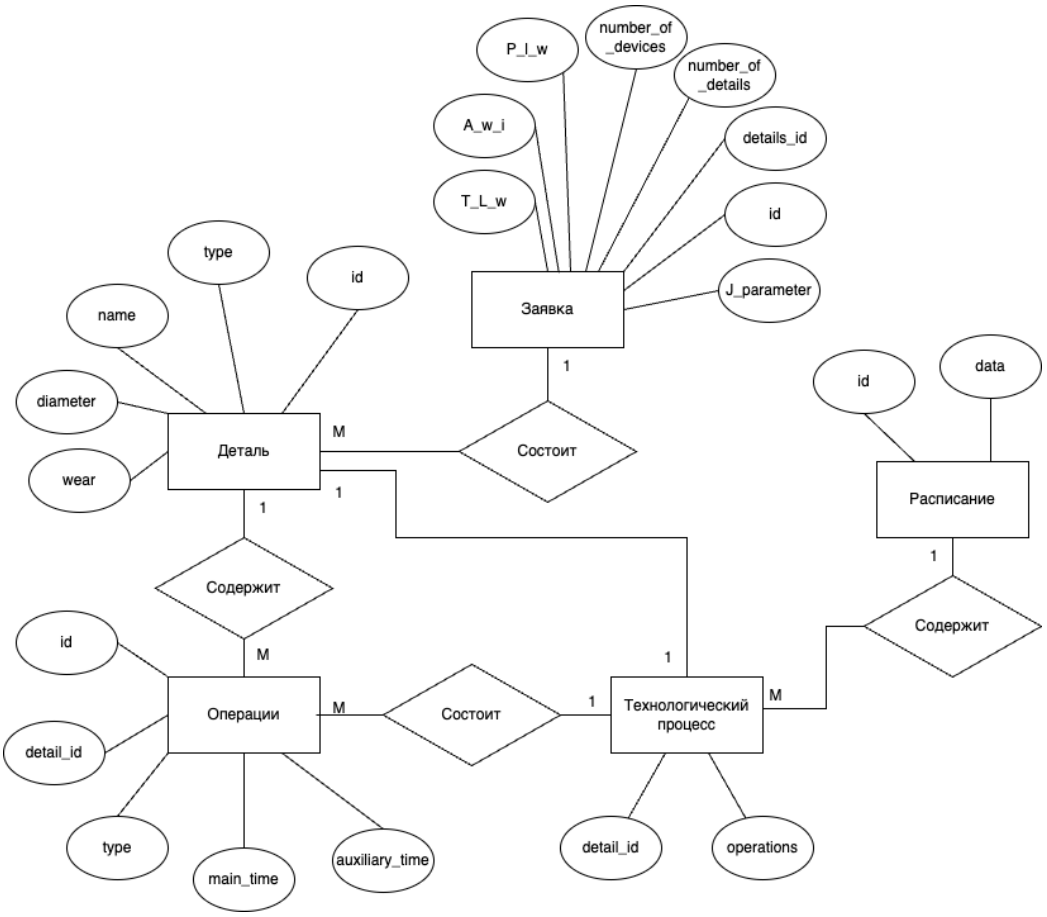


Рисунок 2.8 - ER-диаграмма в нотации П.Чена

Также из таблиц видно, что в системе есть связь многие-ко-многим между сущностями “Деталь” и “Заявка”. Для разбиения данной связи была введена таблица деталь-заявка, в которой в качестве атрибутов будут id заявки и id детали.

В итоге, каждая таблица имеет связь один-ко-многим, либо же один-к-одному. Рисунок 2.9 содержит диаграмму на ключах.

Далее сущности были дополнены не ключевыми атрибутами в соответствии с требованиями к разработке и данными из таблиц 2.2 и 2.3. Результаты полной атрибутивной модели нотации IDEF1X представлены на рисунке 2.10.

В результате было проведено информационное и функциональное моделирование подсистемы ввода и хранения данных в системе оптимизации расписания восстановления узлов и агрегатов транспортного оборудования на специализированных ремонтных предприятиях.

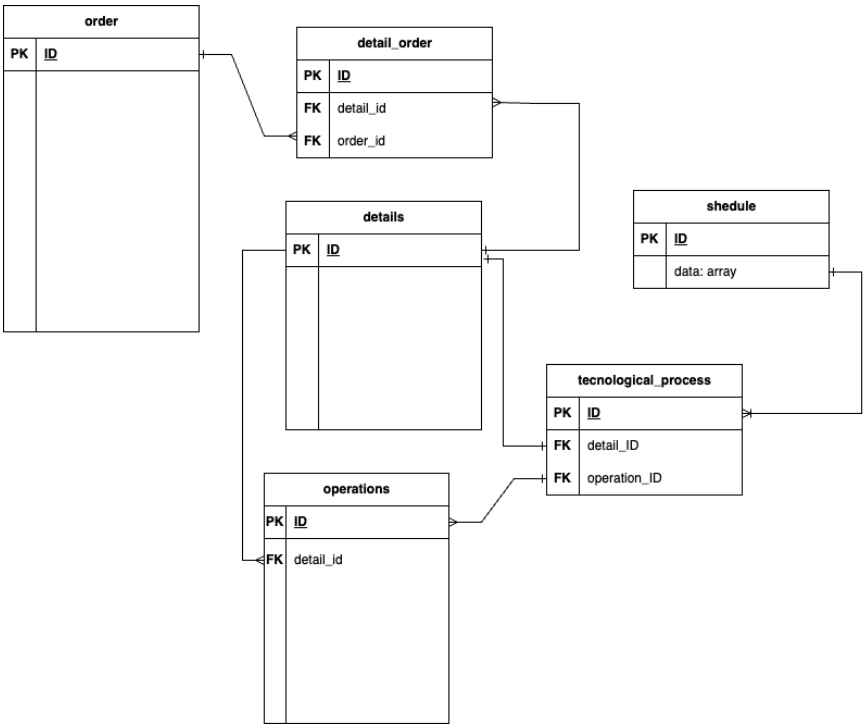


Рисунок 2.9 - Диаграмма на ключах

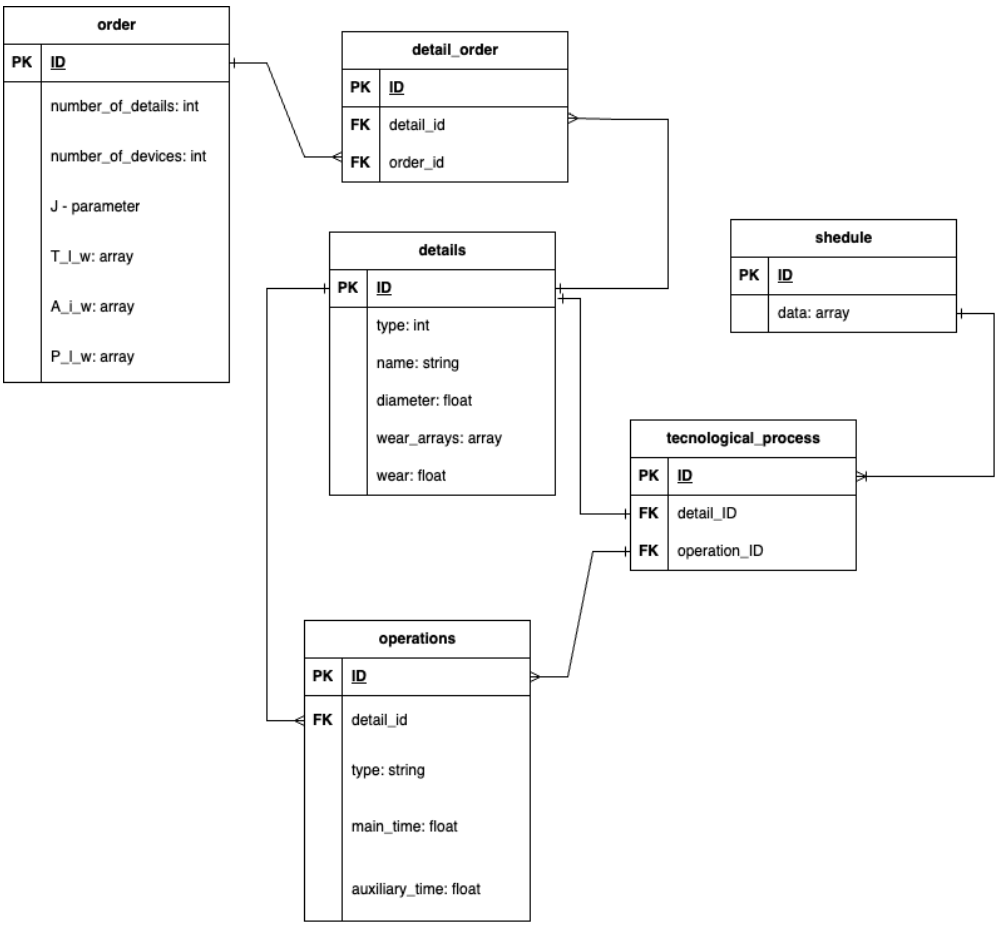


Рисунок 2.10 - Полная атрибутивная модель нотации IDEF1X

2.3 Разработка обобщенной архитектуры и алгоритма функционирования системы

Для разработки обобщенной архитектуры необходимо спроектировать упрощенный алгоритм системы. С помощью диаграммы нотации BPMN и таблиц действий можно описать приблизительный алгоритм системы, и в последующем выбрать архитектуру и инструменты проектирования.

Составим таблицу 2.4, отображающую основной перечень задач, действий и участников.

Таблица 2.4. - Основной перечень задач, действий и участников

№	Задача	Список действий	Участник	Объекты данных
1	Создание детали	Заполнение формы ввода информации о детали, а именно: название, тип детали, количество поврежденных участков, износ, выполнение расчета времени детали	Пользователь, Система	БД Детали, БД Операции
2	Просмотр существующих деталей	Открытие списка всех деталей	Пользователь	БД Детали
3	Создание заявки	Заполнение формы ввода заявки, выполнение расчетов матриц	Пользователь, система	БД Заявок, БД Операций, БД Детали
4	Просмотр существующих заявок	Открытие списка всех заявок	Пользователь	БД Заявок
5	Просмотр расписания	Открытие списка всех расписаний	Пользователь	БД Расписаний
	Загрузка расписания	Загрузка файла через форму, интерпретация данных	Пользователь, Система	БД Расписаний, БД Технологических операций

На основании данной таблицы была построена упрощенная модель бизнес-процессов, представленная на рисунке 2.11.

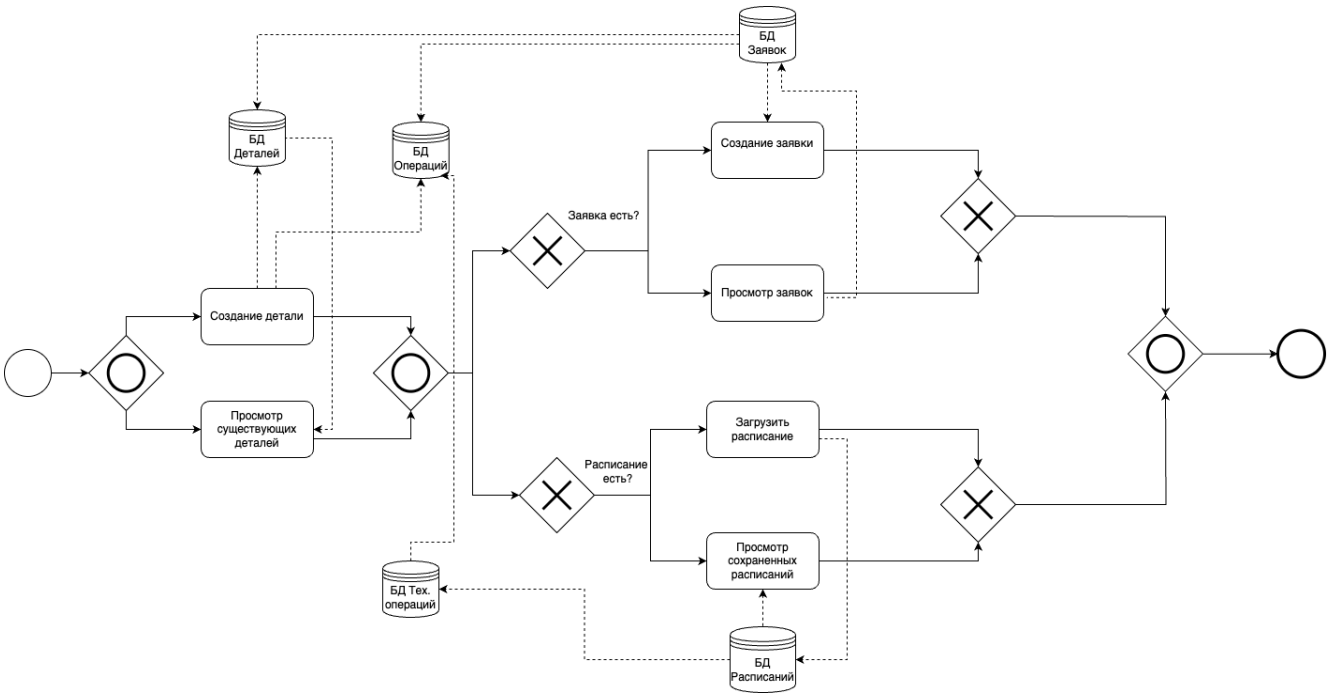


Рисунок 2.11 - Упрощенная модель бизнес-процесса

По аналогии с декомпозицией IDEF0 система может принимать на вход данные о деталях для создания записи в БД, принимать на вход данные о заявках для создания записи в БД и формировании расписания, принимать на вход данные в виде файла расписания для интерпретации и отображения информации пользователю.

Следовательно, исходя из декомпозированного процесса можно выбрать архитектуру. В данном случае актуально будет использовать архитектуру “клиент-сервер”, так как все взаимодействие планируется через пользователя, а ответ будет приходит с сервера.

Плюсами данной архитектуры можно выделить:

- Распределение задач. Распределив задачи между сервером и клиентом мы позволим каждому объекту выполнять четко свою задачу. Таким образом, сервер будет централизованно выполнять свои задачи, так как он хранит все данные.

- Масштабируемость. Легкость в масштабируемости данной архитектуры позволяет без проблем увеличить серверы вне зависимости от клиентов, и поддерживать большое количество пользователей, обрабатывая большое количество данных.
- Безопасность. Хранение данных на сервере облегчает поддержку безопасности данных, так как есть большое количество вариантов защиты серверов.

Задача клиентской части состоит в отображении интерфейса, обработке приходящих данных и отправки запросов. Отображение интерфейса в зависимости от выбранных инструментов может быть реализовано с помощью HTML, CSS, и JavaScript. Также для ускорения разработки можно использовать фреймворки или шаблонизаторы.

Задача серверной части состоит в обработке запросов, управлении данными и отправке ответов. Управление данными подразумевает работу с базой данных и формирование бизнес-логики.

База данных в свою очередь необходима для хранения, записи и чтения данных. Обычно сервер выполняет запросы, и получает данные в ответ.

Реализация серверной части чаще всего выполняется на PHP. Также можно использовать различные фреймворки, помогающие разработать систему быстрее.

Работа с базой данных выполняется с помощью языка запросов SQL. Чаще всего в разработке используют СУБД PostgreSQL, так как это мощная и многофункциональная СУБД, соответствующая стандартам SQL.

Таким образом, была выбрана архитектура будущей подсистемы, а также частично выбраны инструменты разработки.

2.4. Разработка структурных данных подсистемы

При анализе аналогов ERP-систем были поставлены изначальные требования к системе - необходимо строить партии для работы с расписанием, учитывать время восстановления работ с цилиндрической формой. Так как ни

одна система не готова предоставить необходимый функционал, была поставлена задача разработать систему, отвечающая требованиям.

Так как системе предстоит формировать технологические процессы, необходимо рассчитывать режимы обработки деталей и нормы времени выполнения операций на приборах.

Определим, что типы деталей будут обозначены через $i = (\overline{1, n})$, а индексы приборов через l - так как в технологическую цепочку восстановления деталей цилиндрического типа входят 4 операции восстановления, то $l = (\overline{1, 4})$.

Выше было определены 4 операции восстановления:

- механическая обработка;
- вибродуговая или электродуговая наплавка;
- токарная обработка;
- шлифование;

Также стоит учесть, что для деталей первого и второго типов восстанавливаются с использованием электродуговой наплавки, детали третьего и четвертого типов восстанавливаются с использованием вибродуговой наплавки.

К параметрам операций предварительной механической (токарной) обработки, выполняемой на первом (*l=1*) приборе многостадийной (поточной) системы (продольное точение), относятся [9, 13, 20]:

– длина обрабатываемой поверхности детали *i*-го типа с учетом врезания и перебега $L_i\,(i = \overline{1, n})$;

– длина обрабатываемой поверхности детали *i*-го типа по чертежу $l_i\,(i = \overline{1, n})$;

– величина врезания и перебега для детали *i*-го типа $y_i(i = \overline{1, n})$;

− диаметры деталей *i*-ых типов перед токарной обработкой (продольное точение) $D^1{}_i\,(i = \overline{1, n})$, диаметры деталей *i*-ых типов после токарной обработки $d^1{}_i\,(i = \overline{1, n})$;

− припуск на обработку деталей *i*-ых типов на первом приборе $h^1_i (i = \overline{1, n})$ (мм);

− глубина резания для деталей *i*-ых типов на первом приборе $g^1_i (i = \overline{1, n})$ (мм);

– количество проходов для снятия припуска на обработку h^1_i при глубине резания g^1_i для деталей *i*-ых типов $k^1_i (i = \overline{1, n})$;

– подача режущего инструмента для деталей *i*-ых типов $s^1_i (i = \overline{1, n})$ (мм/об);

– скорость резания для деталей *i*-ых типов $v^1_i (i = \overline{1, n})$ (м/ч);

– число оборотов деталей *i*-ых типа в минуту $n^1_i (i = \overline{1, n})$ (об/мин);

– длительность основной токарной обработки деталей *i*-ых типов при продольном точении $t^{o1}_i (i = \overline{1, n})$ (мин);

– длительность вспомогательных операций токарной обработки деталей при продольном точении $t^{в1}_i (i = \overline{1, n})$ (мин).

