

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

**Автоматизированные средства рационального
раскроя линейных и плоских материалов на
исходные заготовки**

Студент

(подпись, дата) Рябцев Никита Олегович
(ФИО полностью)

Руководитель

(подпись, дата) Шибанов С.В.
(фамилия, инициалы)

Нормоконтролёр

(подпись, дата) Такташкин Д.В.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись) Козлов А.Ю.
(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись) Попова Н.А.
(фамилия, инициалы)

Пенза, 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«25» декабря 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Рябцев Никита Олегович гр. 20ВИ2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Автоматизированные средства рационального раскроя линейных и плоских материалов на исходные заготовки

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

2. Руководитель работы Шибанов Сергей Владимирович

3. Задание на работу (назначение разработки, исходные данные и т.п.) _____

Назначение: автоматизированные средства для рационального раскроя линейных и плоских материалов предназначены для решения задач раскроя с целью минимизации используемых ресурсов, используемых в процессе производства.

Средства выполнения: Операционная система Windows 10

Средства разработки: WebStorm, Pycharm, Electron JS, React JS, Fast API Язык программирования: JavaScript, Python

Технологии разработки: клиент-серверная архитектура, компоненты UI, UML

4. Перечень подлежащих разработке вопросов _____

Анализ предметной области и поставка задачи на разработку
Планирование разработки и расчет бюджета
Разработка автоматизированных средств рационального раскроя линейных и плоских материалов на исходные заготовки
Тестирование автоматизированных средств рационального раскроя линейных и плоских материалов на исходные заготовки

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Дата выдачи задания «21» декабря 2023г.

Руководитель бакалаврской работы _____
 (подпись, дата)

Задание к исполнению принял студент _____
 (подпись, дата)

Работу к защите допустить
 Декан ФВТ _____ Л.Р. Фионова
 (подпись, дата)

Реферат

Пояснительная записка содержит 116 листов, 13 таблиц, 45 рисунков, 31 использованный источник и 2 приложения.

РАЦИОНАЛЬНЫЙ РАСКРОЙ МАТЕРИАЛОВ, JAVASCRIPT, PYTHON, UML, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА.

Объект исследования: автоматизированные средства рационального раскроя линейных и плоских материалов на исходные заготовки.

Цель работы: реализовать десктопную версию автоматизированных средств рационального раскроя линейных и плоских материалов на исходные заготовки.

Технологии разработки: клиент-серверная архитектура, компоненты UI, React JS, Electron JS, JavaScript, Python, UML, Fast API.

Задание: спроектировать и реализовать десктопную версию автоматизированных средств рационального раскроя линейных и плоских материалов.

Результат работы: спроектирована и разработана десктопная версия автоматизированных средств рационального раскроя линейных и плоских материалов

					ПГУ 09.03.02 – 8БР202.08 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Рябцев Н. О.			«Автоматизированные средства рационального раскроя линейных и плоских материалов на исходные заготовки» Пояснительная записка.	Лит.	Лист	Листов
Провер.		Шибанов С. В.					4	116
Реценз.						группа 20ВИ2		
Н. Контр.		Такташкин Д. В.						
Утверд.		Козлов А. Ю.						

СОДЕРЖАНИЕ

Введение	8
1 Анализ предметной области и постановка задачи на разработку автоматизированных средств раскроя линейных и плоских материалов	10
1.1 Автоматизация раскроя линейных и плоских материалов.....	10
1.1.1 Классификация материалов для раскроя.....	10
1.1.2 Критерии оценки качества результатов раскроя материалов	11
1.1.3 Задачи автоматизированного раскроя линейных и плоских материалов	12
1.2 Сравнительный анализ приложений-аналогов	13
1.2.1 Цели и задачи сравнительного анализа	13
1.2.2 Выбор приложений-аналогов для анализа	14
1.2.3 Обзор приложений аналогов.....	15
1.2.4 Результаты сравнительного анализа.....	18
1.3 Постановка задачи на разработку автоматизированных средств раскроя материалов	20
1.3.1 Цели разработки и назначение автоматизированных средств раскроя	20
1.3.2 Требования к аппаратно-программному окружению	21
1.3.3 Автоматизируемые бизнес-процессы.....	22
1.4 Планирование процесса разработки программного обеспечения.....	25
2 Проектирование автоматизированных средств раскроя линейных и плоских материалов.....	29
2.1 Моделирование функциональных возможностей.....	29
2.2 Разработка архитектуры программного обеспечения.....	33
2.3 Прототипирование пользовательского интерфейса.....	37
2.4 Обоснование выбора средств разработки	41

2.4.1 Обоснование выбора средств автоматизации управления проектом	41
2.4.2 Обоснование выбора инструментальных средств разработки программного обеспечения	44
2.4.3 Обоснование выбора языковых средств разработки.....	47
3 Разработка программного обеспечения автоматизированных средств раскрытия линейных и плоских материалов.....	52
3.1 Автоматизированное управление проектом	52
3.2 Разработка клиентской части приложения	62
3.2.1 Разработка пользовательского интерфейса клиентского приложения	62
3.2.2 Разработка программного обеспечения клиентского приложения	65
3.3 Разработка серверной части приложения.....	67
3.4 Разработка программной реализации алгоритмов раскрытия материалов.....	70
4 Контроль качества программного обеспечения автоматизированных средств раскрытия линейных и плоских материалов	74
4.1 Планирование процесса контроля качества программного обеспечения	74
4.1.1 Мероприятия по контролю качества программного обеспечения	74
4.1.2 Обоснование выбора модели тестирования	75
4.1.3 Обоснование выбора метрик программного кода.....	76
4.2 Тестирование программных средств.....	77
4.2.1 Тестирование с помощью юнит-тестов	77
4.2.2 Автоматическое сквозное тестирование	81
4.3 Оценка качества программного кода на основе метрик	82
4.3.1 Метрика SLOC	82

4.3.2 Метрика Холстеда.....	83
Заключение	87
Список использованных источников	88
Приложение А (рекомендуемое) Листинг программы.....	91
Приложение Б (рекомендуемое) Результаты выполнения автотестов	112

Введение

В процессе производства изделий, состоящих из различных заготовок, которые имеют определенные параметры появляется необходимость рационального использования имеющихся ресурсов с целью повышения рентабельности бизнеса. Это достигается путем оптимального раскроя материала. С помощью него можно достичь меньшего расхода исходных заготовок, эффективно использовать остаток материала, уменьшить время на изготовление изделия.

Возникает необходимость в средстве автоматизации раскроя линейных и плоских материалов, которые бы позволяли оптимизировать процесс производства с сокращением отходов.

Целью разработки является создание программного обеспечения для автоматизации раскроя линейных и плоских материалов на исходные заготовки. При этом должна будет осуществляться кроссплатформенность исходного приложения.

Для выполнения поставленной цели нужно решить следующие задачи:

- проанализировать предметную область;
- осуществить сравнительный анализ приложений-аналогов;
- выполнить постановку задачи на разработку средства для раскроя линейных и плоских материалов;
- смоделировать функциональные возможности приложения с помощью языка UML;
- выбрать средства для разработки, развертывания, тестирования и интеграции приложения;
- произвести проектирование архитектуры приложения, пользовательского интерфейса;
- разработать приложения для автоматизации раскроя линейных и плоских материалов на исходные заготовки;
- произвести непрерывное развертывание и интеграцию приложения;

- выполнить тестирование программного обеспечения.

Изучение приложений-аналогов было выполнено в статье «Обзор приложений для автоматизации линейного и двумерного раскроя твердых материалов», представленной в 2024 г. на VIII Международной научно-практической конференции студентов и молодых ученых «Модели, системы, сети в технике».

В результате выполнения работы должно быть реализовано средство для автоматизации раскроя линейных и плоских материалов на исходные заготовки.

1 Анализ предметной области и постановка задачи на разработку автоматизированных средств раскроя линейных и плоских материалов

1.1 Автоматизация раскроя линейных и плоских материалов

1.1.1 Классификация материалов для раскроя

Изделия в процессе производства изготавливаются из различных исходных заготовок. Они имеют габаритные параметры, такие как: ширина, длина, толщина, диаметр и другие. Такие заготовки создаются из разнообразного сырья, что влияет на полный процесс раскроя, так как необходимо учитывать его специфику. Не всегда один тот же инструмент может подойти к нарезанию разных материалов. К примеру, раскрой ткани будет иметь один подход, а раскрой железа другой.

Для того, чтобы определить классификацию материалов, которые используются при изготовлении деталей, нужно знать по каким признакам будут делиться исходные заготовки. Классификация материалов раскроя может быть представлена: по типу материала, по толщине материала, по жесткости материала, по размеру материала, по структуре материала, по прочности материала.

По типу материала на:

- металлические;
- деревянные;
- пластиковые;
- тканевые.

По толщине материала на:

- тонкие;
- средние;
- толстые.

По жесткости материала на:

- жесткие;

- полужесткие;
- гибкие.

По структуре материала на:

- однородные;
- слоистые;
- составные.

При линейном раскрое используются материалы из дерева, стали, пластмассы. При двумерном раскрое используются материалы из стекла, ткани, бумаги.

В дальнейшем, будет рассматриваться раскрой твердых линейных и плоских материалов: сталь, дерево, пластмасса, стекло.

1.1.2 Критерии оценки качества результатов раскроя материалов

Для оценки качества раскроя материалов необходимо определить критерии, которые исходят из особенностей исходных заготовок, процесса производства, а также используемых инструментов. Ими будут выступать: эффективность использования материалов, точность размеров, минимизация затрат и времени, учет специфических требований.

Эффективность использования материалов. Данный критерий позволяет оценить, как эффективно распределен и использован материал. Это включает минимизацию отходов и максимизацию количество деталей, вырезанных из исходного материала.

Точность размеров. Необходимо убедиться, что размеры вырезанных деталей соответствуют требуемым спецификациям. Это позволяет избежать несоответствие, что позволяет сэкономить на используемом материале.

Минимизация затрат и времени. Насколько быстро и эффективно раскроить материал, позволяет быстрее производить необходимые детали, а, следовательно, сами изделия. Это включает оптимизацию производственных процессов, например, минимальное количество перестроек режущего инструмента.

Учет специфических требований. Нужно всегда учитывать специфику конкретной задачи раскроя, где могут быть условия, меняющие раскрой. Например, где-то нужно будет учитывать толщину лезвия или угол реза. На некоторых производствах износ лезвия из-за большего числа распилов потребует дополнительные финансовые вложения на покупку нового режущего материала.

Критерии оценки качества результатов раскроя материалов во многом будут зависеть от решения задач раскроя с учетом дополнительных критериев.

1.1.3 Задачи автоматизированного раскроя линейных и плоских материалов

В зависимости от различных требований, можно выделить задачи раскроя: линейный раскрой без учета дополнительных требований, линейный раскрой с учетом угла реза и толщины лезвия, двумерный раскрой.

Задача 1. Линейный раскрой без учета дополнительных требований.

Необходимо из материалов, длиной $L_1, L_2, \dots, L_j, \dots, L_m$ в количестве $B_1, B_2, \dots, B_j, \dots, B_m$, выкроить заготовки, длиной $l_1, l_2, \dots, l_i, \dots, l_n$ в заданном количестве $p_1, p_2, \dots, p_i, \dots, p_n$, где $i = 1, 2, \dots, n$ и $j = 1, 2, \dots, m$. Требуется определить оптимальный план раскроя материала, то есть получить минимальные отходы с учетом комплектности заготовок. Частным случаем, является задача с фиксированной длиной материала L .

В качестве усложнения данной задачи может выступать:

- учет толщины лезвия;
- учет угла реза;
- остаток материала должен группироваться так, чтобы оставалась практически целый материал.

Сформулируем новую задачу раскроя, с учетом дополнительных требований.

Задача 2. Линейный раскрой с учетом угла реза и толщины лезвия.

Необходимо из материалов, длиной $L_1, L_2, \dots, L_j, \dots, L_m$ в количестве $B_1, B_2, \dots, B_j, \dots, B_m$, выкроить заготовки, длиной $l_1, l_2, \dots, l_i, \dots, l_n$ в заданном

количестве $p_1, p_2, \dots, p_i, \dots, p_n$, где $i = 1, 2, \dots, n$ и $j = 1, 2, \dots, m$. Углы реза у каждой заготовки будут: левый – А, правый – В. Толщина лезвия – s . Требуется определить оптимальный план раскроя материала, то есть получить минимальные отходы с учетом комплектности заготовок. Частным случаем, является задача с фиксированной длиной материала L .

Таким образом, были сформулированы задачи линейного раскроя твердых материалов [1].

Задача 3. Двумерный раскрой.

Необходимо из материала площадью S в количестве k , выкроить заготовки площадью $S_1, S_2, \dots, S_i, \dots, S_n$, где в заданном количестве $p_1, p_2, \dots, p_i, \dots, p_n$, где $i = 1, 2, \dots, n$ и $j = 1, 2, \dots, m$. Требуется определить оптимальный план раскроя материала, при котором будет использована наименьшее число материала площадью S .

Таким образом были сформулированы задачи раскроя линейных и плоских материалов.

1.2 Сравнительный анализ приложений-аналогов

1.2.1 Цели и задачи сравнительного анализа

Приложение для автоматизации линейного и двумерного раскроя твердых материалов представляет собой специализированное программное обеспечение, разработанное для оптимизации раскроя линейных, листовых материалов (например, дерева, пластика, металла) [2]. Они позволяют оптимизировать процессы изготовления готового изделия, сократить затраты на их производство, улучшив рентабельность предприятия. Появляется необходимость выбора того приложения, которое наиболее эффективно выполняет поставленную задачу по минимизации отходов. Также важно учитывать, такие факторы, как: удобство использование программы, дополнительный функционал, кроссплатформенность, условия лицензирования приложения и др.

В настоящее время существует большой список приложения для автоматизации раскроя [3]. Большинство не получили широкого распространения из-за высокой конкуренции. Некоторые из них устарели, имеют сложный интерфейс, имеют неудовлетворительную скорость работы для каких-то потребуется внушительное финансовое вложение. Поэтому были выбраны приложения, которые имеют большую популярность.

1.2.2 Выбор приложений-аналогов для анализа

Для сравнительного анализа были выбраны приложения-аналоги, которые в настоящее время активно используются и поддерживаются.

В таблице 1 приведен список приложений по раскрою материалов, а основные функции.

Таблица 1 – Список приложений по раскрою материалов

Наименование приложения	Основные функции	Условия лицензирования	Класс приложений	Операционная система
SmartCut Online	Линейный раскрой	Бесплатный	Web-приложение	Любая
OptCut	Линейный и двумерный раскрой	Платный	Desktop-приложение	Windows 7,8, 10
Astra S-Nesting	Линейный и двумерный раскрой	Бесплатный /Платный	Desktop-приложение	Windows 7,8,10
Cutting 3	Линейный и двумерный раскрой	Платный	Desktop-приложение	Windows 7,8

Как видно из таблицы в основном все приложения являются только десктопными под операционную систему Windows.

Таким образом были выбраны наиболее используемые в настоящее время приложения для раскроя.

1.2.3 Обзор приложений аналогов

SmartCut Online – сайт, который позволяет осуществить оптимальный раскрой линейных материалов. В основе работы лежит собственный алгоритм расчета с наилучшим полезным выходом и минимальными отходами. Имеется возможность импорта данных из Excel. Имеется API для удаленного обращения к калькулятору раскроя данного сайта, а также возможность встроить калькулятор на свой сайт. Раскрой с угловым резом считается несколько иначе, чем рез без углов и углом в 45 градусов, при этом имеется объяснение решения этой задачи. Также планируется разработка двумерного раскроя. Настольной и мобильной версии приложения нет [4]. На рисунке 1 показан интерфейс программы.

▼ Заготовки / Склад:

№	ДЛИНА ЗАГОТОВКИ	КОЛИЧЕСТВО (50)	НАЗВАНИЕ	×
1	6000	50 ∞		×
2		∞		×

≡+ ДОБАВИТЬ ЗАГОТОВКУ

▼ Отрезки:

№	ДЛИНА ОТРЕЗКА	КОЛИЧЕСТВО (29)	НАЗВАНИЕ	×
1	1590	2		×
2	2325	2		×
3	1635	2		×
4	2000	2		×
5	250	7		×
6	150	7		×
7	230	7		×
8				

≡+ ДОБАВИТЬ ОТРЕЗОК

РАСКРОЙ

Рисунок 1 – Интерфейс сайта SmartCut Online

OptCut – программное обеспечение для оптимизации линейных и прямоугольных форм различных материалов. В особенности подходит для производства мебели, лесопилок, производства окон, кузнецов, стекольщиков,

мраморщиков. Основной целью программы является свести расходы к минимуму путем повторного использования запасов предыдущих отрезков, позволяя существенно экономить материалы. Приложение учитывает: доступные материалы, детали производства, технические ограничения (толщина лезвия, возможные обороты, ограничения станка и т. д.), что позволяет генерировать одно из возможных решений, учитывающих установленные приоритеты: меньше отходов, количество попыток, количество выполняемых разрезов [5]. На рисунке 2 показан интерфейс программы.

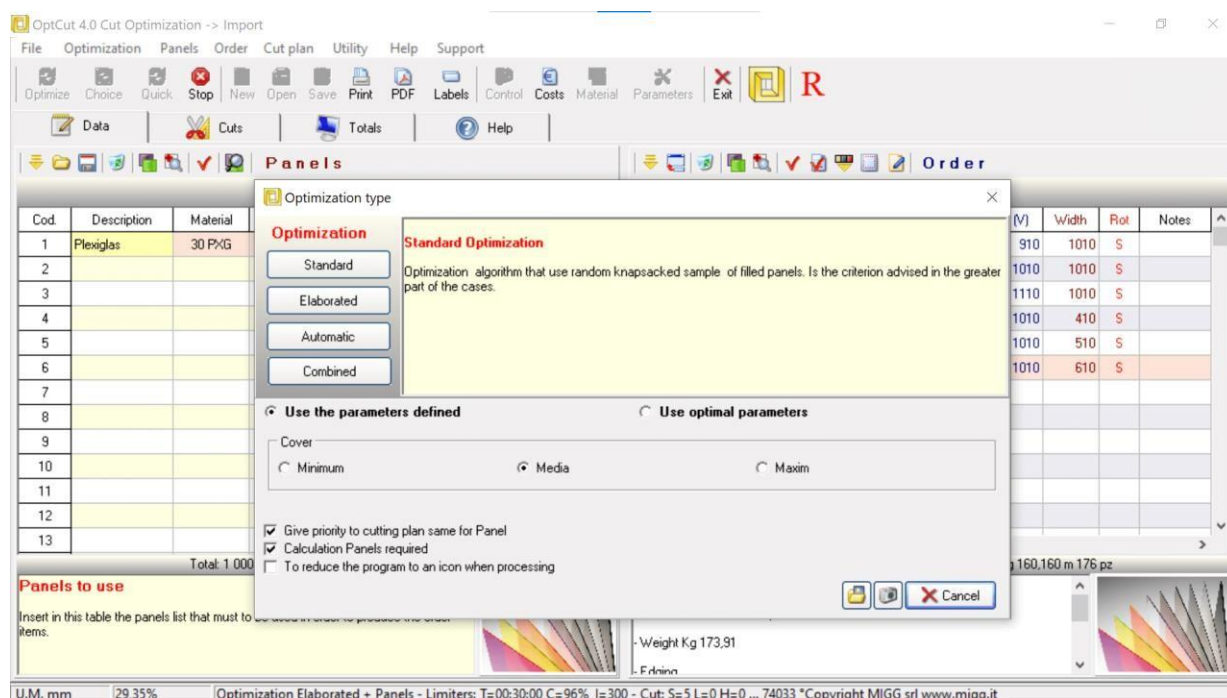


Рисунок 2 – Интерфейс программы OptCut

Astra S-Nesting – является программой, которая оптимизирует раскрой различных материалов произвольной формы. У нее есть функционал как линейного, так и двумерного раскроя. Подобную программу можно использовать на машинах тепловой резки и на фрезерных обрабатывающих станках с ЧПУ, а также для ручного раскроя материала. Astra S-Nesting – приложение с понятным и удобным интерфейсом, отзывчивым для пользователя. Импорт деталей можно осуществить из DXF-файлов. Программа позволяет управлять загруженными деталями, меня угол и расположение. В целом предоставляет широкий функционал для работы с выбранными элементами.

В основе лежит оптимальный алгоритм, достаточно быстрый, имеющий оптимальную последовательность обработки. Также можно осуществить печать по раскрою, экспорт результатов можно сделать в Excel [6]. На рисунке 3 изображен интерфейс программы Astra S-Nesting.

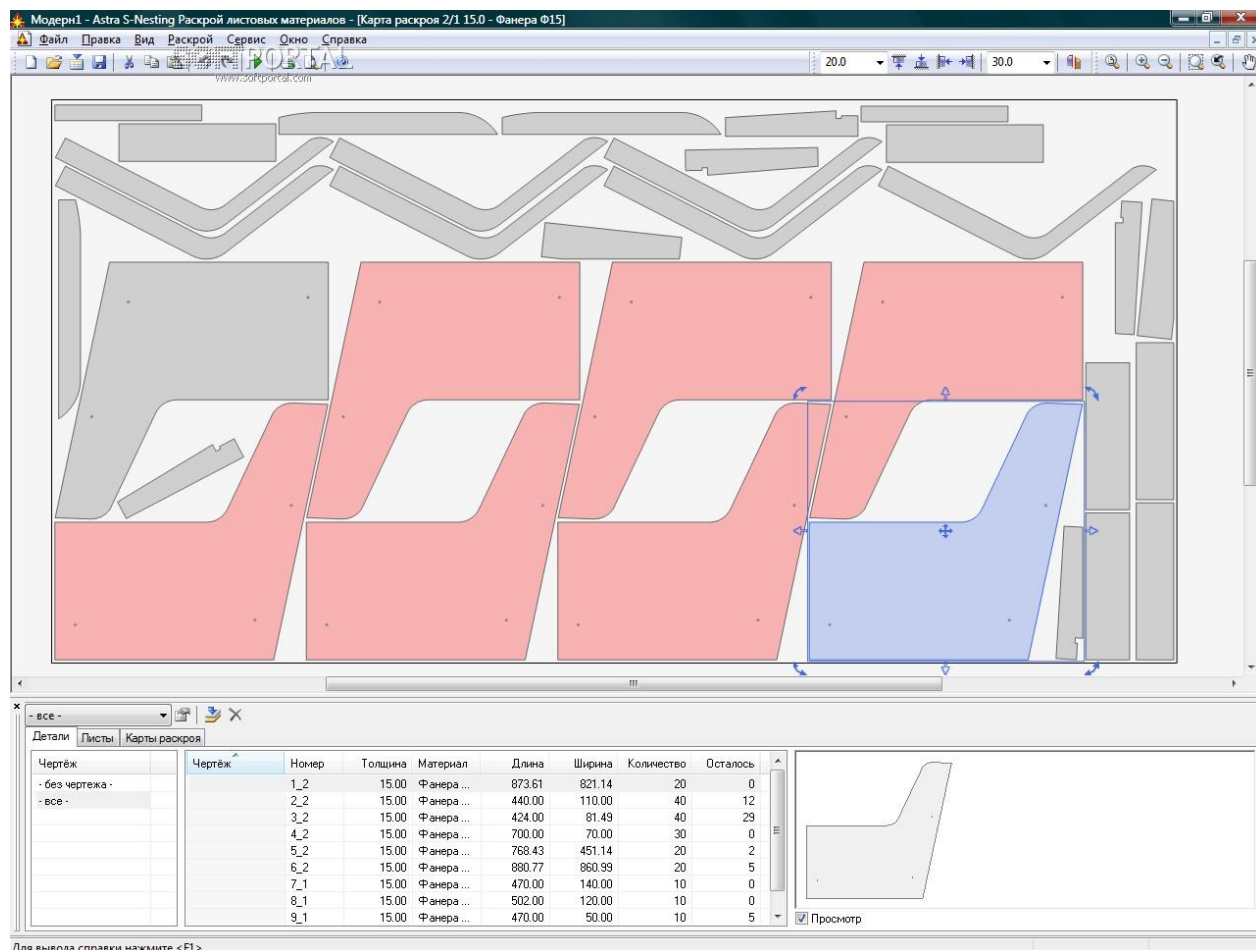


Рисунок 3 – Интерфейс программы Astra S-Nesting

Cutting 3 – это программа семейства CUTTING. Она предназначена для нахождения оптимального раскроя материала. Поддерживает и линейный, и прямоугольный раскрой. Может использоваться в таких областях производствах как: деревообработка, мебельное производство, резка металла, резка стекла. Алгоритм использует принцип минимизации отходов. Программа работает на операционных системах Windows. Сохранение результатов идет в формате Excel [7]. На рисунке 4 показан интерфейс программы.

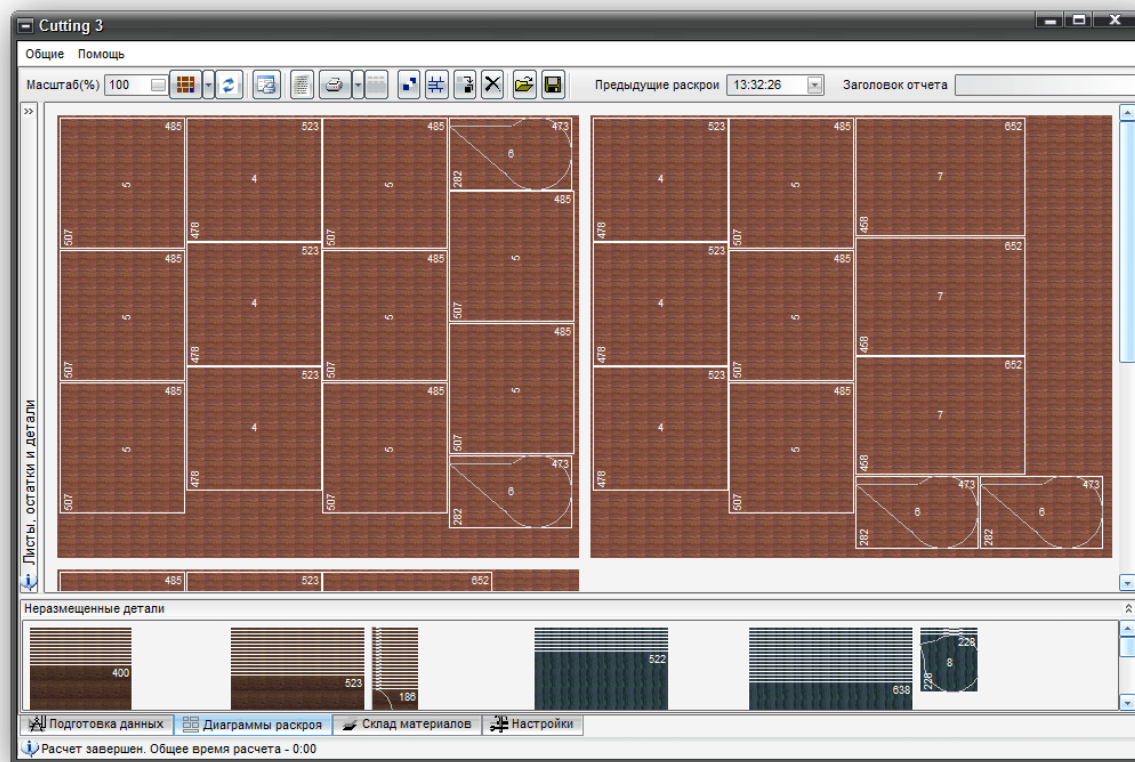


Рисунок 4 – Интерфейс программы Cutting 3

В целом представленные программы удовлетворяют основным функциям, требуемым для раскроя: поддержка как линейного, так и двумерного раскроя, экспорт результатов, а также импорт исходных данных, графическое представление. Не все программы одинаково выполняют основные функции, также необходимо ввести дополнительные критерии для оценки приложений.

1.2.4 Результаты сравнительного анализа

Выделим основные критерии для оценки программ раскроя:

- поддержка и линейного, и двумерного раскроя;
- условия лицензирования;
- удобство интерфейса;
- скорость работы;
- возможность генерации программы для ЧПУ;
- графическое представление результатов раскроя;
- экспорт результатов;

- импорт исходных данных;
- сохранение результатов раскроя для возможности повторного использования;
- кроссплатформенность.

В таблице 2 представлены сравнение программного обеспечения для раскроя.

Таблица 2 – Сравнение программного обеспечения для раскроя

Критерии оценки	Программное обеспечение			
	SmartCut Online	OptCut	Astra S-Nesting	Cutting 3
Поддержка разных видов раскроя	–	+	+	+
Условия лицензирования	Бесплатно	99\$	Ограниченная версия: бесплатно Полная версия: 40000 руб.	200\$
Удобство интерфейса	Легкий	Средний	Сложный	Сложный
Скорость работы	Медленная	Средняя	Быстрая	Быстрая
Графическое представление результатов	+	+	+	+
Возможность генерации программ для ЧПУ	–	–	+	–
Экспорт результатов	+	+	+	+
Импорт исходных данных	+	+	+	+
Кроссплатформенность	–	–	–	–
Сохранение результатов для повторного использования	–	–	–	–

Лучшим приложением с учетом выполняемых функций является Astra S-Nesting, но оно имеет высокую стоимость. Оптимальным выбором можно считать Cutting 3, так как по всем критериям оно имеет неплохие показатели. Рассмотрим недостатки, присущие всем приложениям.

Как видно из таблицы, все программы имеют недостаток в виде отсутствие кроссплатформенности. Необходимость сохранения результатов для повторного использования также может понадобиться для подбора уже готового решения для конкретного случая. Также стоит добавить, что программы не учитывают специфику инструментов для резки и возможность их износа в ходе работы. Например, может возникнуть ситуация, когда было дано решение, при котором требуется большое количество резов. Но при непрерывной работе может возникнуть необходимость замены лезвия, но это обойдется дороже, чем покупка лишнего материала для раскроя. Исходя из этого, появляется необходимость разработки программного обеспечения для линейного и двумерного раскроя твердых материалов, имеющее кроссплатформенность и учет дополнительных условий, указанных выше.

Представленный обзор позволил выявить оптимальное приложение для автоматизации линейного и двумерного раскроя твердых материалов с учетом достижения критериев эффективности. Но с учетом дополнительных условий и недостатков приведенных в статье программ для раскроя, можно сказать, что существует необходимость разработки кроссплатформенного программного обеспечения, удовлетворяющее всем приведенным условиям.

