ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ РЕСПУБЛИКИ ХАКАСИЯ

«ХАКАССКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

**ОТЧЕТ**

по производственной практике

по профессиональному модулю

**ПМ.01. РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ**

специальности 09.02.07 Информационные системы и программирование   
Квалификация Программист

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент гр. | ИС(ПРО)-41 |  |  |  | Моисеев Д.Н. |
|  |  |  | *подпись* |  | *Фамилия И.О.* |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Руководитель практики  от предприятия |  |  |  |  |  |  | Макарьев Д.Д. |
|  | *оценка* |  | *дата* |  | *подпись* |  | *Фамилия И.О.* |
| М.П. |  |  |  |  |  |  |  |
| Руководитель практики  от ГБПОУ РХ ХПК |  |  |  |  |  |  | Черкашин Д.С. |
|  | *оценка* |  | *дата* |  | *подпись* |  | *Фамилия И.О.* |

Абакан 2025 г.

Исследование информационной системы предприятия

ИП Макарьев Данил Дмитриевич — молодое предприятие, располагающееся в г. Минусинск. Основная деятельность — производство медовых напитков брожения (медовуха) на основе натурального хакасского меда. Продукция реализуется через местные рынки, ярмарки и прямые заказы от клиентов. Объем производства на начальном этапе составляет около 1 000 литров в квартал.

Производственный процесс включает несколько этапов. Сначала осуществляется закупка сырья: натуральный мед приобретается у местных пчеловодов, а дополнительные ингредиенты — вода, дрожжи и специи закупаются на оптовых базах. Затем следует смешивание ингредиентов по заранее разработанным рецептурам, которые определяют пропорции. Процесс брожения длится от 2 до 3 недель и проходит в пластиковых емкостях при комнатной температуре. После завершения брожения напиток фильтруется вручную через марлю и разливается в стеклянные бутылки объемом 0,5 литра. В штате предприятия работают три человека: сам предприниматель, который выполняет функции руководителя и отвечает за закупки и продажи, технолог, контролирующий качество и рецептуры, и помощник, занимающийся розливом и доставкой продукции на рынок.

На данный момент учет медовых напитков брожения ведется вручную с использованием минимального набора инструментов. Для закупки сырья данные о поставках (объем меда, стоимость, дата) записываются в таблицу Microsoft Excel, которая ведется самим предпринимателем. В производственном процессе рецептуры фиксируются в документах Microsoft Word, где указаны ингредиенты и их пропорции, а сроки брожения отмечаются в обычном бумажном блокноте. Реализация продукции также сопровождается ручным учетом: заказы клиентов фиксируются либо в отдельной таблице Excel (с указанием имени клиента, объема и даты), либо в мессенджерах, что усложняет их систематизацию.

Отсутствие специализированного программного обеспечения приводит к ряду проблем. Ручной ввод данных в Excel и Word занимает значительное время, особенно при увеличении числа заказов или партий продукции. Ошибки при записи объемов сырья или сроков брожения могут привести к сбоям в производственном цикле, например, к преждевременному завершению брожения или нехватке ингредиентов. Кроме того, отсутствие единой системы учета затрудняет анализ остатков сырья и планирование закупок, что снижает эффективность работы предприятия.

Анализ предметной области

Предметная область данного исследования охватывает процессы производства и реализации медовых напитков брожения на предприятии ИП Макарьев Данил Дмитриевич, расположенном в г. Минусинск. Основной продукт — медовуха, изготавливаемая из натурального хакасского меда с добавлением воды, дрожжей и специй. Производственный цикл включает закупку сырья, подготовку рецептур, брожение, фильтрацию, розлив и продажу готовой продукции через рынки, ярмарки и прямые заказы. На текущий момент все операции учета выполняются вручную с использованием Microsoft Excel, Word и бумажных записей, что создает сложности в управлении процессами.

Ключевые задачи, требующие автоматизации, связаны с учетом медовых напитков брожения на разных этапах:

Учет сырья. Закупка меда и других ингредиентов фиксируется в таблицах Excel, но данные не систематизированы. Отсутствует возможность быстро проверить остатки или спрогнозировать потребности для следующей партии, что приводит к рискам нехватки материалов.

Контроль этапов брожения. Сроки брожения (2–3 недели) записываются в блокноте, и их отслеживание зависит от внимательности персонала. Это может приводить к ошибкам, например, к преждевременной фильтрации или задержке производства.