Значение l^i является заданным (определяется по чертежу восстанавливаемой детали), значение y_i определяется по таблице [9], значение L^i определяется выражением вида:

$$L^i = l^i + y^i;. \quad (2.4.1)$$

Припуск на обработку h^i определяется для заданных значений D^1_i и d^1_i по формуле [9]:

$$h^1_i = \frac{D^1_i - d^1_i}{2}. \quad (2.4.2)$$

Значение g^1_i определяется для вычисленного значения h^1_i и заданного количества проходов $k^1_i (i = \overline{1, n})$: $g^1_i = h^1_a / k^1_i$. Подача режущего инструмента за один оборот детали *i*-го типа $s^1_i (i = \overline{1, n})$, а также скорость резания $v^1_i (i = \overline{1, n})$ для деталей *i*-го типа определяются по таблицам [8, 9]. Количество оборотов $n^1_i (i = \overline{1, n})$ детали *i*-го типа определяется выражением [8, 9]:

$$n^1_i = 318 \frac{v^1_i}{D^1_i}, \quad (2.4.3)$$

где D^1_i – диаметр детали, мм. В соответствии с полученными значениями основное время t^{o1}_i токарной обработки деталей (при продольном точении) на первом приборе определяется выражением вида [8]:

$$t^{o1}_i = \frac{L_i * k^1_i}{n^1_i * s^1_i}. \quad (2.4.4)$$

Вспомогательное время $t^{в1}_i$, связанное с основным временем t^{o1}_i обработки детали *i*-го типа, определяется по таблицам [9]. В соответствии со значениями t^{o1}_i и $t^{в1}_i$ суммарная длительность выполнения операции предварительной токарной обработки детали *i*-го типа на первом приборе перед реализацией операции наплавки определяется следующим образом:

$$t^1_i = t^{o1}_i + t^{в1}_i. \quad (2.4.5)$$

В качестве способов восстановления формы и размеров цилиндрических деталей рассмотрена электродуговая наплавка и вибродуговая наплавка. Для параметров процесса электродуговой наплавки детали *i*-го типа введены следующие обозначения (наплавка выполняется на приборе с номером *l=2,* все параметры проиндексированы номером этого прибора):

– сварочный ток для наплавки детали *i*-го типа на *(l=2)*-ом приборе $I^{\text{св2}}_{i}(i = \overline{1,n})$ (А);

– диаметр проволоки, используемой в наплавочной головке для реализации восстановления детали *i*-го типа на втором приборе с использованием электродуговой наплавки $d^{\text{пр}}_{\text{эi}}(i = \overline{1,n})$ (мм);

– количество проходов при электродуговой наплавке детали *i*-го типа $k^{2}_{\text{эi}}(i = \overline{1,n})$;

– коэффициент наплавки детали *i*-го типа $k^{2}_{\text{нi}}(i = \overline{1,n})$ (г/(А*ч));

– площадь поперечного сечения наплавленного валика *F* (см^2);

– плотность металла проволоки, используемой при наплавке детали *i*-го типа $\rho_{ni}(i = \overline{1,n})$ – плотность наплавляемого шва (г/см^3);

– скорость восстановления детали *i*-го типа с использованием электродуговой наплавки (скорость наплавки) $v^{2}_{\text{эi}}(i = \overline{1,n})$ (м/ч);

– количество оборотов детали при реализации наплавки $n^{2}_{\text{эi}}$ (об/мин);

– шаг наплавки детали *i*-го типа $s^{2}_{\text{эi}}(i = \overline{1,n})$ (мм/об);

– длительность основной операции по восстановлению детали электродуговой наплавкой на втором приборе $t^{o2}_{\text{эi}}(i = \overline{1,n})$ (основное время выполнения операции, мин);

– длительность вспомогательных операций по восстановлению детали электродуговой наплавкой на втором приборе $t^{\text{в2}}_{\text{эi}}(i = \overline{1,n})$ (мин).

Величина тока $I^{\text{св2}}_{i}$ при электродуговой наплавке определяется выражением вида: $I^{\text{св2}}_{i} = 40 * \sqrt[3]{D_i}$, где D_i – диаметр восстанавливаемой детали. Значение $K^{2}_{\text{нi}}$ определяется выражением вида [9, 20]:

$$K^2_{\text{н}i} = 2,3 + 0,065\frac{I^{\text{св2}\ 2}_i}{d^{\text{пр2}}_{\text{э}i}}. \quad (2.4.6)$$

Скорость электродуговой наплавки $v^2_{\text{э}i}$ детали *i*-го типа на (*l=2*)-м приборе определяется в соответствии с выражением вида [9, 20]:

$$v^2_{\text{э}i} = \frac{K^2_{\text{н}i} * I^{\text{св2}}_i}{100 * F * \rho_{ni}}. \quad (2.4.7)$$

где F = 0.06 - 0.2 см^2 при $d^{\text{пр}}_{\text{э}i} = 1.2 - 2.0$ мм. В качестве значения плотности наплавочной проволоки ρ_{ni}, используемой при наплавке детали *i*-го типа, принята плотность металла электрода УОНИ 13/55 (низкоуглеродистая и низколегированная сталь марки СВ-08Г, 7,81 г/см3.). Частота вращения детали (об/мин) определяется выражением вида [9, 20]:

$$n^2_{\text{э}i} = \frac{1000v^2_{\text{э}i}}{60 * \pi * D_i}. \quad (2.4.8)$$

Шаг наплавки детали *i*-го типа $s^2_{\text{э}i}(\overline{1,n})$ (мм/об) в соответствии с [9, 20] определяется выражением вида: $s^2_{\text{э}i} = (2..2.5)d^{\text{пр}}_{\text{э}i\ ё}$. Значение основного времени выполнения операции электродуговой наплавки определяется выражением [9, 20]:

$$t^{o2}_{\text{э}i} = \frac{t^2_{\text{э}i} * L_i}{n^2_{\text{э}i} * s^2_{\text{э}i}}. \quad (2.4.9)$$

В соответствии с [9, 20]: вспомогательное время $t^{\text{в2}}_{\text{э}i}$ определяется как 15% от основного времени электродуговой наплавки $t^{o2}_{\text{э}i}$. В тоже время подготовительно-заключительное время, связанное с установкой оснастки и подготовкой ее к работе, составляет 16 мин [9, 20].

Для параметров процесса вибродуговой наплавки детали *i*-го типа введены следующие обозначения (наплавка выполняется на приборе с номером *l=2,* все параметры проиндексированы номером прибора):

– диаметр проволоки, используемой в наплавочной готовке для реализации восстановления детали *i*-го типа с использованием вибродуговой наплавки $d^{\text{пр}}_{\text{в}i}(i = \overline{1,n})$ (мм);

– сварочный ток для вибродуговой наплавки детали *i*-го типа $I^{\text{св}2}_{i}(i = \overline{1,n})$ (А), напряжение вибродуговой наплавки детали *i*-го типа $U_{i}(i = \overline{1,n})$;

– скорость подачи проволоки при вибродуговой наплавке детали *i*-го типа v^{2}_{ni} (м/ч);

– скорость вибродуговой наплавки детали *i*-го типа $v^{2}_{\text{вн}i}$;

– количество оборотов детали при реализации наплавки $n^{2}_{\text{в}i}$ (об/мин);

– коэффициент перехода электродного материала в наплавляемый металл η (η = 0.8 ÷ 0.9) [13, 20];

– шаг наплавки детали *i*-го типа $s^{2}_{\text{в}i}(i = \overline{1,n})$ (мм/об);

– толщина наплавляемого слоя при вибродуговой наплавке детали *i*-го типа $h^{2}_{\text{в}i}(i = \overline{1,n})$ (мм);

– количество проходов при реализации наплавки $k^{2}_{\text{в}i}(i = \overline{1,n})$;

– коэффициент, учитывающий отклонение фактической толщины наплавляемого слоя от требуемой – *a* (*a* = 0.7 ÷ 0.85) [13, 20];

– длительность основной операции по восстановлению детали вибродуговой наплавкой $t^{o2}_{\text{в}i}(i = \overline{1,n})$ (основное время выполнения операции, мин);

– длительность вспомогательных операций по восстановлению детали вибродуговой наплавкой $t^{\text{в}2}_{\text{в}i}(i = \overline{1,n})$ (мин).

Величина тока $I^{\text{св2}}_{i}$ при вибродуговой наплавке определяется выражением вида: $I^{\text{св2}}_{i} = (60..75)d^{\text{пр}}_{\text{в}i}$ [13, 20]. Величина подачи проволоки при вибродуговой наплавке v_{ni} определяется выражением вида [13, 20]:

$$v_{ni} = \frac{0.1 * I^{\text{св2}}_{i} * U^{i}}{(d^{\text{пр}}_{\text{в}i})^{2}}. \quad (2.4.10)$$

Шаг наплавки ($s^{2}_{\text{в}i}(i = \overline{1,n})$) определяется выражением вида:

$s^{2}_{\text{в}i}(1.6..2.2)d^{\text{пр}}_{\text{в}i}$ [13, 20]. Скорость вибродуговой наплавки $v^{2}_{\text{вн}i}$ детали *i*-го типа на (*l=2*)-м приборе определяется в соответствии с выражением вида [13, 20]:

$$v^{2}_{\text{вн}i} = \frac{0.785\,(d^{\text{пр}}_{\text{в}i})^{2} * v_{ni} * \eta}{h^{2}_{\text{в}i} * s^{2}_{\text{в}i} * 2}. \quad (2.4.11)$$

Частота вращения детали (число оборотов, об/мин) определяется выражением [13, 20]:

$$n^{2}_{\text{в}i} = \frac{1000 v^{2}_{\text{э}i}}{60 * \pi * D_{i}}. \quad (2.4.12)$$

Значение основного времени выполнения операции вибродуговой наплавки определяется выражением:

$$t^{o2}_{\text{в}i} = \frac{k^{2}_{\text{в}i} * L_{i}}{n^{2}_{\text{в}i} * s^{2}_{\text{в}i}}. \quad (2.4.13)$$

Если толщина наплавляемого слоя $h^{2}_{\text{н}i}$ составляет 1.0-1.5 мм, то $d^{\text{пр}}_{\text{в}i}$ = 2.0 мм, а $U_{i} = 15 \div 20$ В [13, 20]. Если толщина наплавляемого слоя $h^{2}_{\text{в}i}$ составляет 2.0-2.5 мм, то $d^{\text{пр}}_{\text{в}i}$ = 2.5 мм, а $U_{i} = 20 \div 25$ В [13, 20].

Если дефекты, связанные с изменением формы и размеров цилиндрических деталей, обнаружены на их периферии, то области восстановления также находятся на периферии деталей. Из-за этого при проведении наплавки образуется слой металла, который требует выравнивания путем точения краев. Таким

образом, механическая обработка деталей после наплавки состоит из двух этапов: 1) выравнивание наплавленного слоя с помощью продольного токарного обработки; 2) выравнивание краев деталей с помощью торцевого токарного обработки. В результате механической обработки деталей на третьем станке после наплавки выполняются как продольное точение, так и удаление излишков металла с краев деталей [11].

Обозначения параметров операции токарной обработки, которая выполняется на третьем инструменте в многостадийной системе (продольное точение), после реализации наплавки (первая операция на третьем инструменте), аналогичны обозначениям параметров для предварительной обработки с изменением индекса l третьего инструмента (l=3):

– длина обрабатываемой поверхности детали *i*-го типа с учетом врезания и перебега L_{i1} ($i = \overline{1, n}$);

– длина обрабатываемой поверхности детали *i*-го типа по чертежу

l_{i1} $(i = \overline{1, n})$;

– величина врезания и перебега для детали *i*-го типа $y_{i1}(i = \overline{1, n})$;

– требуемые диаметры деталей *i*-ых типов (продольное точение)

$D^3_i(i = \overline{1, n})$, диаметры деталей *i*-ых типов перед токарной обработкой $d^3_i(i = \overline{1, n})$;

− припуск на обработку деталей *i*-ых типов на третьем приборе

$h^3_i(i = \overline{1, n})$ (мм);

– глубина резания для деталей *i*-ых типов $g^3_{i1}(i = \overline{1, n})$ (мм);

– количество проходов для снятия припуска на обработку h^3_i при глубине резания g^3_i для деталей *i*-ых типов $k^1_{i1}(i = \overline{1, n})$;

– подача режущего инструмента для деталей *i*-ых типов s^{3}_{i1} $(i = \overline{1, n})$ (мм/об);

– скорость резания для деталей *i*-ых типов v^{3}_{i1} $(i = \overline{1, n})$ (м/ч);

– число оборотов детали *i*-го типа в минуту n^{3}_{i1} $(i = \overline{1, n})$ (об/мин);

– длительность основной токарной обработки деталей при продольном точении на третьем приборе t^{o3}_{i1} $(i = \overline{1, n})$(мин);

– длительность вспомогательных операций токарной обработки деталей при продольном точении на третьем приборе $t^{в3}_{i1}$ $(i = \overline{1, n})$(мин).

Определение значений введенных параметров осуществляется аналогично способам вычисления значений этих же параметров для первого прибора, рассмотренным выше.

В качестве параметров процесса торцевого точения (вторая операция на третьем приборе) рассматриваются:

– длина обрабатываемой поверхности детали *i*-го типа при торцевом точении с учетом врезания и перебега L_{i2} $(i = \overline{1, n})$;

– длина обрабатываемой поверхности детали *i*-го типа при торцевом точении l_{i2} $(i = \overline{1, n})$;

– величина врезания и перебега для детали *i*-го типа при торцевом точении y_{i2} $(i = \overline{1, n})$;

– глубина резания для деталей *i*-ых типов g^{3}_{i2} $(i = \overline{1, n})$ (мм);

– количество проходов при глубине резания g^{3}_{i2} для деталей *i*-ых типов k^{3}_{i2};

– подача инструмента для деталей *i*-ых типов s^{3}_{i2} $(i = \overline{1, n})$ (мм/об);

– скорость резания для деталей *i*-ых типов v^{3}_{i2} $(i = \overline{1, n})$ (м/ч);

– число оборотов детали *i*-го типа n^{3}_{i2} $(i = \overline{1, n})$ (об/мин);

– длительность основной токарной обработки деталей при торцевом точении t^{o3}_{i2} $(i = \overline{1, n})$ (мин);

– длительность вспомогательных операций токарной обработки деталей при торцевом точении $t^{в3}_{i2}$ $(i = \overline{1, n})$ (мин).

Величина l_{i2} определяется как $l_{i2} = D^{1}_{i} / 2$, значение y_{i2} определяется по таблице [8], тогда значение L_{i2} определяется выражением вида:

$$L_{i2} = l_{i1} + y_{i2}. \quad (2.4.14)$$

Глубина резания g^{3}_{i2} и количество проходов k^{3}_{i2} $(i = \overline{1, n})$ являются заданными входными параметрами. В соответствии со значением g^{3}_{i2} подача режущего инструмента за один оборот детали *i*-го типа s^{3}_{i2} $(i = \overline{1, n})$, а также скорость резания v^{3}_{i2} $(i = \overline{1, n})$ для деталей *i*-го типа определяются по таблицам [8]. Количество оборотов n^{3}_{i2} $(i = \overline{1, n})$ детали *i*-го типа определяется выражением [13, 20]:

$$n^{1}_{i} = 318 \frac{v^{3}_{i2}}{D^{1}_{i}}, \quad (2.4.15)$$

где D_{i} – диаметр детали *i*-го типа, мм. В соответствии с полученными значениями основное время t^{o3}_{i2} торцевой токарной обработки деталей на первом приборе определяется выражением вида [9]:

$$t^{o3}_{i2} = \frac{L_{i2} * k^{3}_{i2}}{n^{3}_{i2} * s^{3}_{i2}}. \quad (2.4.16)$$

Вспомогательное время $t^{в3}_{i2}$, связанное с основным временем t^{o3}_{i2} обработки детали *i*-го типа, определяется по таблице [8]. В соответствии со

значениями $t^{в3}_{i1}$, $t^{о3}_{i1}$, $t^{в3}_{i2}$, $t^{о3}_{i2}$ суммарная длительность выполнения операции токарной обработки детали *i*-го типа на третьем приборе после реализацией наплавки детали определяется следующим образом:

$$t^{в3}_{i2} = t^{о3}_{i1} + t^{в3}_{i1} + t^{в3}_{i2} + t^{о3}_{i2}. \quad (2.4.17)$$

Технологическая операция, выполняемая на (*l=4*)-м приборе предусматривает шлифование восстановленной детали после ее механической обработки. Обозначения параметров операции шлифования деталей, выполняемой на четвертом (*l=4*) приборе в многостадийной (поточной) системы после реализации токарной обработки, имеют следующий вид:

– диаметры деталей *i*-ых типов, подвергающиеся шлифовальной обработке $D^4_i (i = \overline{1, n})$;

– длина обрабатываемой поверхности детали *i*-го типа при продольном точении с учетом врезания и перебега $L_{i4} (i = \overline{1, n})$;

– длина обрабатываемой поверхности детали *i*-го типа при шлифовании по чертежу $l_{i4} (i = \overline{1, n})$;

– величина врезания и перебега для детали *i*-го типа при шлифовании $y_{i4}(i = \overline{1, n})$;

– поперечная подача инструмента (шлифовального круга) для деталей *i*-ых типов (глубина шлифования) – $s^{n4}_i (i = \overline{1, n})$ (мм/об);

– количество проходов при глубине шлифования s^{n4}_i для деталей *i*-ых типов $k^4_i (i = \overline{1, n})$;

– продольная подача инструмента (шлифовального круга) для деталей *i*-ых типов $s^{пр4}_i (i = \overline{1, n})$ (мм/об);

– скорость шлифования деталей *i*-ых типов $v^4_i (i = \overline{1, n})$(м/ч);

– число оборотов детали *i*-го типа в минуту n^4_i $(i = \overline{1, n})$ (об/мин);

– коэффициент зачистных ходов K_4 (принимает значения 1.2-1.7 в зависимости от требуемой чистоты обработки) [9];

– длительность основной операции шлифования детали *i*-го типа t^{o4}_i ($i = \overline{1, n}$) (мин);

– длительность вспомогательных операций при шлифовании детали *i*-го типа– $t^{в4}_i$ $(i = \overline{1, n})$ (мин);

– общая длительность выполнения операции шлифования восстановленной детали (при черновой и чистовой обработке) – t^4_i $(i = \overline{1, n})$ (мин);

Значения параметров s^{n4}_i и $s^{пр4}_i$, а также v^4_i определяются по таблицам [8]. Значения параметра n^4_i определяются в соответствии с выражениями вида [9]:

$$n^1_i = 318 \frac{v^4_i}{D^4_i}, \tag{2.4.18}$$

Основное время шлифования t^{o4}_i деталей *i*-ых типов $(i = \overline{1, n})$ определяются выражением вида [9]:

$$t^{o4}_i = \frac{L_i * k^4_i}{n^4_i * s^{пр4}_i} * K_э. \tag{2.4.19}$$

Время вспомогательных операций при шлифовании деталей *i*-ых типов ($t^{в4}_i$) определяется по таблицам [8].