1.3 Постановка задачи на разработку автоматизированных средств раскроя материалов

1.3.1 Цели разработки и назначение автоматизированных средств раскроя

Целью разработки является приложение для автоматизированного раскроя линейных и плоских материалов. Оно будет представлять собой настольное приложение под операционные системы Windows 10, Windows 11, macOS, а также семейства Linux. В дальнейшем планируется разработка веб-версии и мобильной версии приложения.

Разрабатываемое программное обеспечение должно:

- обеспечивать линейный раскрой материалов, в двух версиях: обычный линейный раскрой и линейный раскрой с дополнительными параметрами;
- обеспечивать двумерный раскрой материалов;
- иметь возможность экспорта результатов раскроя в формат Excel;
- иметь возможность импорта исходных данных из формата Excel;
- печать результатов в формат PDF;
- отображать результаты раскроя в виде таблицы, а также в виде рисунка заготовок;
- сохранять результаты раскроя в json-файл с последующим использованием при необходимости;
- давать возможность пользователю просматривать историю раскроя;
- иметь окно справки о последовательности работы в программе.

Разрабатываемое программное обеспечение будет реализовано в виде клиент-серверного приложения в виде модульной монолитной архитектурой.

В результате получится приложение для автоматизации раскроя линейных и плоских материалов на исходные заготовки.

1.3.2 Требования к аппаратно-программному окружению

Приложение для автоматизации раскроя линейных и плоских материалов на исходные заготовки будет иметь следующие требования к программно-аппаратным средствам.

Операционные системы: Windows 10, Windows 11, macOS или операционные системы семейства Linux.

Минимальные параметры компьютера:

- процессор 32-разрядный или 64-разрядный с тактовой частотой 1.1 ГГц и более;
- оперативная память не менее 2 ГБ;
- свободное место на жестком или SSD диске не менее 1 ГБ;
- модель встроенной видеокарты Intel UHD Graphics 600 или дискретной видеокарты AMD Radeon Graphics.

Рекомендуемые параметры компьютера:

- процессор 32-разрядный или 64-разрядный с тактовой частотой 3.7 ГГц и выше;
- оперативная память 4 ГБ и выше;
- накопитель SSD со свободным местом не менее 1 ГБ;
- дискретная видеокарта AMD Radeon Graphics и любая другая выше.

Для настольной версии приложения не требуется подключения к интернету, сервер устанавливается на том же компьютере и запускается в виде службы приложения.

1.3.3 Автоматизируемые бизнес-процессы

Автоматизация бизнес-процессов является одним из главных способов сократить ресурсы в системе любого уровня: от маленького отдела компании до крупного бизнеса. Она помогает ему экономить на одинаковых, постоянно повторяющихся вещах, на которые приходится тратить время и силы. Поэтому автоматизация позволяет составлять алгоритмы действий, решающие данные проблемы.

Диаграмма деятельности представляет собой UML-диаграмму, которая описывает выполняемые действия в системе, специфицирует бизнес-процессы. Основными элементами диаграммы: начальный узел, конечный узел, действие и др.

Начальный узел – это узел, который показывает состояние, где начинается процесс, при этом он может иметь только одно начало.

Конечный узел – это узел, который показывает конечное состояние процесса, при этом процесс может иметь несколько конечных состояний.

Действие – состояние активности, на диаграмме обычно показывает вызов операции, шаг в бизнес-процессе или всего бизнес-процесса в целом.

Переход по потоку – набор действий связывают, упорядочивают действия, существует такие потоки как поток управления и поток объектов.

Узлы принятия решений – аналогично условию if, которое применяется в программировании, принимает на вход поток управления, и в зависимости от различных условий может быть выполнен переход на различные действия.

Узел разъединения – узел, которые из одного потока управления распараллеливает процесс.

Узел соединения – узел, который из нескольких потоков управления при параллельных процессах объединяет в один поток управления.

Вилка – это элемент, который указывает на одновременное начало нескольких параллельных процессов. Визуально вилка обозначается горизонтальной линией, которая раздваивается на несколько стрелок, указывающих на параллельные процессы.

Поток объекта – это элемент диаграммы деятельности, который показывает передачу объектов между действиями. Объект может быть любым конкретным элементом, таким как товар, документ или сообщение.

Объект узла – представляет собой ни что иное как объект, который создается или возникает в процессе выполнения действий [8].

В таблице 3 представлена спецификация бизнес-процесса.

Таблица 3 – Спецификация бизнес-процесса

Бизнес-процесс		Раскрой материалов
Исполнитель		Пользователь
Входные данные	Поставщик	Пользователь
	Содержание	Данные об исходном материале и требуемых заготовках
Выходные данные	Потребитель	Пользователь
	Содержание	Вариант раскроя исходных материалов на требуемые заготовки
Ограничения		Минимум один исходный материал и одна нарезаемая заготовка

На рисунке 5 показана диаграмма деятельности, где отображен раскрой материалов.

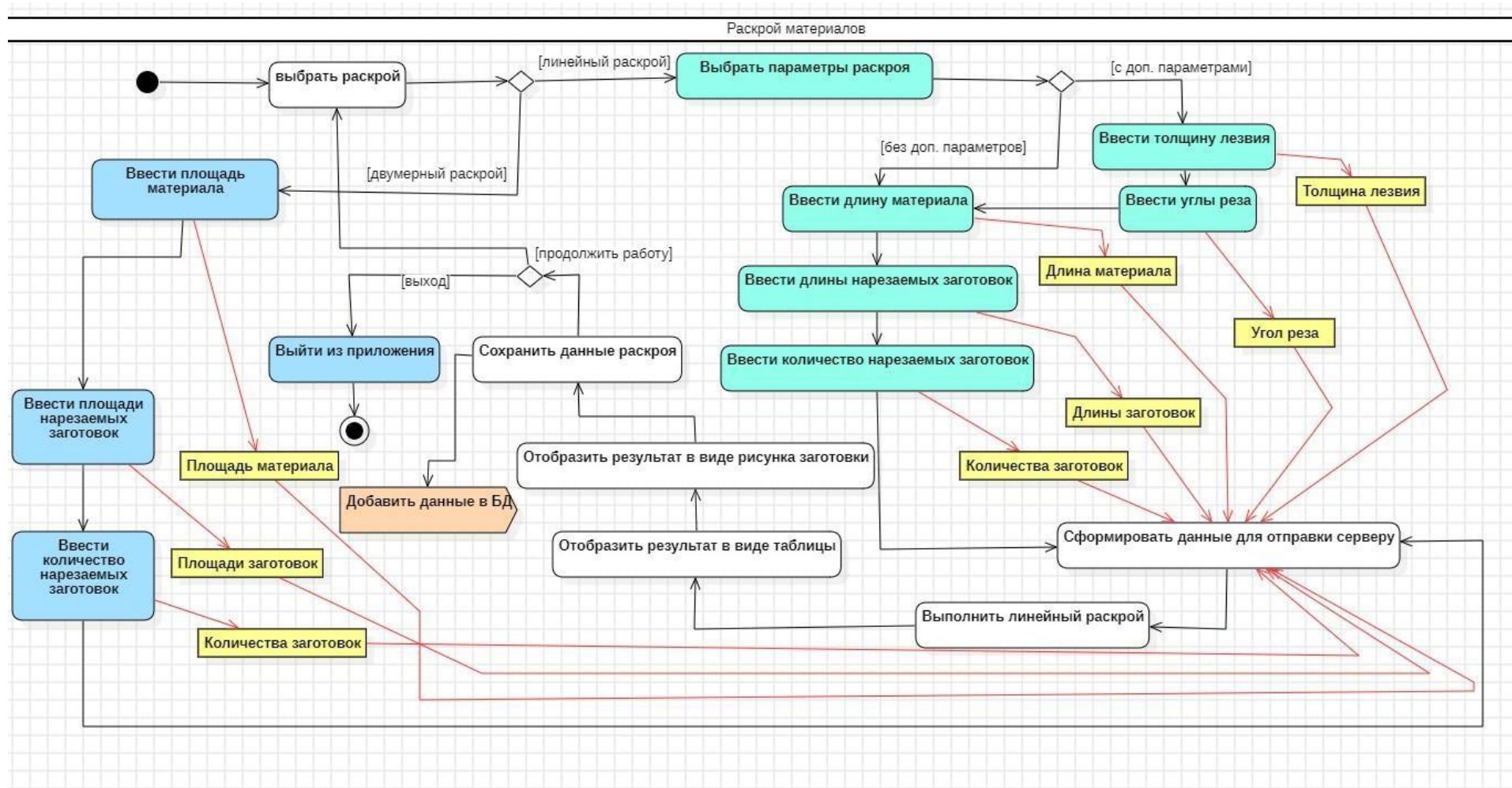


Рисунок 5 – Диаграмма деятельности процесса раскроя материала

В процессе раскроя материала можно выбрать раскрой.

Если выбрать линейный раскрой, то необходимо ввести длину исходного материала, далее вводятся длины нарезаемых заготовок и их количества. При линейном раскрое можно выбрать учет толщины лезвия и углов реза, при этом необходимо будет ввести их значения. Введенные данные формируют исходные параметры для раскроя. Далее выполняется раскрой материалов, при этом идет сохранение результатов раскроя и исходных параметров в json-файл.

Если выбрать двумерный раскрой, то необходимо будет ввести площадь исходного материала, после чего вводятся площади нарезаемых материалов и их количества. Аналогично линейному раскрою, введенные значения являются исходными для раскроя. Результат сохраняется в json-файл.

Таким образом, приложение автоматизирует раскрой линейных и плоских материалов.

1.4 Планирование процесса разработки программного обеспечения

Одним из важнейших пунктов разработки программного обеспечения является планирование процесса разработки. Особенность процесса планирования в проектном виде заключается в постоянной корректировке плана от начала до стадии завершения.

Для планирования разработки использовалась диаграмма Ганта. Эта диаграмма является одной из самых популярных для построения планирования деятельности, в данном случае для разработки приложения. На ней отражены задачи и последовательность их выполнения. На старте проекта сложно планировать: распределять задачи между сотрудниками и определять, сколько времени займёт каждый этап работ. И если неправильно распределить ресурсы, вы не успеете выполнить задачи вовремя [9].

Для построения диаграммы Ганта был выбран Microsoft Project 2016. Он позволяет отдельно создавать ресурсы и задачи, которые требуются на проект. На рисунке 6 изображены ресурсы проекта.

И	Название ресурса	Тип	Единица измерения	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочная	Затраты на использование	Начисление	Базовый календарь
	Аналитик	Трудовой		Ан		100%	110,00 Р/ч	200,00 Р/ч	0,00 Р	Пропорционально	Стандартный
	Разработчик ПО	Трудовой		П		100%	160,00 Р/ч	300,00 Р/ч	0,00 Р	Пропорционально	Стандартный
	DevOps-инженер	Трудовой		Д		100%	100,00 Р/ч	190,00 Р/ч	0,00 Р	Пропорционально	Стандартный
	Тестировщик	Трудовой		Т		100%	100,00 Р/ч	190,00 Р/ч	0,00 Р	Пропорционально	Стандартный
	Архитектор ПО	Трудовой		А		100%	120,00 Р/ч	230,00 Р/ч	0,00 Р	Пропорционально	Стандартный
	Интернет	Затраты		И						Пропорционально	

Рисунок 6 – Ресурсы проекта

При планировании проекта необходимо выделить задачи и подзадачи, которые необходимо решить. Нужно задать даты начала и окончания той или иной работы. В Microsoft Project удобно задать дни, а также выделить совокупность задач для одного шага проекта. Также после этого можно задать ресурсы для всего каждого этапа. Общая таблица, которая отображает задачи и их ресурсы, изображена на рисунке 7.

Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
Начало реализации	0 дней	Пн 04.03.24	Пн 04.03.24		
Анализ предметной области	12 дней	Пн 04.03.24	Вт 19.03.24	1	
Анализ выбранной предметной области	3 дней	Пн 04.03.24	Ср 06.03.24	1	Аналитик; Интернет
Сравнительный анализ приложений-аналогов	5 дней	Чт 07.03.24	Ср 13.03.24	3	Аналитик; Интернет; Архитектор ПО
Постановка задачи на разработку приложения	4 дней	Чт 14.03.24	Вт 19.03.24	4	Аналитик; Архитектор ПО; Интернет
Анализ предметной области завершен	0 дней	Вт 19.03.24	Вт 19.03.24	5	
Разработка требований к приложению	13 дней	Ср 20.03.24	Пт 05.04.24	2	
Выбор архитектуры приложения	2 дней	Ср 20.03.24	Чт 21.03.24	2	Архитектор ПО; Интернет
Определение функциональных требований	5 дней	Пт 22.03.24	Чт 28.03.24	8	Аналитик; Архитектор ПО; Интернет
Определение требований к архитектуре	2 дней	Пт 29.03.24	Пн 01.04.24	9	Архитектор ПО; Интернет
Определение требований к аппаратному окружению	2 дней	Вт 02.04.24	Ср 03.04.24	10	Архитектор ПО; Интернет
Выбор средств разработки приложения	2 дней	Чт 04.04.24	Пт 05.04.24	11	Архитектор ПО; Интернет
Разработка требований завершена	0 дней	Пт 05.04.24	Пт 05.04.24	12	
Разработка и сборка приложения	34 дней	Пн 08.04.24	Чт 23.05.24	7	
Разработка бизнес-логики	14 дней	Пн 08.04.24	Чт 25.04.24	7	Разработчик ПО
Разработка интерфейса	10 дней	Пт 26.04.24	Чт 09.05.24	15	Разработчик ПО
Разработка интерфейса взаимодействия	8 дней	Пт 10.05.24	Вт 21.05.24	16	Разработчик ПО
Создание сборки приложения	2 дней	Ср 22.05.24	Чт 23.05.24	17	DevOps-инженер
Разработка приложений завершена	0 дней	Чт 23.05.24	Чт 23.05.24	18	
Тестирование приложения	5 дней	Пт 24.05.24	Чт 30.05.24	14	
Выбор метода тестирования	1 день	Пт 24.05.24	Пт 24.05.24	14	Тестировщик
Проведение тестирования	4 дней	Пн 27.05.24	Чт 30.05.24	21	DevOps-инженер; Тестировщик
Завершение тестирования	0 дней	Чт 30.05.24	Чт 30.05.24	22	
Завершение проекта	14 дней	Пт 31.05.24	Ср 19.06.24	20	
Составление документации	12 дней	Пт 31.05.24	Пн 17.06.24	20	Аналитик
Интеграция приложения	1 день	Вт 18.06.24	Вт 18.06.24	25	DevOps-инженер; Архитектор ПО
Проект завершен	1 день	Ср 19.06.24	Ср 19.06.24	26	
Конец проекта	0 дней	Ср 19.06.24	Ср 19.06.24		

Рисунок 7 – Перечень задач и ресурсов проекта

После этого можно создать графическое представление календарного плана проекта. На рисунке 8 показана диаграмма Ганта.

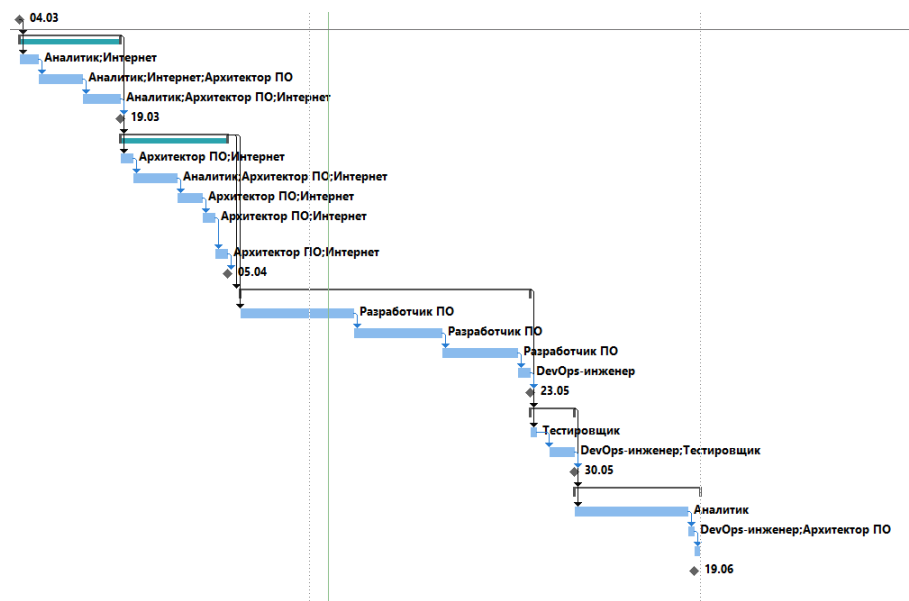


Рисунок 8 – Диаграмма Ганта

Microsoft Project имеет в своем распоряжении набор готовых отчетов и панель мониторинга, контролирующей общее состояние проекта. Также он выполняет проверку и соответствие по графику.

На рисунке 9 показано отчет по затратам.

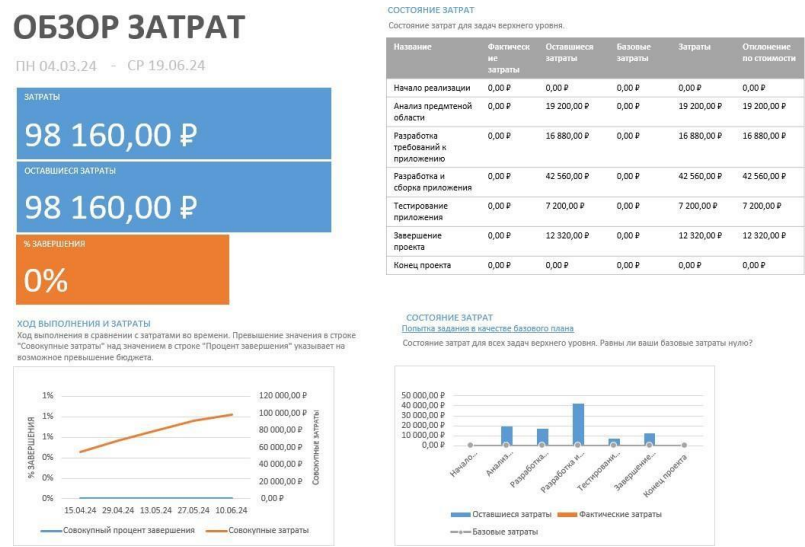


Рисунок 10 – Отчет «Обзор затрат»

По этому отчету видно, что фактические затраты реализации на приложение для автоматизации раскрытия линейных и плоских материалов составили 98 160,00 р., длительность проекта, с учетом всех дней (рабочих, праздников и выходных) составила 107 дней.

По результатам планирования можно сделать вывод, что создание приложения является экономически выгодным.

2 Проектирование автоматизированных средств раскроя линейных и плоских материалов

2.1 Моделирование функциональных возможностей

При проектировании автоматизированных средств раскроя, необходимо определить функциональные возможности программного обеспечения, исходя из постановки задачи.

Анализ предметной области и проектирование являются первыми этапами в жизненном цикле создания программного решения. Для анализа используется диаграмма вариантов использования.

Диаграмма вариантов использования (диаграмма прецедентов) – диаграмма, которая описывает отношения между пользователем и системой. UML-диаграмма use-кейсов позволяет кратко излагать сведения данные сведения. Диаграмма вариантов использования состоит из следующих компонентов:

- актер (Actor) – пользователь или то лицо, которое выполняет какие-то действия в системе;
- варианты использования (Use-cases) – действия, которые выполняет Actor;
- ассоциации – это связи между пользователем и вариантами использования;
- отношения включения – отношения между вариантами использования;
- отношения расширения – необязательная связь между вариантами использования;
- отношения обобщения – это демонстрация отношений типа «порождающая сущность/порожденная сущность»;
- границы системы – задают пределы системы для use кейсов [10].

На рисунке 11 показана диаграмма вариантов использования функциональных возможностей приложения для автоматизации раскроя.

Рисунок 11 – Диаграмма вариантов использования приложения для автоматизации раскроя

Голубым цветом выделены варианты использования, сиреневым варианты использования отношения-включения, желтым вариант использования отношения-расширения, темно-зеленым выделены варианты использования, выполняемые сервером-службой. Описание вариантов использования представлено в таблице 4.

Таблица 4 – Описание вариантов использования

Цель	Описание
1	2
Сделать линейный раскрой	Приложение обеспечивает раскрой линейных материалов
Сделать двумерный раскрой	Приложение обеспечивает раскрой листовых материалов
Просмотреть историю раскроя	Приложение позволяет просматривать историю раскроя
Импорт данных из Excel	Исходные данные можно импортировать из Excel
Ввести данные об исходном материале	В приложении можно ввести параметры исходного материала.
Ввести данные о длинах заготовок	В приложении можно ввести длины требуемых заготовок
Ввести данные о площадях заготовок	В приложении можно ввести площади требуемых заготовок
Выполнить линейный раскрой	Пользователь нажимает кнопку «Раскрой», приложение отправляет исходные данные для расчета серверу-службе
Выполнить двумерный раскрой	Пользователь нажимает кнопку «Раскрой», приложение отправляет исходные данные для расчета серверу-службе
Выполнить печать в PDF	Приложение позволяет распечатать результаты раскроя
Выполнить экспорт в Excel	Приложение позволяет сохранить результаты раскроя в формат Excel

В таблице 5 показан сценарий, а также полное описание варианта использования «Сделать линейный раскрой».

Таблица 5 – Сценарий и полное описание варианта использования «Сделать линейный раскрой»

Название пункта	Описание
Наименование	Сделать линейный раскрой
Краткое описание	Позволяет выполнить раскрой линейных материалов
Основные действующие лица	Пользователь
Второстепенные действующие лица	Нет
Предусловие	Пользователь открыл окно линейный раскрой
Основной сценарий	<ol style="list-style-type: none"> 1. Пользователь выбирает в меню окно с линейным раскром 2. Пользователь вводит данные о исходном материале 3. Пользователь вводит данные о требуемых заготовках 4. Пользователь нажимает кнопку раскроя 5. Приложение выполнит раскрой и выдаст результат в виде рисунка и таблицы
Постусловие	Приложение выдало возможный раскрой материалов

В таблице 6 показан сценарий, а также полное описание варианта использования «Сделать двумерный раскрой».

Таблица 6 – Сценарий и полное описание варианта использования «Сделать линейный раскрой»

Название пункта	Описание
Наименование	Сделать двумерный раскрой
Краткое описание	Позволяет выполнить раскрой плоских материалов
Основные действующие лица	Пользователь
Второстепенные действующие лица	Нет

Продолжение таблицы 6

Название пункта	Описание
Предусловие	Пользователь открыл окно с раскроем плоских материалов
Основной сценарий	<ol style="list-style-type: none"> 1. Пользователь выбирает в меню окно с двумерным раскроем 2. Пользователь вводит данные о исходном материале 3. Пользователь вводит данные о требуемых заготовках 4. Пользователь нажимает кнопку раскроя 5. Приложение выполнит раскрой и выдаст результат в виде рисунка и таблицы
Постусловие	Приложение выдало возможный раскрой материалов

Таким образом были смоделированы функциональные возможности приложения.

2.2 Разработка архитектуры программного обеспечения

При разработке программного обеспечения, производится анализ функциональных возможностей приложения и автоматизируемые им бизнес-процессы с целью выявления наиболее подходящей архитектуры для создаваемой системы или средства.

Архитектурные стили позволяют описать взаимоотношение отдельных компонентов, которые могут охватывать большое число архитектурных свойств. С помощью них можно рассматривать подразделить все архитектуры на монолитные и распределенные.

К монолитным относятся:

- многоуровневая архитектура;
- конвейерная архитектура;
- микроядерная архитектура.

К распределенным:

- сервисная архитектура;
- событийно-ориентированная архитектура;
- архитектура на основе пространства;
- сервис-ориентированная архитектура;
- микросервисная архитектура [11].

Одной из популярных архитектур выступает клиент-серверная архитектура.

Преимущества клиент-серверной архитектуры:

- гибкость и масштабируемость;
- легкость взаимодействия с другими системами;
- удобство обновления и поддержки.

Недостатки клиент-серверной архитектуры:

- необходимость поддержания постоянного соединения между клиентами и серверами;
- возможность возникновения узких мест при большом числе клиентов.

В качестве типов клиент-серверной архитектуры могут выступать: «тонкий клиент», «толстый клиент» и клиент-серверная сеть.

«Тонкий клиент» позволяет только отправлять запросы, получать данные от сервера. При этом подходе у клиента ограниченные возможности, а сервер производит все расчеты.

«Толстый клиент» предусматривает, что на клиентском компьютере будет производиться уже вычисления, выполнять сложные задачи независимо от сервера. К нему идет обращения только в случае запроса дополнительных данных.

Клиент-серверная сеть – ряд клиентов и один или множество серверных узлов.

Уровни, которые есть в данной архитектуре: уровень клиента, уровень сервера и уровень базы данных.

Одноуровневая архитектура. Для нее выделяют следующие особенности:

- сервер и клиент находятся на одном компьютере;
- программное обеспечение формируется и работает без промежуточных узлов;
- получение, обработка данных, вывод результатов – все операции производятся на клиентском компьютере.

Двухуровневая архитектура предполагает:

- присутствие серверного уровня между клиентом и базой данных;
- клиент отправляет запросы к серверу, где будет происходить обработка данных;
- сервер выступает в качестве компонента, отвечающего за бизнес-логику, хранение данных и взаимодействие с БД;
- клиент дает отображение и интерактивность пользовательского интерфейса.

Для трехуровневой архитектуры будет характерно:

- три взаимосвязанных уровня – клиент, сервер, база данных;
- клиент – пользовательский интерфейс;
- сервер – бизнес-логика и управление доступом к данным;
- уровень базы данных, отвечает за хранение и управления.

Многоуровневая архитектура. Такая архитектура уже будет иметь особенности:

- наличие более сложной структуры с несколькими серверными и клиентскими уровнями;
- каждый из них будет отвечает за отдельный функционал и физически разделен на различные компоненты;
- возможность гибко масштабировать систему [12].

В качестве архитектуры программного обеспечения для средства автоматизации раскроя линейных и плоских материалов использовалась локальная клиент-серверная архитектура виде тонкого клиента с одноуровневым подходом. На рисунке 12 показана диаграмма развертывания архитектуры развертывания.

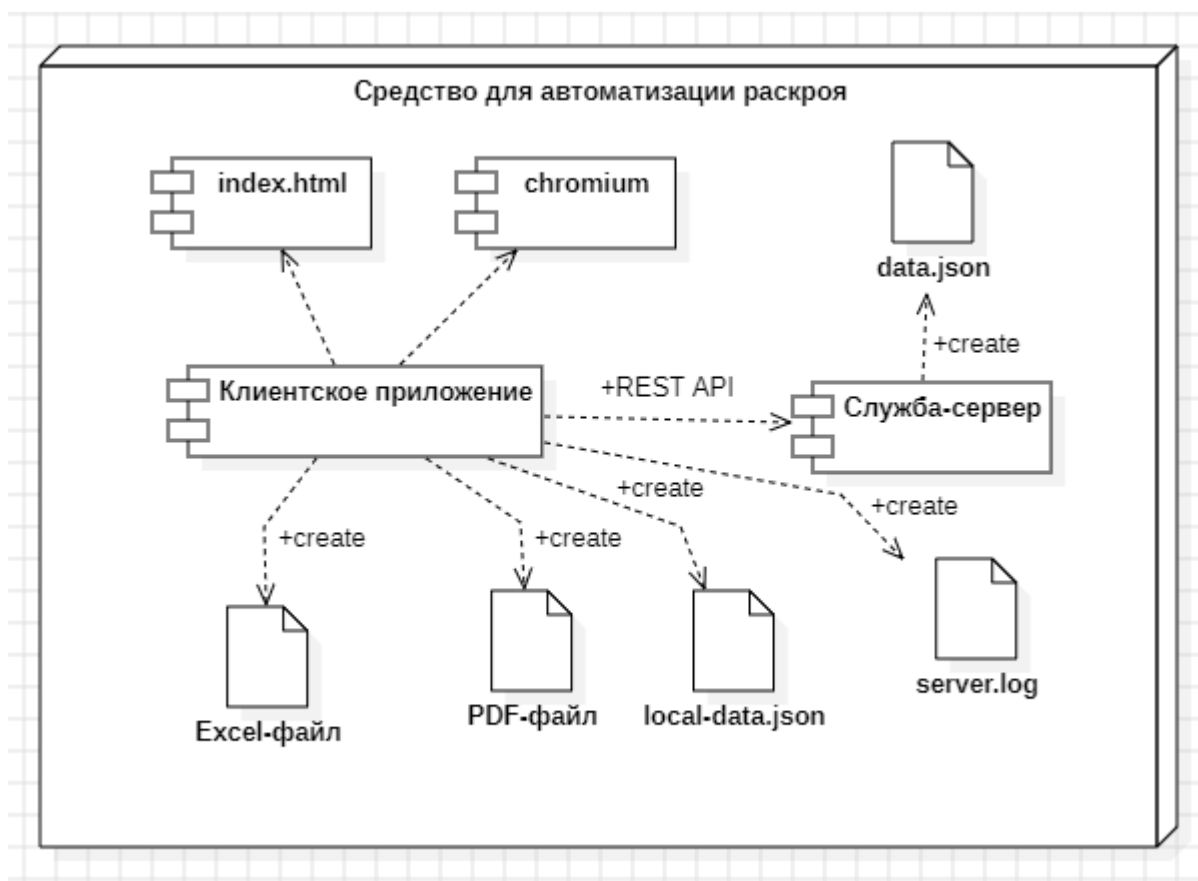


Рисунок 12 – Архитектура системы (диаграмма развертывания)

При запуске клиентского приложения, сервер запускается в виде дочернего процесса клиента. Он в свою очередь взаимодействует с ним посредством REST API интерфейса. Клиентское приложение во время работы производит сохранение файлов, которые содержат результаты раскрытия, а сервер сохраняет историю раскрытия материалов, которая впоследствии доступно пользователю в клиентском приложении.

Взаимодействие элементов архитектуры для сценария раскрытия линейных материалов можно представить с помощью диаграммы последовательности.

Диаграмма последовательности позволяет описывать последовательность работы в приложении, где определенные действия приводят к какому-то результату.

На рисунке 13 представлена диаграмма последовательности взаимодействия компонентов приложения для автоматизации раскрытия.