Учет заказов. Заказы клиентов фиксируются в Excel или мессенджерах без единой базы, что затрудняет их обработку и анализ спроса. При увеличении числа заказов ручной учет становится неэффективным.

Разработка программного модуля направлена на решение этих проблем. Приложение должно обеспечивать:

* Ведение базы данных сырья с указанием объемов, даты закупки и остатков.
* Автоматический расчет сроков брожения для каждой партии с уведомлениями о готовности.
* Хранение информации о заказах (имя клиента, объем, дата) с возможностью быстрого доступа и анализа.

Автоматизация позволит сократить время на рутинные операции, минимизировать ошибки и повысить контроль над производственным процессом. Например, при вводе данных о закупке 50 кг меда приложение сможет рассчитать, на сколько партий этого хватит (при норме 1 кг на 5 литров медовухи — на 250 литров), а также уведомить о необходимости новой закупки при достижении минимального остатка. Аналогично, учет заказов упростит планирование производства и доставки.

Проектирование программного модуля

Проектирование диаграмм

Диаграмма вариантов использования описывает взаимодействие пользователей с приложением. Она отражает основные функции системы и роли сотрудников.

Актёры:

* Предприниматель — отвечает за производственные процессы (закупки, создание партий, учет готовой продукции, управление заказами и клиентами), а также формирование отчетов. Обычно это сам Макарьев Д.Д.
* Администратор — управляет учетными записями пользователей, добавляя или редактируя их данные. Может совпадать с Предпринимателем.
* Технолог — занимается производством, включая учет сырья, создание рецептов и регистрацию партий.
* Помощник — работает с заказами и клиентами, а также отслеживает готовую продукцию.

Варианты использования:

* Вход в систему — общий для всех актеров, обеспечивает аутентификацию через логин и пароль.
* Добавить закупку сырья — регистрация закупленного сырья (например, меда или дрожжей).
* Просмотреть остатки сырья — отображение текущих запасов на складе.
* Добавить рецепт — создание нового рецепта с указанием названия и состава.
* Создать партию — регистрация новой партии с выбором рецепта и объема.
* Просмотреть статус партий — проверка состояния партий (например, "в брожении").
* Просмотреть готовую продукцию — учет доступных напитков после розлива.
* Добавить клиента — внесение данных о физическом или юридическом лице.
* Добавить заказ — оформление заказа с выбором готовой продукции.
* Просмотреть заказы — просмотр списка заказов с их статусами.
* Добавить пользователя — создание новой учетной записи для сотрудника.
* Сформировать отчет — генерация сводки по производству, запасам или заказам.
* Диаграмма демонстрирует разделение ролей: Предприниматель имеет доступ ко всем функциям, кроме управления пользователями, которое закреплено за Администратором; Технолог фокусируется на производстве, а Помощник — на заказах. Это обеспечивает гибкость и контроль доступа.