Общее время выполнения операций шлифования с деталями *i*-ых типов определяется следующим образом:

$$t^4_i = t^{o4}_i + t^{в4}_i. \tag{2.4.20}$$

Рассмотренные в работе математические модели процесса выполнения ПЗ, математической модели иерархической игры оптимизации

составов ПЗ и расписаний их выполнения, методы оптимизации составов ПЗ и расписаний, реализованные в виде соответствующего комплекса программ, применены для решения задач планирования процесса восстановления деталей цилиндрической формы в составе партий. Для реализации восстановления формы и размеров деталей разных типов применены электродуговая и вибродуговая наплавки.

Технологический процесс восстановления включает этап предварительной механической обработки (продольная токарная обработка на первом приборе), этап непосредственно наплавки (на втором приборе), этап последующей механической обработки (продольная и торцевая токарные обработки на третьем приборе) и этап шлифования (на четвертом приборе).

Выводы по разделу 2

В ходе выполнения информационного и функционального моделирования системы системы оптимизации расписаний восстановления узлов и агрегатов технологического и транспортного оборудования на специализированных ремонтных предприятиях и ее подсистемы, ввода и хранения данных, были построены и декомпозированы различные диаграммы.

Были выделены внешние сущности подсистемы ввода и хранения – пользователи и подсистема MILP, были также выделены сценарии их взаимодействия с разрабатываемой подсистемой. Также была представлена информация, передаваемая между компонентами в процессе, и последовательность событий, которые происходят в системе при действиях пользователя.

Соответствующе были составлены диаграммы разных нотаций - DFD, IDEF0-IDEF1X, ER-диаграмма нотации Чена, BPMN.

DFD-диаграмма помогла выделить основные внешние сущности и потоки данных. Как было указано, внешними сущностями в подсистеме являются подсистема решения MILP и пользователь.

С помощью IDEF0 были описаны процессы взаимодействия пользователя с системой в целом, также процессы были декомпозированы. IDEF1X и ER-диаграмма нотации Чена отразила описание связей между компонентами и их ключевыми и не ключевыми атрибутами. Для описания бизнес-процессов была использована диаграмма нотации BPMN.

Также была описана архитектура приложения, и описаны возможности выбранной архитектуры “клиент - сервер”.

В одном из разделов была описана структурных данных системы, которые можно использовать для расчетов в дальнейшем.

В результате, после проектирования системы в данном разделе, можно приступать к разработке.

3 РАЗРАБОТКА ПОДСИСТЕМЫ ВВОДА И ХРАНЕНИЯ ДАННЫХ

3.1 Обоснование выбора средств программной реализации системы ввода и хранения данных

Алгоритм ввода и хранения данных будет реализован с помощью фреймворка для создания веб-приложений Laravel.

Laravel – фреймворк для веб-разработки, созданный на языке программирования PHP. Laravel предназначен для упрощения задач, связанных с созданием веб-приложений, и предлагает множество встроенных инструментов и функций. Фреймворк также имеет активную поддержку, и имеет удобную работу с объектно-реляционным отображением (ORM) Eloquent, что позволяет работать с базами данных с помощью моделей PHP, и упрощает выполнение CRUD операций и взаимодействие с реляционными данными. Также фреймворк содержит встроенный шаблонизатор Laravel, который предлагает легковесный, но мощный способ создания динамических HTML-шаблонов с использованием логики на PHP. [5]

Для выполнения операций с базой данных (БД) в работе был выбран PostgreSQL. Это мощная, объектно-реляционная система управления базами данных (СУБД) с открытым исходным кодом, которая соответствует стандартам SQL и поддерживает большое количество функций – расширяемость, сложные запросы и транзакции. Также имеет активное и большое сообщество поддержки с большим количеством документации. [16]

3.2 Разработка функциональной схемы подсистемы ввода и хранения данных системы оптимизации расписания восстановления узлов и агрегатов

Разработка функциональной схемы является ключевым этапом проектирования системы.

Процесс оптимизации расписания начинается с ввода данных о деталях. Далее, в процессе расчета данных они формируются в технологические процессы и в последующем могут формироваться в заявку. В процессе формирования заявки формируются матрицы, которые в последующем будут использованы при оптимизации расписания.

После того как система MILP передала результаты, в блоке загрузки результатов можно загрузить оптимизированное расписание для интерпретации. В последующем, блоки расчета данных, формирования заявки и загрузки результатов расписания отображают полученные результаты вычислений, и результаты работы реализованного решения представлены на рисунке 3.2.1.

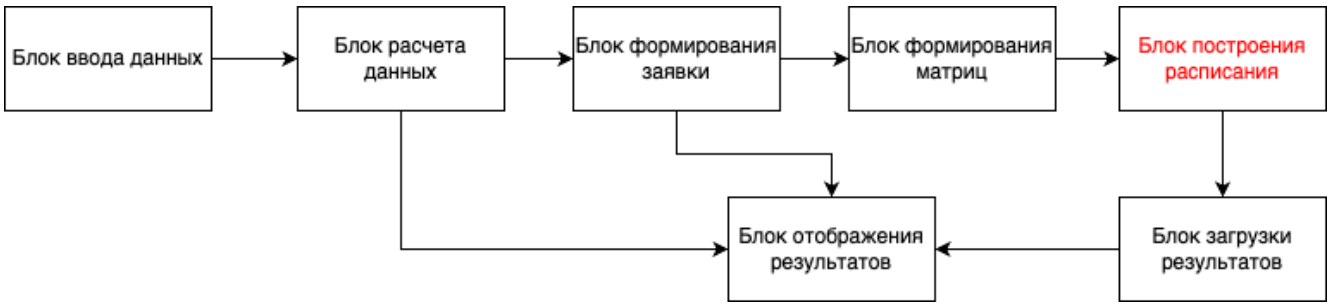


Рисунок 3.2.1 - Структурно-функциональная схема системы

В соответствии с построенной структурно-функциональной схемой системы можно выделить следующие этапы:

- Разработка форм ввода данных;
- Обеспечение хранения данных;
- Обработка и анализ данных для построения решения;
- Разработка форм отображения результатов.

3.3. Выбор и обоснование языка программирования

В качестве языка программирования был выбран язык PHP. Фреймворк Laravel, выбранный ранее, работает на языке программирования PHP, реализация интерфейса реализована с помощью встроенного шаблонизатора Blade.

PHP (Hypertext Preprocessor) – современный, кроссплатформенный, интерпретируемый скриптовый язык программирования с открытым исходным кодом. Данный язык программирования имеет большое количество библиотек, активно поддерживается сообществом и используется для разработки статических и динамических сайтов и веб-приложений. [15]

В качестве инструмента для реализации интерфейса был выбран встроенный в Laravel шаблонизатор Blade. Это простой, но мощный движок шаблонов, входящий в состав Laravel. Все шаблоны Blade компилируются в обычный PHP-код и кэшируются до тех пор, пока не будут изменены, что добавляет фактически нулевую нагрузку приложения. Данный шаблонизатор поддерживает язык разметки HTML, написание сценариев на Javascript а также поддержка стилей с помощью CSS [14]. Для улучшения UI использовалась библиотека Bootstrap.

HTML – гипертекстовый язык разметки. Данный язык программирования используется для создания различных гипертекстовых документов, которые можно просматривать с помощью любого типа веб-браузеров. [14]

CSS – каскадная таблица стилей, которая позволяет управлять размером и стилем шрифтов, таблиц, позиционированием элементов и многими другими параметрами отображения HTML – документов [14].

JavaScript – это высокоуровневый, мультипарадигменный, функциональный часто компилируемый в режиме реального времени язык, соответствующий стандарту ECMAScript. Используется для создания различных скриптов и сценариев, которые встраиваются в HTML-документы или включаются в них и взаимодействуют с DOM. [4]

Bootstrap – открытый HTML и CSS фреймворк, который используется для ускорения верстки адаптивного дизайна. Включает в себя CSS и HTML-шаблоны оформления для веб-форм, меток, типографики, кнопок, блоков навигации и других компонентов веб-интерфейса. [18]

Composer - это инструмент для управления зависимостями в разработке на PHP. Он позволяет легко устанавливать, обновлять и контролировать библиотеки, которые используются в проекте. Composer работает с файлами composer.json, где указаны все необходимые зависимости и их версии.

В качестве IDE, для разработки была выбрана PhpStorm от JetBrains. PHPStorm предлагает обширный набор инструментов, которые делают процесс разработки эффективным и удобным Это мощная интегрированная среда разработки для PHP.

Преимущества данной IDE:

- Поддержка различных языков: PHP, HTML, CSS, JavaScript и другие языки веб-разработки.
- Интеграция с системами контроля версий: Поддержка Git и других систем;
- Инструменты для работы с базами данных: Поддержка различных СУБД, включая MySQL, PostgreSQL, SQLite.
- Встроенный терминал и управление задачами: Встроенный терминал и интеграция с инструментами управления задачами, такими как Composer и npm.
- Поддержка современных фреймворков и CMS: Laravel, Symfony, Drupal, WordPress и другие.

Проанализировав, можно сделать вывод, что данная IDE является идеальной для работы с Laravel и PHP.

Помимо написания кода важно вести контроль над изменениями проекта. С этим может помочь Git - система контроля версий, позволяющая отследить где, когда и кем были произведены изменения. Помимо отслеживания изменений можно выполнять большое количество операций с данными в ветках. Система выстроена в виде дерева, и в процессе работы над проектом используются ветки.

Github - это платформа, которая позволяет удобно работать с Git в веб и десктоп режиме. Она предоставляет возможность работы с репозиториями и совместной работой, что является немаловажным при разработке проектов.

3.4 Описание программных модулей

После проектирования и разработки структуры основных данных определим порядок модулей, которые необходимо реализовать в системе. Исходя из всех данных, всего будет шесть модулей:

- Создание записи о детали;
- Просмотр данных о всех деталях;
- Создание заявки на оптимизацию расписания;
- Каталог созданных заявок;
- Загрузка файла с расписанием для интерпретации;
- Просмотр всех загруженных расписаний.

Далее, на основе перечисленных модулей были созданы подразделы. Подразделы были оформлены в виде меню, и отображаются на каждой странице. Для удобства навигации была разработана страница “Главная”, которая также содержит список модулей приложения. Листинг 1 отображает содержание меню на каждой странице. Код главной страницы представлен более подробно в приложении А.

Листинг 1- Меню веб-приложения

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
   <div class="container-fluid">
       <div class="collapse navbar-collapse" id="navbarNav">
           <ul class="navbar-nav">
                <li class="nav-item"><a class="nav-link" href="{{ route('main.index')
}}">Главная</a></li>
                                <li  class="nav-item"><a  class="nav-link"  href="{{
route('detail.allDetails') }}">Справочник деталей</a></li>
                                <li  class="nav-item"><a  class="nav-link"  href="{{
route('detail.create') }}">Добавить деталь</a></li>
                                <li  class="nav-item"><a  class="nav-link"  href="{{
route('order.create') }}">Создать заявку</a></li>
                                <li  class="nav-item"><a  class="nav-link"  href="{{
route('order.index') }}">Все заявки</a></li>
```

Продолжение листинга 1 - Меню веб-приложения

```
                                <li  class="nav-item"><a  class="nav-link"  href="{{
route('upload.form') }}">Загрузить расписание</a></li>
                    <li class="nav-item"><a class="nav-link" href="{{ route('data.list')
}}">Загруженные расписания</a></li>
            </ul>
        </div>
    </div>
</nav>
```

3.4.1. - Разработка программного модуля “Деталь”

Так как вся информация идет изначально от деталей, первым спроектированным модулем был модуль “Деталь”. Так как Laravel поддерживает MVC-паттерн, то у сущности есть модель Detail, контроллер Detail Controller и отображение ввода информации о детали.

При сохранении детали в БД в системе сразу же идет расчет времени выполнения каждой операции восстановления, и эти данные также сохраняются в БД. Листинг 2 показывает контроллер с выполнением логики модели. Контроллер имеет только один метод `store(Request $request)`. Более полный код представлен в приложении А.

Листинг 2 - Detail Controller

```
public function store(Request $request)
{
// валидация входных данных
    $validatedData = $request->validate([
        'name' => 'required|string|max:255',
        'initialDiameter' => 'required|numeric',
        'wearsCount' => 'required|integer|min:1',
        'wear_areas' => 'required|array|min:1',
        'wear_areas.*' => 'numeric',
        'type' => 'required|integer'
    ]);

    $wear_areas = $validatedData['wear_areas'];
// проверка на массив
    if (!is_array($wear_areas)) {
        $wears_area = json_decode($wear_areas, true);
    }
//создание детали
    $detail = Detail::create([
        'name' => $validatedData['name'],
```

Продолжение листинга 2 - Detail Controller

```
        'diameter' => $validatedData['initialDiameter'],
        'wear_areas' => $wear_areas,
        'type' => $validatedData['type'],
        'wear' => $request->wear,
    ]);

// логирование
    Log::info($validatedData);

// создание и расчет операций
    $operationsData = [
        'turning' => $detail->turningTime()
    ];

//определение типа детали для выбора операции
    if ($detail->type == 1 || $detail->type == 2) {
        $operationsData['electricityWelding'] = $detail->electricityWelding();
    } else {
        $operationsData['vibroWelding'] = $detail->vibroWelding();
    }

//слияние всех данных в массив операций
    $operationsData = array_merge($operationsData, [
        'secondTurning' => $detail->secondTurningTime(),
        'grinding' => $detail->grinding(),
    ]);

//создание записи операции для каждого типа из созданного массива
    foreach ($operationsData as $type => $time) {
        $main_time = $time['main_time'] ?? 0;
        $auxiliary_time = $time['auxiliary_time'] ?? 0;
        $operation = Operation::create([
            'detail_id' => $detail->id,
            'type' => $type,
            'main_time' => $main_time,
            'auxiliary_time' => $auxiliary_time,
        ]);

        $operations[] = $operation->toArray(); // Добавление операции в список
    }

    // Создание технологического процесса
    TechnologicalProcess::create([
        'detail_id' => $detail->id,
        'operations' => json_encode($operations), // Сохранение списка операций в
формате JSON
    ]);

    // Возвращаем ответ или выполняем другие действия

    return redirect()->route('detail.create')->with('success', 'Detail created
successfully with operations calculated.');
}
```

Далее обратимся к методам в модели. В соответствии со структурой данных, необходимо рассчитать время для 4-х операций восстановления. Листинг 3 представляет методы расчета времени выполнения восстановления по операции

токарных работ. Листинг 4, 5 содержит полный код с комментариями вспомогательных методов, которые рассчитывают данные длины обрабатываемой поверхности и диаметр детали после обработки.

Листинг 3 - Функция turningTime()

```
public function turningTime(
   float $yi = 3.5,  // Величина врезания и перебега для детали
   int $ki = 1,      // Количество проходов для снятия припуска при глубине резания
   float $s1i = 0.5, // Подача режущего инструмента
   float $v1i = 50,  // Скорость резания
   float $tv1i = 3   // Длительность вспомогательных операций токарной обработки
): array
{
   $this->li = array_sum($this->wear_areas); // Длина обрабатываемой поверхности по
чертежу
   $Li = $this->lengthDetail($this->li, $yi); // Длина обрабатываемой поверхности
детали с учетом врезания и перебега

   $D1i = $this->diameter; // Диаметр детали перед токарной обработкой
   $this->calculateNewDiameter($D1i, $this->wear_areas);
   $d1i =$this->damageDiameter; // Диаметр детали после токарной обработки

   // Припуск на обработку
   $h1i = ($D1i - $d1i) / 2;

   // Глубина резания
   $g1i = $h1i / $ki;

   // Число оборотов деталей в минуту
   $n1i = 318 * ($v1i / $D1i);

   // Длительность основной токарной обработки
   $to1i = ($Li * $ki) / ($n1i * $s1i);

   // Суммарная длительность выполнения операции предварительной токарной обработки
   $t1i = $to1i + $tv1i;

   // Возврат суммы основной и вспомогательной длительности обработки
   return [
       'main_time' => ceil($t1i),
       'auxiliary_time' => ceil($tv1i),
   ];
}
```

Листинг 4 - Функция расчета длины детали

```
private function lengthDetail(
   float $li, // Длина обрабатываемой поверхности по чертежу (обработка поврежденной
поверхности)
   float $yi // Величина врезания и перебега для детали
): float {
   return $li + $yi;
}
```

Листинг 5 - Функция расчета диаметра после обработки

```
   private function calculateNewDiameter(
   float $initialDiameter, // Начальный диаметр
   array $wears // Массив повреждений
) {
   // Рассчитываем новый диаметр после токарной обработки
   $totalWear = array_sum($wears) / count($wears);
   $this->damageDiameter = $initialDiameter - 2 * $totalWear;
}
```

Листинг 6 отображает выполнение операций вибродуговой наплавки и электродуговой наплавки. Выбор данной операции осуществляется в зависимости от типа детали.

Листинг 6 - Метод расчета электродуговой наплавки

```
public function electricityWelding(
   float $k2i = 1 // Количество проходов при наплавке
): array {
   $density = 7.81; // Плотность проволоки (г/см³)
   $lengthDetail = $this->li; // Длина обрабатываемой поверхности детали с учетом
врезания и перебега
   $dpri = $this->diameter - $this->damageDiameter; // Диаметр проволоки (мм)
   // Сварочный ток (А)
   $isv2i = 40 * pow($this->diameter, 1/3);

   // Коэффициент наплавки (г/(А*ч))
   $k2ni = 2.3 + 0.065 * pow($isv2i, 2) / $dpri;

   // Площадь поперечного сечения наплавленного валика (см²)
   $F = 0.06; // Допустим фиксированное значение для dпрэi = 1.2 - 2.0 мм

   // Скорость наплавки (м/ч)
   $v2ei = ($k2ni * $isv2i) / (100 * $F * $density);

   // Частота вращения детали (об/мин)
   $n2i = (1000 * $v2ei) / (60 * pi() * $this->diameter);

   // Шаг наплавки детали (мм/об)
   $s2i = $this->stepWelding($dpri, 2, 2.5);
   // Длительность основной операции по восстановлению детали электродуговой
наплавкой (мин)
   $to2i = ($k2i * $lengthDetail) / ($n2i * $s2i);

   // Вспомогательное время (15% от основного времени)
   $tv2i = $to2i * 0.15;
   // Подготовительно-заключительное время (мин)
   $preparatoryTime = 16;

   return [
       'main_time' => ceil($to2i + $tv2i),
       'auxiliary_time' => ceil($tv2i),
   ];

}
```

Также необходимо рассчитать шаг наплавки для функции наплавки. Листинг 7 демонстрирует написанный код.

Листинг 7 - Метод вычисления шага наплавки