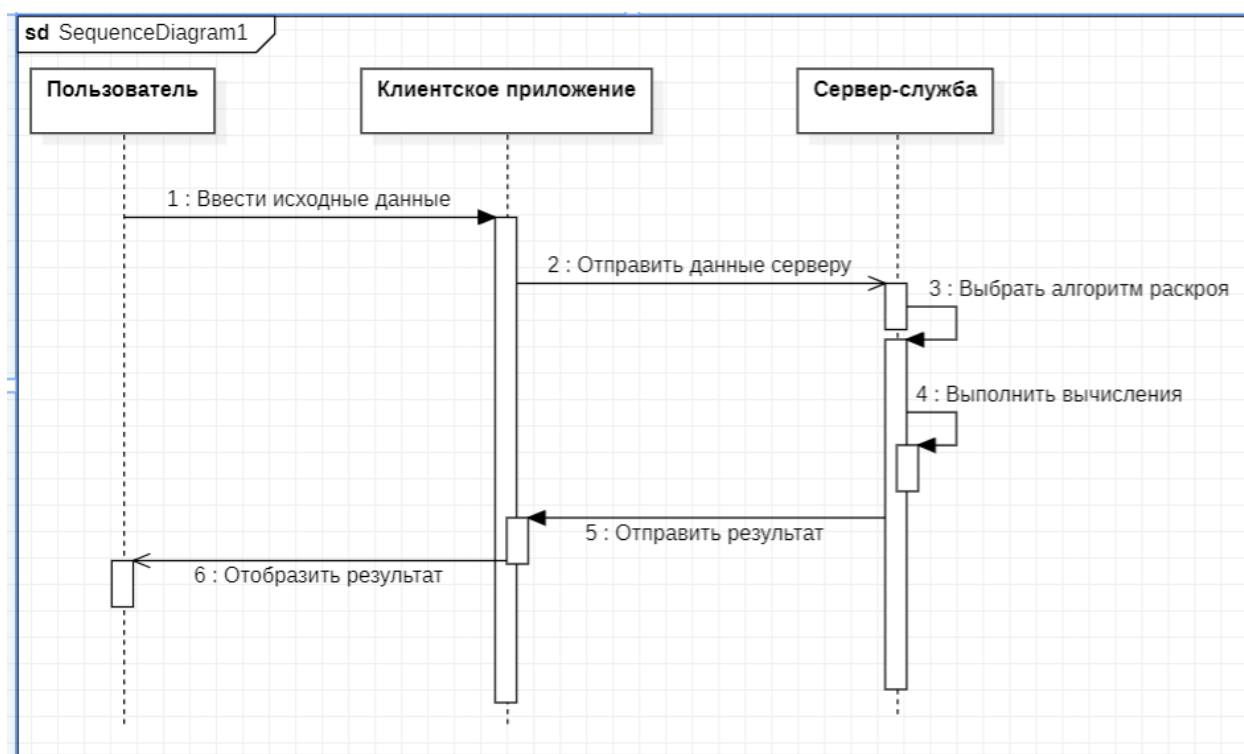


Рисунок 13 – Диаграмма последовательности взаимодействия компонентов приложения

Пользователь вводит исходные данные, которые клиентское приложение сериализует и отправляет серверу. Тот в свою очередь выбирает нужный алгоритм для линейного раскроя, после чего производит вычисления. Результат отправляется клиенту, который в свою очередь отображает данные пользователю.

Таким образом была спроектирована архитектура средства для автоматизации раскроя.

2.3 Прототипирование пользовательского интерфейса

Прототипирование интерфейса является важным шагом в процессе разработки приложения. В данном случае прототипирование – это процесс создания макета рабочего интерфейса на основе нарисованных и готовых макетов. Это позволяет изобразить приложение до написания кода.

После создания прототипа его можно:

- продемонстрировать заказчику с целью выявления недостатков в приложении;

- сбор качественных и количественных характеристики прототипа приложения исходя из демонстрации пользователям;
- демонстрация разработчикам для объяснения логики работы пользовательских путей в интерфейсе.

Базовыми принципами проектирования прототипов пользовательского интерфейса являются:

- создание сигнализированных состояний системы;
- использования опыта пользователя;
- уменьшений усилий, прилагаемых пользователем;
- создание путей выхода из сценария;
- возможность настройки интерфейса под себя;
- сохранение консистентности системы;
- предотвращение ошибок.

Прототип будет создаваться только после того, как определена целевая аудитория для программного продукта. Необходимо учитывать то, что в нем должны будут учтены пользовательские сценарии. Прототип должен представлять собой интерактивную модель, которая проверить возможные действия пользователя. С помощью прототипа можно взаимодействовать с продуктом и проводить пользовательские тестирования.

Прототипирование экономит денежные затраты на разработку того, что может не сработать. И даёт возможность посмотреть на «рабочее приложение» до того, как программисты начнут писать код.

Создание прототипов производится различными способами, вплоть до простого рисунка на бумаге или электронном носителе. Однако, чаще всего используются: Figma, Adobe XD, Framer [13].

При прототипировании пользовательского интерфейса приложения для автоматизации раскроя линейных и плоских материалов использовался Software Ideas Modeler. Схема пользовательского интерфейса изображена на рисунке 14.

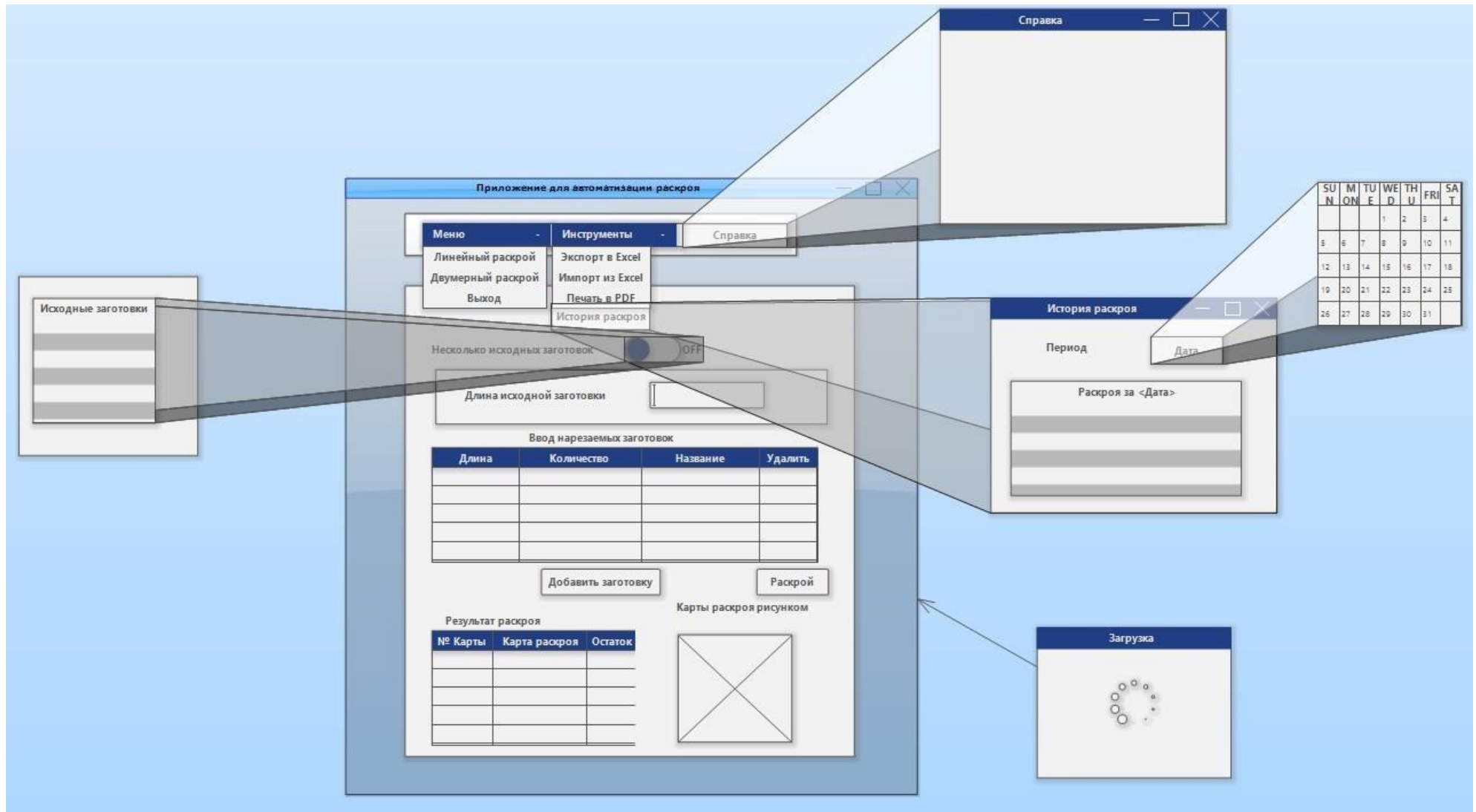


Рисунок 14 – Схема пользовательского интерфейса

В таблице 7 описаны спецификации окон.

Таблица 7 – Спецификация окон

Наименование	Спецификация
Окно загрузки	Перед запуском приложения появляется окно загрузки, где проигрывается анимация загрузки
Главное окно приложения для автоматизации раскроя	Основное окно, содержащее меню, основной контент, включающий ввод исходных данных и вывод результата. При этом линейный и двумерный раскрой имеют аналогичное меню, различие основного контента заключается лишь в вводе площадей, а не длин
Окно «История раскроя»	Окно, отображающее историю раскроя за выбранный период времени
Окно справки	Окно справки, которое содержит сведения о программе и последовательности работы в ней

На рисунке 15 показана диаграмма иерархии диаграмму окон

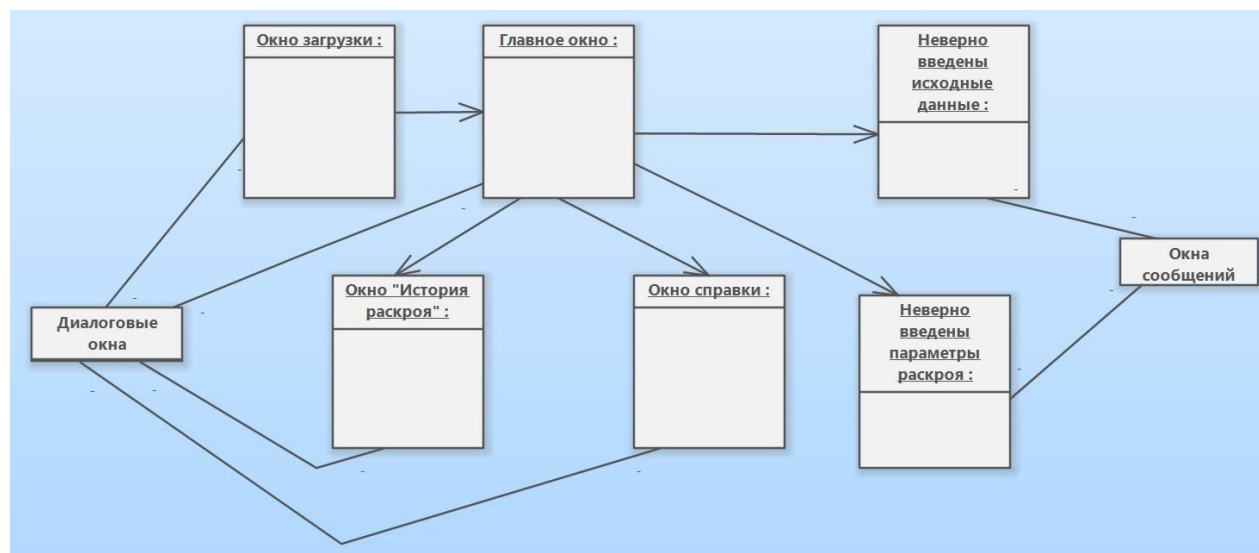


Рисунок 15 – Диаграмма иерархии окон

Основная работа осуществляется в основном окне. Если пользователь вводит некорректные данные, то ему выдаются соответствующие окна об ошибках. Окно открывается при выборе в меню справки. Там пользователь может ознакомиться с приложением и последовательностью действий в нем. В окне «История раскроя» пользователь может выбрать период, за который будет показана информация.

Таким образом было выполнено прототипирование пользовательского интерфейса средства для автоматизации раскроя.

2.4 Обоснование выбора средств разработки

2.4.1 Обоснование выбора средств автоматизации управления проектом

Для управления процессом разработки, системой контроля версий, сбором данных и построением отчетов использовался продукт Azure DevOps.

Azure DevOps – представляет собой web-приложение для управления разработкой, тестированием, развертыванием и сопровождением продукта. На рисунке 16 показана главная страница с текущими проектами пользователя в Azure DevOps.

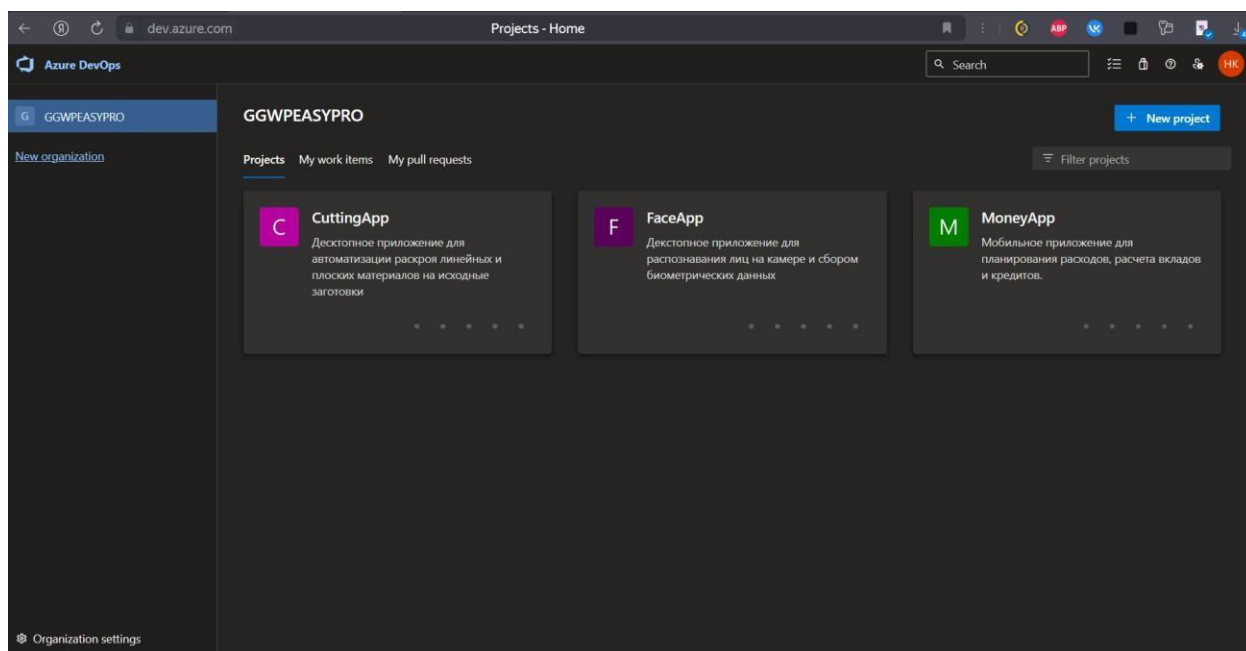


Рисунок 16 – Главная страница с проектами пользователя в Azure DevOps

В проекте Azure DevOps доступны вкладки: Azure Pipelines, Azure Boards, Azure Artifacts, Azure Repos Azure Test Plans.

Azure Pipelines. Здесь можно настроить непрерывную интеграцию и развертывания с поддержкой любых языков программирования, платформ и облачных сервисов.

Azure Boards. инструменты для планирования и контроля рабочего процесса: доски, журналы невыполненных работ, журнал текущих работ, командные информационные панели и настраиваемые отчеты.

Azure Artifacts. Каналы пакетов Maven, npm, NuGet, Python из открытых и закрытых источников. Позволяет добавить интегрированное управление пакетами в конвейеры CI/CD.

Azure Repos. Облачные репозитории Git для хранения файлов проекта. Возможность добавление запросов на включение и добавление файлов в проект, настройка управления слияния веток кода и их разветвление.

Azure Test Plans. Представляет собой комплексное решение для планирования тестирования или произвольного тестирования [14].

В процессе разработки использовался Azure Boards для управления задачами проекта CuttingApp – средства для автоматизации раскрытия линейных и плоских материалов. На рисунке 17 показаны задачи проекта.

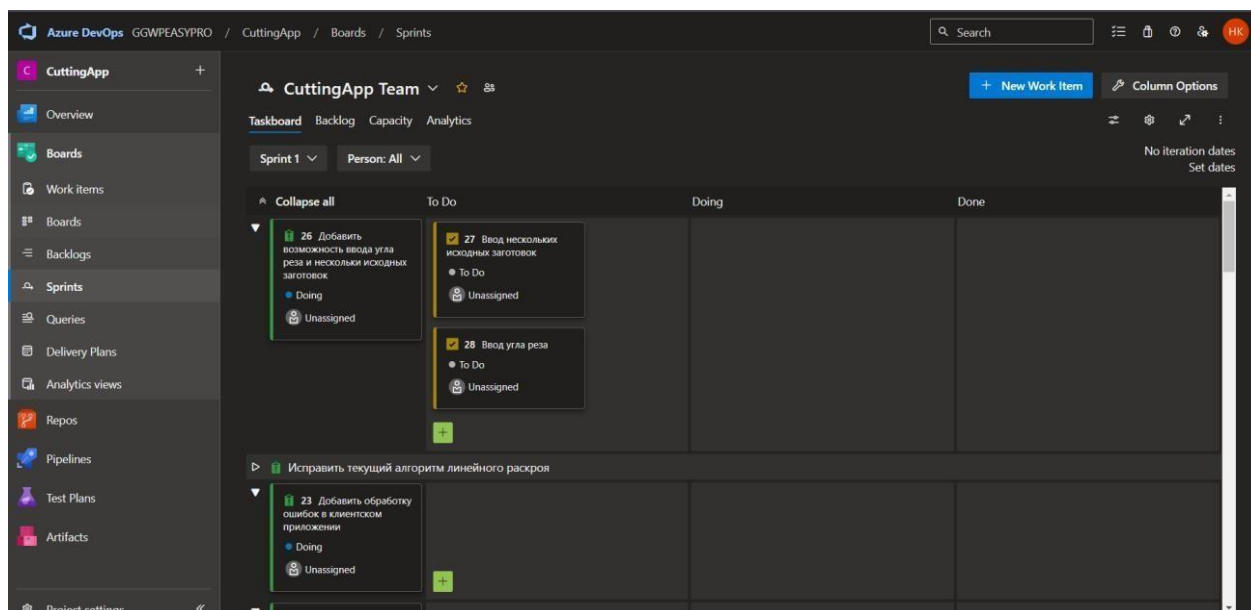


Рисунок 17 – Задачи проекта CuttingApp

Для проведения тестирования и сборки приложения использовался инструмент CI/CD TeamCity. На рисунке 18 показан интерфейс программы.

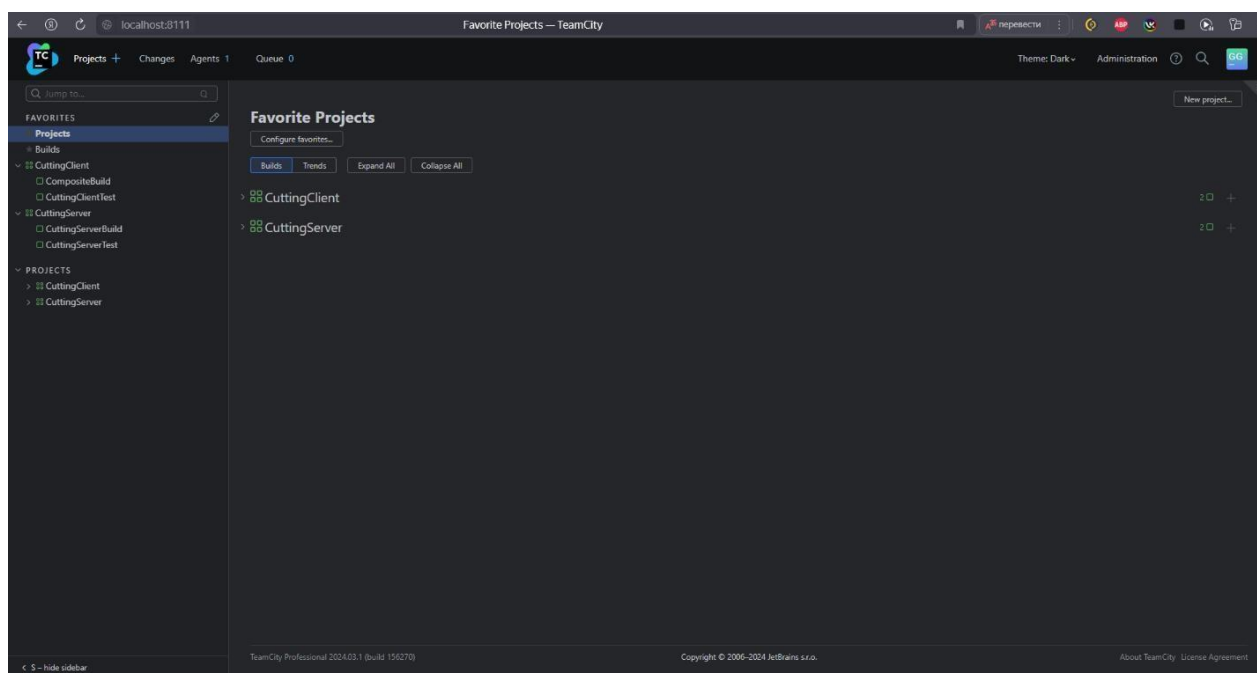


Рисунок 18 – Главная страница TeamCity

TeamCity – представляет собой Java-приложение, запускаемое в виде сервера локально. TeamCity имеет следующие свойства:

- гибкость и расширяемость;
- обеспечивает полную функциональность для процессов CI/CD;
- требует настройки, так как дополнительно можно установить множество плагинов.
- возможность размещать ресурсы сборок в виде «артефактов», позволяющая использовать их в других сборках;
- интеграция и дополнительными плагинами.

Большинство продуктов CI/CD являются платными, более громоздкими и менее удобными по интерфейсу. Основные преимущества TeamCity: наличие бесплатной версии для небольших команд, интеграция с облачными сервисами, гибкость и расширяемость, поэтому он выбран в качестве основного средства CI/CD [15].

Таким образом были выбраны средства для непрерывной интеграции и поставки с целью автоматизации и упрощения реализации средства для автоматизации раскрытия линейных и плоских материалов.

2.4.2 Обоснование выбора инструментальных средств разработки программного обеспечения

Для быстрого и качественного написания кода, необходимо выбрать наиболее подходящее инструментальное средство разработки программного обеспечения. В качестве таких продуктов были выбраны: WebStorm, Pycharm.

WebStorm – интегрированная среда разработки на JavaScript, CSS и HTML от компании JetBrains. Разработка была осуществлена на основе платформы IntelliJ IDEA. Веб-разработка полагается на мощные инструменты, которые должны адаптироваться под изменяющийся дизайн. Наличие современного и многофункционального программного обеспечения необходимо для создания web-приложений, площадок электронной коммерции и т. п.

WebStorm IDE обладает достаточно внушительным спектром функций, которые позволяют разработчикам повысить производительность, а также поддержку таких параметров как:

- интеллектуальное редактирование кода;
- поиск кода;
- проверка кода;
- интеграция с системой управления версиями;
- отладка кода;
- поддержка пользовательской настройки среды разработки;
- экосистема JetBrains;
- предварительные просмотры кода;
- понятный и простой пользовательский интерфейс;
- поддержка сообщества веб-разработчиков для WebStorm.

Также совсем недавно появилась возможность добавления AI-ассистента в процессе разработки [16].

На рисунке 19 показан интерфейс WebStorm.

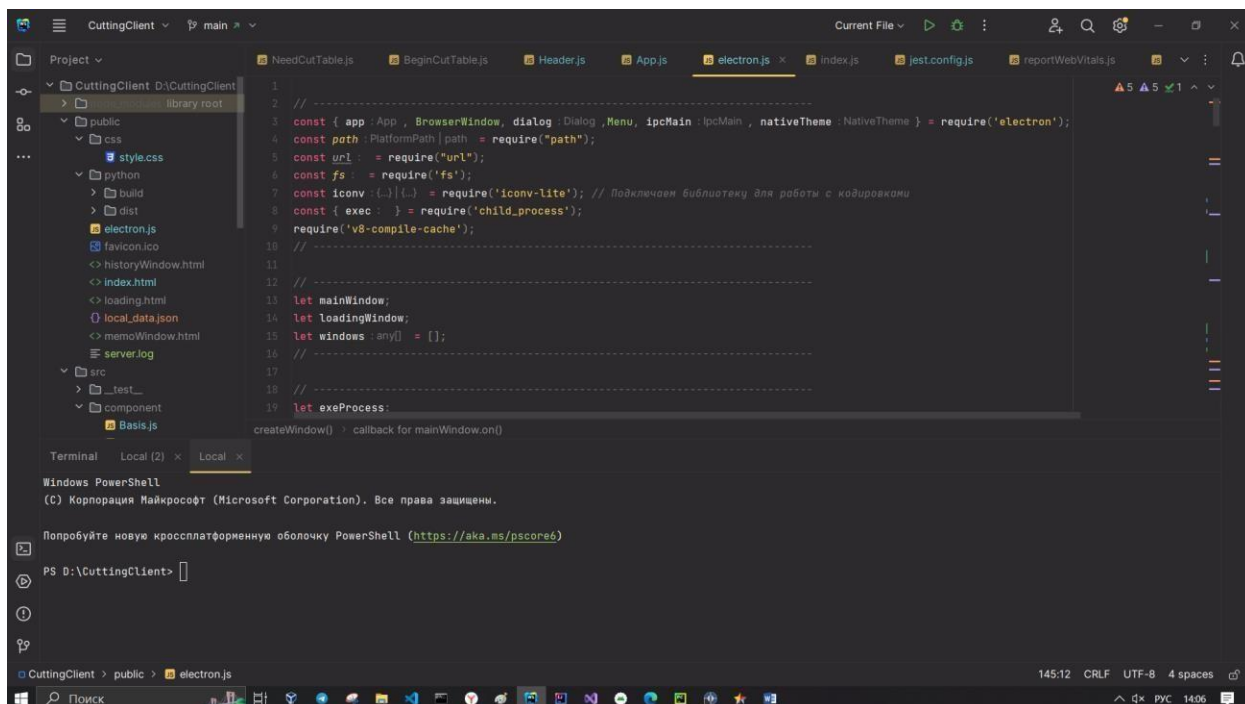


Рисунок 19 – Интерфейс WebStorm

WebStorm является платное программное обеспечение, однако JetBrains предлагает 30-дневный бесплатный период, а также для студентов доступны все платные продукты JetBrains по специальной подписке на период обучения. Аналогами WebStorm являются: Apache NetBeans, Visual Studio Code, Notepad ++, Sublime Text, Atom.

WebStorm был выбран в качестве средства разработки, так имеет: более удобный интерфейс и внешнего оформления, автозаполнение, позволяющее быстро писать код, возможность простой интеграции с GitHub и управлении версиями кода.

Pycharm – кроссплатформенная интегрированная среда разработки для языка программирования Python, которую разработала компания JetBrains на основе IntelliJ IDEA.

Данный продукт имеет две версии: Pycharm Community Edition – является бесплатной версией, и Pycharm Professional Edition – расширенная версия продукта, с дополнительным функционалом.

Pycharm имеет следующие возможности:

- умный редактор кода;

- быстрая навигация и поиск – быстрое нахождение нужных, переменных, классов, функций, файлов;
- возможность изменения внешнего вида пользовательского интерфейса;
- встроенные инструменты для управления версиями кода – Git и GitHub, а также интеграция с Docker.

В расширенной версии доступны:

- инструмент AI Assistant – встроенные функции, основанные на ИИ, умный-чат бот, позволяет выбрать оптимальные планы рефакторинга, составление и написание документации к коду;
- удаленная разработка – возможность настройки удаленного интерпретатора и подключение к удаленной машине через защищенное соединение, проект может быть размещен на удаленной среде, например Google Cloud Workstations или GitHub Codespaces;
- поддержка баз данных [17].

На рисунке 20 показан пользовательский интерфейс Pycharm.

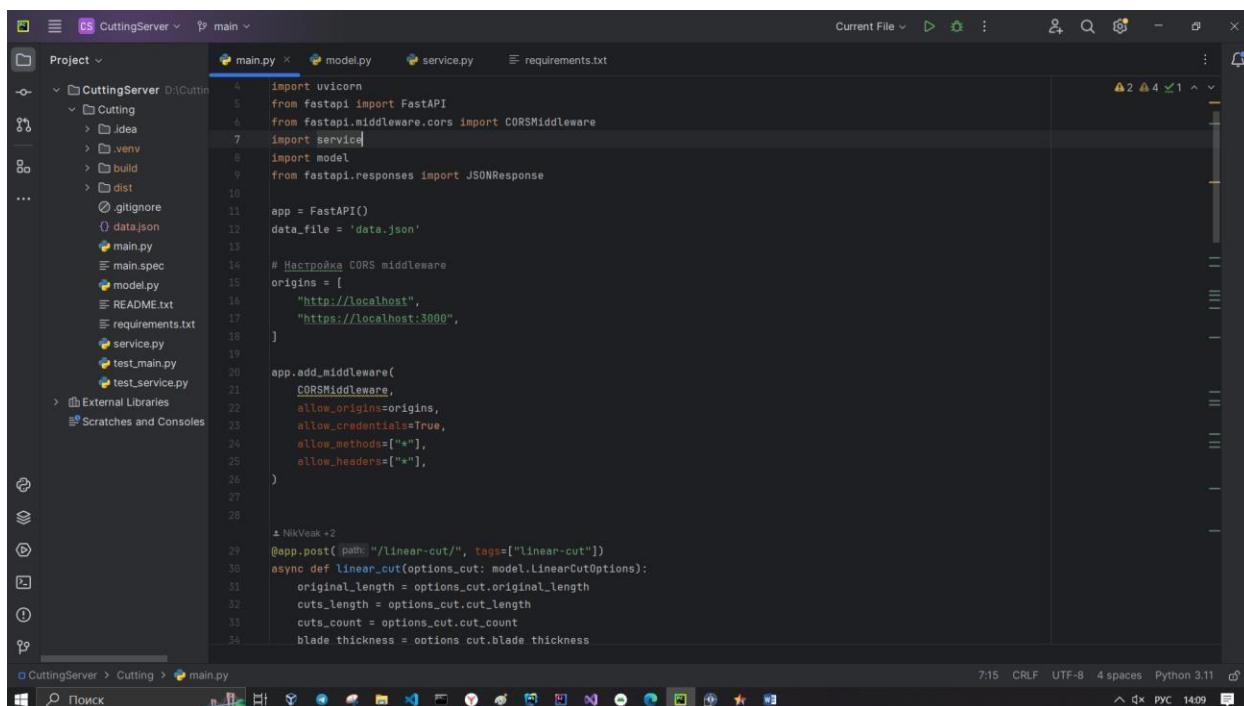


Рисунок 20 – Интерфейс Pycharm

Аналогами Pycharm являются: Visual Studio Code, Sublime Text, Atom, Notepad ++.

Pyscharm был выбран в качестве основного средства разработки, так предоставляет наиболее полный функционал в плане написания кода и интеграции с GitHub, а также имеет удобный пользовательский интерфейс.

2.4.3 Обоснование выбора языковых средств разработки

Разработанное средство для автоматизации раскроя линейных и плоских материалов на исходные заготовки состоит из клиентской и серверной части.

Серверная часть была написана с использованием языка программирования Python, а также фреймворка FastAPI и библиотеки Pulp.

Python – язык программирования, появившийся в начале 90-х годов прошлого века, общего назначения, имеет динамическую типизацию и автоматическое управление памяти. Он ориентирован на повышение читабельности кода и его кода, благодаря минимализму в синтаксисе. Популярен в анализе данных, машинном обучении, DevOps и веб-разработке, а также в разработке игр.

Для написания сервера использовался фреймворк FastAPI, имеющий широкий функционал. Он позволяет быстрое написание HTTP API-серверов со встроенной валидацией, сериализацией и асинхронностью. В качестве реализации одного из алгоритмов раскроя использовалась библиотека для решения задач линейного программирования Pulp.

Клиентская часть была написана с использованием языка программирования JavaScript с использованием фреймворка Electron JS и библиотеки React JS.

JavaScript является языком программирования, поддерживающим императивный, функциональный, объектно-ориентированный стили. Он имеет широкое применение в браузерах, обладает следующими чертами:

- динамическая типизация;
- функции являются объектами первого класса;
- автоматическое управление памятью;
- основан на прототипах.

Стандартами для JavaScript являются спецификации языка ECMAScript.

В настоящее время этот язык программирования имеет применение не только для сценариев на сайтах, но при разработке игр, десктопных и мобильных приложений. В качестве инструмента JavaScript для разработки настольного программного обеспечения был выбран фреймворк Electron JS.

Electron JS – представляет собой фреймворк для разработки приложений под ПК, с использованием HTML, CSS, JavaScript. Такие приложения являются кроссплатформенными и могут работать на Windows, Mac, Linux. Разработчиком Electron JS является компания GitHub Inc. На рисунке 21 показан официальный сайт Electron JS.

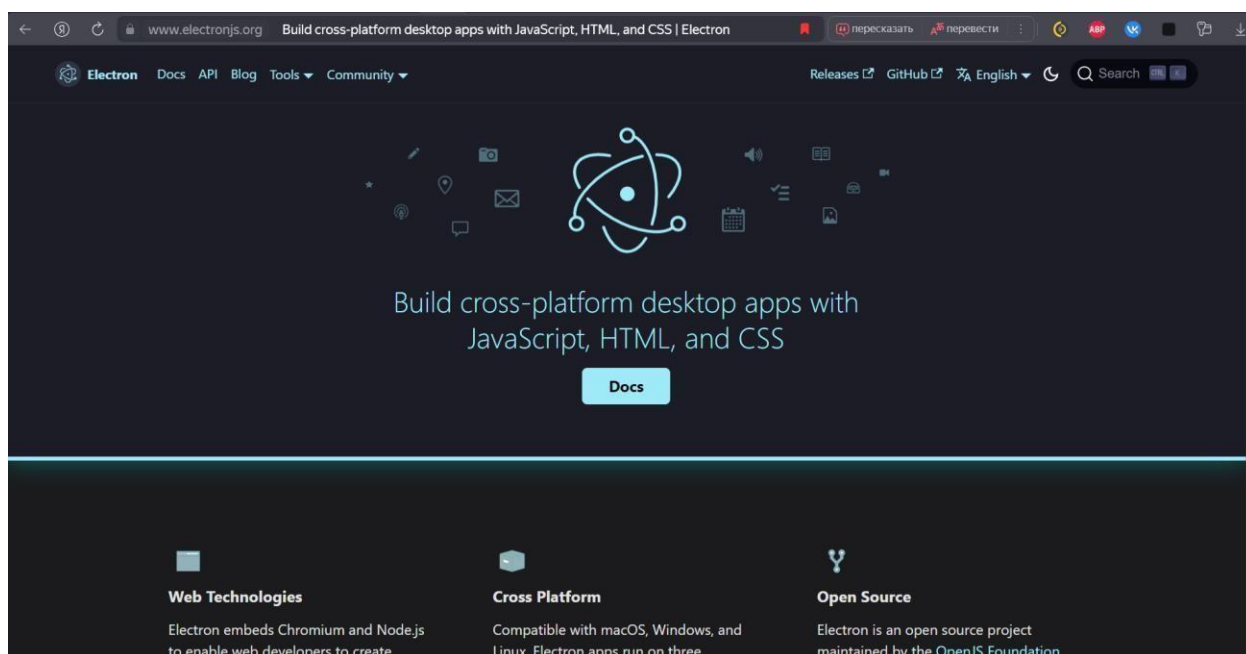


Рисунок 21 – Официальный сайт Electron JS

В основе работы фреймворка лежит использования браузерного движка Chromium и Node JS, которые объединены в единую среду, позволяющую писать настольные приложения с использованием веб-технологий.

Преимущества использования Electron:

- работа на всех платформах;
- низкий порог входа;
- различные интеграции с операционными системами;

- большое сообщество разработчиков и пользователей;
- автоматические обновления.

Недостатки использования Electron для создания приложений:

- каждое приложение поставляется со всей средой выполнения;
- некоторые приложения выходят низкого качества;
- большие размеры загрузки;
- высокое использование памяти.

В настоящее время производится доработка и улучшения фреймворка с целью оптимизации использования ресурсов. Однако современные устройства также имеют неплохую производительность, поэтому приложения, написанные с использованием Electron JS работают достаточно хорошо.

Наиболее популярным программным обеспечением, разработанным на этом фреймворке является:

Atom – текстовый редактор кода, позволяющий вести разработку на различных языках программирования;

Figma – приложение для прототипирования пользовательского интерфейса при разработке различных продуктов;

Docker Desktop – программное обеспечение для контейнеризации, предлагающее разработчикам инструмент для создания и запуска приложений любом месте, активно используется при CI/CD для быстрого развертывания приложений;

Discord – мессенджер для мгновенного обмена сообщениями, а также поддержка видеоконференций;

MongoDB Comprass – система управления документно-ориентированными базами данных;

Spotify – приложения для прослушивания музыки;

Visual Studio Code – популярный редактор кода с возможностью добавления различных плагинов, позволяющие упрощать и улучшать написание кода [18].

Для быстрого и качественного написания пользовательского интерфейса использовалась библиотека React JS, которая была интегрирована в Electron JS приложение.

React JS – библиотека с открытым исходным кодом, используемая для разработки веб-интерфейсов. Широко используется при реализации web-приложений наряду с другими популярными библиотеками такие как Angular JS, Vue JS. Официальный сайт показан на рисунке 22.

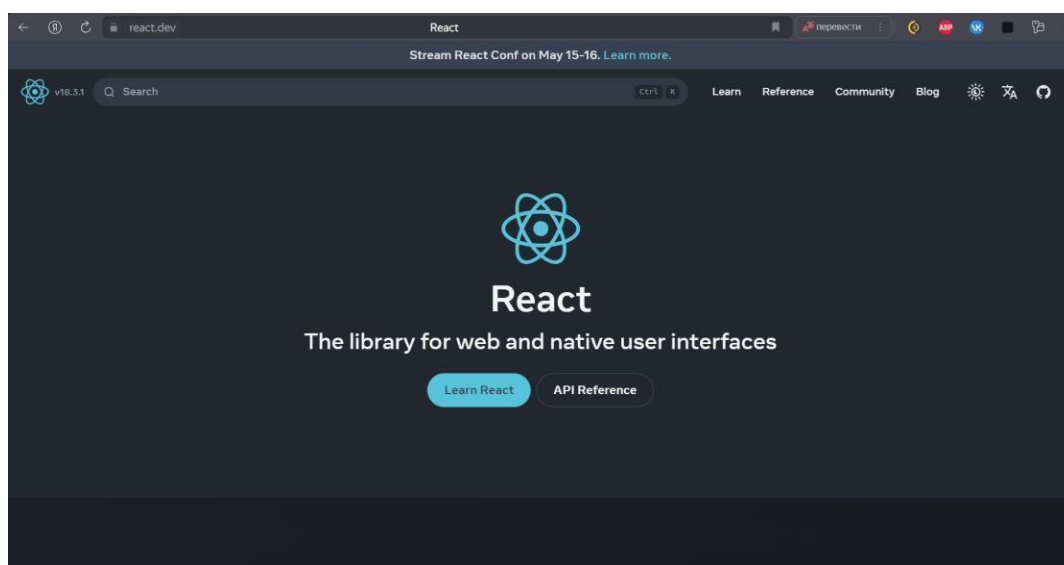


Рисунок 22 – Официальный сайт React JS

Основой React является создание компонент, которые отображаются в браузере в виде блоков HTML кода. При написании компонент можно использовать два стиля написания: функциональный и объектно-ориентированный. Последний используется обычно, если разрабатываемое приложение создается с помощью языка программирования TypeScript, являющегося расширением языка JavaScript. Также в приложения на React можно добавлять интерактивность, различные стили поведения объектов на странице. React JS имеет большое сообщество разработчиков [19].

Средство для автоматизации раскрытия линейных и плоских материалов на исходные заготовки должно быть кроссплатформенным, включая возможность работы на Android, IOS, в виде настольного приложения под Windows, Linux, Mac OS, а также в виде web-приложения. Использование Electron JS позволит обеспечить кроссплатформенность. Библиотека React JS имеет расширение

библиотеки React Native, позволяющее писать мобильные приложения под Android и IOS. В свою очередь, она будет использоваться в web-приложении.

3 Разработка программного обеспечения автоматизированных средств раскрытия линейных и плоских материалов

3.1 Автоматизированное управление проектом

Одним из главных конкурентных преимуществ в реализации программного обеспечения является скорость разработки. Поэтому на смену старых концепций разработки пришла модель CI/CD.

CI (Continuous Integration) – непрерывная интеграция. Данная методология разработки и набор практик предполагает, что основную ветку репозитория проекта после внесения изменений можно использовать множество раз, запуская при этом не только локальные проверки, но и в рамках задач выполнять автоматические тесты, unit-тесты. С точки зрения технического характера, цель CI – обеспечить последовательный и автоматизированный способ сборки приложения. При этом разработчики вносят частые commits, в результате чего происходит более тесная коммуникация между ними при разработке программного обеспечения.

CD (Continuous Delivery) – непрерывная поставка. Представляет собой автоматическое развертывание приложения в нужное окружение. Для данного процесса используются инструменты CI/CD такие как Jenkins, TeamCity, Travis CI, CircleCI. Простой конвейер CD состоит из сборки, тестирования и развертывания. Сложные конвейеры уже включают следующие этапы:

- получение кода из системы контроля версий и выполнение сборки;
- настройка инфраструктуры тестирования;
- копирования кода в целевую среду;
- настройка переменных окружения целевой среды;
- развертывание компонентов приложения, которые могут включать веб-серверы, базы данных, различные API-сервисы;
- запуск сервисов или их вызов, необходимых для работоспособности приложения;
- выполнение тестирования;

- логирование и отправка оповещений о состоянии поставки [20].

Для автоматизации разработки использовался GitHub, который позволял непрерывно поставлять код в облачное окружение, откуда TeamCity использовал для сборки приложения и выполнения тестов с измерением покрытия ими [21].

Для выполнения тестов и сборки серверной части приложения был создан проект TeamCity – CuttingServer. На рисунке 23 показан этот проект.

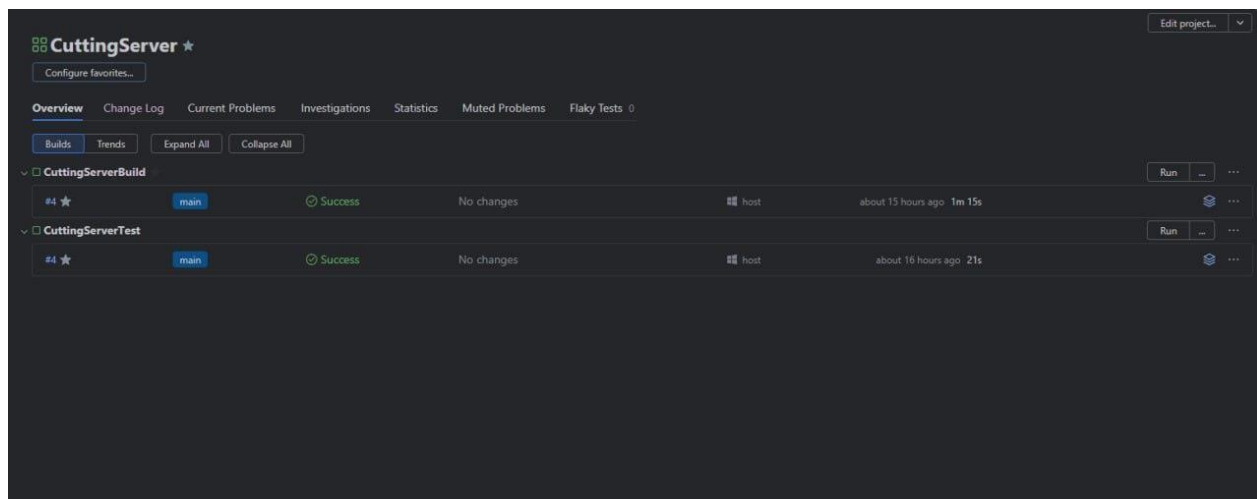


Рисунок 23 – Проект CuttingServer

Для проекта CuttingServer были созданы две build-конфигурации: CuttingServerTest, CuttingServerBuild. Каждая из них выполняет определенные задачи, позволяя разделить выполнение тестов и сборку приложения. Впоследствии, результаты выполнения тестов будут видны в одной сборке, а артефакты с исполняемыми файлами сервера в другой.

CuttingServerTest предназначен для выполнения автоматизации выполнения тестов и создания отчетов о покрытии тестами.

Существуют следующие настройки:

- имя сборки;
- идентификатор сборки;
- тип сборки;
- формат сборок;
- путь к артефактам.

На рисунке 24 показаны главные настройки build-конфигурации.

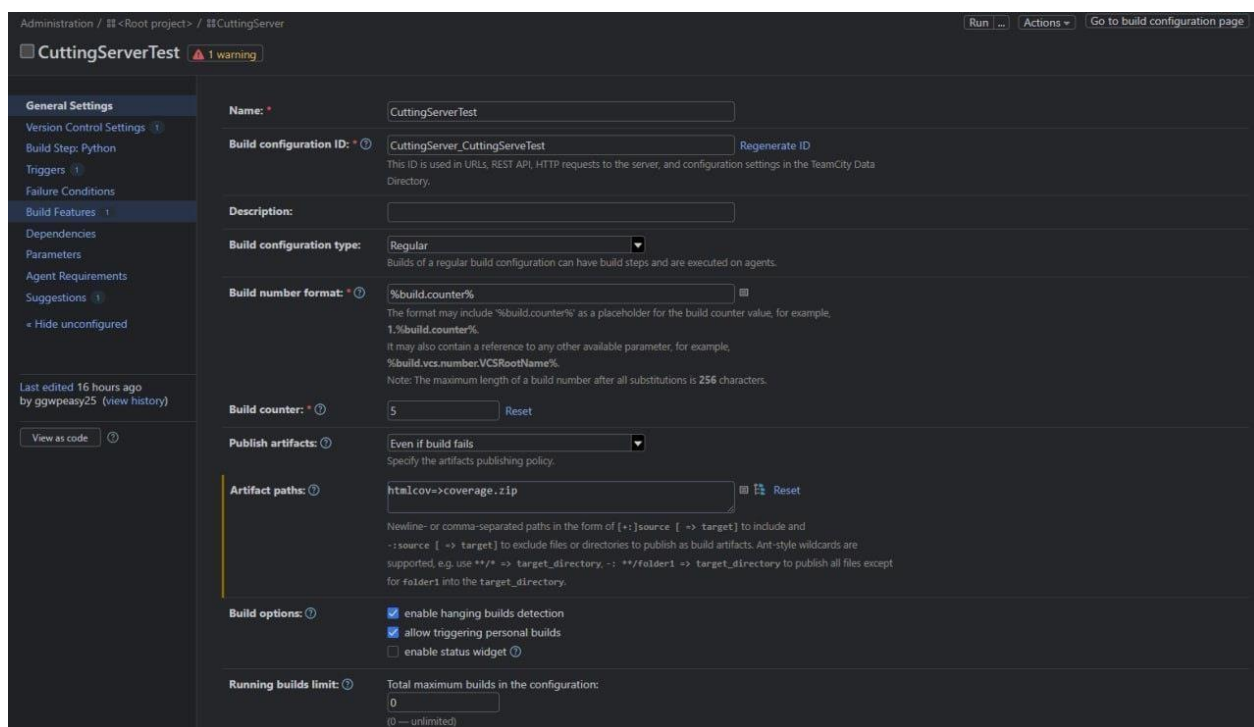


Рисунок 24 – Главные настройки build-конфиграции CuttingServerTest

Сборка состоит из одного шага. В качестве средства выполнения шага был выбран python-скрипт осуществляющий выполнение тестов (листинг 1).

Листинг 1 – Код python-скрипта, осуществляющий выполнение тестов

```
# -*- coding: utf-8 -*-
import subprocess
print("Прохождение и покрытие тестами")
try:
    # Обновление pip
    result = subprocess.run(["python.exe", "-m", "pip", "install", "--upgrade", "pip"], capture_output=True, text=True, encoding='utf-8')

    # Вывод результата на экран
    print(result.stdout)
except subprocess.CalledProcessError as e:
    print(f"An error occurred: {e}")
except FileNotFoundError as e:
    print(f"File not found error: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```

subprocess.run(["coverage", "run", "--
omit=test_main.py", "test_main.py"], encoding='utf-8')

subprocess.run(["coverage", "report", "-m"], encoding='utf-8')

subprocess.run(["coverage", "html"], encoding='utf-8')

```

В начале производится установка необходимых пакетов, после идет выполнение тестов с формированием результатов покрытия.

Для сборки серверной-части приложения была создана build-конфигурация CuttingServerBuild. На рисунке 25 показаны главные настройки сборки.

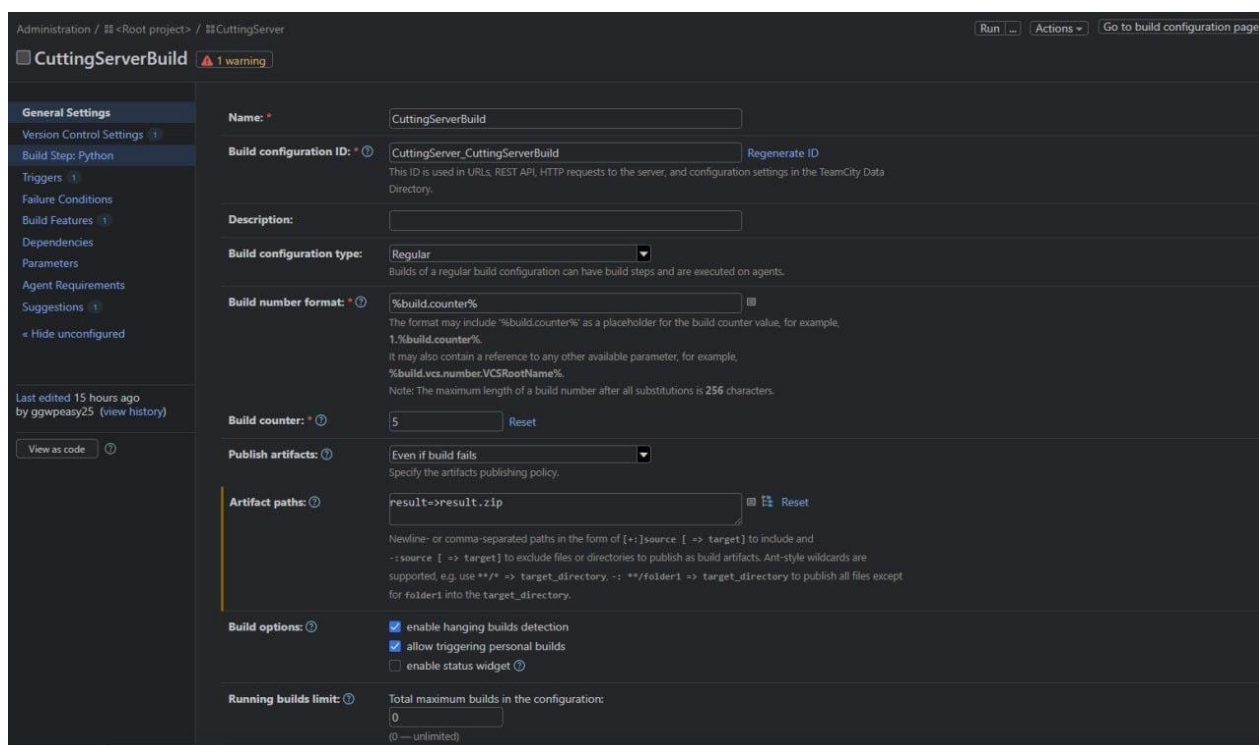


Рисунок 25 – Build-конфигурация CuttingServerBuild

В шаге используется python-скрипт осуществляющий сборку серверной части приложения в исполняемый файл Windows (листинг 2).

Листинг 2 – Код python-скрипта для сборки серверной части приложения

```

# -*- coding: utf-8 -*-

import subprocess

print("Build exe server")

try:

    # Обновление pip

```

```

    result = subprocess.run(["python.exe", "-m", "pip", "install", "--
upgrade", "pip"], capture_output=True, text=True, encoding='utf-8')

    # Вывод результата на экран
    print(result.stdout)

    # Сборка проекта с помощью PyInstaller
    subprocess.run(["pyinstaller", "-y", "--clean", "--additional-hooks-
dir", "extra-hooks", "--hiddenimport", "pulp", "main.py", "--onefile", "--
collect-data", "pulp"], shell=True)

    # Организация результатов
    subprocess.run(["dir"], shell=True)
    subprocess.run(["mkdir", "result"], shell=True)
    subprocess.run(["move", "build", "result"], shell=True)
    subprocess.run(["move", "dist", "result"], shell=True)
except subprocess.CalledProcessError as e:
    print(f"An error occurred: {e}")
except FileNotFoundError as e:
    print(f"File not found error: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")

```

Сборка осуществляется с помощью pyinstaller под операционную систему Windows. В дальнейшем планируется сборки под разные операционные системы.

Для выполнения тестов клиентской части приложения, а также полной сборки приложения, был создан проект TeamCity – CuttingClient. На рисунке 26 показаны build-конфигурации.

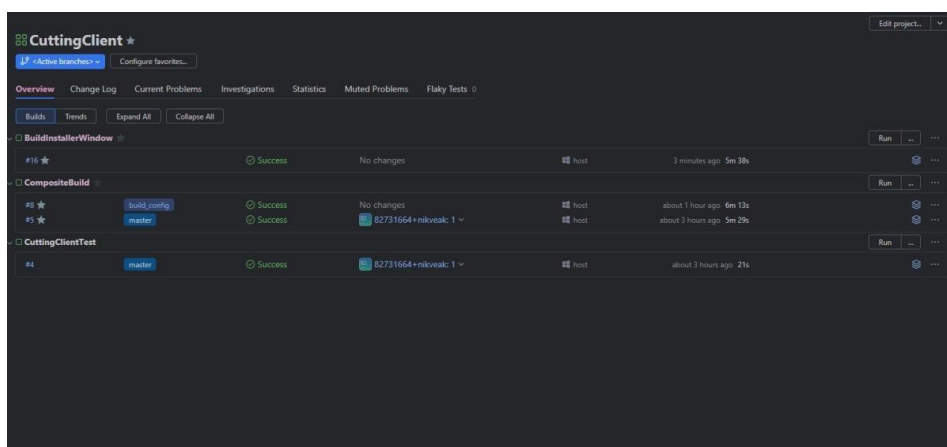


Рисунок 26 – Проект CuttingClient

Для проекта CuttingClient было создано две build-конфигурации: CuttingClientTest, CompositeBuild, BuildInstallerWindow.

CuttingClientTest автоматизирует выполнение тестов и создание отчета о покрытии. На рисунке 27 показаны настройки сборки.

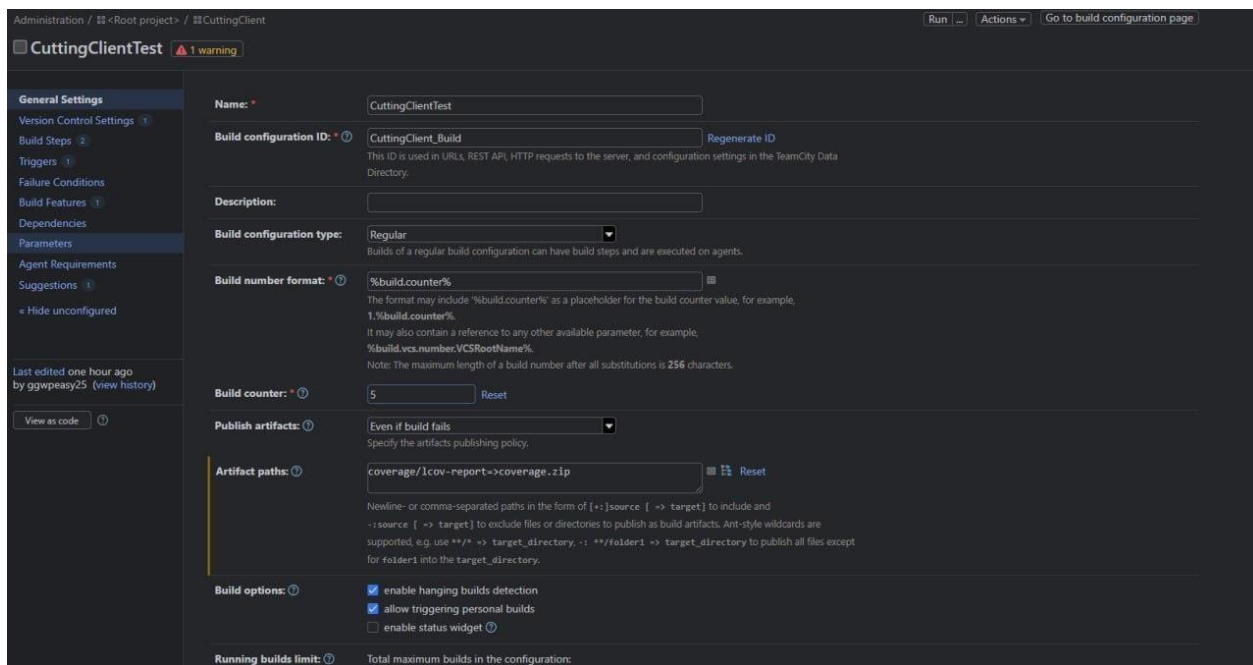


Рисунок 27 – Сборка CuttingClientTest

Сборка состоит из двух шагов:

- установка пакетов с помощью команды `npm install`;
- выполнение тестов с помощью команды `npm test`.

Для выполнения сборки приложения под Windows была создана build-конфигурация CompositeBuild.

Сборка приложения под Windows на Electron JS может производиться различными способами:

- с помощью `electron-packager`;
- с помощью `electron-builder`.

Второй способ подходит, если нужно сразу создать установщик для приложения и не нужны другие настройки.

На рисунке 28 показаны настройки build-конфигурации.

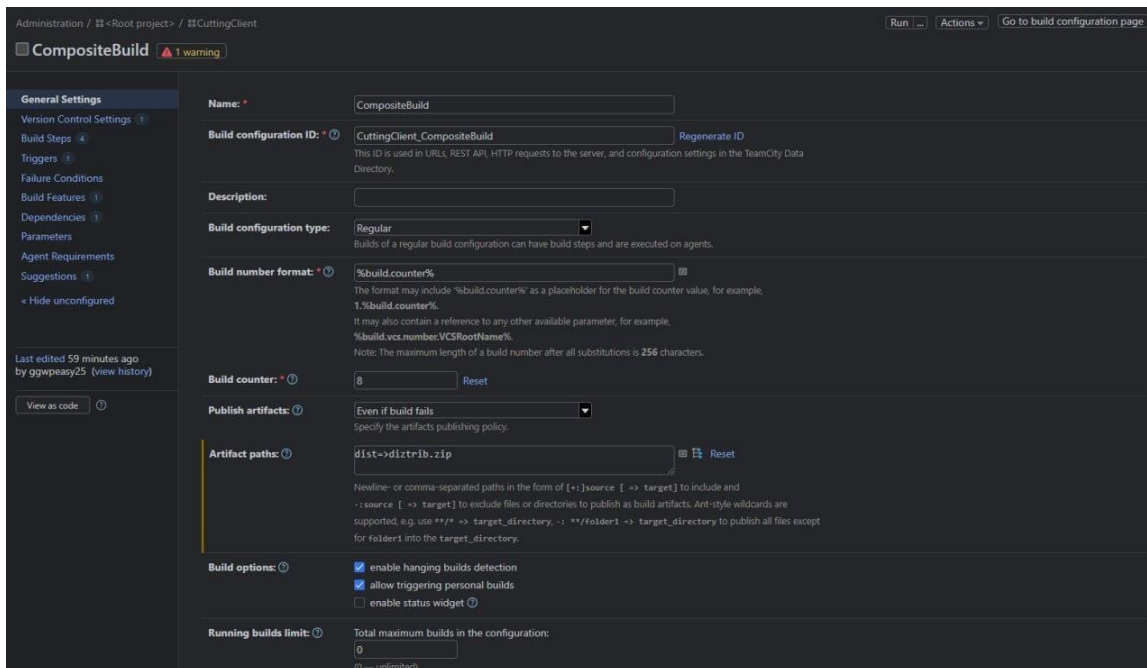


Рисунок 28 – Build-конфигурация CompositeBuild

В ней в качестве зависимости был указан путь до артефактов другой сборки: CuttingServerBuild. На рисунке 29 показаны настройки артефактов сборки.