```
private function stepWelding(float $dpri, float $minFactor, float $maxFactor): float
{
   $factor = mt_rand($minFactor * 100, $maxFactor * 100) / 100; // Случайный
коэффициент в диапазоне от minFactor до maxFactor
   return $factor * $dpri;
}
```

Рассмотрим функцию вибродуговой наплавки. Листинг 8 демонстрирует функцию расчета времени восстановления методом вибродуговой наплавки. Также листинги 9, 10 показывают дополнительные методы расчета напряжения и сварочного тока, которые необходимы при вибродуговой наплавке.

Листинг 8 - Метод вибродуговой наплавки

```
public function vibroWelding(
   float $k2i = 2, // Количество проходов при наплавке
   float $h2vi = 3 // Толщина наплавляемого слоя
): array {
   $nu = 0.8 / 0.9; // Коэффициент перехода
   $alpha = 0.7 / 0.85; // Коэффициент отклонения толщины
   $dpri = $this->diameter - $this->damageDiameter; // Диаметр проволки
   $ui = $this->voltage($h2vi); // Напряжение
   $isv2i = $this->electricity($dpri); // Сварочный ток
   $s2i = $this->stepWelding($dpri, 1.6, 2.2); // Шаг наплавки
   $vni = (0.1 * $isv2i * $ui * pow($dpri, 2)); // Величина подачи проволки
   $v2vi = (0.785 * pow($dpri, 2) * $vni * $nu) / ($h2vi * $s2i * $alpha); //
Скорость наплавки
   $n2i = (1000 * $v2vi) / (60 * pi() * $this->diameter); // Частота вращения детали
   $to2i = ($k2i * $this->li) / ($n2i * $s2i); // Основное время
   $tv2i = ($to2i * 15) / 100; // Вспомогательное время
   return [
       'main_time' => ceil($to2i + $tv2i),
       'auxiliary_time' => $tv2i
   ];
}
```

Листинг 9 - Расчет сварочного тока

```
private function electricity(float $dpri): float {
   $factor = mt_rand(60 * 100, 75 * 100) / 100;
   return $factor * $dpri;
}
```

Листинг 10 - Расчет напряжения

```
private function voltage(float $h2vi): float {
   $ui = 0.0;
   if ($h2vi >= 1.0 && $h2vi <= 1.5) {
       $ui = 15 / 20;
   } elseif ($h2vi >= 2.0 && $h2vi <= 2.5) {
       $ui = 20 / 25;
   } else {
       $ui =  30 / 40;
   }

   return $ui;
}
```

Далее, следующим этапом восстановления будет торцевая и продольная токарная обработка. Так как операция одна, но несколько процессов, она разбита на несколько функций. В соответствии со структурой данных, продольное точение соответствует первой операции, но продольное будет отличаться. Листинг 11 отображает всю операцию токарного восстановления включая и продольную и торцевую обработку и возвращение результатов.

Листинг 11 - Расчет времени торцевой и продольной токарной обработки

```
public function secondTurningTime(): array {
   $turningTime = $this->turningTime();
   $faceTurning = $this->faceTurning();

   $main_time_total = $turningTime['main_time'] + $faceTurning['main_time'];
   $auxiliary_time_total = $turningTime['auxiliary_time'] + $faceTurn
ing['auxiliary_time'];

   return [
       'main_time' => $main_time_total,
       'auxiliary_time' => $auxiliary_time_total,
   ];
}

private  function faceTurning(
   int $ki = 1,      // Количество проходов для снятия припуска при глубине резания
   float $s1i = 0.5, // Подача режущего инструмента
   float $v1i = 50,  // Скорость резания
   float $tv2i = 3,  // Длительность вспомогательных операций токарной обработки
   float $yi = 3.5  // Величина врезания и перебега для детали
   ): array
{
   $li2 = $this->diameter / 2;
   $Li2 = $li2 / + $yi;

   $n1i = 318 * ($v1i / $this->diameter); // Число оборотов деталей в минуту

   // Расчет длительности основной токарной обработки
   $to1i = ($Li2 * $ki) / ($n1i * $s1i);
```

Продолжение листинга 11 - Расчет времени торцевой и продольной токарной обработки

```
    return [
        'main_time' => ceil($to1i + $tv2i),
        'auxiliary_time' => ceil($tv2i),
    ];
}
```

Окончательным этапом завершения реализации операций является шлифование. Листинг 12 демонстрирует реализацию расчета времени шлифовки.

Листинг 12 - Метод расчета времени шлифовки

```
public function grinding(
    float $snp4i = 0.4 // Продольная подача инструментов
): array {
    $ki4 = rand(4, 10); // Количество проходов
    $v4i = rand(400, 600) / 100; // Скорость шлифования деталей
    $n4i = 318 * ($v4i / $this->diameter); // число оборотов детали
    $k = rand(1.2, 1.7) / 100; // коэффициент зачистных ходов

    // Вспомогательное время
    $tv4i = rand(270, 340) / 100;
    // Основное время
    $to4i = (($this->li * $ki4) / ($n4i * $snp4i)) * $k;

    return [
        'main_time' => ceil($to4i + $tv4i),
        'auxiliary_time' => ceil($tv4i)
    ];
}
```

Также важно отметить, что у модели Details есть связи с таблицами “Операции” как один-ко-многим, и таблицей “Технологический процесс” как один-к-одному. В Листинге 13 продемонстрированы все связи модели Деталь с таблицами.

Листинг 13 - Связи модели “Деталь”

```
protected $table = 'details'; // основная таблица

protected $fillable = [
    'type', 'name', 'diameter', 'wear_areas', 'wear',
]; // поля в таблице

protected $casts = [
    'wear_areas' => 'array',
]; // каст данных в массив

// связь с таблицей “Операции”
public function operations()
{
```

Продолжение листинга 13 - Связи модели “Деталь”

```
    return $this->hasMany(Operation::class, 'detail_id');
}

//связь с таблицей “Технологический процесс”
public function technologicalProcess()
{
    return $this->hasOne(TechnologicalProcess::class);
}
```

3.4.2. Разработка модели “Заявка”

Основная задача подсистемы - хранение и ввод данных. В подсистеме также есть возможность оформить заявку, которая в последующем передаст данные подсистеме вычисления MILP для оптимизации расписания выполнения восстановления узлов и агрегатов.

Рассмотрим модуль составления заявки. У модуля также есть контроллер, модель и представление. В Листингах будет продемонстрирована работа контроллера и модели, основной код представлен в приложении А.

Рассмотрим связи с таблицами. Исходя из диаграмм, можно сделать вывод, что таблица “Заявки” имеет связь один-ко-многим с таблицей “Детали”. Листинг 14 демонстрирует модель Order и связи с таблицами.

Листинг 14 - Модель Order

```
class Order extends Model
{
    use HasFactory;

    protected $table = 'orders';

    protected $fillable = [
        'number_of_details',
        'number_of_devices',
        'J_parameter',
        'T_l_w',
        'A_w_i',
    ];

// связь с таблицей “Детали”
    public function details()
    {
        return $this->belongsToMany(Detail::class);
    }
```

Продолжение листинга 14 - Модель Order

```
    public function order()
    {
        return $this->belongsTo(Order::class);
    }
}
```

Далее рассмотрим саму логику формирования заказа. Для оформления заказа есть форма ввода, полный код представления отражен в приложении А.

Контроллер с логикой также содержит метод `store(Request $request)`, который сохраняет данные и формирует необходимые матрицы.

Определим входные данные. Для составления расписания подсистеме решения MILP необходимы такие данные: 1) количество деталей; 2) количество приборов (l = 4); 3) J - параметр - общее количество позиций, которые занимают ПЗ в последовательностях их выполнения на приборах КС (J>W); 4) T_l_w – длительности реализации операций с заданиями w-х типов (входящих в наборы заданий с индексами (номерами) w) на l-х приборах КС;5) A_w_i – параметр, характеризующий принадлежность i-го задания w-му набору заданий (awi =1 в случае, если i-е задание принадлежит w-му набору (если i-е задание имеет w-й тип); awi =0 в случае, если i-е задание не принадлежит w-му их набору); 6) P_l_w– длительности переналадок l-х приборов на выполнение заданий w-х типов; – матрица длительностей переналадок приборов на выполнение заданий w-х типов;

Рассмотрим в листинге 15 формирование новой заявки.

Листинг 15 - Формирование новой заявки

```
// валидация данных
$validated = $request->validate([
    'number_of_details' => 'required|integer|min:2',
    'detail_ids' => 'required|array|min:2',
    'detail_ids.*' => 'exists:details,id|distinct',
]);

// Создание новой заявки
$order = Order::create([
    'number_of_details' => $validated['number_of_details'],
    'number_of_devices' => 4,
    'J_parameter' => 44
]);
// присоединение деталей к заявке
$order->details()->attach($validated['detail_ids']);
```

Далее необходимо выполнить расчеты параметров. В начале выполним вычисления параметра J. Реализация кода представлена в листинге 16.

Листинг 16 - Расчеты J-параметра

```
//метод store
$order->J_parameter = $this->countJ($numberOfTypes, $numberOfTasks);

 private function countJ(int $n, $n_w): int
{
   // Проверяем, что $n_w не равно нулю, чтобы избежать деления на ноль
   if ($n_w == 0) {
        return 0; // Возвращаем ноль или другое подходящее значение, если $n_w равно
нулю
   }

   // Вычисляем верхнюю и нижнюю границу
   $lowerBound = 2 * $n;
   $upperBound = $n * (floor($n_w / 2));

   // Проверяем, что верхняя граница больше или равна нижней границе
   if ($upperBound < $lowerBound) {
        // Если нет, меняем местами значения, чтобы обеспечить корректные аргументы
для mt_rand()
        $temp = $lowerBound;
        $lowerBound = $upperBound;
        $upperBound = $temp;
   }

   // Возвращаем случайное целое число в заданном диапазоне
   return mt_rand($lowerBound, $upperBound);
}
```

После того, как были выполнены расчеты параметра J, выполним расчеты матриц. Для начала необходимо составить матрицу соответствия задания типу задания - A_l_w. Листинг 17 демонстрирует алгоритм формирования матрицы соответствия.

Листинг 17 - Формирование матрицы A_l_w

```
//метод store
$typeMatrix = $this->createTypeMatrix($validated['detail_ids']);
$order->A_w_i = json_encode($typeMatrix);

private function createTypeMatrix(array $detailIds)
{
    // Получение уникальных типов деталей из базы данных и преобразование их в
массив
```

Продолжение листинга 17 - Формирование матрицы A_l_w

```
    $types = Detail::select('type')->distinct()->pluck('type')->toArray();

    // Инициализация пустой матрицы типов
    $typeMatrix = [];

    // Проход по каждому detailId из переданного массива
    foreach ($detailIds as $detailId) {
        // Поиск детали по её ID
        $detail = Detail::find($detailId);

        // Создание строки типов для текущей детали
        $typeRow = $this->createTypeRow($detail->type, $types);

        // Добавление строки типов в матрицу типов
        $typeMatrix[] = $typeRow;
    }

    // Возврат готовой матрицы типов
    return $typeMatrix;
}

private function createTypeRow($detailType, $types)
{
    // Создание строки, заполненной нулями, с длиной равной количеству уникальных
типов
    $typeRow = array_fill(0, count($types), 0);

    // Проход по всем уникальным типам для создания строки
    foreach ($types as $index => $type) {
        // Если текущий тип совпадает с типом детали, устанавливаем значение 1
        if ($type == $detailType) {
            $typeRow[$index] = 1;
        } else {
            // Иначе оставляем значение 0
            $typeRow[$index] = 0;
        }
    }

    // Возвращаем строку типов для одной детали
    return $typeRow;
}
```

Далее, после формирования матрицы соответствия типу задания самого задания, необходимо рассчитать несколько матриц времени - матрица основного времени выполнения восстановления T_l_w и матрица вспомогательного времени выполнения (матрица переналадок приборов) P_l_w.

Алгоритм формирования матриц представлен в листинге 19 и листинге 20 с комментариями.

Листинг 19 - Формирование матрицы основного времени восстановления узлов

```
// метод store
$operationsMatrix =
$this->calculateSummarizedOperations($validated['detail_ids']);
$order->T_l_w = json_encode($operationsMatrix);

private function calculateSummarizedOperations(array $detailIds)
{
    // Определение типов работ
    $workTypes = ['turning', 'electricityWelding', 'vibroWelding',
'secondTurning', 'grinding'];

    // Инициализация массива для суммарных операций
    $summarizedOperations = [];

    // Инициализация сумм всех типов работ нулями
    foreach ($workTypes as $workType) {
        $summarizedOperations[$workType] = 0;
    }

    // Проход по каждому идентификатору детали
    foreach ($detailIds as $detailId) {
        // Получение всех операций, связанных с текущей деталью
        $operations = Operation::where('detail_id', $detailId)->get();

        // Обновление суммарных операций для каждой операции
        foreach ($operations as $operation) {
            // Добавление времени основной операции к соответствующему типу
работы
            $summarizedOperations[$operation->type] += $operation->main_time;
        }
    }

    // Возврат массива суммарных операций
    return $summarizedOperations;
}
```

Листинг 20 - Формирование матрицы вспомогательного времени (матрица переналадок приборов)

```
// метод store

      $operationsMatrixAuxiliary =
$this->calculateSummarizedOperationsAuxiliary($validated['detail_ids']);
$order->P_L_w = json_encode($operationsMatrixAuxiliary);


private function calculateSummarizedOperationsAuxiliary(array $detailIds)
{
// Определение типов работ
          $workTypes = ['turning', 'electricityWelding', 'vibroWelding',
'secondTurning', 'grinding'];
```

Продолжение листинга 20 - Формирование матрицы вспомогательного времени (матрица переналадок приборов)

```
        // Инициализация массива для суммарных операций
        $summarizedOperations = [];

        // Инициализация сумм всех типов работ нулями
        foreach ($workTypes as $workType) {
            $summarizedOperations[$workType] = 0;
        }

        // Проход по каждому идентификатору детали
        foreach ($detailIds as $detailId) {
            // Получение всех операций, связанных с текущей деталью
            $operations = Operation::where('detail_id', $detailId)->get();

            // Обновление суммарных операций для каждой операции
            foreach ($operations as $operation) {
                // Добавление времени основной операции к соответствующему типу
работы
                $summarizedOperations[$operation->type] +=
$operation->auxiliary_time;
            }
        }

        // Возврат массива суммарных операций
        return $summarizedOperations;

}
```

В результате, подсистеме решения задач MILP отправляются данные в заявке для реализация оптимизации расписания восстановления узлов и агрегатов технологического и транспортного оборудования на специализированных ремонтных предприятиях.

3.4.3 Модули просмотра заявок и деталей

Данные модули отвечают за отображение данных пользователю информации о том, что содержится в базе данных. Модуль отображения информации о деталях содержит метод в контроллере Details Controller и вызывает представление. Для отображения делается выборка всех деталей в БД и передается в представление. В свою очередь, представление имеет фильтр по типу детали, реализация которого представлена в листинге 21. Полный код представлен в приложении А.

Листинг 21 - Фильтрация по типу детали

```
<script>
    // Событие запускается, когда вся страница загружена
    document.addEventListener('DOMContentLoaded', (event) => {
        // Получаем доступные типы из переменной PHP и преобразуем их в JSON
        const availableTypes = @json($availableTypes);

        // Получаем элемент input с id 'filterType'
        const filterInput = document.getElementById('filterType');

        // Добавляем обработчик события 'input' для элемента filterInput
        filterInput.addEventListener('input', function() {
            // Проверяем введенный тип на допустимость
            validateType(this.value, availableTypes);
            // Фильтруем детали по введенному типу
            filterDetails(this.value);
        });

        // Функция проверки введенного типа
        function validateType(value, types) {
            // Если введенный тип не содержится в массиве доступных типов и не
пустой, устанавливаем кастомное сообщение об ошибке
            if (!types.includes(value) && value !== '') {
                filterInput.setCustomValidity('Тип не существует');
            } else {
                // Иначе убираем сообщение об ошибке
                filterInput.setCustomValidity('');
            }
        }

        // Функция фильтрации деталей по введенному значению
        function filterDetails(filterValue) {
            // Получаем все строки таблицы в <tbody>
            const rows = document.querySelectorAll('tbody tr');
            // Проходим по каждой строке таблицы
            rows.forEach(row => {
                // Получаем текстовое содержимое третьей ячейки строки
(предположительно это тип детали)
                const type = row.querySelector('td:nth-child(3)').textContent;
                // Если тип детали содержит введенное значение или введенное
значение пустое, показываем строку
                if (type.includes(filterValue) || filterValue === '') {
                    row.style.display = '';
                } else {
                    // Иначе скрываем строку
                    row.style.display = 'none';
                }
            });
        }
    });
</script>
```

Также отобразим формирование представления с отображением всех деталей в БД. Листинг 21 отображает запрос в базу данных и передачу данных в представление.

Листинг 22 - Метод для отображения представления со всеми деталями

```
public function allDetails()
{
   $details = Detail::all()->toArray();
   $availableTypes = Detail::select('type')->distinct()->pluck('type')->toArray();
   return view('details', compact('details', 'availableTypes' ));
}
```

По аналогии с представлением всех деталей работает и отображение всех заказов. Для отображения данных выполняется запрос в БД в таблицу “Детали”, и далее они передаются в представление. Полный код представления представлен в приложении А.

3.4.4 Модуль интерпретации и отображения оптимизированного расписания

Рассмотрим модуль интерпретации расписания. Важно отметить, что расписание от подсистемы решения MILP приходит в формате .xlsx. Для интерпретации данных из формата .xlsx была использована библиотека PhpOffice\PhpSpreadsheet. Данная библиотека отлично работает с файлами различных форматов, включая .xlsx, .xls, .ods, .csv и другие.

С помощью данной библиотеки был реализован алгоритм загрузки и интерпретации данных. Полный код представления загрузки файла представлен в приложении А, а в листинге 23 отражен алгоритм загрузки и чтения данных из файла.

Листинг 23 - Метод загрузки файла с расписанием

```
public function uploadAndProcessFile(Request $request)
{
   // Проверка наличия файла в запросе
   if (!$request->hasFile('file')) {
       return response()->json(['error' => 'No file uploaded'], 400);
   }

   // Получение файла из запроса
   $file = $request->file('file');

   // Загружаем Excel файл
   $spreadsheet = IOFactory::load($file->getRealPath());

   // Считываем все листы и сохраняем данные в базу данных
```

Продолжение листинга 23 - Метод загрузки файла с расписанием

```
    foreach ($spreadsheet->getSheetNames() as $sheetName) {
        $sheet = $spreadsheet->getSheetByName($sheetName);
        $sheetData = $sheet->toArray();