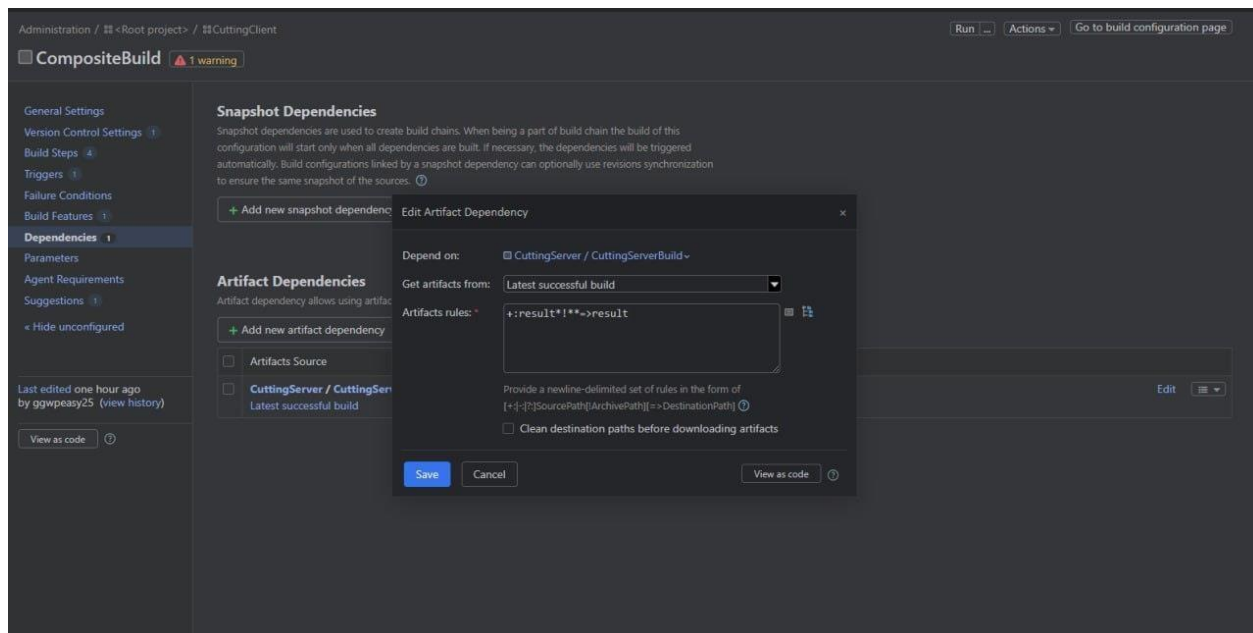


Рисунок 29 – Зависимости сборки build-конфигурации CompositeBuild

CompositeBuild состоит из четырех шагов.

Первый шаг. Установка необходимых пакетов командой npm install.

Второй шаг. Подготовка к сборке приложения. Для этого был написан python-скрипт (листинг 3).

Листинг 3 – Код python-скрипт, осуществляющий подготовку к сборке

```
import os
import shutil
import subprocess
print("#-----#")
print("Перетаскиваем папку с билдами сервера в нужную папку")
source = r'%system.teamcity.build.workingDir%/result'
destination = r'%system.teamcity.build.workingDir%/public'
def copy_build_server(source_path, destination_path):
    for item in os.listdir(source_path):
        s = os.path.join(source_path, item)
        d = os.path.join(destination_path, item)
        if os.path.isdir(s):
            shutil.copytree(s, d, dirs_exist_ok=True)
        else:
            shutil.copy2(s, d)
def check_current_dir(folder):
    subprocess.run(f'cd {folder} && dir', shell=True, check=True)
print("Директория с билдами сервера")
check_current_dir(source)
copy_build_server(source, destination)
print("Директория с местоположением сервера")
check_current_dir(destination)
```

Из зависимостей мы переносим нужные папки в директорию, откуда будет запускаться сервер как служба клиентского приложения. Также производится проверка на наличие нужных файлов.

Третий шаг. Сборка React приложения с помощью команды `npm run build`.

Четвертый шаг. Сборка Electron приложения с помощью Команды `npm package-npm-win`.

После сборки CompositeBuild выкладывает архив приложения, который можно скачать. На рисунке 30 показаны артефакты сборки.

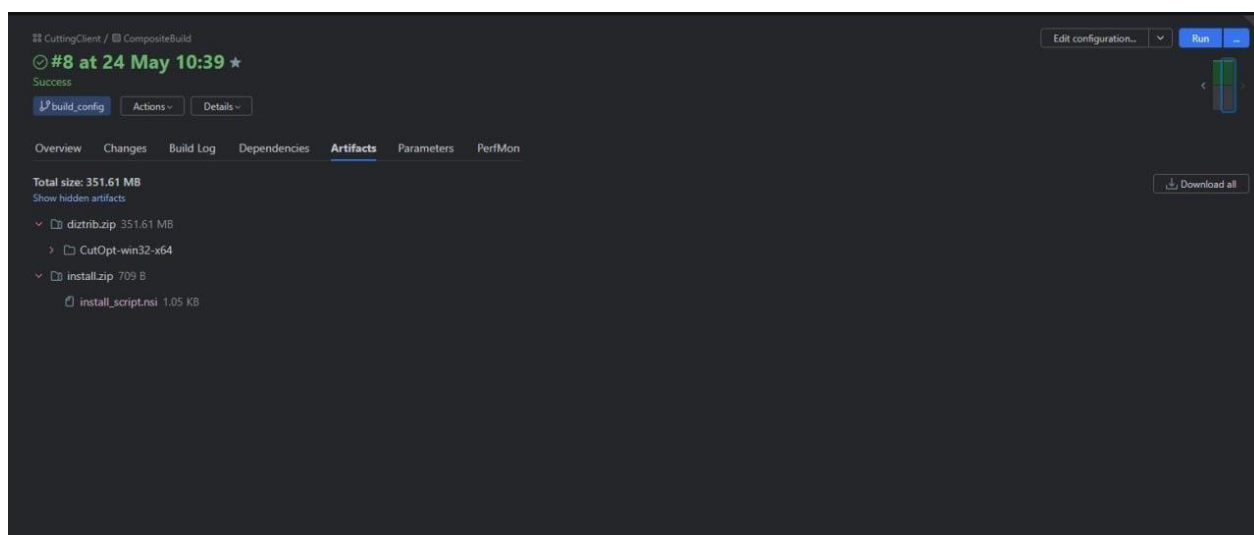


Рисунок 30 – Артефакты сборки CompositeBuild

Build-конфигурация BuildInstallerWindow позволяет создавать установщик программного обеспечения раскроя линейных и плоских материалов на исходные заготовки для операционной системы Windows.

В качестве зависимости используются артефакты сборки CompositeBuild. На рисунке 31 показаны зависимости артефактов сборки.

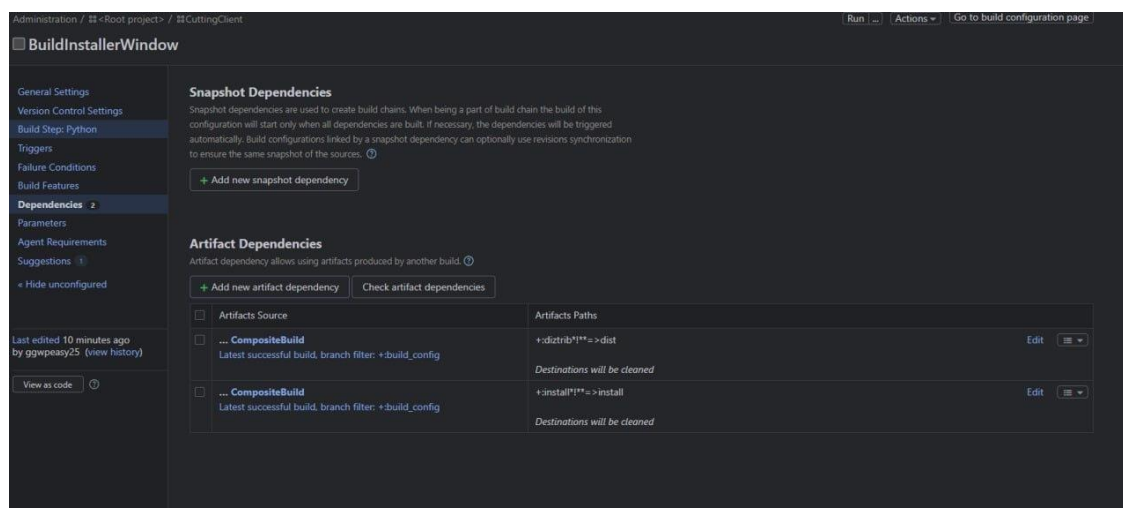


Рисунок 31 – Зависимости build-конфигурации BuildInstallerWindow

В качестве шага использовался python-скрипт, который осуществляет проверку папок и создания установщика с помощью утилиты NSIS (листинг 4).

Листинг 4 – Код python-скрипта, используемый для создания установщика

```
import os
import subprocess
```

```

import shutil

def check_current_dir(folder):
    subprocess.run(f'cd {folder} && dir', shell=True, check=True,
encoding='utf-8')

project = r'%system.teamcity.build.workingDir%/dist/CutOpt-win32-x64'
installer_dir_script = r'%system.teamcity.build.workingDir%/install'

check_current_dir(r'%system.teamcity.build.workingDir%/')
check_current_dir(project)
check_current_dir(installer_dir_script)
subprocess.run(["makensis", "install_script.nsi"], shell=True,
check=True, encoding='utf-8')

```

Скрипт NSI, создающий непосредственно установщик представлен в листинге 5.

Листинг 5 – Код скрипта NSI

```

!include LogicLib.nsh

; Начало скрипта установки

; Заголовок установщика
Name "Installer CutOpt"
OutFile "CutOptInstaller.exe"
RequestExecutionLevel user

; Задание директорий установки
InstallDir "$PROGRAMFILES\CuttingApp"

; Стандартные страницы, присутствующие в любом установщике, типа
"Выберите каталог для установки" и т.д.

Page directory
Page instfiles

; Установка Electron JS
Section "Electron JS" SecElectron
    SetOutPath $INSTDIR
    File /r "dist\CutOpt-win32-x64\"
SectionEnd
; Завершение установки
Section "Complete" SecComplete
; Дополнительные действия после установки
SectionEnd

; Конец скрипта установки

```

По окончании сборки можно скачать установщик. На рисунке 32 показаны артефакты сборки BuildInstallerWindow.

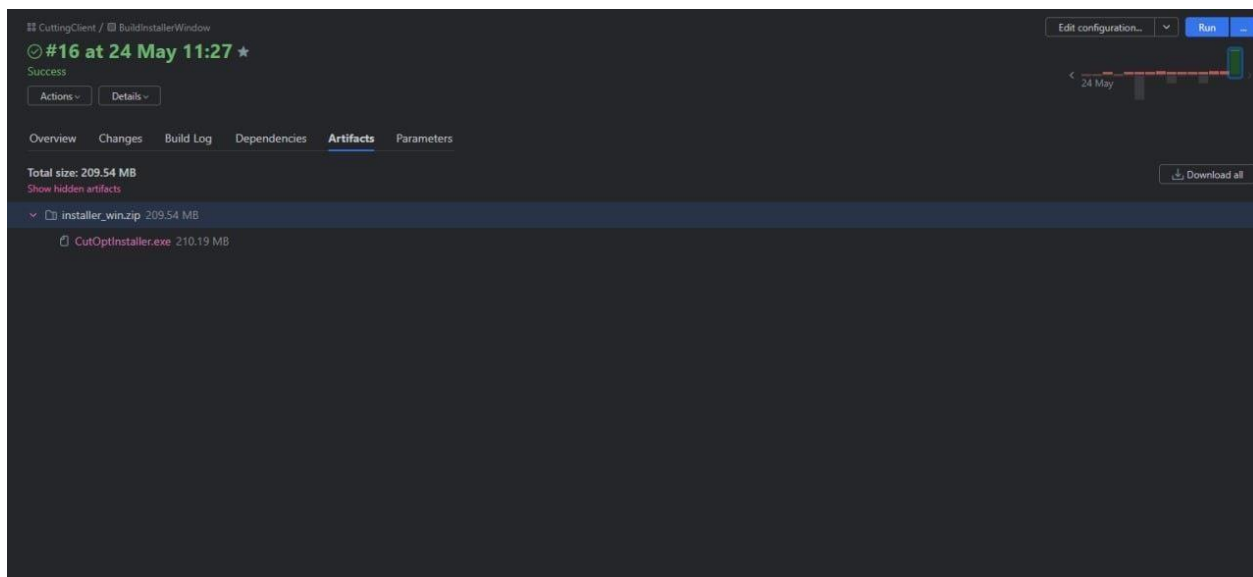


Рисунок 32 – Артефакты сборки BuildInstallerWindow

В дальнейшем планируется сборки под другие платформы. Таким образом была произведена настройка процессов CI/CD.

3.2 Разработка клиентской части приложения

3.2.1 Разработка пользовательского интерфейса клиентского приложения

Пользовательский интерфейс разрабатывался исходя из прототипа, спроектированного в разделе 2.3.

Существуют несколько различных цветовых схем в приложениях:

- аналоговая цветовая схема;
- монохромная цветовая схема;
- составная цветовая схема.

Выбор зависит от различных требований к UI-интерфейсу. В качестве цветовой схемы использовалась составная цветовая схема, где основными цветами выступали серый и белый, дополнительными черный, темно-зеленый, зелено-голубой [22].

На рисунке 33 показано главное окно линейного раскроя.

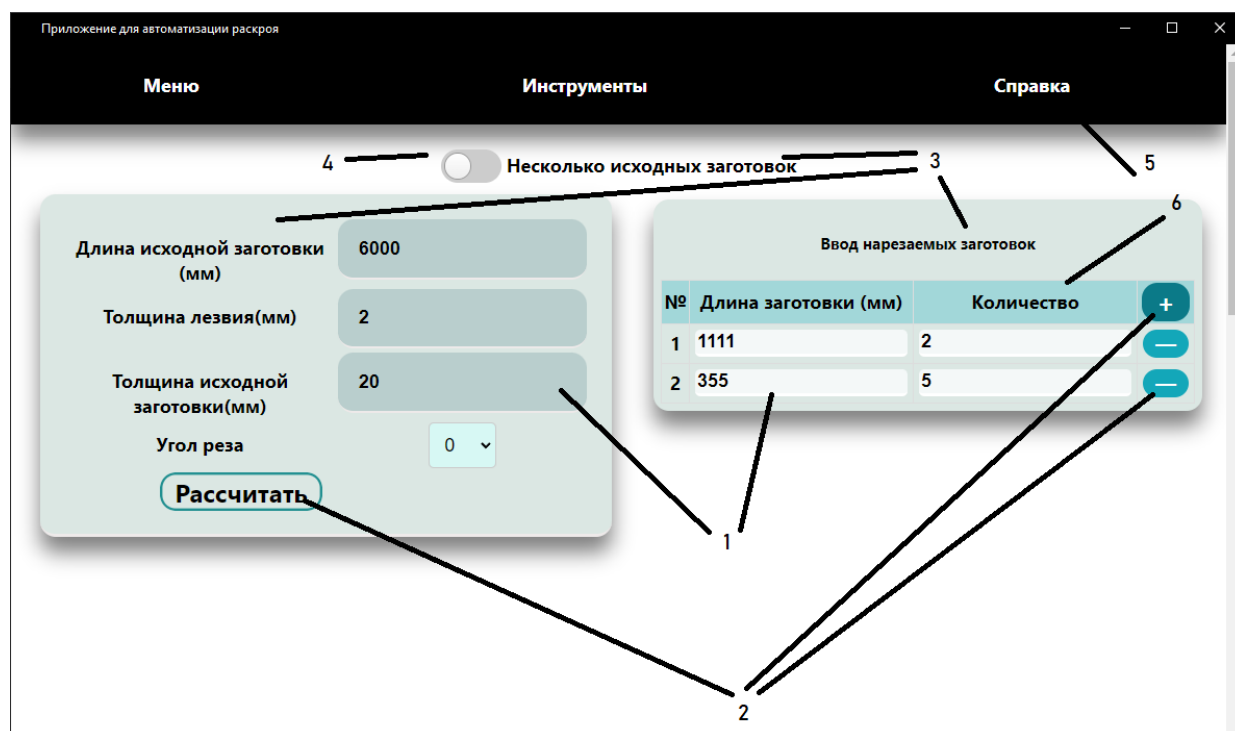


Рисунок 33 – Главное окно линейного раскроя

На рисунке расположены элементы:

- 1 – поля ввода;
- 2 – кнопки;
- 3 – поля наименований;
- 4 – кнопка-переключатель;
- 5 – список, отображающий меню.

На рисунке 34 показана страница линейного раскроя с несколькими исходными заготовками.

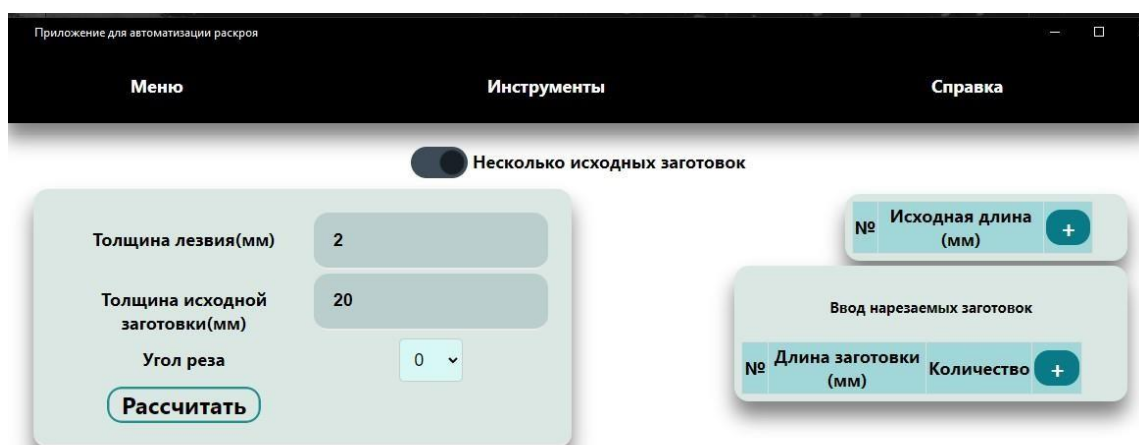


Рисунок 34 – Страница линейного раскроя с несколькими заготовками

Элементы страницы будут аналогичными, как и в главной странице линейного раскроя.

На рисунке 35 показана главная страница после выполнения линейного раскроя с отображенными на ней результатами.

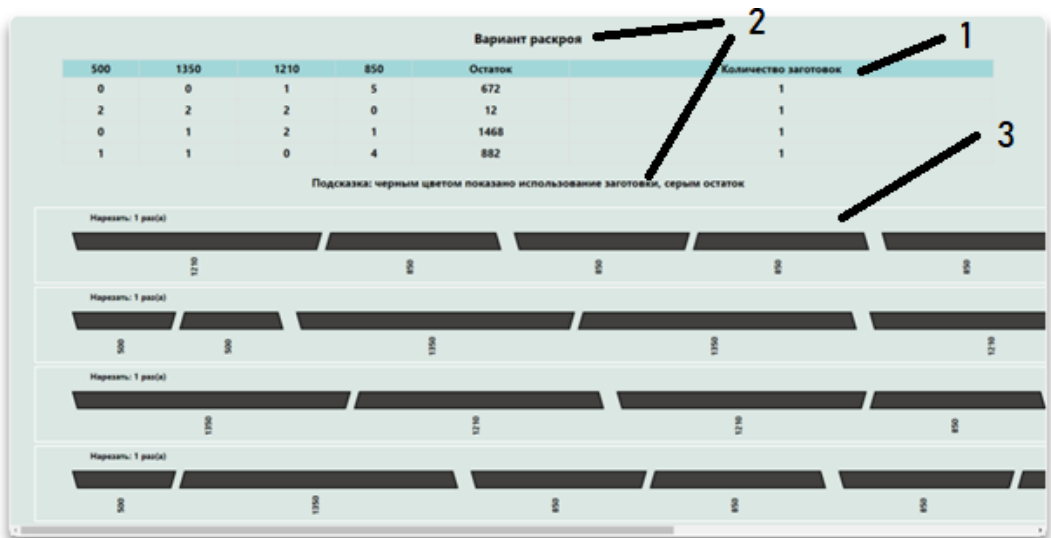


Рисунок 35 – Страница с результатами линейного раскроя

На рисунке показаны элементы:

- 1 – таблица с отображенными результатами раскроя;
- 2 – поля наименований;
- 3 – элемент svg графики для отображения заготовок.

На рисунке 36 показано окно с выводом истории раскроя.

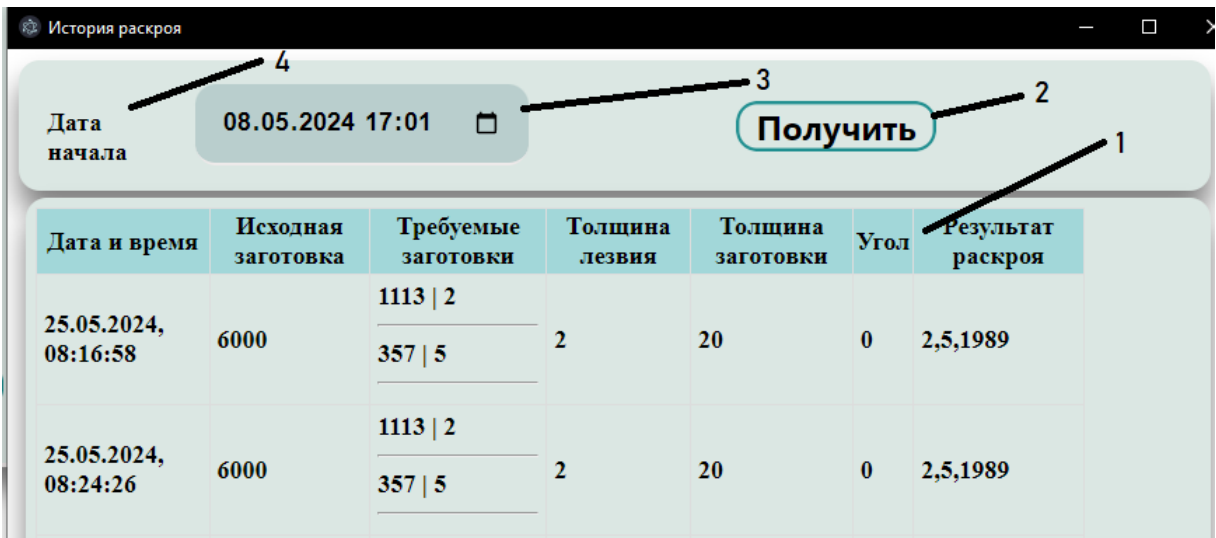


Рисунок 36 – Окно с результатами раскроя

На рисунке показаны следующие элементы:

- 1 – таблица, отображающая результаты раскроя;
- 2 – кнопка;
- 3 – поле ввода даты и времени;
- 4 – поле наименования.

На рисунке 37 показано окно справки.

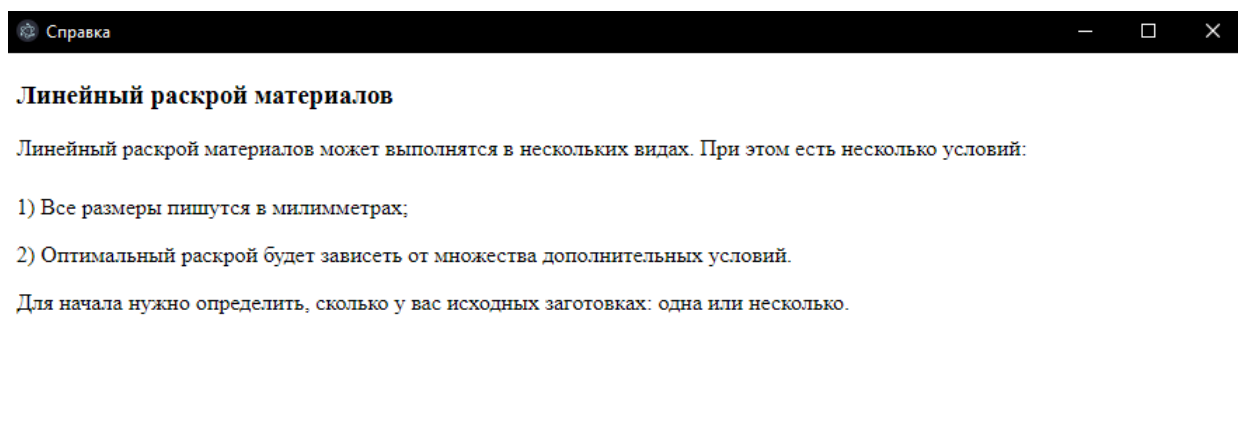


Рисунок 37 – Окно справки

В данное окно будет добавляться информация о последовательности работы в программе.

Таким образом был разработан пользовательский интерфейс для приложения раскроя линейных и плоских материалов.

3.2.2 Разработка программного обеспечения клиентского приложения

Программное обеспечение клиентского приложения разрабатывалось на языке программирования JavaScript с использованием Electron JS и React JS.

Компоненты React JS могут писаться в функциональном стиле или в стиле объектно-ориентированного программирования. Второй способ подходит, если разработка ведется на языке TypeScript.

В работе клиентское приложение писалось на языке JavaScript в функциональном стиле.

На рисунке 38 показана диаграмма компонентов клиентского приложения.

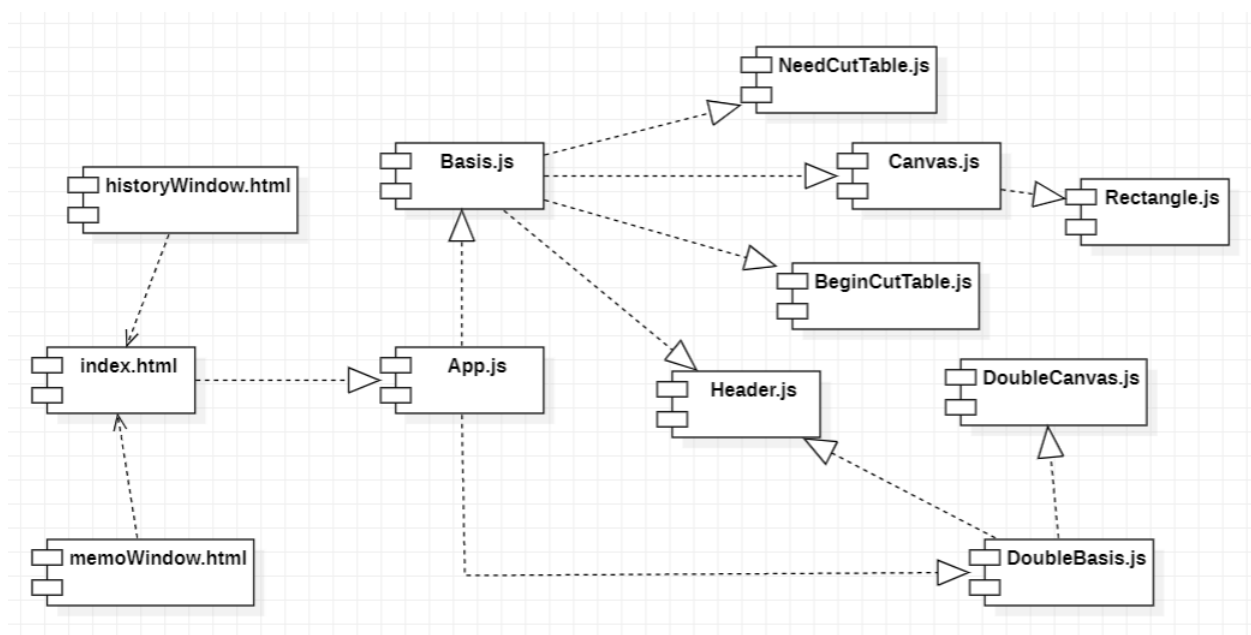


Рисунок 38 – Диаграмма компонентов клиентского приложения

Приведем описание каждого компонента:

- компонент `index.html` – главная страница приложения, в которой будут создаваться компоненты React JS приложения;
- компонент `historyWindow` – страница, выводющая историю раскроя, открывается как отдельное окно Electron JS приложения;
- компонент `memoWindow` – страница, содержащая информацию о программном обеспечении, открывается как отдельное окно Electron JS приложения;
- компонент `App.js` – элемент, который отображает всё React JS приложение;
- компонент `Basis.js` – элемент, отображающий страницу с линейным раскроем;
- компонент `NeedCutTable.js` предназначен для отображения таблицы с требуемыми заготовками;
- компонент `BeginCutTable.js` – элемент для отображения таблицы с исходными заготовками;
- компонент `Canvas.js` предназначен для отображения поля для рисования результатов раскроя;
- компонент `Rectangle.js` предназначен для рисования заготовок;

- компонент Header.js – элемент, отображающий меню приложения;
- компонент DoubleBasis.js предназначен для отображения страницы с двумерным раскроем;
- компонент DoubleCanvas.js предназначен для отображения поля с рисованием двумерных заготовок.

Код главного компонента App.js представлен в листинге 6.

Листинг 6 – Код главного компонента App.js

```
import './App.css';
import {lazy} from "react";
import { BrowserRouter, Routes, Route } from 'react-router-dom';
const Basis = lazy(()=> import('./component/Basis'));
const DoubleBasis = lazy(()=>import ('./component/DoubleBasis'));
function App() {
  return (
    <div className="App">
      <BrowserRouter>
        <Routes>
          <Route path="/" element={
            <Basis/>}>

          </Route>
          <Route path="/double"
            element={<DoubleBasis/>}>
          </Route>
        </Routes>
      </BrowserRouter>

    </div>
  );
}

export default App;
```

Полный исходный код клиентской части, расположен в приложении.

Таким образом была разработана клиентская часть автоматизированного средства рационального раскроя линейных и плоских материалов.

3.3 Разработка серверной части приложения

Серверная часть была реализована на Python с использованием фреймворка Fast API. В таблице 8 представлено описание REST API интерфейса.

Таблица 8 – Описание REST API интерфейса

Название функции	Строка адреса	Метод REST API	Описание
linear_cut	/linear-cut/	post	Реализует использование метода линейного программирования для решения задачи линейного раскроя
linear_cut_dynamic	/linear-cut-dynamic	post	Реализует использование «жадного» алгоритма для решения задачи линейного раскроя
linear_multi_cut	/linear-multi-cut	post	Реализует использование метода линейного программирования для решения задачи линейного раскроя с несколькими заготовками
history_cut	/history-cut	get	Реализует обращение к json файлу, содержащему историю раскроя
bivariate_cut	/bivariate-cut	post	Реализует использование жадного алгоритма для двумерного раскроя

Все методы являются асинхронными, то есть не требуют обратного ответа от клиентского приложения. В листинге 7 представлен код метода linear_cut REST API интерфейса серверного приложения.

Листинг 7 – Код REST API интерфейса приложения

```
@app.post("/linear-cut/", tags=["linear-cut"])
async def linear_cut(options_cut: model.LinearCutOptions):
    original_length = options_cut.original_length
    cuts_length = options_cut.cut_length
    cuts_count = options_cut.cut_count
    blade_thickness = options_cut.blade_thickness
    cutting_angle = options_cut.cutting_angle
    original_thickness = options_cut.original_thickness

    length_cutting = service.prepare_cuts(original_length, cuts_length,
blade_thickness, cutting_angle,
original_thickness)
    maps = service.linear_cut_method(original_length, cuts_length,
cuts_count)

    if len(maps) == 0:
        return []
    else:
        new_maps = service.recycle_maps_remains(original_length,
cuts_length, maps)
        result_maps = service.recycle_maps(original_length, cuts_length,
maps)
```

```

        service.restore_cuts(result_maps, length_cutting,
blade_thickness)

        data_in_bd = service.create_cutting_options(original_length,
cuts_length, cuts_count,
                                                    blade_thickness,
original_thickness, cutting_angle)
        record_in_bd = {
            'id': datetime.now().isoformat(),
            'data_input': data_in_bd,
            'data_result': new_maps
        }
        service.write_in_json(record_in_bd, data_file)
        return JsonResponse(content={"result_maps": result_maps, "maps":
new_maps})

```

Для того, чтобы передавать данные между клиентом и сервером необходимо формировать json строку с определенными полями. Сервер получает от клиента эти данные и десериализует их для передачи их в функции решения задач раскроя. Для этого были реализованы модели, содержащие необходимые типы данных (листинг 8).

Листинг 8 – Код моделей, содержащие необходимые типы данных

```

from pydantic import BaseModel
from typing import List
class LinearCutOptions(BaseModel):
    # длина исходного материала
    original_length: int
    # массив заготавливаемых длин
    cut_length: List[int]
    # массив заготавливаемых количеств
    cut_count: List[int]
    # толщина лезвия
    blade_thickness: float
    # угол реза
    cutting_angle: float
    # толщина исходной заготовки
    original_thickness: float

class SquareCutOptions(BaseModel):
    pieces: List[tuple]
    material_height: int
    material_width: int
class LinearMultiCutOptions(BaseModel):
    # длина исходного материала
    originals_length: List[int]
    # массив заготавливаемых длин
    cut_length: List[int]
    # массив заготавливаемых количеств
    cut_count: List[int]
    # толщина лезвия

```

```
blade_thickness: float
# угол реза
cutting_angle: float
# толщина исходной заготовки
original_thickness: float
```

`LinearCutOptions` будет содержать данные, которые необходимы для расчета, если введена одна исходная заготовка, а `LinearMultiCutOptions` будет содержать данные, где есть несколько исходных заготовок [23].

Таким образом была реализована серверная часть приложения для раскроя материалов.

3.4 Разработка программной реализации алгоритмов раскроя материалов

Задача о раскрое, описанная в разделе 1.1.3, может решаться несколькими способами. Существуют методы решения задачи линейного раскроя: методы линейного программирования, методы динамического программирования, генетический алгоритм и его виды. В автоматизированном средстве рационального раскроя были использованы методы: линейного программирования, динамического программирования.

Для решения задачи линейного раскроя методом линейного программирования использовалась библиотека `python pulp`. В ней были реализованы готовые методы для решения задач линейного программирования. В листинге 8 представлена функция, решающая задачу линейного раскроя методом линейного программирования.

Листинг 8 – Функция, реализующая решение задачи линейного раскроя методом линейного программирования

```
def linear_cut_method(original_length, cuts_length, cuts_count):
    # создаем карты раскроя и остатки
    possible_cuts, remainders = linear_cutting(original_length,
        cuts_length, cuts_count)

    # транспонируем матрицу для системы уравнений
    A = transpose_matrix(possible_cuts)

    # создаем описание задачи раскроя
    prob = pulp.LpProblem("Cutting Problem", pulp.LpMinimize)
```

```

# создаем переменные, одна переменная - вариант раскроя
num_variables = len(possible_cuts)
x = [pulp.LpVariable(f'x{i}', lowBound=0, cat='Integer') for i in
range(1, num_variables + 1)]

# задаем функцию, которую нужно минимизировать
prob += pulp.lpDot(remainders, x)

# задаем ограничения (коэффициенты справа)
rhs_values = cuts_count

# строим матрицу коэффициентов
constraints_coefficients = [tuple(comb) for comb in A]

# составляем систему уравнений
for i, constraint_coefficients in
enumerate(constraints_coefficients):
    prob += pulp.lpDot(constraint_coefficients, x) == rhs_values[i]

status = prob.solve(PULP_CBC_CMD(timeLimit=5))

if status == pulp.LpStatusOptimal:
    result_maps = []
    for v in prob.variables():
        if v.varValue != 0.0:
            var_index = int(v.name[1:]) - 1
            for j in range(int(v.varValue)):
                result_maps.append(possible_cuts[var_index])
            print(v.name, "=", v.varValue)

    print("Суммарная потеря материала:", pulp.value(prob.objective))
    print_arr(result_maps)
    return result_maps
else:
    print(pulp.LpStatus)
    return []

```

В данном случае, библиотека `pulp` сама подбирает наиболее оптимальный метод для решения задачи линейного программирования [24].

Вторым алгоритмом, решающим задачу линейного раскроя является «жадный» алгоритм. В листинге 9 представлен код функции, реализующий «жадный» алгоритм для линейного раскроя.

Листинг 9 – Функция, реализующая «жадный» алгоритм для линейного раскроя

```

def find_optimal_maps(original_length, cuts_length, counts):
    maps, remains = linear_cutting(original_length, cuts_length, counts)
    for i in range(len(maps)):
        maps[i].append(remains[i])
    sorted_maps = sorted(maps, key=lambda x: (-x[0], x[-1]))
    # print_arr(sorted_maps)
    selected_maps = []
    current_counts = [0] * len(counts)

```

```

for i in range(len(sorted_maps)):
    if sorted_maps[i][-1] == original_length:
        continue
    temp_map = sorted_maps[i]
    temp_map = temp_map[:-1]
    if sum_and_compare(temp_map, current_counts, counts):
        selected_maps.append(sorted_maps[i])
        current_counts = sum_array(current_counts, temp_map)

print_arr(selected_maps)
return selected_maps

```

Работа алгоритма заключается в следующем, сначала составляются карты раскроя заготовок, после чего происходит их отбор, по наименьшему остатку, подходящие карты раскроя сохраняются в итоговый вариант раскроя. В результате получается вариант раскроя для данной ситуации. Минусом алгоритма можно считать, что он не всегда может найти оптимальное решение, но может решить задачу раскроя, где обычное линейного программирование не работает.

В качестве алгоритм, реализующий двумерный раскрой был выбран «жадный» алгоритм (листинг 10)

Листинг 10 – Функция, реализующая «жадный» алгоритм для решения задачи двумерного раскроя.

```

def greedy_cutting_stock(pieces, material_width, material_height):
    pieces = sorted(pieces, key=lambda x: x[0] * x[1], reverse=True) #
    Сортируем заготовки по убыванию площади
    remaining_material = material_width * material_height
    used_pieces = []

    while remaining_material > 0:
        best_piece_index = -1
        best_fit = 0

        for i, piece in enumerate(pieces):
            if piece[0] <= material_width and piece[1] <=
material_height:
                fit = min(material_width // piece[0], material_height //
piece[1])
                if fit > best_fit:
                    best_fit = fit
                    best_piece_index = i

        if best_piece_index == -1: # Не нашли подходящей заготовки
            break

        used_pieces.append(pieces[best_piece_index])
        piece_width, piece_height = pieces[best_piece_index][0],

```

```

pieces[best_piece_index][1]
    area = piece_width * piece_height
    remaining_material -= area
    material_width -= piece_width
    material_height -= piece_height
    pieces.pop(best_piece_index)

return used_pieces

```

«Жадный» алгоритм для двумерного раскроя выбирает на каждом шаге самый подходящий прямоугольник из доступных и помещает его в текущее положение на плоскости. В самом простом случае, это может означать выбор прямоугольника с наименьшей высотой или шириной, который подходит для размещения в текущем свободном месте. «Жадный» алгоритм продолжает выбирать и размещать прямоугольники до тех пор, пока все прямоугольники не будут размещены. Однако «жадный» алгоритм может не давать оптимального решения, так как он не учитывает глобальную оптимизацию и может привести к неправильному распределению прямоугольников на плоскости. Также алгоритм реализует размещение только прямоугольных фигур. В дальнейшем планируется усовершенствование алгоритма [25].

Таким образом были реализованы алгоритмы для решения задач линейного и двумерного раскроя.

4 Контроль качества программного обеспечения автоматизированных средств раскроя линейных и плоских материалов

4.1 Планирование процесса контроля качества программного обеспечения

4.1.1 Мероприятия по контролю качества программного обеспечения

В ходе реализации автоматизированных средств рационального раскроя линейных и плоских материалов были выполнены мероприятия по контролю качества программного обеспечения. Было проведено тестирование и оценка качества кода.

Тестирование приложения – важная часть реализации программного обеспечения. В ходе тестирования производится выявления ошибок и проверяется корректность работы. Существуют следующие модели тестирования: функциональное тестирование, нефункциональное тестирование.

Функциональное тестирование представляет собой проверку функций приложения, которые уже готовы, но могут показывать некорректную работу на различные варианты поведения приложения.

К нефункциональному тестированию можно отнести:

- юнит-тестирование;
- тестирование с помощью интеграционных тестов;
- нагрузочное тестирование;
- тестирование пользовательского интерфейса;
- конфигурационное тестирование;
- тестирование с помощью автотестов.

В качестве моделей тестирования для программного обеспечения раскроя линейных и плоских материалов были выбраны тестирование с помощью юнит-тестов, тестирование с использованием интеграционных тестов.

Оценка качества кода – этап контроля качества программного обеспечения, которое позволяет произвести анализ программного кода с целью

выявления его недостатков с возможностью последующего рефакторинга. В качестве метрик программного кода были выбраны две метрики: SLOC и Холстеда.

4.1.2 Обоснование выбора модели тестирования

Для тестирования автоматизированных средств рационального раскроя линейных и плоских материалов было выбрано юнит-тестирование, автоматическое сквозное тестирование.

Юнит-тестирование – это вид тестирования, заключающаяся в проверке работоспособности отдельных функциональных модулей, процессов или частей кода приложения. Оно позволяет избежать ошибок, быстро исправить их при обновлении или дополнении программного обеспечения новыми компонентами, при этом не затрачивать время на его проверку целиком.

Преимущества использования модели юнит-тестирования:

- простота;
- параллельная разработка;
- информативность;
- имеется возможность повторного использования.

Недостатки использования системы юнит-тестирования:

- модульное тестирование не гарантирует, что будут найдены все ошибки;
- применяется к изолированным фрагментам кода, поэтому может выявить только ошибки проверяемого модуля;
- не способно выявить системные ошибки продукта в целом [26].

Сквозное тестирование – подход к тестированию программного обеспечения, который проверяет всё ПО, включая его взаимодействие с внешними интерфейсами от начала до конца. Также оно проверяет все ПО на предмет зависимостей, целостности данных и возможности подключения к другим системам, интерфейсам и базам данных для имитации полной производственной среды, пакетную обработку данных из различных систем восходящего или нисходящего потока в дополнение к системе программы.

Основная цель сквозного тестирования – идентифицировать системные зависимости и убедиться в целостности данных и связи с другими системами, интерфейсами и базами данных для обеспечения полной производительности.

Преимущества сквозного тестирования:

- комплексное тестирование;
- упрощённый процесс тестирования;
- реалистичные сценарии;
- тестирование, ориентированное на клиента;
- снижение будущих рисков;
- повышение уверенности в качестве продукта;
- раннее обнаружение ошибок.

Недостатки сквозного тестирования:

- трудоемкость;
- дороговизна;
- ограниченный охват;
- проблемы, которые трудно изолировать;
- обслуживание.

Автоматизированное тестирование предоставляет более продвинутые возможности. Автоматическое тестирование может быть более точным во многих ситуациях. Его сложно настроить и у него обычно более крутая кривая обучения. Требуется более высоких начальных затрат, но обеспечивает большую отдачу от вложенных сил по мере роста и развития процесса тестирования [27].

Таким образом выбранные модели тестирования обеспечили проверку работы программного обеспечения средства рационального раскроя линейных и плоских материалов.

4.1.3 Обоснование выбора метрик программного кода

Применение метрик в общем случае позволяет изучить сложность разработанного или разрабатываемого проекта, оценить объем работ, стилистику разрабатываемой программы и усилия, потраченные каждым разработчиком для

реализации того или иного решения. Существуют количественные метрики, метрики сложности потока программы, метрики сложности потока управления данных.

Количественные метрики: SLOC, метрика Холстеда, метрика Джилба.

SLOC – количество строк кода, является самой элементарной метрикой.

Данная метрика была изначально разработана для оценки трудозатрат по проекту. Различают физические и логические строки кода. Кроме SLOC к количественным характеристикам относят также: количество пустых строк, количество комментариев, процент комментариев, среднее число строк для функций, среднее число строк, среднее число строк для модулей.

Метрики Холстеда – группа метрик, основанных на подсчете некоторых единиц в коде программы. При использовании этих метрик частично компенсируются недостатки, связанные с возможностью записи одной и той же функциональности разным количеством строк и операторов.

Метрики Джилба – ещё один тип метрик программного обеспечения, относящийся к количественным. Они показывают сложность ПО на основе насыщенности программы операторами цикла или условными операторами [28].

Основными метриками были выбраны метрика SLOC и метрика Холстеда так как они позволяют в целом оценить объем и сложность программного кода.

4.2 Тестирование программных средств

4.2.1 Тестирование с помощью юнит-тестов

Для тестирования серверной части приложения были реализованы юнит-тесты. В таблице 9 представлены список юнит-тестов.

Таблица 9 – Юнит тесты серверной части приложения

Название юнит-теста	Описание
test_linear_cutting_multi	Тестирование функции для составления карт раскроя с несколькими исходными заготовками
test_merge_3d_array_to_2d	Тестирование функции преобразования трехмерного массива в двумерный
test_recycle_maps	Тестирование функции обработки результатов раскроя

Название юнит-теста	Описание
test_sum_and_compare	Тестирование функции поэлементного сравнения двух массивов
test_sum_array	Тестирование функции суммирования массива
test_compare_equal	Тестирование функции равенства двух массивов
test_linear_cutting	Тестирование функции для составления карт раскроя с одной исходной заготовкой
test_transpose_matrix	Тестирование функции транспонирования матрицы
test_linear_cut_method	Тестирование функции линейного раскроя с помощью линейного программирования
test_find_optimal_maps	Тестирование функции линейного раскроя жадным алгоритмом
test_bivariate_cut	Тестирование функции двумерного раскроя
test_linear_cut_method_multi	Тестирование функции линейного раскроя с несколькими исходными заготовками

Результаты прохождения тестов серверной части приложения представлены на рисунке 39.

```

Step 1/1: Python 20s
13:04:10 [0, 0, 2, 0, 1, 5, 2]
13:04:10 [1, 1, 0, 0, 0, 0, 3]
13:04:10 [1, 1, 0, 0, 0, 2, 0]
13:04:10 OK, linear_cut_method_multi
13:04:10 OK, linear_cutting
13:04:10 OK
13:04:10 Incorrect
13:04:10 OK
13:04:10 [[1500, 1500, 3000], [1500, 1500, 1000, 1000, 1000, 0], [1500, 1500, 1000, 1000, 1000, 0]]
13:04:10 OK, function recycle_maps was tested (1)
13:04:10 [[1500, 1500, 3000], [1500, 1500, 1000, 1000, 1000, 0], [1500, 1500, 1000, 1000, 1000, 0]]
13:04:10 OK, function recycle_maps was tested (2)
13:04:10 OK, sum_and_compare (1)
13:04:10 OK, sum_and_compare (2)
13:04:10 OK
13:04:10 OK, transpose_matrix
13:04:10
13:04:10
Name      Stats    Miss Cover Missing
-----
main.py    80      0  92%  42, 111-115, 125
model.py   20      0 100%
service.py 241    10   96%  29-30, 55, 57, 59, 230, 291-292, 330-339
test_main.py 27     27   0%  1-48
test_service.py 127    14   89%  27, 36, 44, 50, 63, 73, 91, 100, 126, 135, 149, 171, 181
TOTAL      495     57   88%
13:04:10 Wrote HTML report to htalcov/index.html
13:04:10 Process exited with code 0
13:04:10 Process successfully exited with code 0
13:04:20 Publishing artifacts < 1s
13:04:20 Publishing internal artifacts < 1s
13:04:20 Publishing internal artifacts < 1s
13:04:20 Publishing artifacts < 1s

```

Рисунок 39 – Результаты прохождения тестов серверной части приложения

Процент покрытия кода тестами серверной части приложения показан на рисунке 40.

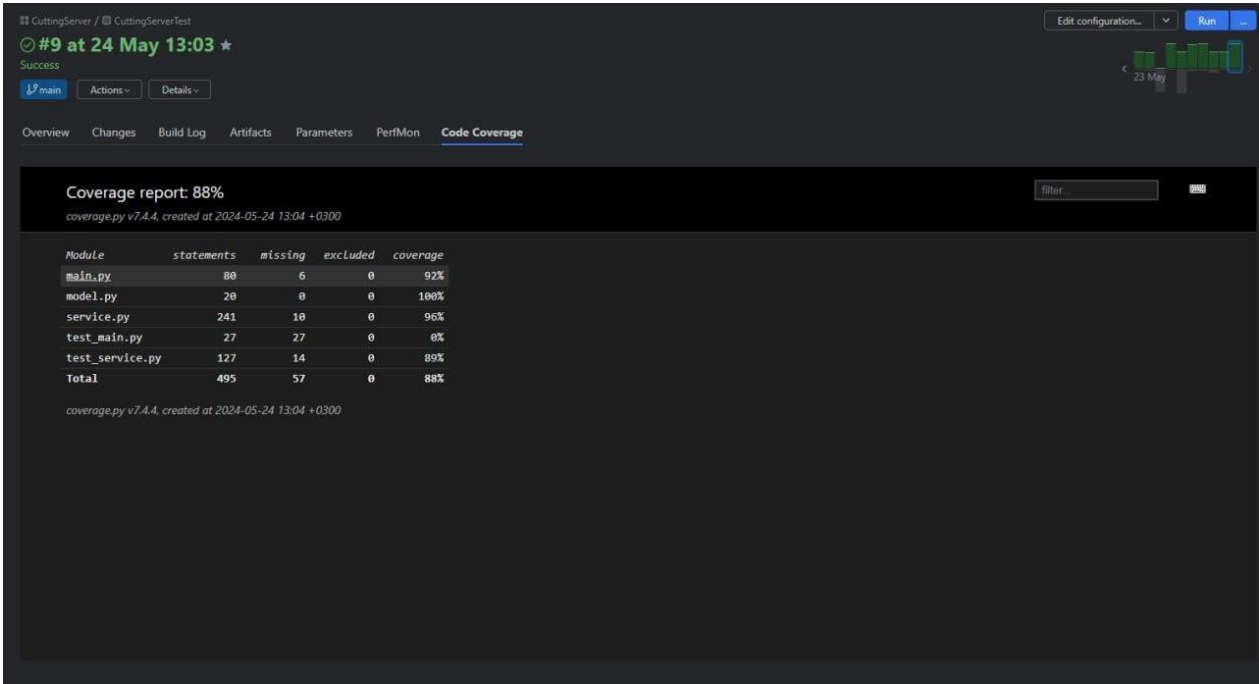


Рисунок 40 – Процент покрытия кода тестами серверной части приложения

Можно сделать вывод, что серверное часть приложения покрыто тестами на 88 процента, что является хорошим результатом.

Тестирование клиентской части частично было выполнено юнит-тестами. В таблице 10 представлены описание юнит-тестов клиентской части.

Таблица 10 – Юнит-тесты для клиентской части приложения

Название	Описание теста
NeedCutTable.test.jsx	Тестируем рендеринг компонента с таблицей требуемых заготовок
BeginCutTable.test.jsx	Тестируем рендеринг с таблицей исходных заготовок
Rectangle.test.jsx	Тестируем рисование прямоугольников при раскрое
CanvasDouble.test.jsx	Тестируем рисование холста и отображение на нем двумерного раскроя
Canvas.test.jsx	Тестируем рисование холста и отображением на нем линейного раскроя

Данные тесты будут производить проверку компонентов, которые являются основными в процессе рендеринга. Необходимо проверить как будут отрисовываться заготовки после нахождения вариантов раскроя.

На рисунке 41 показаны результаты выполнения тестов.

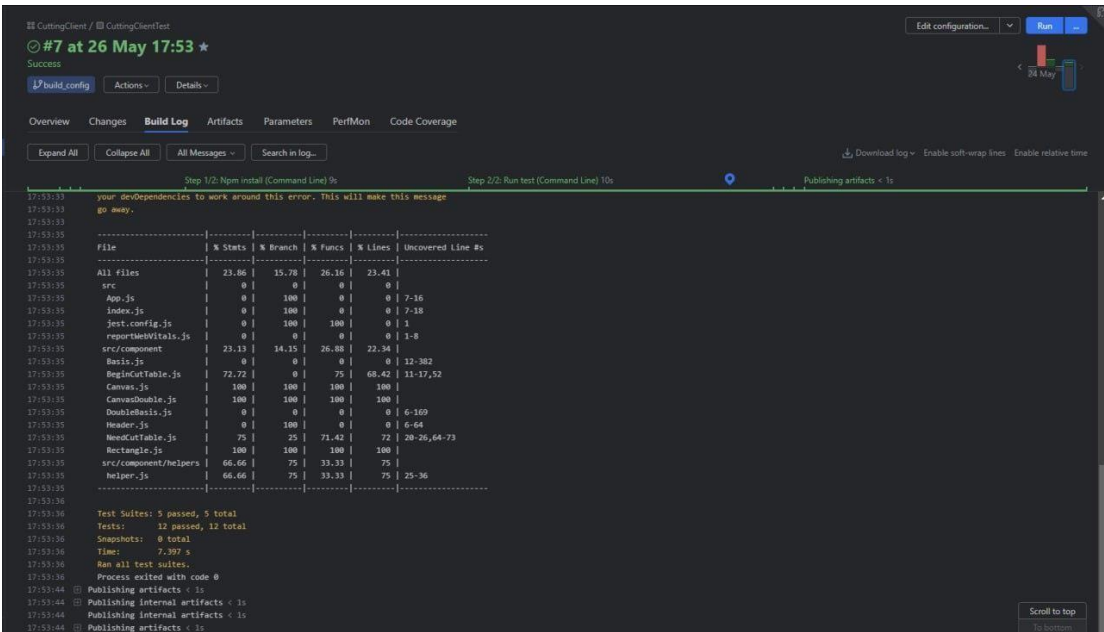


Рисунок 42 – Результат выполнения юнит-тестов

Отчет о покрытии тестами показан на рисунке 38.

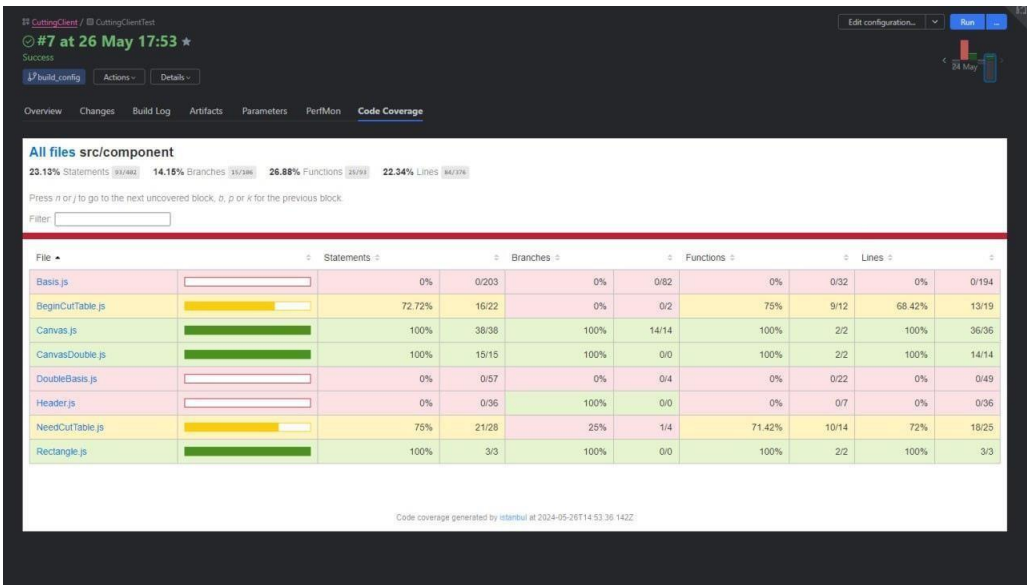


Рисунок 43 – Результат покрытия тестами

В целом по компонентам показан низкое покрытие (около 25%), но основные компоненты, для которых написаны юнит-тесты имеют хорошее покрытие. Другие компоненты были протестированы помощью автоматического сквозного тестирования.

4.2.2 Автоматическое сквозное тестирование

Для тестирования клиентской части, а также взаимодействия ее с серверной частью были написаны автотесты на языке JavaScript с помощью библиотеки Cypress [29]. В таблице 11 приведена их спецификация.

Таблица 11 – Спецификации автотестов

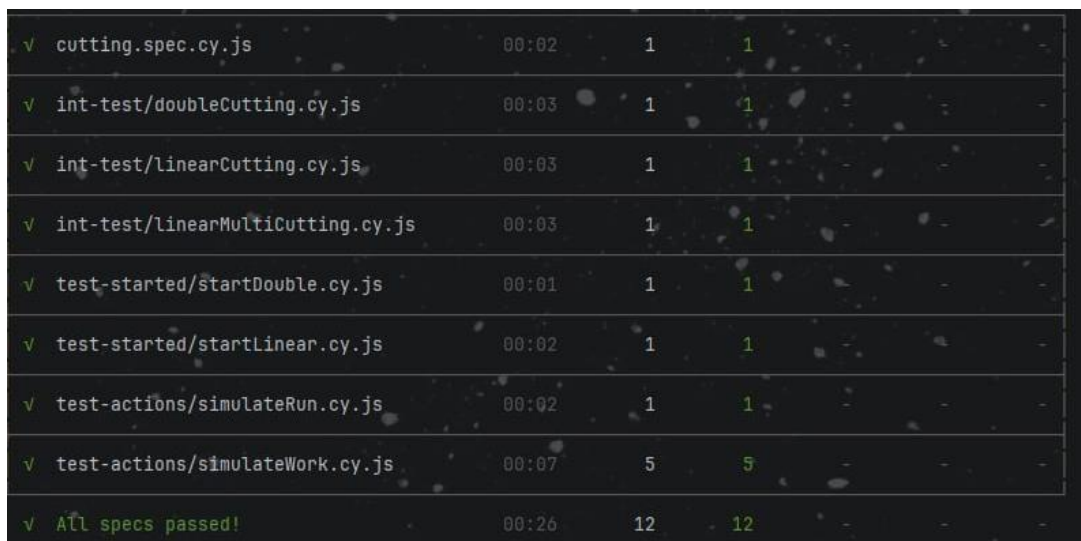
Название	Описание
startLinear	Проверяет рендер всех компонентов линейного раскроя
startDouble	Проверяет рендер всех компонентов двумерного раскроя
simulateWork	Симулирует работу с компонентами в странице, выполняет такие действия как, ввод исходных значений для раскроя, добавление и удаление строк таблицы и др.
simulateRun	Проверяет переход между страницами двумерного и линейного раскроя
linearCutting	Симулирует выполнение линейного раскроя, включающий ввод параметров и нажатие кнопки, после чего идет обращение к работающему в этот момент серверу
linearMultiCutting	Симулирует выполнение линейного раскроя с несколькими исходными заготовками, включающий ввод параметров и нажатие кнопки, после чего идет обращение к работающему в этот момент серверу
doubleCutting	Симулирует выполнение двумерного раскроя, включающий ввод параметров и нажатие кнопки, после чего идет обращение к работающему в этот момент серверу
cutting.spec	Проверяет доступ к странице со всеми видами раскроя

В процессе выполнения автотестов, было запущено средство для их управления и редактирования Cypress.

Cypress представляет собой фреймворк для написания сквозных тестов для React JS приложения.

Также был запущен сервер, для того чтобы проверить связь с ним, а также выполнение линейного и двумерного раскроя.

Общий результат выполнения автотестов представлен на рисунке 44.



✓ cutting.spec.cy.js	00:02	1	1	-	-
✓ int-test/doubleCutting.cy.js	00:03	1	1	-	-
✓ int-test/linearCutting.cy.js	00:03	1	1	-	-
✓ int-test/linearMultiCutting.cy.js	00:03	1	1	-	-
✓ test-started/startDouble.cy.js	00:01	1	1	-	-
✓ test-started/startLinear.cy.js	00:02	1	1	-	-
✓ test-actions/simulateRun.cy.js	00:02	1	1	-	-
✓ test-actions/simulateWork.cy.js	00:07	5	5	-	-
✓ All specs passed!	00:26	12	12	-	-

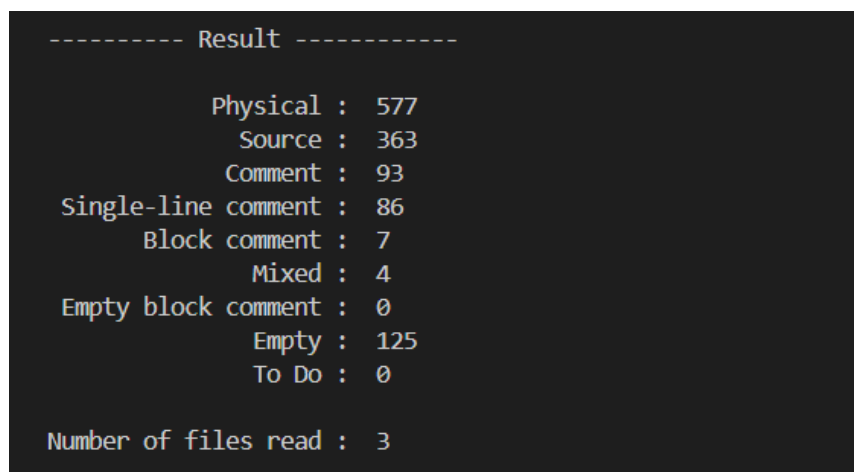
Рисунок 44 – Результат выполнения автотестов

Результат выполнения отдельных автотестов представлен в приложении Б.