        ExcelData::create([
            'sheet_name' => $sheetName,
            'data' => $sheetData, // Laravel автоматически преобразует массив в JSON
для сохранения
        ]);
    }

    return response()->json(['message' => 'File uploaded and processed
successfully'], 200);
}
```

Также одной из задач подсистемы было отправлять данные расчетов времени восстановления узлов подсистеме MILP. Метод записи данных файл был реализован также с помощью библиотеки Php Spreadsheet. Листинг 24 демонстрирует метод загрузки данных расчетов данной подсистемы в файл с форматом .xlsx. Файл можно скачать при успешном создании заявки на создание расписания. Полный код представления отражен в приложении А.

Листинг 24 - Метод загрузки данных в файл

```
public function download(Request $request)
{
    $id = $request->route('id');
    $order = Order::with('details')->findOrFail($id);

    $spreadsheet = new Spreadsheet();

    // Добавляем лист в книгу
    $sheet = $spreadsheet->getActiveSheet();

    // Заполняем данные в лист
    $sheet->setCellValue('A1', 'ID');
    $sheet->setCellValue('B1', 'Number of Details');
    $sheet->setCellValue('C1', 'Number of Devices');
    $sheet->setCellValue('D1', 'J Parameter');
    $sheet->setCellValue('E1', 'T L W');
    $sheet->setCellValue('F1', 'A W I');
    $sheet->setCellValue('G1', 'P L W');

    $row = 2;
    $sheet->setCellValue('A'. $row, $order->id);
    $sheet->setCellValue('B'. $row, $order->number_of_details);
    $sheet->setCellValue('C'. $row, $order->number_of_devices);
    $sheet->setCellValue('D'. $row, $order->J_parameter);
    $sheet->setCellValue('E'. $row, json_encode($order->T_l_w));
    $sheet->setCellValue('F'. $row, json_encode($order->A_w_i));
    $sheet->setCellValue('G'. $row, $order->P_l_w?? '');
```

Продолжение листинга 24 - метод загрузки данных в файл

```
    // Сохраняем файл в памяти
    // Сохраняем файл в памяти и отправляем на скачивание
    $writer = new Xlsx($spreadsheet);
    $fileName = 'order_'. $order->id. '.xlsx';

    // Буферизация вывода
    ob_start();
    $writer->save('php://output');
    $content = ob_get_contents();
    ob_end_clean();