4.3 Оценка качества программного кода на основе метрик

4.3.1 Метрика SLOC

Количество строк кода (SLOC) – метрика программного обеспечения, позволяющая определить объем кода путем подсчета строк в тексте исходного кода. Возьмем код серверной части приложения раскроя линейных и плоских материалов. Код приведен в приложение А. Для расчета метрики используем открытый исходный код, написанный на JavaScript, который позволяет автоматически подсчитать метрику SLOC [30]. На рисунке 45 показан результат измерений кода серверной части приложения.



```

----- Result -----
      Physical : 577
        Source : 363
        Comment : 93
Single-line comment : 86
      Block comment : 7
          Mixed : 4
Empty block comment : 0
          Empty : 125
          To Do : 0

Number of files read : 3

```

Рисунок 45 – Результат измерений серверной части приложения

Из приведенных данных можно выделить:

- физические строки кода – 577;
- логические строки кода – 363;
- комментарии – 93;
- пустые строки – 125.

Как видно из результатов измерений, достаточно большое число комментариев обеспечивает хорошую документированность кода, а в совокупности с пустыми строками обеспечивают лучшую читаемость кода.

4.3.2 Метрика Холстеда

Метрика Холстеда – это метрика, которая вычисляет число строк кода и синтаксических элементов исходного кода программы. Для расчета метрики были взяты такие элементы как: количество уникальных операторов, количество уникальных операндов, общее количество операторов, общее количество операндов [31]. В качестве измерений взять код service.py, расположенный в приложении. Операторы представлены в таблице 12.

Таблица 12 – Операторы

Операторы	Число вхождений
import	1
def	6
if	7
for	21
return	5
+	4
-	3
!=	3
<=	2
>	1
==	2
=	25
/	1
break	1
in	21

Продолжение таблицы 12

Операторы	Число вхождений
LpProblem	1
Find_combination	1
Find_closest_combination	1
Transpose_matrix	1
Count_remains	1
Doc_count_cut	1
lpDot	2
solve	1
[]	1
n1 = 24	N1 = 117

В таблице 13 представлены операнды.

Таблица 13 – Операнды

Операнды	Число вхождений
target	3
limits	2
pulp	6
count	4
combinations	11
_sum	3
Current	6
prices	3
i	30
new_sum	3
LpProblem	1
Problem	1
LpMinize	1
Paper	1
Result	3
L	2
sum_comb	2
remain	6
comb	16
j	12
rows	3

Продолжение таблицы 10

Операнды	Число вхождений
cols	3
Transposed	2
matrix	4
original_length	3
cuts_length	6
cuts_count	3
prob	5
closest_combinations	2
new_comb	2
comb_transpose	2
num_variables	2
x	6
coefficients	2
constraint_coefficients	2
rhs_values	2
var_index	2
varValue	2
name	1
corresponding_column	7
n2 = 44	N2 = 189

На основе этих показателей словарь программы будет рассчитываться по формуле:

$$n = n1 + n2 = 68,$$

где $n1$ – количество уникальных операторов;

$n2$ – количество уникальных операндов.

Длина программы рассчитывается по формуле:

$$N = N1 + N2 = 306,$$

где $N1$ – общее количество операторов;

$N2$ – общее количество операндов.

Объем программы рассчитывается по формуле:

$$HV = N * \log_2(n) = 1862.76.$$

Сложность программы:

$$HD = \left(\frac{n1}{2}\right) * \left(\frac{N2}{n2}\right) = 51.5.$$

Таким образом программа средний уровень сложности, при этом есть возможность рефакторинга с целью улучшения качества, что, следовательно, позволить понизить уровень сложности программы.

Заключение

В ходе выполнения выпускной квалификационной работы было выполнено:

- анализ предметной области;
- сравнительный анализ приложений аналогов;
- постановка задачи на разработку автоматизированных средств рационального раскроя линейных и плоских материалов;
- планирование процесса разработки программного обеспечения;
- проектирование автоматизированных средств рационального раскроя линейных и плоских материалов;
- разработка автоматизированных средств рационального раскроя линейных и плоских материалов;
- контроль качества программного обеспечения.

Была реализована десктопная версия приложения для раскроя линейных и плоских материалов. В дальнейшем планируется разработка мобильной версии и web-приложения и реализация новых функций в программном обеспечении.

Список использованных источников

1. Никитенков В. Л. О целочисленном решении задачи линейного раскроя / Никитенков В. Л. // Вестник Сыктывкарского университета. Сер. 1. Вып. 6. 2006. – 166 с.
2. П. Ю. Бунаков, Н. В. Каскевич. Новые технологии автоматизированного раскроя материалов для мебельного производства / П. Ю. Бунаков, Н. В. Каскевич. – М.: Московский государственный областной социально-гуманитарный институт, 2012. – 170 с.
3. Список (перечень) программ для раскроя материалов. [Электронный ресурс]. URL: <http://fk40.ru/programmy/718-spisok-perechen-programm-dlya-raskroya-materialov.html> (дата обращения 13.03.2024).
4. SmartCut Online оптимальный линейный раскрой. [Электронный ресурс]. URL: <https://smartcut.pro/ru/> (дата обращения 14.03.2024).
5. OptCut. [Электронный ресурс]. URL: <https://OptCut.html> (дата обращения 15.03.2024).
6. Astra S-Nesting. [Электронный ресурс]. URL: <https://astra-km.ru/software/astra-s-nesting> (дата обращения 16.03.2024).
7. Cutting. [Электронный ресурс]. URL: <https://cuttinghome.com/indexr.html> (дата обращения 17.03.2024).
8. Activity Diagram Tutorial [Электронный ресурс]. URL: <https://online.visual-paradigm.com/diagrams/tutorials/activity-diagram-tutorial/> (дата обращения 19.03.2024).
9. Диаграмма Ганта: как и где построить, шаблоны и примеры. [Электронный ресурс]. URL: <https://getcomprass.ru/blog/posts/diagramma-ganta> (дата обращения 20.03.2024).
10. Диаграмма прецедентов (use case диаграмма). [Электронный ресурс]. URL: <https://testengineer.ru/use-case-diagram/> (дата обращения 27.03.2024).

11. Марк Ричардс и Нил Форд – Фундаментальный подход к программной архитектуре. Паттерны, свойства, проверенные методы – СПб.:Питер, 2023 – 448 с.

12. Основные особенности и виды архитектур клиент-сервер. [Электронный ресурс]. URL: <https://www.ittelo.ru/news/osnovnye-osobennosti-i-vidy-arkhitektur-klient-server/> (дата обращения 7.04.2024).

13. Зачем нужно прототипирование интерфейсов. [Электронный ресурс]. URL: <https://inostudio.com/blog/articles-design/zachem-nuzhno-prototipirovanie-interfeysov/> (дата обращения 14.07.2024).

14. Знакомство с Azure DevOps. [Электронный ресурс]. URL: <https://habr.com/ru/companies/microsoft/articles/423367/> (дата обращения 16.04.2024).

15. TeamCity: настраиваем CI/CD в вашей команде [Электронный ресурс]. URL: <https://habr.com/ru/companies/tinkoff/articles/532546/> (дата обращения 19.04.2024).

16. WebStorm IDE Review [Электронный ресурс]. URL: <https://www.developer.com/languages/webstorm-ide-review/> (дата обращения 21.04.2024).

17. Pycharm: IDE для Data Science и веб-разработки на Python [Электронный ресурс]. URL: <https://www.jetbrains.com/ru-ru/pycharm/> (дата обращения 21.04.2024).

18. 23 приложения на Electron, о которых нужно знать [Электронный ресурс]. URL: <https://omgubuntu.ru/23-prilozheniia-dlia-linux/> (дата обращения 22.04.2024).

19. React. The library for web and native user interfaces [Электронный ресурс]. URL: <https://react.dev/> (дата обращения 23.04.2024).

20. Что такое CI/CD? Разбираемся с непрерывной интеграции и непрерывной поставкой [Электронный ресурс]. URL: <https://habr.com/ru/companies/otus/articles/515078/> (дата обращения 18.04.2024).

21. TeamCity: the Hassle-Free CI/CD Tool by JetBrains [Электронный ресурс]. URL: <https://www.jetbrains.com/teamcity/> (дата обращения 24.04.2024).
22. Цветовая схема приложения – 14 трендовых вариантов [Электронный ресурс]. URL: <https://webdevnet.ru/14-color-schemes-for-mobile-app/> (дата обращения 26.04.2024).
23. Знакомство с FastAPI [Электронный ресурс]. URL: <https://habr.com/ru/articles/488468/> (дата обращения 1.05.2024).
24. PULP БЖУ [Электронный ресурс]. URL: <https://habr.com/ru/articles/578796/> (дата обращения 2.05.2024).
25. Жадные алгоритмы | Алгоритмы на графах [Электронный ресурс]. URL: https://ru.hexlet.io/courses/algorithms-graphs/lessons/greedy-algorithms/theory_unit (дата обращения 4.05.2024).
26. Unit-тестирование [Электронный ресурс]. URL: <https://blog.skillfactory.ru/glossary/unit-testirovanie/> (дата обращения 7.05.2024).
27. E2E Testing tutorial: Complete Guide to End to End Testing With Examples [Электронный ресурс]. URL: <https://dev.to/lambdatest/e2e-testing-tutorial/> (дата обращения 10.05.2024).
28. Программный код и его метрики [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082> (дата обращения 12.05.2024).
29. Testing Types | Cypress Documentation [Электронный ресурс]. URL: <https://docs.cypress.io/guides/core-concepts/> (дата обращения 14.05.2024).
30. GitHub [Электронный ресурс]. URL: <https://docs.cypress.io/guides/core-concepts/testing-types> (дата обращения 17.05. 2024).
31. Halstead's Software Metrics – Software Engineering [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/software-halsteads> (дата обращения 19.05.2024).

Приложение А
(рекомендуемое)
Листинг программы

Листинг А.1 – Код компонента Basis.js

```
import React, {lazy, Suspense, useEffect, useState} from "react";
import axios from 'axios';
import Canvas from "./Canvas";
import Header from "./Header";
import {findAndCountDuplicates,
  transformArray,
  hasZeros,
  hasZeroIn,
  isLessThanAny} from "./helpers/helper"
//const { ipcRenderer } = window.require('electron');
let ipcRenderer;
try {
  const electronWindow = window.require('electron');
  ipcRenderer = electronWindow.ipcRenderer;
} catch (error) {
}

// подгружаем динамические таблицы
const NeedCutTable = lazy(()=> import('./NeedCutTable'));
const BeginCutTable = lazy(()=> import('./BeginCutTable'));

// базовый компонент
const Basis = React.memo(function Basis(props) {
  // вана коса проверяем занпужены ли данные
  const [isLoadingLocalData, setIsLoadingLocalData] = useState(false);

  // вана коса сохраняем исходные данные
  const [optionCutting, setOptionCut] = useState({});

  // Используем хук для обновления контекста данных между компонентами
  const [headers, setExportData] = useState([]);

  // вана коса результат загружать будем
  const [data, setData] = useState([]);

  // вана коса результат загружать в таблицу будем
  const [tableData, setTableResultData] = useState([]);

  // вана тоса проверка будет загрузились ли результаты
  const [loading, setLoading] = useState(true);

  // вана тоса проверка будет на идет ли процесс вычисления
  const [counting, setCounting] = useState(false);

  // вана коса установка длин заготовок
  const [cuts, setCuts] = useState([]);

  // вана коса ввод исходных материалов
  //-----
  const [inputValue, setInputValue] = useState(0);
  const handleInputChange = (event) => {
    if (event.target.value === '') {
      //setInputValue(0);
    }else{
```

```

        setInputValue(event.target.value);
    }

}
//-----

// вана коса установка данных в таблицу с исходными заготовками
//-----

const handleBeginDataChange = (inputValueTable)=>{
    setInputValueTable(inputValueTable)
}
const [inputValueTable, setInputValueTable] = useState([]);
//-----

// вана коса требуемые заготовки загружать будем
//-----

const [rows, setTableData] = useState([]);
// вана коса установка данных в таблицу с требуемыми заготовками
const handleTableDataChange = (data)=>{
    if (isLoadingLocalData){
        setTableData(data);
    }
};
//-----

// вана коса импортированные данные
//-----

const [importTableData, setImportTableData] = useState([]);
const handleImportData = (data)=>{

    setTableData([]);
    setImportTableData(transformArray(data));
}
//-----

// вана коса установка угла реза
//-----

const [angle, setAngle] = useState(0);
const handleAngle = (event)=>{
    setAngle(event.target.value);
}
//-----

// вана коса установка толщины лезвия
//-----

const [blade, setBlade] = useState(0);

```

```

const handleBlade = (event)=>{
  if (event.target.value === ''){
    //setBlade(0);
  }else{
    setBlade(event.target.value);
  }
}
//-----

```

```

// вана коса установка толщины заготовки
//-----

```

```

const [thickness, setThickness] = useState(0);
const handleThickness = (event)=>{
  if (event.target.value === ''){
    //setThickness(0);
  }else{
    setThickness(event.target.value);
  }
}
//-----

```

```

// вана коса установка раскрой с несколькими исходными материалами
//-----

```

```

const [multiLinear, setMultiLinear] = useState(false);
const handleMultiLinear = (e) =>{
  setMultiLinear(e.target.checked);
}
//-----

```

```

// сохранение локальных данных
useEffect(() => {
  if (isLoadingLocalData){
    let localData = [];
    localData.splice(0, localData.length);
    localData.push({"inputValue": inputValue});
    localData.push({"angle": angle});
    localData.push({"blade": blade});
    localData.push({"thickness": thickness});
    localData.push({"rows": rows});
    setOptionCut(localData);
    console.log(localData)
  }
}

```

```

}, [inputValue, angle, blade, thickness, rows]);

```

```

useEffect(()=>{
  if (ipcRenderer !== undefined){
    ipcRenderer.send('load-data');
  }
}

```

```

ipcRenderer.on('data-loaded', (event, load_data)=>{
  if (load_data.length === 0){
    setIsLoadingLocalData(true);
  }else{
    console.log(load_data);
    setInputValue(load_data[0].inputValue);
    setBlade(load_data[2].blade);
    setAngle(load_data[1].angle);
    setThickness(load_data[3].thickness);
    setImportTableData(load_data[4].rows);
    setIsLoadingLocalData(true);
    console.log(importTableData);
  }
});
return () => {
  ipcRenderer.removeAllListeners('data-loaded');
};
}, [isLoadingLocalData]);

// создание результата-таблицы заголовков
const renderTableHeader = () => {
  if (cuts.length === 0) {
    return null;
  }
  return (
    <tr>
      {cuts.map((column, index) => {
        return <th key={index}>{column}</th>;
      })}
      <th>Остаток</th>
      <th>Количество заготовок</th>
    </tr>
  );
};

// создание результата-таблицы данных
const renderTableData = (data) => {
  if (data.length === 0) {
    return <tr><td colSpan="3">Нет данных</td></tr>;
  }

  const columns = Object.keys(data[0]);

  return data.map((item, index) => {
    return (
      <tr key={index}>
        {
          columns.map((column, columnIndex) => {
            return <td
key={columnIndex}>{item[column]}</td>;
          })
        }
      </tr>
    );
  });
};

```

```

    });
};
// линейный раскрой с одним исходным материалом
const countCutsLinear = (event) => {
    event.preventDefault();
    setLoading(true);
    if(multiLinear){
        try {
            const cut_lengths = rows.map(row => Number(row.cuts));
            const counts = rows.map(row => Number(row.counts));
            const input_values = inputValueTable.map(row=>
                Number(row.cuts));

            if (cut_lengths.length === 0 || counts.length === 0){
                ipcRenderer.send(
                    "incorrect-enter", {
                        "type": 1,
                        "message": "Не введены заготовки или их
количества !"
                    });
                return;
            }else if (cut_lengths.length !== counts.length) {
                ipcRenderer.send("incorrect-enter", {
                    "type": 1,
                    "message": "Количество заготовок и их количества
не совпадают !"
                });
                return;
            }else if (blade > 5){
                ipcRenderer.send("incorrect-enter", {"type": 1,
"message": "Толщина лезвия не может быть больше 5 мм !"});
                return;
            }else if (angle !== 0 && angle !== '0' && (thickness ===
0 || thickness === '')){
                ipcRenderer.send("incorrect-enter", {"type": 1,
"message": "При угле реза, отличным от нуля, требуется ввод толщины
заготовки !"});
                return;
            }else if (hasZeros(cut_lengths) || hasZeros(counts) ||
hasZeroIn(cut_lengths) || hasZeroIn(counts)){
                ipcRenderer.send("incorrect-enter", {"type": 1,
"message": "Длины или количества заготовок содержат нули !"});
                return;
            }else if (thickness > 100){
                ipcRenderer.send("incorrect-enter", {"type": 1,
"message": "Ограничение по толщине заготовки ! (Не больше 100 мм)"});
                return;
            }else if (hasZeros(input_values) ||
hasZeroIn(input_values)){
                ipcRenderer.send("incorrect-enter", {"type": 1,
"message": "Исходные заготовки не могут быть равны 0"});
                return;
            }
        }
        const cutOptions = {
            'originals_length': input_values,
            'cut_length':cut_lengths,
            'cut_count':counts,

```

```

        'blade_thickness':blade,
        'cutting_angle':angle,
        'original_thickness':thickness
    };
    setCounting(true);
    let result_data_draw;
    let result_data_table;
    let headers = [];
    console.log(cutOptions)
    axios.post('http://localhost:8000/linear-multi-cut',
cutOptions)
        .then(response=>{
            console.log(response.data)
            result_data_draw = response.data["result_maps"];
            result_data_table = response.data["maps"];
            setCuts(cut_lengths);
            for (let i = 0; i < cut_lengths.length; i++){
                headers.push(cut_lengths[i].toString());
            }
            headers.push("Остаток");
            headers.push("Количество заготовок")
            setExportData(headers);

setTableResultData(findAndCountDuplicates(result_data_table));
            setData(result_data_draw);
            setLoading(false);
            setCounting(false);
        }).catch(error=>{
            console.error(error);
        });

    } catch (e) {
        console.error('Error send', e);
    }
} else {
    try {
        const cut_lengths = rows.map(row => Number(row.cuts));
        const counts = rows.map(row => Number(row.counts));

        if (cut_lengths.length === 0 || counts.length === 0){
            ipcRenderer.send("uncorrect-enter", {"type": 1,
"message": "Не введены заготовки или их количества !"});
            return;
        } else if (cut_lengths.length !== counts.length){
            ipcRenderer.send("uncorrect-enter", {"type": 1,
"message": "Количество заготовок и их количества не совпадают !"});
            return;
        } else if (isLessThanAny(inputValue, cut_lengths)){
            ipcRenderer.send("uncorrect-enter", {"type": 1,
"message": "Одна из заготовок больше чем исходная заготовка!"});
            return;
        } else if (blade > 5){
            ipcRenderer.send("uncorrect-enter", {"type": 1,
"message": "Толщина лезвия не может быть больше 5 мм !"});
            return;
        } else if (angle !== 0 && angle !== '0' && (thickness ===
0 || thickness === '')){

```

```

        ipcRenderer.send("uncorrect-enter", {"type": 1,
"message": "При угле реза, отличным от нуля, требуется ввод толщины заготовки !"});
        return;
    }else if (hasZeros(cut_lengths) || hasZeros(counts) ||
hasZeroIn(cut_lengths) || hasZeroIn(counts)){
        ipcRenderer.send("uncorrect-enter", {"type": 1,
"message": "Длины или количества заготовок содержат нули !"});
        return;
    }else if (thickness > 100){
        ipcRenderer.send("uncorrect-enter", {"type": 1,
"message": "Ограничение по толщине заготовки ! (Не больше 100 мм)"});
        return;
    }else if (inputValue === 0 || inputValue === ''){
        ipcRenderer.send("uncorrect-enter", {"type": 1,
"message": "Исходная заготовка не может быть равна 0"});
        return;
    }

    const cutOptions = {
        'original_length': inputValue,
        'cut_length': cut_lengths,
        'cut_count': counts,
        'blade_thickness': blade,
        'cutting_angle': angle,
        'original_thickness': thickness
    };
    setCounting(true);
    let result_data_draw;
    let result_data_table;
    let headers = [];
    axios.post('http://localhost:8000/linear-cut/',
cutOptions)
        .then(response=>{
            console.log(response.data)
            result_data_draw = response.data["result_maps"];
            result_data_table = response.data["maps"];
            if (result_data_draw.length === 0)
            {
                axios.post('http://localhost:8000/linear-cut-
dynamic', cutOptions).then(
                    response=>{
                        result_data_draw =
response.data["result_maps"];
                        result_data_table =
response.data["maps"];
                        setCuts(cut_lengths);
                        setData(result_data_draw);
                        for (let i = 0; i <
cut_lengths.length; i++){
                            headers.push(cut_lengths[i].toString());
                        }
                        headers.push("Остаток");
                        headers.push("Количество заготовок")
                    }
                )
            }
        })
    }
}

```

```

        setExportData(headers);

setTableResultData(findAndCountDuplicates(result_data_table));
        setLoading(false);
        setCounting(false);
    }
    ).catch(error=>{
        console.log(error);
    })
} else {
    setCuts(cut_lengths);
    for (let i = 0; i < cut_lengths.length; i++) {
        headers.push(cut_lengths[i].toString());
    }
    headers.push("Остаток");
    headers.push("Количество заготовок");
    setExportData(headers);

setTableResultData(findAndCountDuplicates(result_data_table));
        setData(result_data_draw);
        setLoading(false);
        setCounting(false);
    }
    }).catch(error=>{
        console.error(error);
    });

    } catch (e) {
        console.error('Error send', e);
    }
}

};

return(
    <div data-testid="basis-component">
        <Header exportTable={tableData} headers={headers}
cuttingOption={optionCutting}
            onImportData={handleImportData}/>
        <div className="divBasis" data-testid="basis">
            <label className="toggle">
                <input className="toggle-checkbox" type="checkbox"
checked={multiLinear}
                    onChange={handleMultiLinear}/>
                <div className="toggle-switch" data-
testid="toggle"></div>
                <span className="toggle-label">Несколько исходных
заготовок</span>
            </label>
            {multiLinear ? (<form data-testid="form-input-multi"
onSubmit={countCutsLinear}>
                <div className="begin">

                    <ul>
                        <li>
                            <label className="form__label">Толщина
лезвия (мм)</label>

                        </li>

```

```

        <li>
            <input placeholder="Толщина лезвия (мм) "
onChange={handleBlade}
                                name="blade"
                                className="form__input"
type="number" value={blade} min="0"/>
        </li>
    </ul>
    <ul>
        <li>
            <label className="form__label">Толщина
исходной заготовки (мм)</label>

            </li>
            <li>
                <input placeholder="Толщина исходной
заготовки (мм) "
                                onChange={handleThickness}
                                name="thickness"
                                className="form__input"
                                type="number" value={thickness}
                                min="0"/>
            </li>
        </ul>
        <ul>
            <li>
                <label className="form__label">Угол
реза</label>

            </li>
            <li>
                <div className="select-container">
                    <select className="selectAngle"
value={angle} onChange={handleAngle}>
                        {/* Опции для выбора */}
                        <option value="0">0</option>
                        <option value="30">30</option>
                        <option value="45">45</option>
                        <option value="60">60</option>
                    </select>
                </div>
            </li>
        </ul>
        <button className="btnAction" id="countBtnMulti"
type="submit">Рассчитать</button>

    </div>
    <Suspense fallback={<div>Загрузка...</div>}>
        <BeginCutTable
setInputValueTable={handleBeginDataChange}/>
    </Suspense>
    <Suspense fallback={<div>Загрузка...</div>}>
        <NeedCutTable data={importTableData}
onSetTableData={handleTableDataChange}/>
    </Suspense>

    </form>) : (<form data-testid="form-input-simple"
onSubmit={countCutsLinear}>

```

```

<div className="begin">
  <ul>
    <li>
      <label className="form__label">Длина
исходной заготовки (мм)</label>

      </li>
      <li>
        <input placeholder="Длина (мм) "
onChange={handleInputChange}
              name="original_length"
              className="form_input"
              type="number"
              value={inputValue} min="0"/>

      </li>
    </ul>
    <ul>
      <li>
        <label className="form__label">Толщина
лезвия (мм)</label>

      </li>
      <li>
        <input placeholder="Толщина лезвия (мм) "
onChange={handleBlade}
              name="blade"
              className="form_input"
              type="number"
              value={blade} min="0"/>

      </li>
    </ul>
    <ul>
      <li>
        <label className="form__label">Толщина
исходной заготовки (мм)</label>

      </li>
      <li>
        <input placeholder="Толщина исходной
заготовки (мм) "
onChange={handleThickness}
              name="thickness"
              type="number" value={thickness}
              min="0"/>

      </li>
    </ul>
    <ul>
      <li>
        <label className="form__label">Угол
пеза</label>

```

```

        </li>
        <li>
            <div className="select-container">
                <select className="selectAngle"
value={angle} onChange={handleAngle}>
                    { /* Опции для выбора */ }
                    <option value="0">0</option>
                    <option value="30">30</option>
                    <option
value="45">45</option><option
value="60">60</option>
                </select>
            </div>
        </li>
    </ul>
    <button className="btnAction" id="countBtn"
type="submit">Рассчитать</button>
    </div>
    <Suspense fallback={<div>Загрузка...</div>}>
        <NeedCutTable data={importTableData}
onSetTableData={handleTableDataChange}/>
    </Suspense>
</form>)}
{counting ? (
    <div className="wrapperLoader">
        <div className="loader">
            <div className="inner one"></div>
            <div className="inner two"></div>
            <div className="inner three"></div>
        </div>
    </div>

) : (<p></p>)}
<div className="res">
    {loading ? (<p></p>) : (
        <div className="res-wrapper" data-testid="res">
            <h3> Вариант раскроя </h3>
            <table className="resultTable">
                <thead>
                    {renderTableHeader()}
                </thead>
                <tbody>
                    {renderTableData(tableData)}
                </tbody>
            </table>
            <p>Подсказка: черным цветом показано
использование заготовки, серым остаток</p>
            <Canvas maps={data} thickness={thickness}
angle={angle}/>
            <div className="padd"></div>
        </div>
    )}
</div>

</div>

</div>

);
});

```

```
export default Basis;
```

Листинг А.2 – Код service.py, содержащий функции для выполнения раскроя

```
import json

import pulp
from pulp import PULP_CBC_CMD
import math

""" здесь начинается описание сервисных функций """
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% """

# функция составления опций резки для бд
def create_cutting_options(original_length, cuts_length, cuts_count,
blade, thickness, angle):
    dict_option = {}
    dict_option['original_length'] = original_length
    dict_option['cuts_length'] = cuts_length
    dict_option['cuts_count'] = cuts_count
    dict_option['blade'] = blade
    dict_option['thickness'] = thickness
    dict_option['angle'] = angle

    return dict_option

# функция записи в json файл
def write_in_json(data, file_name):
    try:
        with open(file_name, 'r') as file:
            data_json = json.load(file)
    except (FileNotFoundError, json.JSONDecodeError):
        data_json = []

    data_json.append(data)
    with open(file_name, 'w') as file:
        json.dump(data_json, file, indent=4)

# функция для восстановления исходных данных с учетом угла реза и толщины
лезвия
# -----
def restore_cuts(maps, linear_cutting, blade_thickness):
    for i in range(len(maps)):
        for j in range(len(maps[i])):
            # if j == 0:
            # maps[i][j] = int(math.ceil(maps[i][j] + 2 * linear_cutting
            - blade_thickness))
            if j != len(maps[i]) - 1:
                maps[i][j] = int(math.ceil(maps[i][j] + 2 *
linear_cutting - blade_thickness))

# -----
```

```

# функция для подготовки исходных заготовок с учетом угла реза
# -----

def prepare_cuts(original_length, cuts_length, blade_thickness,
cutting_angle, original_thickness):
    length_cutting = 0
    if cutting_angle == 0:
        length_cutting = 0
    elif cutting_angle == 30:
        length_cutting = math.ceil((math.sqrt(3) * original_thickness) /
3)
    elif cutting_angle == 60:
        length_cutting = math.ceil(math.sqrt(3) * original_thickness)
    elif cutting_angle == 45:
        length_cutting = original_thickness

    for i in range(len(cuts_length)):
        cuts_length[i] = math.ceil(cuts_length[i] - 2 * length_cutting)
        cuts_length[i] += blade_thickness
        cuts_length[i] = int(cuts_length[i])

    original_length += blade_thickness
    return length_cutting

# -----

# функция приведения результатов раскроя к нужному виду (добавление
остатков)
# -----

def recycle_maps_remains(original_length, cuts_length, maps):
    result_maps = []
    for i in range(len(maps)):
        summ = 0
        map = []
        for j in range(len(maps[i])):
            map.append(maps[i][j])
            if maps[i][j] != 0:
                for k in range(maps[i][j]):
                    summ += cuts_length[j]
        map.append(original_length - summ)
        result_maps.append(map)
    return result_maps

# функция приведения результатов раскроя к нужному виду
# -----

def recycle_maps(original_length, cuts_length, maps):
    result_maps = []
    if len(cuts_length) != len(maps[0]):
        for i in range(len(maps)):
            map = []
            for j in range(len(maps[i]) - 1):
                if maps[i][j] != 0:
                    for k in range(maps[i][j]):
                        map.append(cuts_length[j])
            map.append(maps[i][-1])
            result_maps.append(map)

```

```

else:
    for i in range(len(maps)):
        map = []
        summ = 0
        for j in range(len(maps[i])):
            if maps[i][j] != 0:
                for k in range(maps[i][j]):
                    summ += cuts_length[j]
                    map.append(cuts_length[j])
        map.append(original_length - summ)
        result_maps.append(map)

    return result_maps

# -----
# -----

# функция поэлементного сравнения двух массивов
# -----
# -----

def compare_equal(arr1, arr2):
    for el1, el2 in zip(arr1, arr2):
        if el1 != el2:
            return True
        else:
            return False

# -----
# -----

# функция печати массива
# -----
# -----

def print_arr(A):
    for i in range(len(A)):
        print(A[i])

# -----
# -----

# поэлементное суммирование массива
# -----
# -----

def sum_array(arr1, arr2):
    result = []
    for i in range(len(arr1)):
        # Суммируем элементы поэлементно первых двух массивов
        sum_elements = arr1[i] + arr2[i]
        result.append(sum_elements)
    return result

# поэлементное сравнение элементов на превышение
# -----

```

```

-----
def sum_and_compare(arr1, arr2, arr3):
    result = sum_array(arr1, arr2)
    for i in range(len(result)):
        if result[i] > arr3[i]:
            return False
    return True

# -----

# транспонирование матрицы
# -----

def transpose_matrix(matrix):
    rows, cols = len(matrix), len(matrix[0])
    transposed = [[0 for _ in range(rows)] for _ in range(cols)]

    for i in range(rows):
        for j in range(cols):
            transposed[j][i] = matrix[i][j]

    return transposed

# -----

# составление всех возможных карт раскроя, а также остатков, получаемых
# из каждой карты
# -----

def linear_cutting(length, cut_lengths, cut_counts):
    def generate_cuts(length, cut_lengths, cut_counts, current_cut,
result, remainders):
        if not cut_lengths:
            result.append(current_cut.copy() + cut_counts)
            remainders.append(length)
            return

        for i in range(min(length // cut_lengths[0], cut_counts[0]) + 1):
            generate_cuts(length - i * cut_lengths[0], cut_lengths[1:],
cut_counts[1:], current_cut + [i], result,
remainders)

    result = []
    remainders = []
    generate_cuts(length, cut_lengths, cut_counts, [], result,
remainders)
    return result, remainders
# -----

# составление всех возможных карт раскроя, с несколькими заготовками
# -----

```

```

def linear_cutting_multi(original_lengths, cut_lengths, cut_counts):
    results = []
    remainders_list = []

    for length in original_lengths:
        result, remainders = linear_cutting(length, cut_lengths,
cut_counts)
        results.append(result)
        remainders_list.append(remainders)

    return results, remainders_list
# -----

# преобразование трехмерного массива в двумерный при мульти линейном
раскрое
# -----

def merge_3d_array_to_2d(arr):
    merged_array = []
    for i in range(len(arr)):
        for j in range(len(arr[i])):
            merged_array.append(arr[i][j])
    return merged_array
# -----

# размещение заготовок в определенной площади
# -----

def place_rectangles(total_area, smaller_rectangles, smaller_rect_count):
    remaining_area = total_area
    result = []

    for i in range(len(smaller_rectangles)):
        for j in range(smaller_rect_count[i]):
            if smaller_rectangles[i] <= remaining_area:
                result.append(smaller_rectangles[i])
                remaining_area -= smaller_rectangles[i]

    if remaining_area == 0:
        return result
    else:
        return []
# -----

"""
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% """
""" здесь заканчивается описание сервисных функции"""

""" здесь начинается описание методов решения задач раскроя"""
"""
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% """
def linear_cut_method_multi(original_length, cuts_length, cuts_count):

```

```

# создаем карты раскроя и остатки
possible_cuts, remainders = linear_cutting_multi(original_length,
cuts_length, cuts_count)

possible_cuts = merge_3d_array_to_2d(possible_cuts)
remainders = merge_3d_array_to_2d(remainders)

# транспонируем матрицу для системы уравнений
A = transpose_matrix(possible_cuts)

# создаем описание задачи раскроя
prob = pulp.LpProblem("Cutting Problem", pulp.LpMinimize)

# создаем переменные, одна переменная - вариант раскроя
num_variables = len(possible_cuts)
x = [pulp.LpVariable(f'x{i}', lowBound=0, cat='Integer') for i in
range(1, num_variables + 1)]

# задаем функцию, которую нужно минимизировать
prob += pulp.lpDot(remainders, x)

# задаем ограничения (коэффициенты справа)
rhs_values = cuts_count

# строим матрицу коэффициентов
constraints_coefficients = [tuple(comb) for comb in A]

# составляем систему уравнений
for i, constraint_coefficients in
enumerate(constraints_coefficients):
    prob += pulp.lpDot(constraint_coefficients, x) == rhs_values[i]

status = prob.solve(PULP_CBC_CMD(timeLimit=5))

if status == pulp.LpStatusOptimal:
    result_maps = []
    for v in prob.variables():
        if v.varValue != 0.0:
            var_index = int(v.name[1:]) - 1 # Получение индекса
переменной из имени
            for j in range(int(v.varValue)):
                result_maps.append(possible_cuts[var_index])
            print(v.name, "=", v.varValue)

    print("Суммарная потеря материала:", pulp.value(prob.objective))
    print_arr(result_maps)
    return result_maps
else:
    print(pulp.LpStatus)
    return []

# решение задачи линейного раскроя методом линейного программирования
# -----
def linear_cut_method(original_length, cuts_length, cuts_count):
    # создаем карты раскроя и остатки
    possible_cuts, remainders = linear_cutting(original_length,

```

```

cuts_length, cuts_count)

# транспонируем матрицу для системы уравнений
A = transpose_matrix(possible_cuts)

# создаем описание задачи раскроя
prob = pulp.LpProblem("Cutting Problem", pulp.LpMinimize)

# создаем переменные, одна переменная - вариант раскроя
num_variables = len(possible_cuts)
x = [pulp.LpVariable(f'x{i}', lowBound=0, cat='Integer') for i in
range(1, num_variables + 1)]

# задаем функцию, которую нужно минимизировать
prob += pulp.lpDot(remainders, x)

# задаем ограничения (коэффициенты справа)
rhs_values = cuts_count

# строим матрицу коэффициентов
constraints_coefficients = [tuple(comb) for comb in A]

# составляем систему уравнений
for i, constraint_coefficients in
enumerate(constraints_coefficients):
    prob += pulp.lpDot(constraint_coefficients, x) == rhs_values[i]

status = prob.solve(PULP_CBC_CMD(timeLimit=5))

if status == pulp.LpStatusOptimal:
    result_maps = []
    for v in prob.variables():
        if v.varValue != 0.0:
            var_index = int(v.name[1:]) - 1 # Получение индекса
переменной из имени
            for j in range(int(v.varValue)):
                result_maps.append(possible_cuts[var_index])
            print(v.name, "=", v.varValue)

    print("Суммарная потеря материала:", pulp.value(prob.objective))
    print_arr(result_maps)
    return result_maps
else:
    print(pulp.LpStatus)
    return []

# -----

# решение задачи линейного раскроя жадным алгоритмом
# -----

def find_optimal_maps(original_length, cuts_length, counts):
    maps, remains = linear_cutting(original_length, cuts_length, counts)
    for i in range(len(maps)):
        maps[i].append(remains[i])

```

```

sorted_maps = sorted(maps, key=lambda x: (-x[0], x[-1]))
# print_arr(sorted_maps)
selected_maps = []
current_counts = [0] * len(counts)
for i in range(len(sorted_maps)):
    if sorted_maps[i][-1] == original_length:
        continue
    temp_map = sorted_maps[i]
    temp_map = temp_map[:-1]
    if sum_and_compare(temp_map, current_counts, counts):
        selected_maps.append(sorted_maps[i])
        current_counts = sum_array(current_counts, temp_map)

print_arr(selected_maps)
return selected_maps
# -----

# размещение двумерных заготовок на площади
# -----

def greedy_cutting_stock(pieces, material_width, material_height):
    pieces = sorted(pieces, key=lambda x: x[0] * x[1], reverse=True) #
Сортируем заготовки по убыванию площади
    remaining_material = material_width * material_height
    used_pieces = []

    while remaining_material > 0:
        best_piece_index = -1
        best_fit = 0

        for i, piece in enumerate(pieces):
            if piece[0] <= material_width and piece[1] <=
material_height:
                fit = min(material_width // piece[0], material_height //
piece[1])
                if fit > best_fit:
                    best_fit = fit
                    best_piece_index = i

        if best_piece_index == -1: # Не нашли подходящей заготовки
            break

        used_pieces.append(pieces[best_piece_index])
        piece_width, piece_height = pieces[best_piece_index][0],
pieces[best_piece_index][1]
        area = piece_width * piece_height
        remaining_material -= area
        material_width -= piece_width
        material_height -= piece_height
        pieces.pop(best_piece_index)

    return used_pieces
# -----

```

```
# -----  
-----  
""" здесь заканчивается описание методов раскрыя"""
```

Приложение Б
(рекомендуемое)
Результаты выполнения автотестов

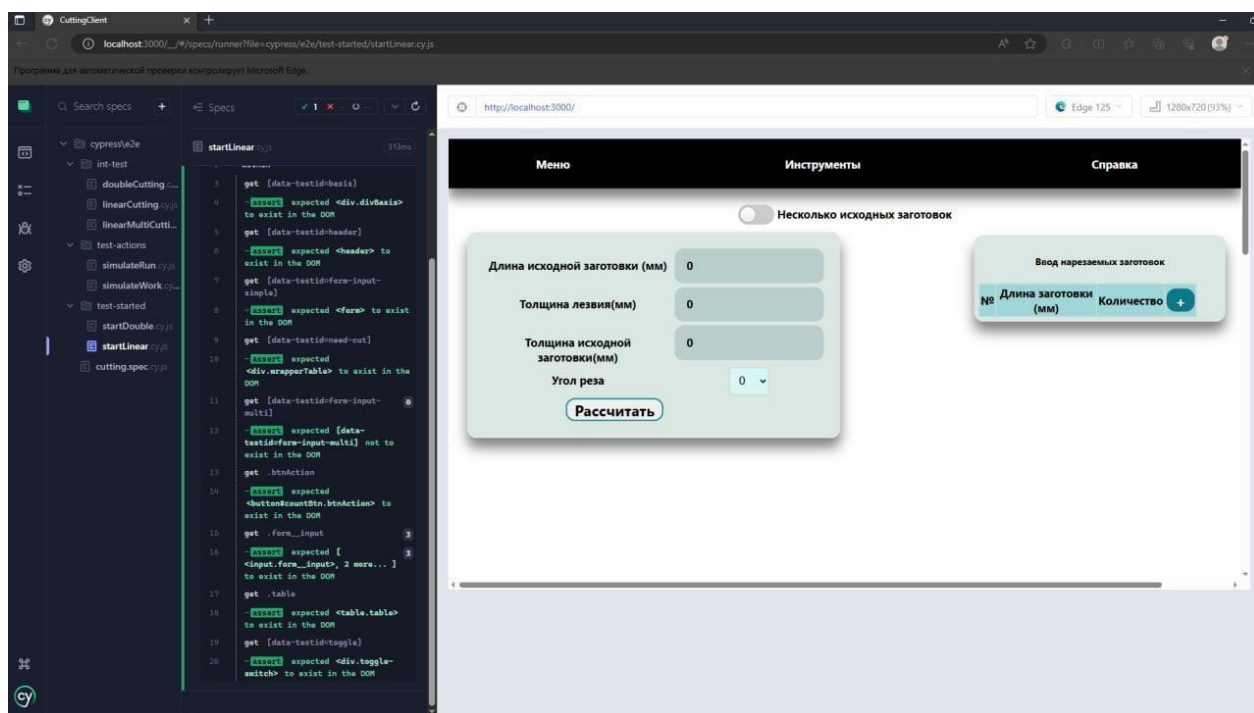


Рисунок Б.1 – Результат выполнение автотеста startLinear

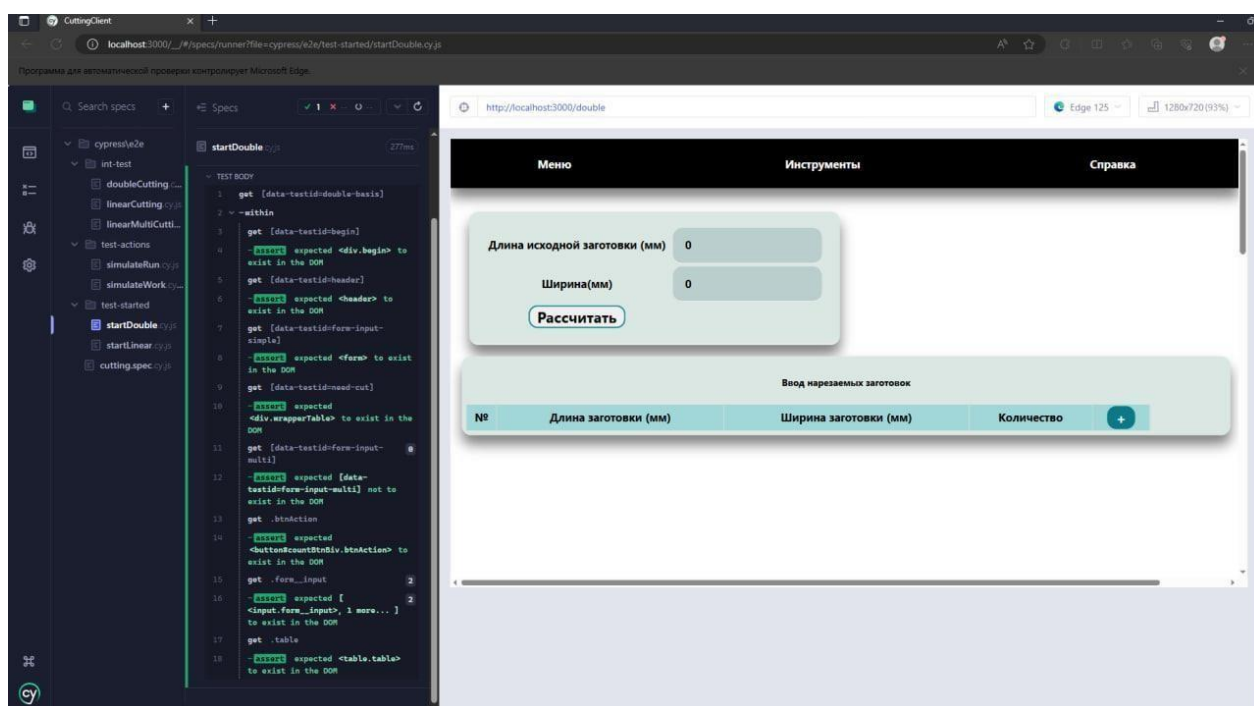


Рисунок Б.2 – Результат выполнение автотеста startDouble

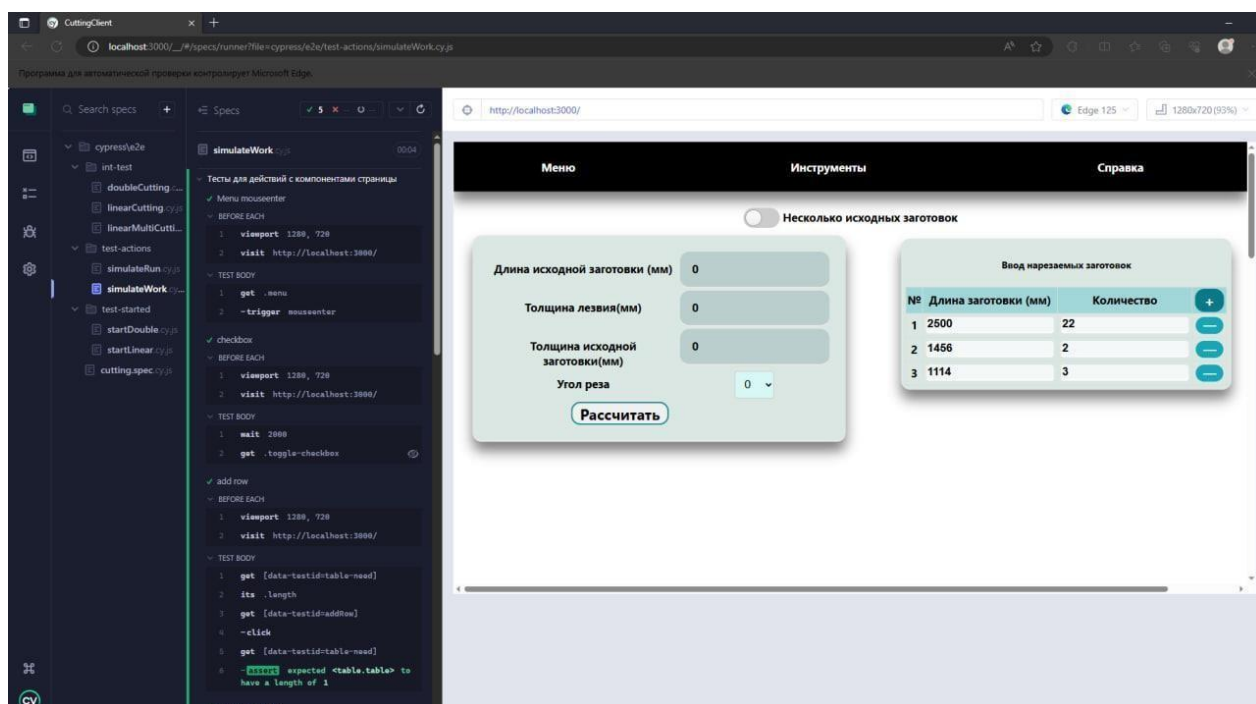


Рисунок Б.3 – Результат выполнение автотеста simulateWork

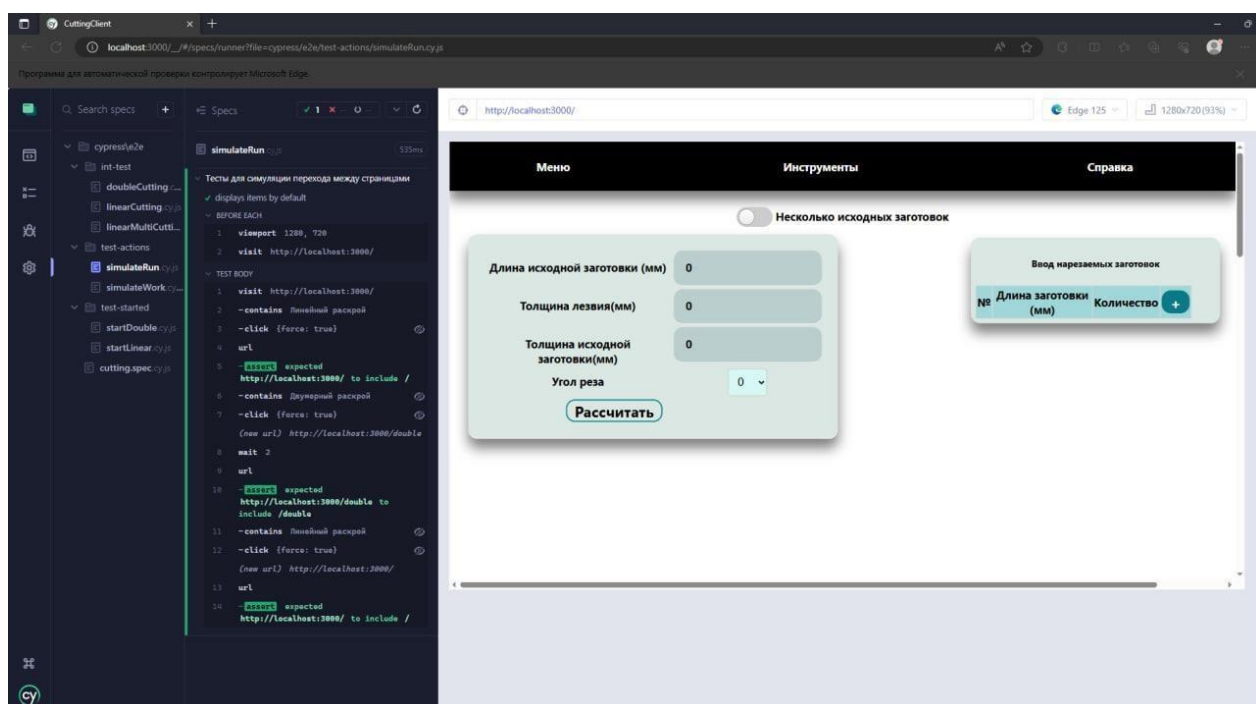


Рисунок Б.4 – Результат выполнение автотеста simulateRun

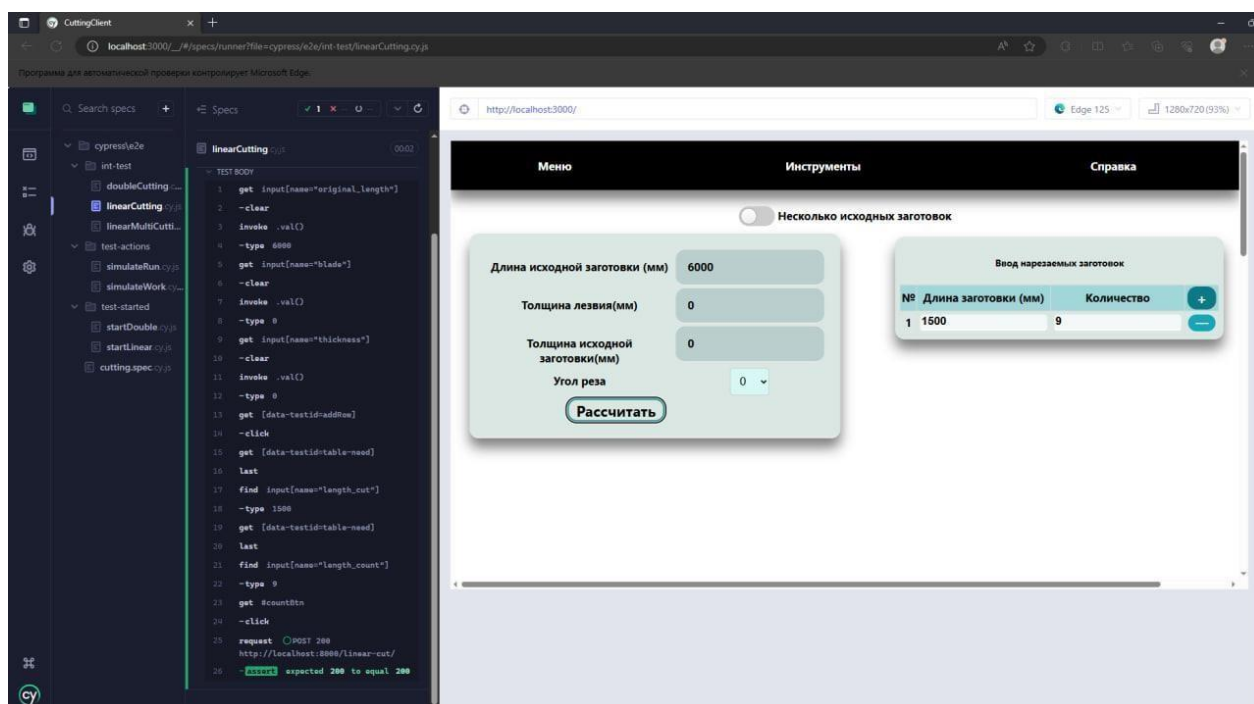


Рисунок Б.5 – Результат выполнение автотеста linearCutting

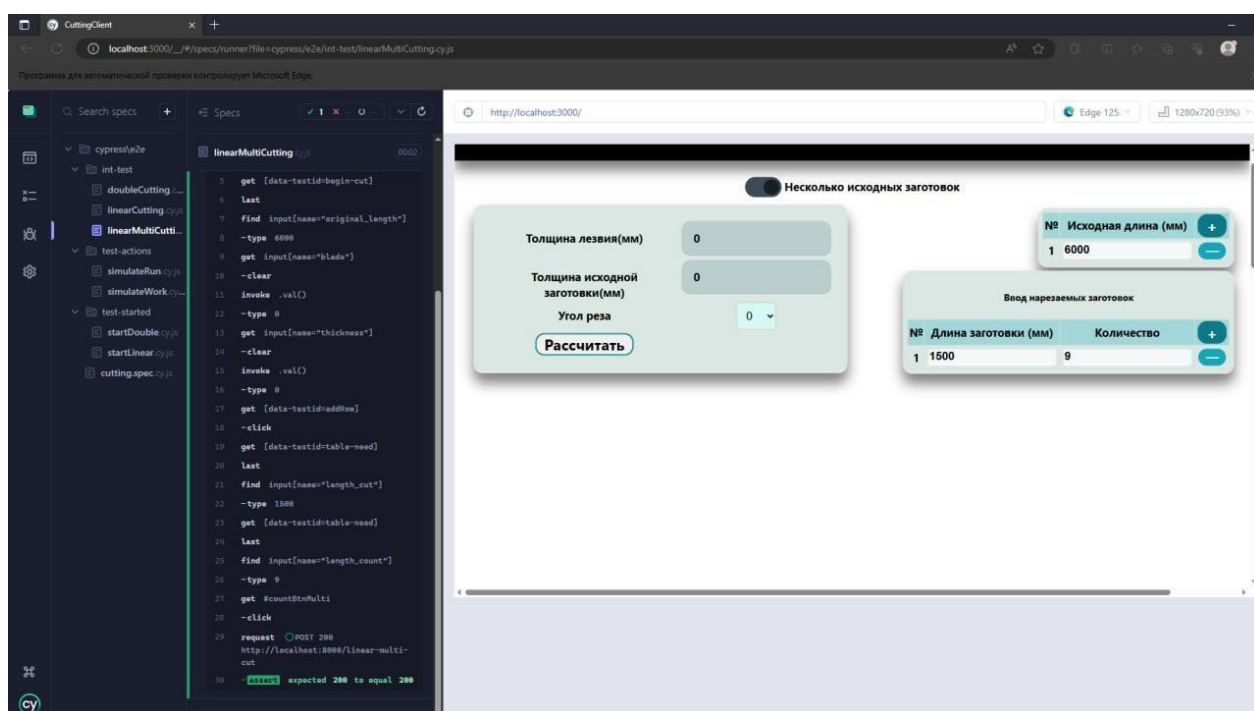


Рисунок Б.6 – Результат выполнение автотеста linearMultiCutting

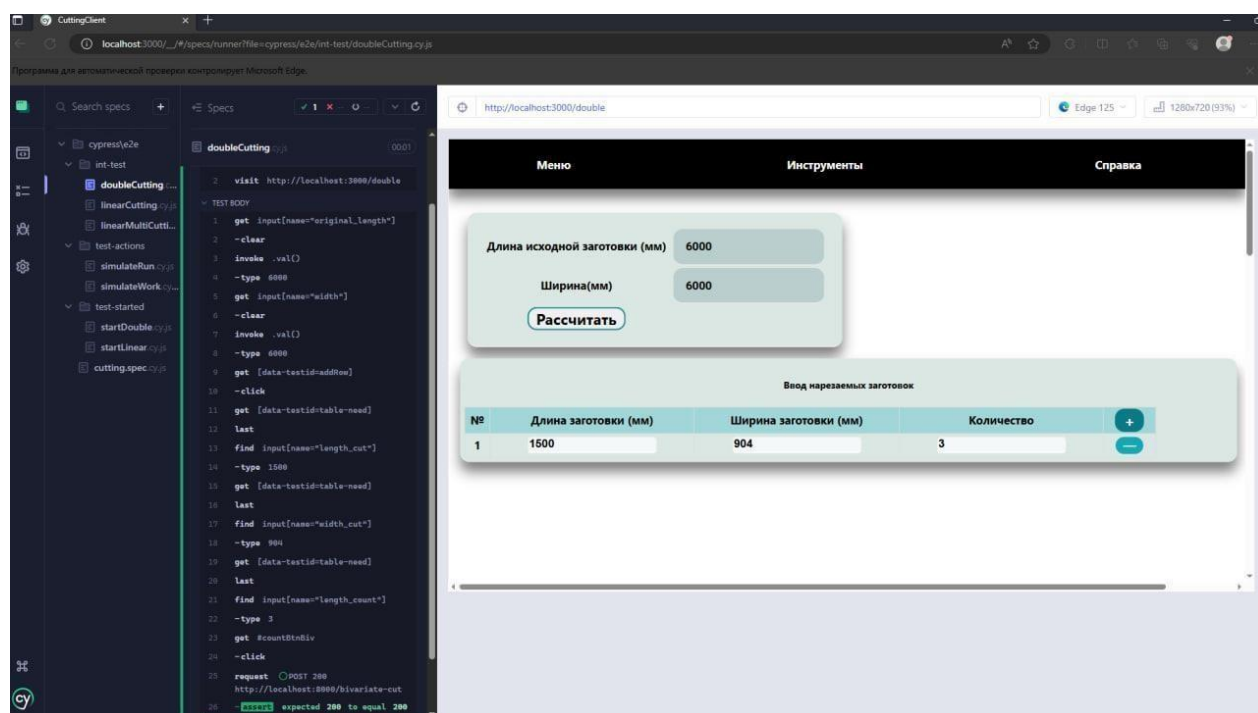


Рисунок Б.7 – Результат выполнение автотеста doubleCutting

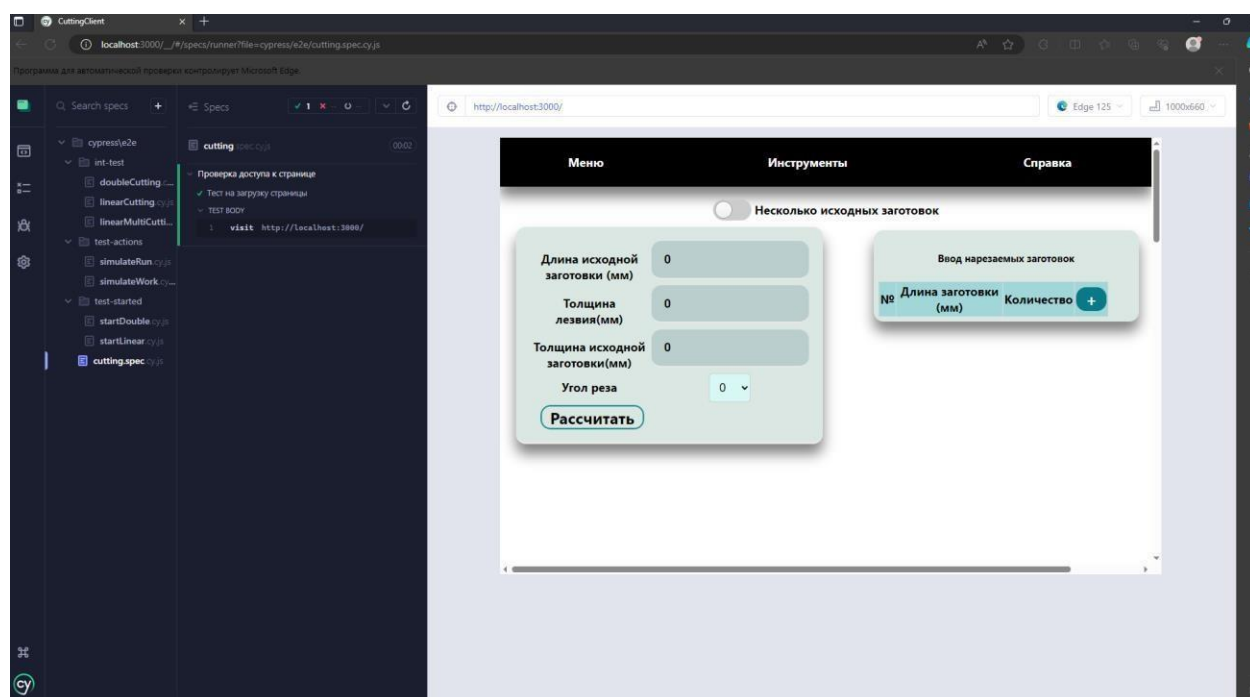


Рисунок Б.8 – Результат выполнение автотеста cutting.spec