    return response($content)
        ->header('Content-Type',
'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet')
        ->header('Content-Disposition', 'attachment; filename="'. $fileName . '"')
        ->header('Cache-Control', 'max-age=0');
}
```

3.5 Тестирование, верификация, валидация подсистемы ввода и хранения данных системы оптимизации расписания восстановления узлов и агрегатов

Целью тестирования и верификации является обеспечение корректности и надежности данных,а также соответствию поставленных задач ВКР.

Тестирование информационной системы является необходимым этапом для признания ее полноценной, поскольку в процессе тестирования могут быть обнаружены различные недочеты, что в свою очередь способствует улучшению качества продукта.

При открытии приложения пользователь видит главную страницу. Рисунок 3.5.1. демонстрирует наполнение и функционал главной страницы.

Из рисунка видно, что в приложении есть различный функционал, который отображен в меню. По тапу на первый раздел “Справочник деталей” пользователь перейдет на таблицу с демонстрацией всех деталей что есть в БД. На рисунке 3.5.2 отображен интерфейс данных о деталях.

Как видно из изображения, данные могут быть отфильтрованы по типу детали. Рисунок 3.5.3 демонстрирует, что фильтр работает с типами данными из БД. Результаты фильтрации отражены на рисунке 3.5.4. На данный момент в системе всего один вид детали второго типа.

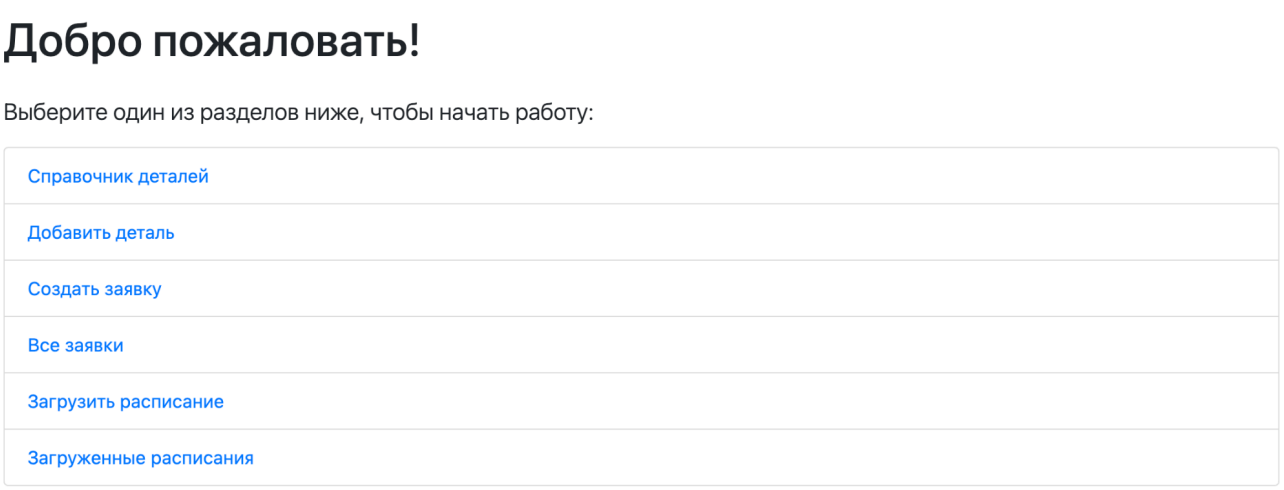


Рисунок 3.5.1 - Главная страница приложения

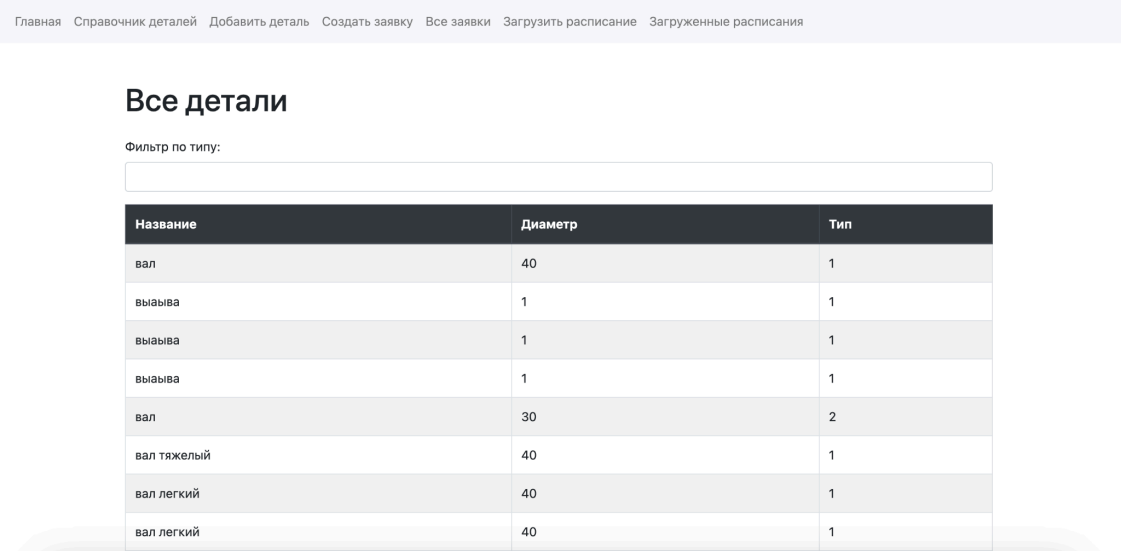


Рисунок 3.5.2 - Отображение данных о деталях.

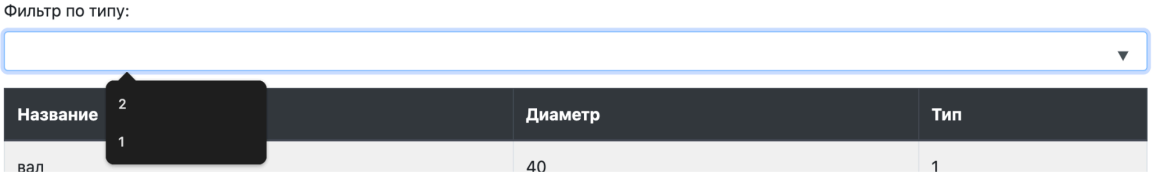


Рисунок 3.5.3 - Количество типов в БД доступных для фильтрации

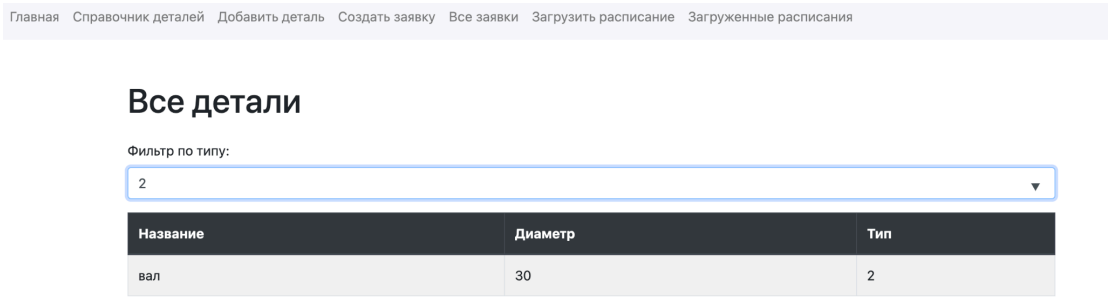


Рисунок 3.5.4 - Результат фильтрации

Далее, при переходе в меню в раздел “Добавить деталь” пользователю отобразится форма ввода данных о детали. Рисунок 3.5.5. демонстрирует интерфейс формы ввода.

Введем данные. При попытке отправить полностью пустую форму появится сообщение об ошибке. Попап с ошибкой отображен на рисунке 3.5.6. В системе все поля данной формы обязательны, и ввести пустые значения невозможно.

Создадим тестовую запись. На рисунке 3.5.7 видно, что если указать что у детали несколько участков повреждений, количество полей для ввода значений увеличится. При добавлении данных можно проверить, записываются ли данные. Рисунок 3.5.8 демонстрирует, что запись успешно создана и ее можно увидеть в списке деталей.

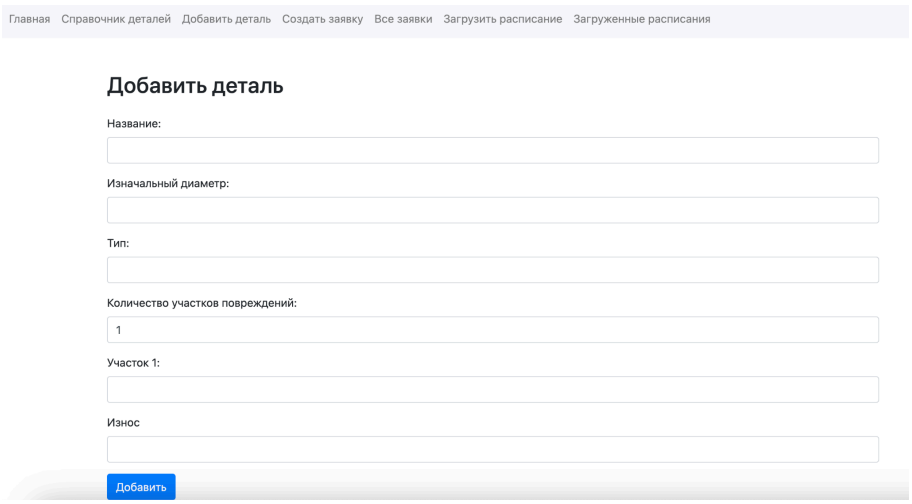


Рисунок 3.5.5 - Форма ввода данных детали

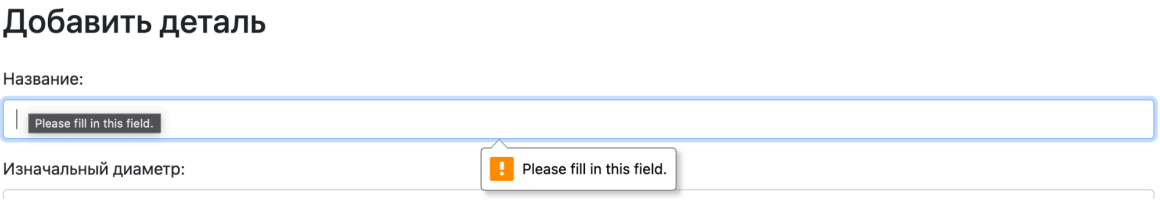


Рисунок 3.5.6 - Валидация на полях

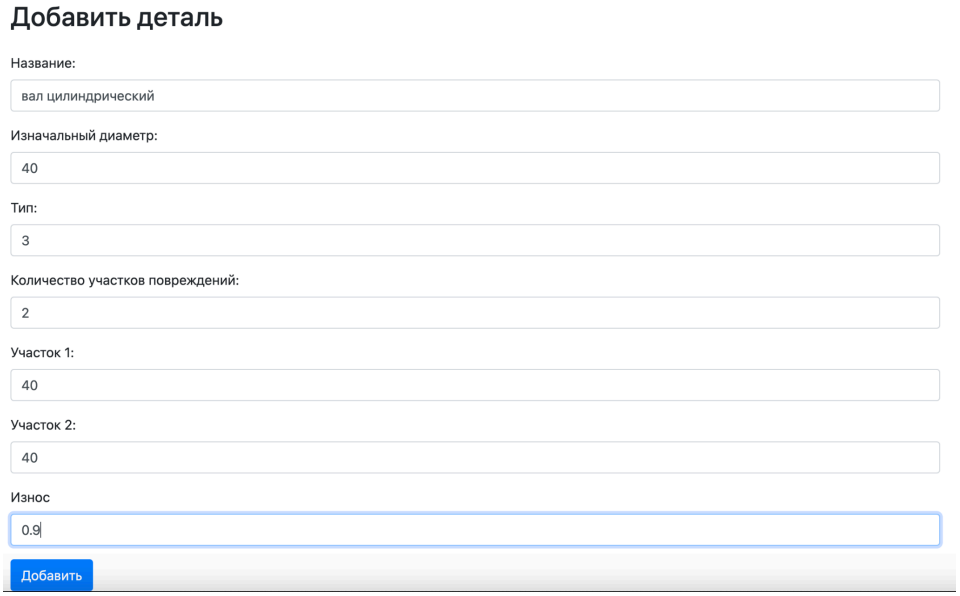


Рисунок 3.5.7 - Заполнение данными формы ввода

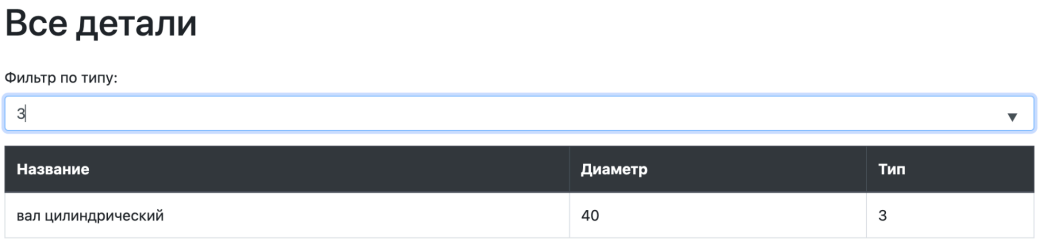


Рисунок 3.5.8 - Успешная запись в системе

Переходя по вкладкам меню, следующим пунктом будет создание заявки. Процесс создания заявки происходит путем ввода количества деталей, а после необходимо выбрать детали из справочника. Рисунки 3.5.9 и 3.5.10 демонстрируют процесс ввода данных.

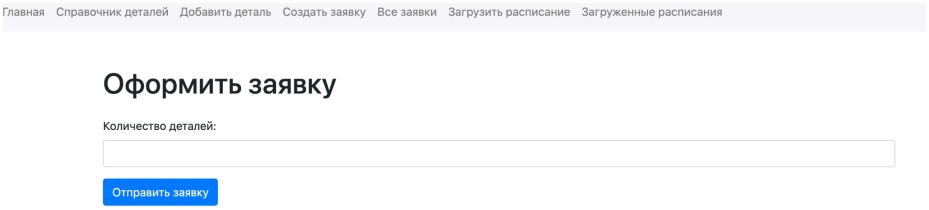


Рисунок 3.5.9 - Начало оформления заявки

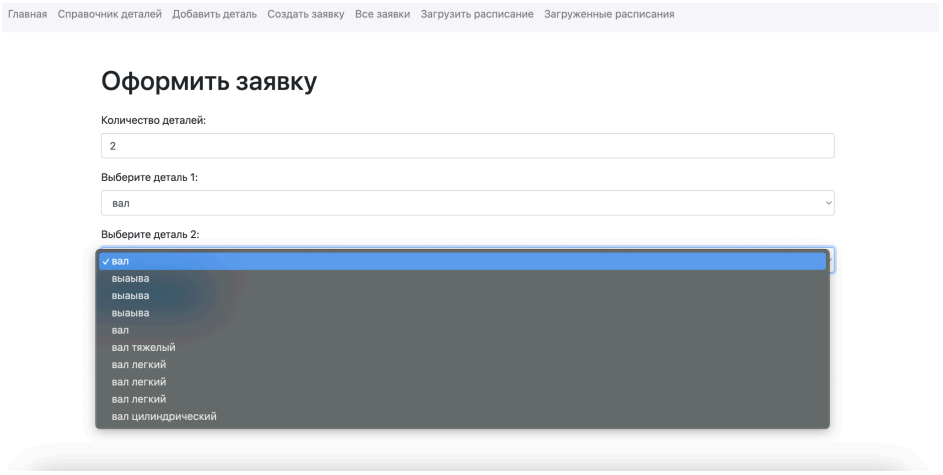


Рисунок 3.5.10 - Выбор деталей из базы данных

После того, как заявка была успешна создана, отображаются результаты заявки и выбранные детали. После можно создать новую заявку, либо же выгрузить результаты. Рисунок 3.5.11 демонстрирует интерфейс успешно созданной заявки.

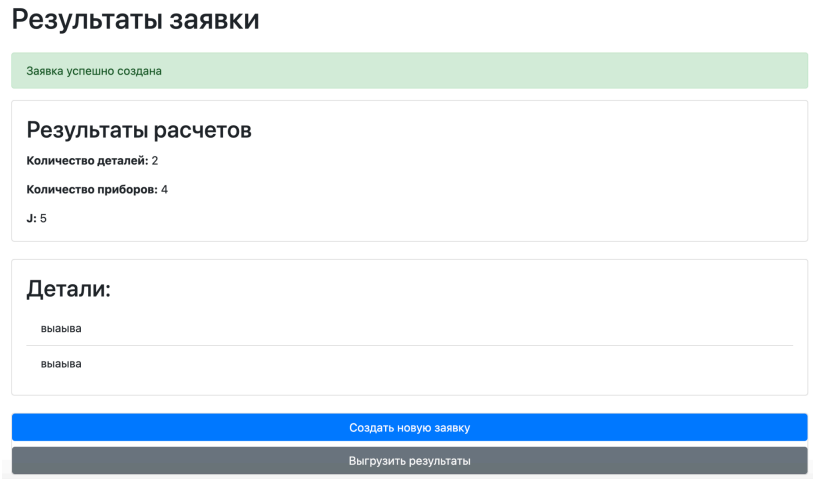


Рисунок 3.5.11 - Выбор деталей из базы данных

Попробуем выгрузить результаты. При нажатии на кнопку появляется диалоговое окно с возможностью сохранить данные (рис. 3.5.12).

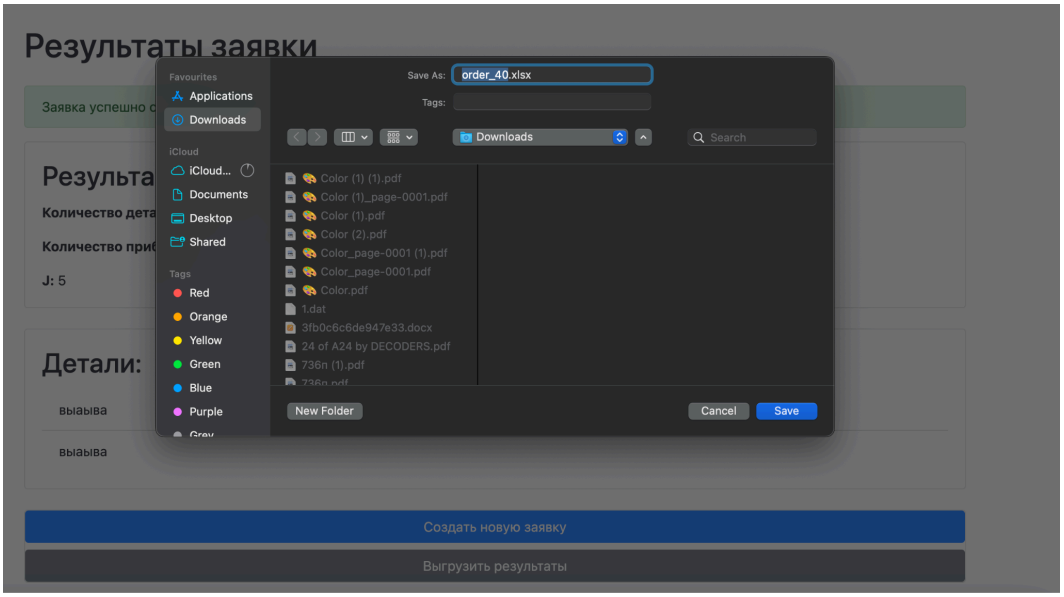


Рисунок 3.5.12 - Диалоговое окно для выгрузки результатов

После того как данные были выгружены, их успешно можно просмотреть. Рисунок 3.5.13 демонстрирует успешную выгрузку данных.

ID	Number of	Number of	J Paramete	T L W	A W I	P L W
40	2	4	5	"{\"turning"	"[[0,0,1],[0,0,1]]"	{“turning":1,"electricityWelding":4,"vibroWelding":30,"secondTurning":44,"grinding":50}
						{“turning”:4,"electricityWelding":3,"vibroWelding":24,"secondTurning":11,"grinding":3}

Рисунок 3.5.13 - Диалоговое окно для выгрузки результатов

Далее идет блок загрузки расписания. При осуществлении загрузки появляется диалоговое окно с возможностью выбор файла. Рисунки 3.5.14 и 3.5.15 демонстрируют интерфейс загрузки расписания.

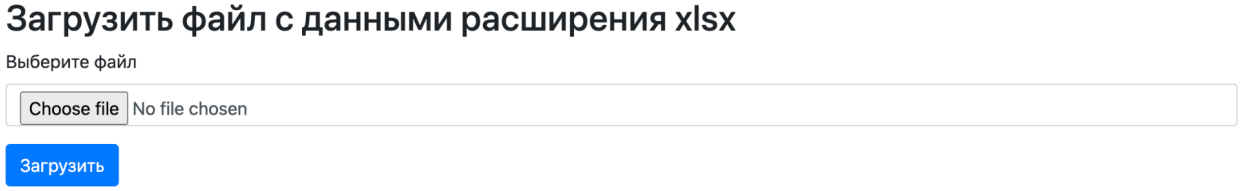


Рисунок 3.5.14 - Интерфейс загрузки файла с расписанием

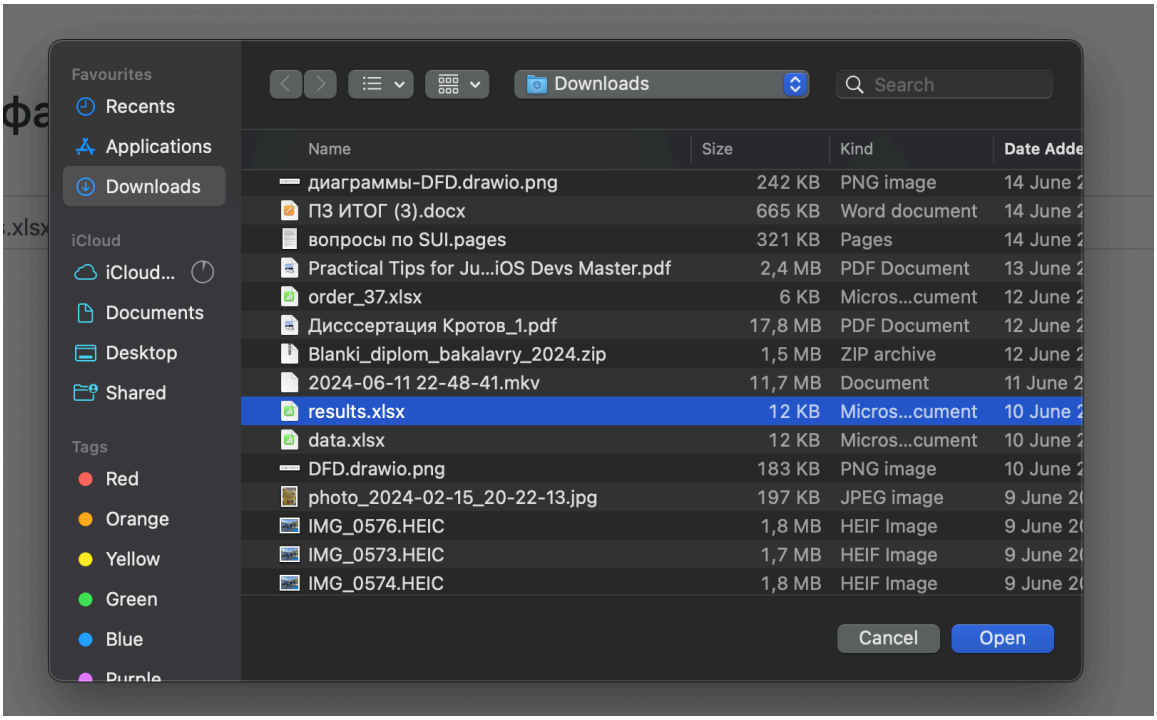


Рисунок 3.5.15 - Диалоговое окно загрузки расписания

После загрузки расписания, посмотреть данные расписания можно во вкладке меню “Загруженные расписания”. Рисунок 3.5.16 отображает интерфейс загруженных расписаний.

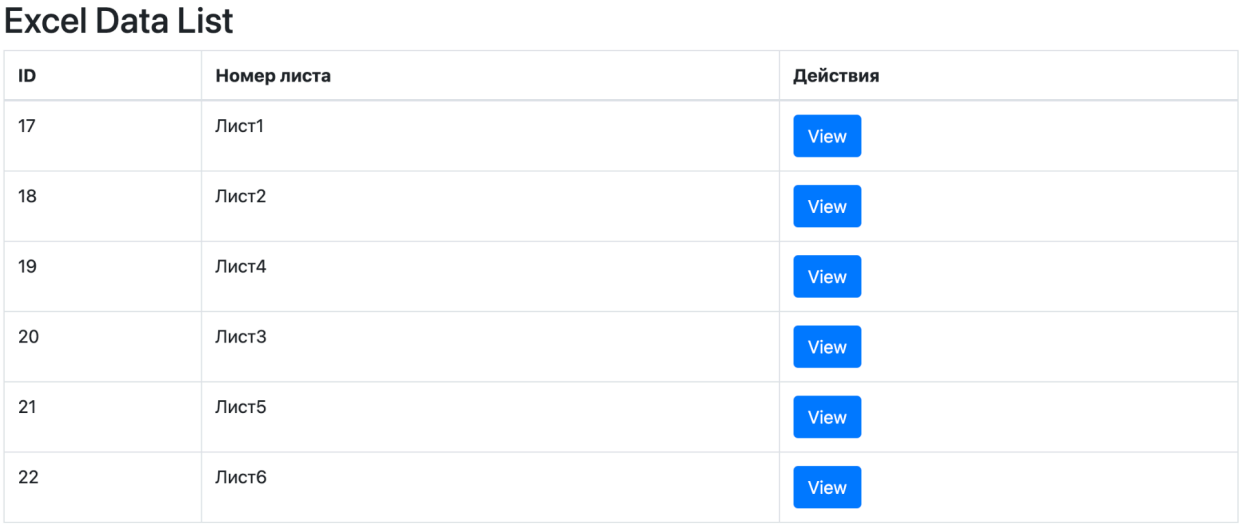
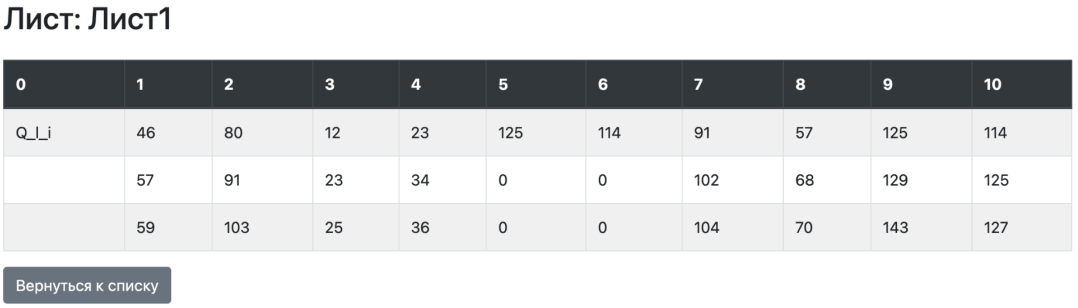


Рисунок 3.5.16 - Интерфейс отображения загруженных расписаний

Каждый лист отображает результаты составленного расписания. Рисунок 3.5.17 отображает результат выгруженного файла.



Лист: Лист1

0	1	2	3	4	5	6	7	8	9	10
Q_l_i	46	80	12	23	125	114	91	57	125	114
	57	91	23	34	0	0	102	68	129	125
	59	103	25	36	0	0	104	70	143	127

Вернуться к списку

Рисунок 3.5.17 - Результаты загруженного расписания

Проверим БД на нормализацию. Из полной атрибутивной схемы таблицы видно, что некоторые неключевые атрибуты имеют тип “массив”. Соответственно, БД не нормализована и не соответствует 1НФ.

При проектировании было принято решение о денормализации базы данных для того, чтобы упростить расчеты и удовлетворить специфические требования приложения - в системе выполняется расчет и формирование матриц, логически эти расчеты необходимо группировать именно в массивы для дальнейшей выгрузки результатов. При приведении БД к 1НФ было бы большое количество таблиц и связей, а также пришлось бы выполнять сложные трудоемкие запросы на соединение для получения и записи данных.

В пример приведем таблицу orders. Запрос с данными представлен на рисунке 3.5.18. Обратим внимание на поля P_l_w, T_l_w и A_w_i - поля хранят матрицы времени выполнения заданий, а также соответствия типам заданий. Эти значения удобнее группировать и записывать в массивы, так как данные в виде матриц используются дальше при построении оптимизированного расписания.

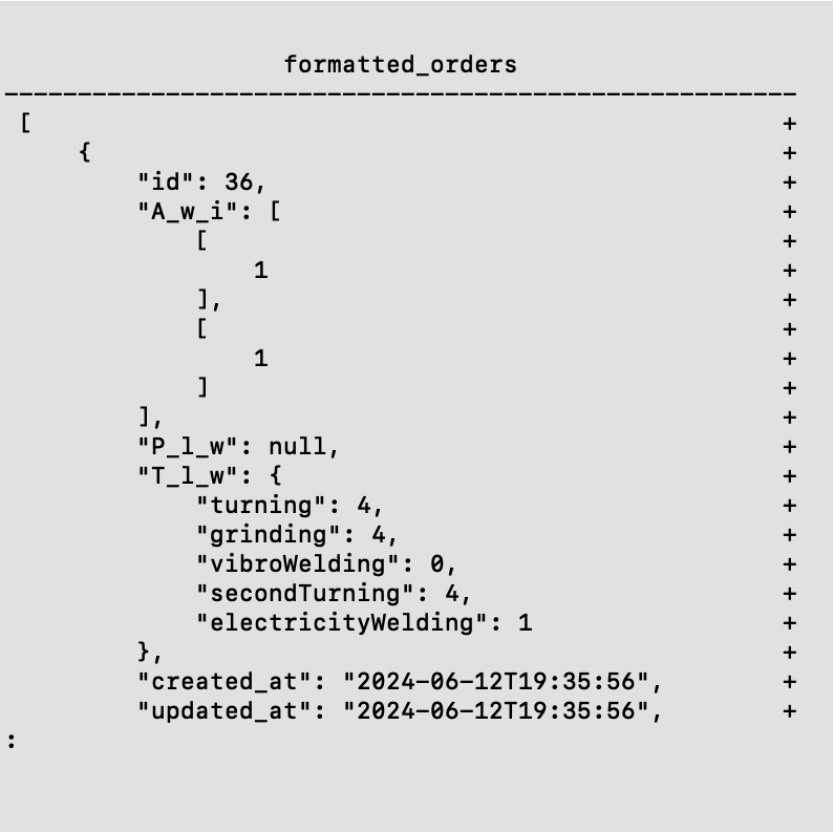


Рисунок 3.5.18 - Результаты запроса таблицы orders

Выводы по разделу 3

В данном разделе был проведен анализ и выбор программных средств для программной реализации подсистемы ввода и хранения данных. В результате анализа был выбран фреймворк Laravel с языком программирования PHP, и встроенным шаблонизатором Blade. В качестве работы с БД была выбрана СУБД PostgresSQL, а для поддержания UI были выбраны язык разметки и каскадных стилей HTML и CSS, язык скриптов JavaScript а также библиотека Bootstrap.

В процессе работы были реализованы модули оформления заявки на оптимизацию расписания, модуль интерпретации оптимизированного расписания а также ввода и хранения данных о деталях и технологических процессах данных деталей.

После разработки подсистемы было проведено тестирование, вследствии которого было выявлено что система содержит ввод и хранение данных, а также

может выполнять необходимые расчеты времени восстановления и интерпретации данных. Также была проверена БД, в результате тестирования было выявлено что БД де нормализована для удовлетворения специфических требований к расчетам.

ЗАКЛЮЧЕНИЕ

В данной дипломной работе были исследованы возможности реализации восстановления деталей цилиндрической формы на различных этапах. В соответствии с целью, была построена система хранения и ввода данных деталей.

В контексте разработки системы оптимизации расписаний восстановления узлов и агрегатов технологического и транспортного оборудования на специализированных ремонтных предприятиях, подсистема хранения и ввода данных играет ключевую роль в обеспечении эффективности и надежности процессов оптимизации расписания для восстановления оборудования.

Проведенный анализ предметной области выявил, что есть системы решающие похожие задачи, но нет таких систем, которые могли бы работать с построением расписаний с помощью пакетов заданий. Также, так как ремонтные предприятия специализированные, важно учитывать тип деталей. В данном случае ключевую роль специализации предприятия играет цилиндрический тип детали - различные виды валов, пальцев, валиков и осей.

Второй раздел данной работы посвящен проектированию. Диаграммы потоков данных (DFD) и диаграммы нотации IDEF0 помогли декомпозировать и разбить систему на функциональные блоки. Диаграммы на ключах, нотации Чена и нотации IDEF1X помогли качественно разработать систему БД в 3НФ. Также благодаря декомпозиции по бизнес-процессам было легко выявить взаимодействия данных в системе, и определен основной функционал:

- ввод и хранение данных о деталях, заявках и расписаниях;
- расчет времени восстановления узлов и агрегатах на различных приборах;
- интерпретация и отображение данных об оптимизированном расписании;
- выгрузка и отображение выполненных расчетов.

Третий раздел содержит обоснование программных средств, проектирование структурно - функциональной схемы, которая помогла структурировать систему и взаимодействия блоков. В ходе обоснования программных средств был выбран фреймворк Laravel, и использование языка PHP соответственно. Для реализации UI был выбран шаблонизатор Blade, который поддерживает различные инструменты для верстки: HTML, CSS, JavaScript.

Также третий раздел содержит разработку подсистемы и полное ее тестирование. В процессе тестирования был проверен не только основной функционал, но и проверена БД на соответствие нормальным формам. В процессе проектирования было принято решение о денормализации БД с целью удовлетворения специфических расчетов системы и для упрощения получения данных без сложных связей и запросов.

В результате выполнения работы можно сделать выводы, что приложение может быть доработано специализированно для предприятия и внедрено.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 1С:ERP Управление предприятием // Отраслевые и специализированные решения 1С:Предприятие [Электронный ресурс]. URL https://solutions.1c.ru/catalog/1cerp

2. IDEF0 Diagram // vitechcorp.com: [сайт]. – URL: https://vitechcorp.com/resources/CORE/onlinehelp/desktop/Views/IDEF0.htm

3. IDEF1X Notation // vertabelo.com: [сайт]. – URL: https://vertabelo.com/blog/idef1x-notation/

4. JavaScript Documentation [Электронный ресурс]. URL: https://devdocs.io/javascript/

5. Laravel 10.x Руководство пользования [Электронный ресурс]. URL: https://laravel.su/docs/10.x/documentation

6. SAP Risk Management Overview // Enterprise risk management software [Электронный ресурс]. URL: https://www.sap.com/products/financial-management/risk-management.html

7. Багринцев О.О. ТЕХНОЛОГИЧЕСКИЕ ПРОЦЕССЫ ВОССТАНОВЛЕНИЯ ИЗНОШЕННЫХ ДЕТАЛЕЙ / Багринцев О.О, Харин М.В., Мурлыкин Р.Ю - Научный журнал молодых ученых

8. Воловик Е.Л. Справочник по восстановлению деталей / Е.Л. Воловик. – М.: Из-во «Колос», 1981.– 351 с.

9. Воловик Е.Л. Ремонт машин. Под ред. 13 Н.Ф. – М.: Из-во «Агропромиздат», 1992. – 560 с.

10. Галактика ERP // Enterprise risk management software [Электронный ресурс]. URL: https://galaktika.ru/erp

11. Кротов Кирилл Викторович МАТЕМАТИЧЕСКИЕ МОДЕЛИ И МЕТОДЫ МНОГОУРОВНЕВОЙ ОПТИМИЗАЦИИ РАСПИСАНИЙ МНОГОСТАДИЙНЫХ ПРОЦЕССОВ С АДАПТАЦИЕЙ: дис. доктор технических наук: 05.13.08.. - Севастополь, 2022. - 417 с.

12. Матвеев В.А. Техническое нормирование ремонтных работ в сельском хозяйстве / В.А. Матвеев, И.И. Пустовалов. – М.: Из-во «Колос», 1979. – 288 с.

13. Н.А. Кривошеева, М.Г. Таспаева БАЗЫ ДАННЫХ: Теория нормализации. Методические рекомендации [Электронный ресурс]. URL: http://elib.osu.ru/bitstream/123456789/14480/1/142385_20210429.pdf

14.

15. Руководство по HTML5 и CSS3 [Электронный ресурс]. URL: https://metanit.com/web/html5/

16. Руководство по PHP ¶ [Электронный ресурс]. URL: https://www.php.net/manual/ru/index.php

17. Руководство по PostgreSQL [Электронный ресурс]. URL: https://www.postgresql.org/docs/

18. Руководство пользования Bootstrap // https://bootstrap-4.ru/ – URL: https://bootstrap-4.ru/

19. Сведения о GIT // https://docs.github.com/ – URL: https://docs.github.com/ru/get-started/using-git/about-git

20. 20 А.Г. Ремонт технологических машин и оборудования: учебное пособие / А.Г. 20, В.А. Скрябин, О.В. Пименова, А.С. Репин, Н.Я. Карасев. – Пенза: Информационно-издательский центр Пензенского государственного университета, 2009.– 328 с.

21. Схиртладзе А.Г. Ремонт технологических машин и оборудования: учебное пособие / А.Г.Схиртладзе, В.А.Скрябин, В.П.Борискин.– Старый Оскол: Из-во «Тонкие наукоемкие технологии», 2020. – 434 с.

22. Что такое ERP // https://dynamics.microsoft.com – URL: https://dynamics.microsoft.com/ru-ru/erp/what-is-erp/

ПРИЛОЖЕНИЕ А

Полный код приложения

```
<?php

namespace App\Http\Controllers;

use App\Models\Detail;
use App\Models\Operation;
use App\Models\TechnologicalProcess;
use Illuminate\Routing\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Log;

class DetailController extends Controller
{
   public function allDetails()
   {
       $details = Detail::all()->toArray();
       $availableTypes =
Detail::select('type')->distinct()->pluck('type')->toArray();
       return view('details', compact('details', 'availableTypes' ));

   }

   public function create()
   {
       return view('addDetail');
   }

   public function store(Request $request)
   {
       $validatedData = $request->validate([
           'name' => 'required|string|max:255',
           'initialDiameter' => 'required|numeric',
           'wearsCount' => 'required|integer|min:1',
           'wear_areas' => 'required|array|min:1',
           'wear_areas.*' => 'numeric',
           'type' => 'required|integer'
       ]);

       $wear_areas = $validatedData['wear_areas'];

       if (!is_array($wear_areas)) {
           $wears_area = json_decode($wear_areas, true);
       }

       $detail = Detail::create([
           'name' => $validatedData['name'],
           'diameter' => $validatedData['initialDiameter'],
           'wear_areas' => $wear_areas,
           'type' => $validatedData['type'],
           'wear' => $request->wear,
       ]);

       Log::info($validatedData);

       $operationsData = [
           'turning' => $detail->turningTime()
       ];
```

```
        if ($detail->type == 1 || $detail->type == 2) {
            $operationsData['electricityWelding'] = $detail->electricityWelding();
        } else {
            $operationsData['vibroWelding'] = $detail->vibroWelding();
        }

        $operationsData = array_merge($operationsData, [
            'secondTurning' => $detail->secondTurningTime(),
            'grinding' => $detail->grinding(),
        ]);

        $operations = [];

        foreach ($operationsData as $type => $time) {
            $main_time = isset($time['main_time']) ? $time['main_time'] : null;
            $auxiliary_time = isset($time['auxiliary_time']) ?
$time['auxiliary_time'] : null;
            $operation = Operation::create([
                'detail_id' => $detail->id,
                'type' => $type,
                'main_time' => $main_time,
                'auxiliary_time' => $auxiliary_time,
            ]);

            $operations[] = $operation->toArray(); // Добавление операции в список
        }

        // Создание технологического процесса
        TechnologicalProcess::create([
            'detail_id' => $detail->id,
            'operations' => json_encode($operations), // Сохранение списка операций в
формате JSON
        ]);

        // Возвращаем ответ или выполняем другие действия

        return redirect()->route('detail.create')->with('success', 'Detail created
successfully with operations calculated.');
    }
}

<?php

namespace App\Http\Controllers;

use Illuminate\Routing\Controller;

class MainController extends Controller
{
    function index() { return view('main'); }
}
<?php

namespace App\Http\Controllers;

use App\Models\Detail;
use App\Models\ExcelData;
use App\Models\Operation;
use App\Models\Order;
use Illuminate\Http\Request;
use Illuminate\Routing\Controller;
use PhpOffice\PhpSpreadsheet\IOFactory;
```

```
use PhpOffice\PhpSpreadsheet\Spreadsheet;
use PhpOffice\PhpSpreadsheet\Writer\Xlsx;

class OrderController extends Controller
{
   public  function index()
   {
       $orders = Order::all();
       return view('allOrders', compact('orders'));
   }

   public function create()
   {
       $details = Detail::all();
       return view('order', compact('details'));
   }

   public function store(Request $request)
   {
       $validated = $request->validate([
           'number_of_details' => 'required|integer|min:2',
           'detail_ids' => 'required|array|min:2',
           'detail_ids.*' => 'exists:details,id|distinct',
       ]);

       // Создание новой заявки
       $order = Order::create([
           'number_of_details' => $validated['number_of_details'],
           'number_of_devices' => 4,
           'J_parameter' => 44
       ]);

       // Присоединение деталей к заявке
       $order->details()->attach($validated['detail_ids']);

       $numberOfTypes =
count(Detail::select('type')->distinct()->pluck('type')->toArray());
       $numberOfTasks = count($validated['detail_ids']);

       $order->J_parameter = $this->countJ($numberOfTypes, $numberOfTasks);

       $operationsMatrix =
$this->calculateSummarizedOperations($validated['detail_ids']);
       $order->T_l_w = json_encode($operationsMatrix);

       $operationsMatrixAuxiliary =
$this->calculateSummarizedOperationsAuxiliary($validated['detail_ids']);
       $order->P_L_w = json_encode($operationsMatrixAuxiliary);

       $typeMatrix = $this->createTypeMatrix($validated['detail_ids']);
       $order->A_w_i = json_encode($typeMatrix);

       $order->save();


       return redirect()->route('order.show', $order->id)->with('success', 'Заявка
успешно создана');
   }

   public function show($id)
   {
       $order = Order::with('details')->findOrFail($id);
       $details = $order->details;
```

```
        return view('orderResults', compact('order', 'details'));
    }

    private function calculateSummarizedOperations(array $detailIds)
    {
        $workTypes = ['turning', 'electricityWelding', 'vibroWelding',
'secondTurning', 'grinding'];

        $summarizedOperations = [];

        foreach ($workTypes as $workType) {
            $summarizedOperations[$workType] = 0;
        }

        foreach ($detailIds as $detailId) {
            $operations = Operation::where('detail_id', $detailId)->get();

            // Для каждой операции обновляем суммарные операции
            foreach ($operations as $operation) {
                $summarizedOperations[$operation->type] += $operation->main_time;
            }
        }

        return $summarizedOperations;
    }

    private function calculateSummarizedOperationsAuxiliary(array $detailIds)
    {
        $workTypes = ['turning', 'electricityWelding', 'vibroWelding',
'secondTurning', 'grinding'];

        $summarizedOperations = [];

        foreach ($workTypes as $workType) {
            $summarizedOperations[$workType] = 0;
        }

        foreach ($detailIds as $detailId) {
            $operations = Operation::where('detail_id', $detailId)->get();

            // Для каждой операции обновляем суммарные операции
            foreach ($operations as $operation) {
                $summarizedOperations[$operation->type] +=
$operation->auxiliary_time;
            }
        }

        return $summarizedOperations;
    }

    private function createTypeMatrix(array $detailIds)
    {
        $types = Detail::select('type')->distinct()->pluck('type')->toArray();  //
Получение уникальных типов деталей
        $typeMatrix = [];

        foreach ($detailIds as $detailId) {
            $detail = Detail::find($detailId);
            $typeRow = $this->createTypeRow($detail->type, $types);
            $typeMatrix[] = $typeRow;
        }
```

```
        return $typeMatrix;
    }

    private function createTypeRow($detailType, $types)
    {
        $typeRow = array_fill(0, count($types), 0); // Создаем строку, заполненную
нулями

        foreach ($types as $index => $type) {
            if ($type == $detailType) {
                $typeRow[$index] = 1;
            } else {
                $typeRow[$index] = 0;
            }
        }

        return $typeRow;
    }


    private function countJ(int $n, $n_w): int
    {
        // Проверяем, что $n_w не равно нулю, чтобы избежать деления на ноль
        if ($n_w == 0) {
            return 0; // Возвращаем ноль или другое подходящее значение, если $n_w
равно нулю
        }

        // Вычисляем верхнюю и нижнюю границу
        $lowerBound = 2 * $n;
        $upperBound = $n * (floor($n_w / 2));

        // Проверяем, что верхняя граница больше или равна нижней границе
        if ($upperBound < $lowerBound) {
            // Если нет, меняем местами значения, чтобы обеспечить корректные
аргументы для mt_rand()
            $temp = $lowerBound;
            $lowerBound = $upperBound;
            $upperBound = $temp;
        }

        // Возвращаем случайное целое число в заданном диапазоне
        return mt_rand($lowerBound, $upperBound);
    }

    public function download(Request $request)
    {
        $id = $request->route('id');
        $order = Order::with('details')->findOrFail($id);

        $spreadsheet = new Spreadsheet();

        // Добавляем лист в книгу
        $sheet = $spreadsheet->getActiveSheet();

        // Заполняем данные в лист
        $sheet->setCellValue('A1', 'ID');
        $sheet->setCellValue('B1', 'Number of Details');
        $sheet->setCellValue('C1', 'Number of Devices');
        $sheet->setCellValue('D1', 'J Parameter');
        $sheet->setCellValue('E1', 'T L W');
        $sheet->setCellValue('F1', 'A W I');
        $sheet->setCellValue('G1', 'P L W');
```

```
        $row = 2;
        $sheet->setCellValue('A'. $row, $order->id);
        $sheet->setCellValue('B'. $row, $order->number_of_details);
        $sheet->setCellValue('C'. $row, $order->number_of_devices);
        $sheet->setCellValue('D'. $row, $order->J_parameter);
        $sheet->setCellValue('E'. $row, json_encode($order->T_l_w));
        $sheet->setCellValue('F'. $row, json_encode($order->A_w_i));
        $sheet->setCellValue('G'. $row, $order->P_l_w?? '');


        // Сохраняем файл в памяти
        // Сохраняем файл в памяти и отправляем на скачивание
        $writer = new Xlsx($spreadsheet);
        $fileName = 'order_'. $order->id. '.xlsx';

        // Буферизация вывода
        ob_start();
        $writer->save('php://output');
        $content = ob_get_contents();
        ob_end_clean();

        return response($content)
            ->header('Content-Type',
'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet')
            ->header('Content-Disposition', 'attachment; filename="'. $fileName .
'"')
            ->header('Cache-Control', 'max-age=0');
    }

    public function showUploadForm()
    {
        return view('upload');
    }

    public function uploadAndProcessFile(Request $request)
    {
        // Проверка наличия файла в запросе
        if (!$request->hasFile('file')) {
            return response()->json(['error' => 'No file uploaded'], 400);
        }

        // Получение файла из запроса
        $file = $request->file('file');

        // Загружаем Excel файл
        $spreadsheet = IOFactory::load($file->getRealPath());

        // Считываем все листы и сохраняем данные в базу данных
        foreach ($spreadsheet->getSheetNames() as $sheetName) {
            $sheet = $spreadsheet->getSheetByName($sheetName);
            $sheetData = $sheet->toArray();

            ExcelData::create([
                'sheet_name' => $sheetName,
                'data' => $sheetData, // Laravel автоматически преобразует массив в
JSON для сохранения
            ]);
        }

        return response()->json(['message' => 'File uploaded and processed
successfully'], 200);
    }
```

```
    public function listData()
    {
        $excelData = ExcelData::all();
        return view('list', compact('excelData'));
    }

    public function showinfoData($id)
    {
        $excelData = ExcelData::findOrFail($id);
        return view('showData', compact('excelData'));
    }
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Detail extends Model {
    private float $li = 0.0; // Общая длина обрабатываемой поверхности
    private float $damageDiameter = 0.0; // Диаметр, который меняется в процессе
работы

    protected $table = 'details';

    protected $fillable = [
        'type', 'name', 'diameter', 'wear_areas', 'wear',
    ];

    protected $casts = [
        'wear_areas' => 'array',
    ];

    public function operations()
    {
        return $this->hasMany(Operation::class, 'detail_id');
    }

    public function technologicalProcess()
    {
        return $this->hasOne(TechnologicalProcess::class);
    }

    /**
     * Расчет времени токарной обработки
     */
    public function turningTime(
        float $yi = 3.5,  // Величина врезания и перебега для детали
        int $ki = 1,      // Количество проходов для снятия припуска при глубине
резания
        float $s1i = 0.5, // Подача режущего инструмента
        float $v1i = 50,  // Скорость резания
        float $tv1i = 3   // Длительность вспомогательных операций токарной обработки
    ): array
    {
        $this->li = array_sum($this->wear_areas); // Длина обрабатываемой поверхности
по чертежу
        $Li = $this->lengthDetail($this->li, $yi); // Длина обрабатываемой
поверхности детали с учетом врезания и перебега

        $D1i = $this->diameter; // Диаметр детали перед токарной обработкой
        $this->calculateNewDiameter($D1i, $this->wear_areas);
```

```
        $d1i =$this->damageDiameter; // Диаметр детали после токарной обработки

        // Припуск на обработку
        $h1i = ($D1i - $d1i) / 2;

        // Глубина резания
        $g1i = $h1i / $ki;

        // Число оборотов деталей в минуту
        $n1i = 318 * ($v1i / $D1i);

        // Длительность основной токарной обработки
        $to1i = ($Li * $ki) / ($n1i * $s1i);

        // Суммарная длительность выполнения операции предварительной токарной
обработки
        $t1i = $to1i + $tv1i;

        // Возврат суммы основной и вспомогательной длительности обработки
        return [
            'main_time' => ceil($t1i),
            'auxiliary_time' => ceil($tv1i),
        ];
    }

    /**
     * Расчет длины обрабатываемой поверхности детали с учетом врезания и пробега
     */
    private function lengthDetail(
        float $li, // Длина обрабатываемой поверхности по чертежу (обработка
поврежденной поверхности)
        float $yi // Величина врезания и перебега для детали
    ): float {
        return $li + $yi;
    }

    /**
     * Расчет нового диаметра после обработки
     */
    private function calculateNewDiameter(
    float $initialDiameter, // Начальный диаметр
    array $wears // Массив повреждений
) {
    // Рассчитываем новый диаметр после токарной обработки
    $totalWear = array_sum($wears) / count($wears);
    $this->damageDiameter = $initialDiameter - 2 * $totalWear;
}

    /**
     * Расчет времени электродуговой наплавки
     */
    public function electricityWelding(
        float $k2i = 1 // Количество проходов при наплавке
    ): array {
        $density = 7.81; // Плотность проволоки (г/см³)
        $lengthDetail = $this->li; // Длина обрабатываемой поверхности детали с
учетом врезания и перебега
        $dpri = $this->diameter - $this->damageDiameter; // Диаметр проволоки (мм)

        // Сварочный ток (А)
        $isv2i = 40 * pow($this->diameter, 1/3);

        // Коэффициент наплавки (г/(А*ч))
```

```
        $k2ni = 2.3 + 0.065 * pow($isv2i, 2) / $dpri;

        // Площадь поперечного сечения наплавленного валика (см²)
        $F = 0.06; // Допустим фиксированное значение для dпрэi = 1.2 - 2.0 мм

        // Скорость наплавки (м/ч)
        $v2ei = ($k2ni * $isv2i) / (100 * $F * $density);

        // Частота вращения детали (об/мин)
        $n2i = (1000 * $v2ei) / (60 * pi() * $this->diameter);

        // Шаг наплавки детали (мм/об)
        $s2i = $this->stepWelding($dpri, 2, 2.5);

        // Длительность основной операции по восстановлению детали электродуговой
наплавкой (мин)
        $to2i = ($k2i * $lengthDetail) / ($n2i * $s2i);

        // Вспомогательное время (15% от основного времени)
        $tv2i = $to2i * 0.15;

        // Подготовительно-заключительное время (мин)
        $preparatoryTime = 16;

        return [
            'main_time' => ceil($to2i + $tv2i),
            'auxiliary_time' => ceil($tv2i),
        ];

    }

    /**
     * Вычисление шага наплавки
     */
    private function stepWelding(float $dpri, float $minFactor, float $maxFactor):
float {
        $factor = mt_rand($minFactor * 100, $maxFactor * 100) / 100; // Случайный
коэффициент в диапазоне от minFactor до maxFactor
        return $factor * $dpri;
    }

    /**
     * Расчет времени вибродуговой наплавки
     */
    public function vibroWelding(
        float $k2i = 2, // Количество проходов при наплавке
        float $h2vi = 3 // Толщина наплавляемого слоя
    ): array {
        $nu = 0.8 / 0.9; // Коэффициент перехода
        $alpha = 0.7 / 0.85; // Коэффициент отклонения толщины

        $dpri = $this->diameter - $this->damageDiameter; // Диаметр проволки
        $ui = $this->voltage($h2vi); // Напряжение
        $isv2i = $this->electricity($dpri); // Сварочный ток
        $s2i = $this->stepWelding($dpri, 1.6, 2.2); // Шаг наплавки
        $vni = (0.1 * $isv2i * $ui * pow($dpri, 2)); // Величина подачи проволки

        $v2vi = (0.785 * pow($dpri, 2) * $vni * $nu) / ($h2vi * $s2i * $alpha); //
Скорость наплавки
        $n2i = (1000 * $v2vi) / (60 * pi() * $this->diameter); // Частота вращения
детали

        $to2i = ($k2i * $this->li) / ($n2i * $s2i); // Основное время
```

```
        $tv2i = ($to2i * 15) / 100; // Вспомогательное время

        return [
            'main_time' => ceil($to2i + $tv2i),
            'auxiliary_time' => $tv2i
        ];
    }

    /**
     * Расчет сварочного тока
     */
    private function electricity(float $dpri): float {
        $factor = mt_rand(60 * 100, 75 * 100) / 100;
        return $factor * $dpri;
    }

    /**
     * Расчет напряжения
     */
    private function voltage(float $h2vi): float {
        $ui = 0.0;

        if ($h2vi >= 1.0 && $h2vi <= 1.5) {
            $ui = 15 / 20;
        } elseif ($h2vi >= 2.0 && $h2vi <= 2.5) {
            $ui = 20 / 25;
        } else {
            $ui =  30 / 40;
        }

        return $ui;
    }

    /**
     * Расчет времени 2-3 токарной обработки
     */

    public function secondTurningTime(): array {
        $turningTime = $this->turningTime();
        $faceTurning = $this->faceTurning();

        $main_time_total = $turningTime['main_time'] + $faceTurning['main_time'];
        $auxiliary_time_total = $turningTime['auxiliary_time'] +
$faceTurning['auxiliary_time'];

        return [
            'main_time' => $main_time_total,
            'auxiliary_time' => $auxiliary_time_total,
        ];
    }

    private  function faceTurning(
        int $ki = 1,      // Количество проходов для снятия припуска при глубине
резания
        float $s1i = 0.5, // Подача режущего инструмента
        float $v1i = 50,  // Скорость резания
        float $tv2i = 3,  // Длительность вспомогательных операций токарной обработки
        float $yi = 3.5  // Величина врезания и перебега для детали
        ): array
    {
        $li2 = $this->diameter / 2;
        $Li2 = $li2 / + $yi;
```

```
        $n1i = 318 * ($v1i / $this->diameter); // Число оборотов деталей в минуту

        // Расчет длительности основной токарной обработки
        $to1i = ($Li2 * $ki) / ($n1i * $s1i);

        return [
            'main_time' => ceil($to1i + $tv2i),
            'auxiliary_time' => ceil($tv2i),
        ];
    }

    /**
     * Расчет времени шлифования
     */
    public function grinding(
        float $snp4i = 0.4 // Продольная подача инструментов
    ): array {
        $ki4 = rand(4, 10); // Количество проходов
        $v4i = rand(400, 600) / 100; // Скорость шлифования деталей
        $n4i = 318 * ($v4i / $this->diameter); // число оборотов детали
        $k = rand(1.2, 1.7) / 100; // коэффициент зачистных ходов

        // Вспомогательное время
        $tv4i = rand(270, 340) / 100;
        // Основное время
        $to4i = (($this->li * $ki4) / ($n4i * $snp4i)) * $k;

        return [
            'main_time' => ceil($to4i + $tv4i),
            'auxiliary_time' => ceil($tv4i)
        ];
    }
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class ExcelData extends Model
{
    use HasFactory;

    protected $fillable = [
        'sheet_name',
        'data',
    ];

    protected $casts = [
        'data' => 'array',
    ];
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Operation extends Model
{
    use HasFactory;
```

```
    protected $table = 'operations';

    protected $fillable = [
        'detail_id', 'type', 'main_time', 'auxiliary_time'
    ];

    public function detail()
    {
        return $this->belongsTo(Detail::class, 'detail_id');
    }
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Order extends Model
{
    use HasFactory;

    protected $table = 'orders';

    protected $fillable = [
        'number_of_details',
        'number_of_devices',
        'J_parameter',
        'T_l_w',
        'A_w_i',
        'P_l_w',
    ];

    public function details()
    {
        return $this->belongsToMany(Detail::class);
    }
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class TechnologicalProcess extends Model
{
    use HasFactory;

    protected $table = 'technological_processes';

    protected $fillable = [
        'detail_id',
        'operations',
    ];

    public function detail()
    {
        return $this->belongsTo(Detail::class);
    }
}
<?php
```

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class TechnologicalProcess extends Model
{
   use HasFactory;

   protected $table = 'technological_processes';

   protected $fillable = [
       'detail_id',
       'operations',
   ];

   public function detail()
   {
       return $this->belongsTo(Detail::class);
   }
}
<nav class="navbar navbar-expand-lg navbar-light bg-light">
   <div class="container-fluid">
       <div class="collapse navbar-collapse" id="navbarNav">
           <ul class="navbar-nav">
               <li class="nav-item"><a class="nav-link" href="{{ route('main.index')
}}">Главная</a></li>
               <li class="nav-item"><a class="nav-link" href="{{
route('detail.allDetails') }}">Справочник деталей</a></li>
               <li class="nav-item"><a class="nav-link" href="{{
route('detail.create') }}">Добавить деталь</a></li>
               <li class="nav-item"><a class="nav-link" href="{{
route('order.create') }}">Создать заявку</a></li>
               <li class="nav-item"><a class="nav-link" href="{{
route('order.index') }}">Все заявки</a></li>
               <li class="nav-item"><a class="nav-link" href="{{
route('upload.form') }}">Загрузить расписание</a></li>
               <li class="nav-item"><a class="nav-link" href="{{ route('data.list')
}}">Загруженные расписания</a></li>
           </ul>
       </div>
   </div>
</nav>
<!DOCTYPE html>
<html>
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
   <title>Добавить деталь</title>
   <script>
       function updateWearFields() {
           const wearCount = document.getElementById('wearsCount').value;
           const container = document.getElementById('wear-container');
           container.innerHTML = '';

           for (let i = 0; i < wearCount; i++) {
               const formGroup = document.createElement('div');
               formGroup.classList.add('form-group');

               const label = document.createElement('label');
```

```
                label.innerText = `Участок ${i + 1}:`;
                const input = document.createElement('input');
                input.type = 'number';
                input.step = '0.1';
                input.name = 'wear_areas[]';
                input.classList.add('form-control');
                input.required = true;
                formGroup.appendChild(label);
                formGroup.appendChild(input);
                container.appendChild(formGroup);
            }
        }
    </script>
</head>
<body>
@include('layout.menu')

<div class="container mt-5">
    <h2 class="mb-4">Добавить деталь</h2>
    <form action="{{ route('detail.store') }}" method="POST">
        @csrf
        <div class="form-group">
            <label for="name">Название:</label>
            <input type="text" class="form-control" id="name" name="name" required>
        </div>
        <div class="form-group">
            <label for="initialDiameter">Изначальный диаметр:</label>
            <input type="number" step="0.1" class="form-control" id="initialDiameter"
name="initialDiameter" required>
        </div>
        <div class="form-group">
            <label for="type">Тип:</label>
            <input type="number" class="form-control" id="type" name="type" required>
        </div>
        <div class="form-group">
            <label for="wearsCount">Количество участков повреждений:</label>
            <input type="number" class="form-control" id="wearsCount"
name="wearsCount" value="1" min="1" onchange="updateWearFields()" required>
        </div>
        <div id="wear-container">
            <div class="form-group">
                <label>Участок 1:</label>
                <input type="number" step="0.1" class="form-control"
name="wear_areas[]" required>
            </div>
        </div>
        <div class="form-group">
            <label for="wear">Износ</label>
            <input type="text" class="form-control" id="wear" name="wear"
pattern="[0-9]*([.,][0-9]+)?" title="Введите число (можно десятичное)">
        </div>
        <button type="submit" class="btn btn-primary">Добавить</button>
    </form>
</div>

</body>
</html>

<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
```

```
   <title>Детали</title>
</head>
<body>
@include('layout.menu')

<div class="container mt-5">
   <h1 class="mb-4">Заявки</h1>
   <table class="table table-striped table-bordered">
       <thead class="thead-dark">
       <tr>
           <th>Номер</th>
           <th>Количество приборов</th>
           <th>Количество деталей</th>
       </tr>
       </thead>
       <tbody>
       @foreach ($orders as $order)
           <tr>
               <td>{{ $order->id }}</td>
               <td>{{ $order->number_of_devices }}</td>
               <td>{{ $order->number_of_details }}</td>
           </tr>
       @endforeach
       </tbody>
   </table>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.3/dist/umd/popper.min.js"></scr
ipt>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></scrip
t>
</body>
</html>
<!DOCTYPE html>
<html>
<head>
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
   <title>Детали</title>
   <script>
       document.addEventListener('DOMContentLoaded', (event) => {
           const availableTypes = @json($availableTypes);

           const filterInput = document.getElementById('filterType');
           filterInput.addEventListener('input', function() {
               validateType(this.value, availableTypes);
               filterDetails(this.value);
           });

           function validateType(value, types) {
               if (!types.includes(value) && value !== '') {
                   filterInput.setCustomValidity('Тип не существует');
               } else {
                   filterInput.setCustomValidity('');
               }
           }

           function filterDetails(filterValue) {
               const rows = document.querySelectorAll('tbody tr');
               rows.forEach(row => {
```

```
                    const type = row.querySelector('td:nth-child(3)').textContent;
                    if (type.includes(filterValue) || filterValue === '') {
                        row.style.display = '';
                    } else {
                        row.style.display = 'none';
                    }
                });
            }
        });
    </script>
</head>
<title>Все детали</title>
</head>
<body>
@include('layout.menu')

<div class="container mt-5">
    <h1 class="mb-4">Все детали</h1>

    <!-- Поле для фильтрации -->
    <div class="form-group">
        <label for="filterType">Фильтр по типу:</label>
        <input type="text" class="form-control" id="filterType" list="typeOptions">
        <datalist id="typeOptions">
            @foreach ($availableTypes as $type)
                <option value="{{ $type }}"></option>
            @endforeach
        </datalist>
    </div>

    <table class="table table-striped table-bordered">
        <thead class="thead-dark">
        <tr>
            <th>Название</th>
            <th>Диаметр</th>
            <th>Тип</th>
        </tr>
        </thead>
        <tbody>
        @foreach ($details as $detail)
            <tr>
                <td>{{ $detail['name'] }}</td>
                <td>{{ $detail['diameter'] }}</td>
                <td>{{ $detail['type'] }}</td>
            </tr>
        @endforeach
        </tbody>
    </table>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.3/dist/umd/popper.min.js"></scr
ipt>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></scrip
t>
</body>
</html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
   <title>Excel Data List</title>
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
@include('layout.menu')
<div class="container mt-5">
   <h2>Excel Data List</h2>
   <table class="table table-bordered">
       <thead>
       <tr>
           <th>ID</th>
           <th>Номер листа</th>
           <th>Действия</th>
       </tr>
       </thead>
       <tbody>
       @foreach($excelData as $data)
           <tr>
               <td>{{ $data->id }}</td>
               <td>{{ $data->sheet_name }}</td>
               <td>
                   <a href="{{ route('data.show', $data->id) }}" class="btn
btn-primary">View</a>
               </td>
           </tr>
       @endforeach
       </tbody>
   </table>
</div>
</body>
</html>
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
   <title>Добро пожаловать!</title>
</head>
<body>
<div class="container mt-5">
   <h1 class="mb-4">Добро пожаловать!</h1>
   <p class="lead">Выберите один из разделов ниже, чтобы начать работу:</p>
   <ul class="list-group">
       <li class="list-group-item"><a href="{{ route('detail.allDetails')
}}">Справочник деталей</a></li>
       <li class="list-group-item"><a href="{{ route('detail.create') }}">Добавить
деталь</a></li>
       <li class="list-group-item"><a href="{{ route('order.create') }}">Создать
заявку</a></li>
       <li class="list-group-item"><a href="{{ route('order.index') }}">Все
заявки</a></li>
       <li class="list-group-item"><a href="{{ route('upload.form') }}">Загрузить
расписание</a></li>
       <li class="list-group-item"><a href="{{ route('data.list') }}">Загруженные
расписания</a></li>
   </ul>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
```

```
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.3/dist/umd/popper.min.js"></scr
ipt>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></scrip
t>
</body>
</html>

<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
   <title>Оформить заявку</title>
   <script>
       function updateDetailFields() {
           var numberOfDetails = document.getElementById('number_of_details').value;
           var detailFieldsContainer =
document.getElementById('detail_fields_container');
           detailFieldsContainer.innerHTML = '';

           for (var i = 0; i < numberOfDetails; i++) {
               var formGroup = document.createElement('div');
               formGroup.classList.add('form-group');

               var label = document.createElement('label');
               label.innerText = 'Выберите деталь ' + (i + 1) + ':';
               label.setAttribute('for', 'detail_select_' + i);

               var select = document.createElement('select');
               select.name = 'detail_ids[]';
               select.classList.add('form-control', 'detail-select');
               select.setAttribute('data-index', i);
               select.id = 'detail_select_' + i;
               select.onchange = updateSelectOptions;

               @foreach ($details as $detail)
               var option = document.createElement('option');
               option.value = '{{ $detail->id }}';
               option.text = '{{ $detail->name }}';
               select.appendChild(option);
               @endforeach

               formGroup.appendChild(label);
               formGroup.appendChild(select);
               detailFieldsContainer.appendChild(formGroup);
           }
       }

       function updateSelectOptions() {
           var selects = document.getElementsByClassName('detail-select');
           var selectedValues = Array.from(selects).map(select => select.value);

           Array.from(selects).forEach(select => {
               var currentValue = select.value;
               select.innerHTML = '';

               @foreach ($details as $detail)
               var option = document.createElement('option');
               option.value = '{{ $detail->id }}';
```

```
                option.text = '{{ $detail->name }}';
                if (!selectedValues.includes(option.value) || option.value ===
currentValue) {
                    select.appendChild(option);
                }
                @endforeach

                    select.value = currentValue;
            });
        }

        function updateNumberOfDetailsField() {
            var numberOfDetailsInput = document.getElementById('number_of_details');
            var hiddenInput = document.getElementById('hidden_number_of_details');
            hiddenInput.value = numberOfDetailsInput.value;
        }

        document.addEventListener('DOMContentLoaded', function () {
            document.getElementById('number_of_details').addEventListener('change',
updateNumberOfDetailsField);
        });
    </script>
</head>
<body>
@include('layout.menu')

<div class="container mt-5">
    <h1 class="mb-4">Оформить заявку</h1>

    <form action="{{ route('order.store') }}" method="POST">
        @csrf
        <div class="form-group">
            <label for="number_of_details">Количество деталей:</label>
            <input type="number" class="form-control" id="number_of_details"
name="number_of_details" min="2" onchange="updateDetailFields()" required>
            <input type="hidden" id="hidden_number_of_details"
name="hidden_number_of_details">
        </div>
        <div class="form-group" id="detail_fields_container"></div>
        <button type="submit" class="btn btn-primary">Отправить заявку</button>
    </form>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.3/dist/umd/popper.min.js"></scr
ipt>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></scrip
t>
</body>
</html>
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Результаты заявки</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
@csrf
@include('layout.menu')
```

```
<div class="container mt-5">
   <h1 class="mb-4">Результаты заявки</h1>

   @if (session('success'))
       <div class="alert alert-success" role="alert">
           {{ session('success') }}
       </div>
   @endif

   <div class="card mb-4">
       <div class="card-body">
           <h2 class="card-title">Результаты расчетов</h2>
           <p class="card-text"><strong>Количество деталей:</strong> {{
$order->number_of_details }}</p>
           <p class="card-text"><strong>Количество приборов:</strong> {{
$order->number_of_devices }}</p>
           <p class="card-text"><strong>J:</strong> {{ $order->J_parameter }}</p>
       </div>
   </div>

   <div class="card mb-4">
       <div class="card-body">
           <h2 class="card-title">Детали:</h2>
           <ul class="list-group list-group-flush">
               @foreach ($details as $detail)
                   <li class="list-group-item">{{ $detail->name }}</li>
               @endforeach
           </ul>
       </div>
   </div>

   <div class="card mb-4">
   <a href="{{ route('order.create') }}" class="btn btn-primary mb-2">Создать новую
заявку</a>
   <a href="{{ route('order.download', ['id' => $order->id]) }}" class="btn
btn-secondary">Выгрузить результаты</a>
   </div>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.3/dist/umd/popper.min.js"></scr
ipt>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></scrip
t>
</body>
</html>
<!-- showData.blade.php -->

<h2>Лист: {{ $excelData->sheet_name }}</h2>
<table class="table table-bordered">
   <thead>
   <tr>
       @foreach(array_keys($excelData->data[0]) as $header)
           <th>{{ $header }}</th>
       @endforeach
   </tr>
   </thead>
   <tbody>
   @foreach($excelData->data as $row)
       <tr>
```

```
            @foreach($row as $cell)
                <td>{{ $cell }}</td>
            @endforeach
        </tr>
    @endforeach
    </tbody>
</table>
<a href="{{ route('data.list') }}" class="btn btn-secondary">Вернуться к списку</a>
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Загрузить файл</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
@include('layout.menu')
<div class="container mt-5">
    <h2>Загрузить файл с данными расширения xlsx</h2>
    <form action="{{ route('upload.process') }}" method="POST"
enctype="multipart/form-data">
        @csrf
        <div class="form-group">
            <label for="file">Выберите файл</label>
            <input type="file" class="form-control" id="file" name="file" required>
        </div>
        <button type="submit" class="btn btn-primary">Загрузить</button>
    </form>
</div>
</body>
</html>

<?php

use App\Http\Controllers\DetailController;
use App\Http\Controllers\MainController;
use App\Http\Controllers\OrderController;
use Illuminate\Support\Facades\Route;

Route::get('/', [MainController::class, 'index'])->name('main.index');
Route::get('/details', [DetailController::class,
'allDetails'])->name('detail.allDetails');
Route::get('/detail/create', [DetailController::class,
'create'])->name('detail.create');
Route::post('/detail', [DetailController::class, 'store'])->name('detail.store');
Route::get('/submit-order', [OrderController::class,
'create'])->name('order.create');
Route::post('/submit-order', [OrderController::class,
'store'])->name('order.store');
Route::get('/order/{id}/results', [OrderController::class,
'show'])->name('order.show');
Route::get('/orders', [OrderController::class, 'index'])->name('order.index');
Route::get('/order/{id}/download', [OrderController::class,
'download'])->name('order.download');
Route::get('/upload', [OrderController::class,
'showUploadForm'])->name('upload.form');
Route::post('/upload', [OrderController::class,
'uploadAndProcessFile'])->name('upload.process');
Route::get('/data', [OrderController::class, 'listData'])->name('data.list');
Route::get('/data/{id}', [OrderController::class,
'showinfoData'])->name('data.show');
```