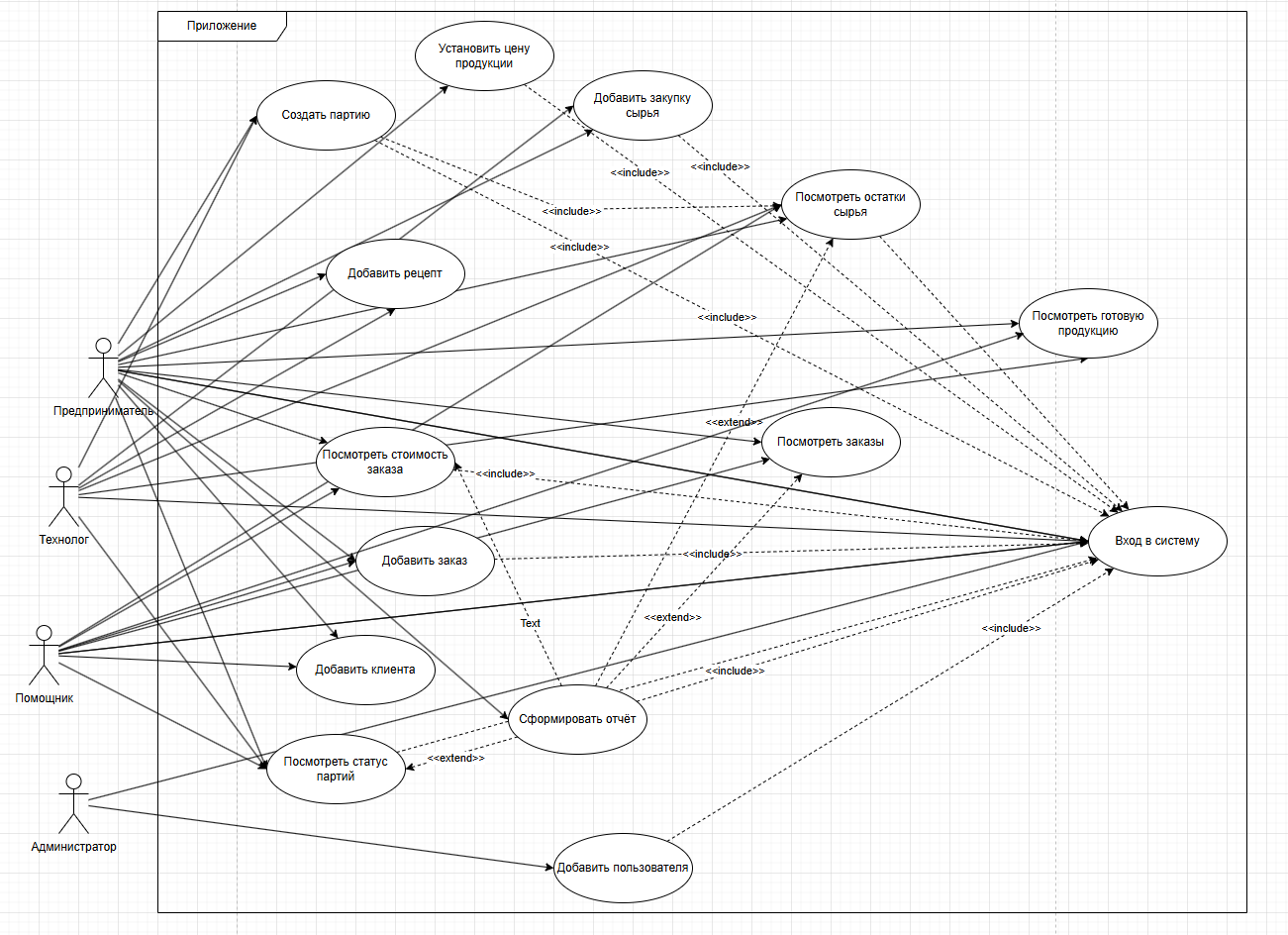


Рисунок 1 – Use Case диаграмма

Реляционная база данных представляет собой организованный набор данных с заранее определенными взаимосвязями. В реляционной модели информация структурируется в таблицы, где каждая строка соответствует отдельной записи, а каждый столбец содержит атрибуты с конкретными значениями. Такой табличный подход упрощает установление связей между данными и обеспечивает гибкий доступ к информации без необходимости изменения структуры базы. Это делает реляционные базы данных эффективным инструментом для хранения и обработки данных в системах управления.

Схема «сущность-связь» (ER-диаграмма) — это разновидность блок-схемы, которая иллюстрирует взаимосвязи между различными «сущностями» (такими как люди, объекты или процессы) внутри системы. ER-диаграммы широко применяются для проектирования и анализа реляционных баз данных в области разработки программного обеспечения, информационных систем и производственных процессов. Они позволяют визуализировать структуру данных, определить ключевые объекты и их взаимодействия, что особенно важно на этапе создания приложений для автоматизации бизнес-процессов.

* Пользователи (Users) — хранит данные сотрудников, участвующих в производстве и управлении.
* Сырье (RawMaterials) — фиксирует информацию о закупленных материалах, таких как мед или специи.
* Рецепты (Recipes) — содержит рецептуры напитков с названиями и составами.
* Партии (Batches) — регистрирует производственные партии с указанием объема и статуса.
* Связь сырья и партий (BatchMaterials) — отражает расход сырья на каждую партию.
* Готовая продукция (FinishedProducts) — учитывает произведенные напитки после розлива.
* Клиенты (Clients) — хранит данные о покупателях, включая тип (физлицо или юрлицо).
* Заказы (Orders) — управляет заказами клиентов с датой и статусом.
* Элементы заказа (OrderItems) — детализирует позиции в каждом заказе, связывая их с готовой продукцией.

Эти таблицы связаны между собой: пользователи создают партии и заказы, рецепты применяются к партиям, сырье расходуется через промежуточную таблицу, партии превращаются в готовую продукцию, а клиенты размещают заказы, которые содержат несколько позиций. Такая структура обеспечивает полный учет производственного цикла и продаж.

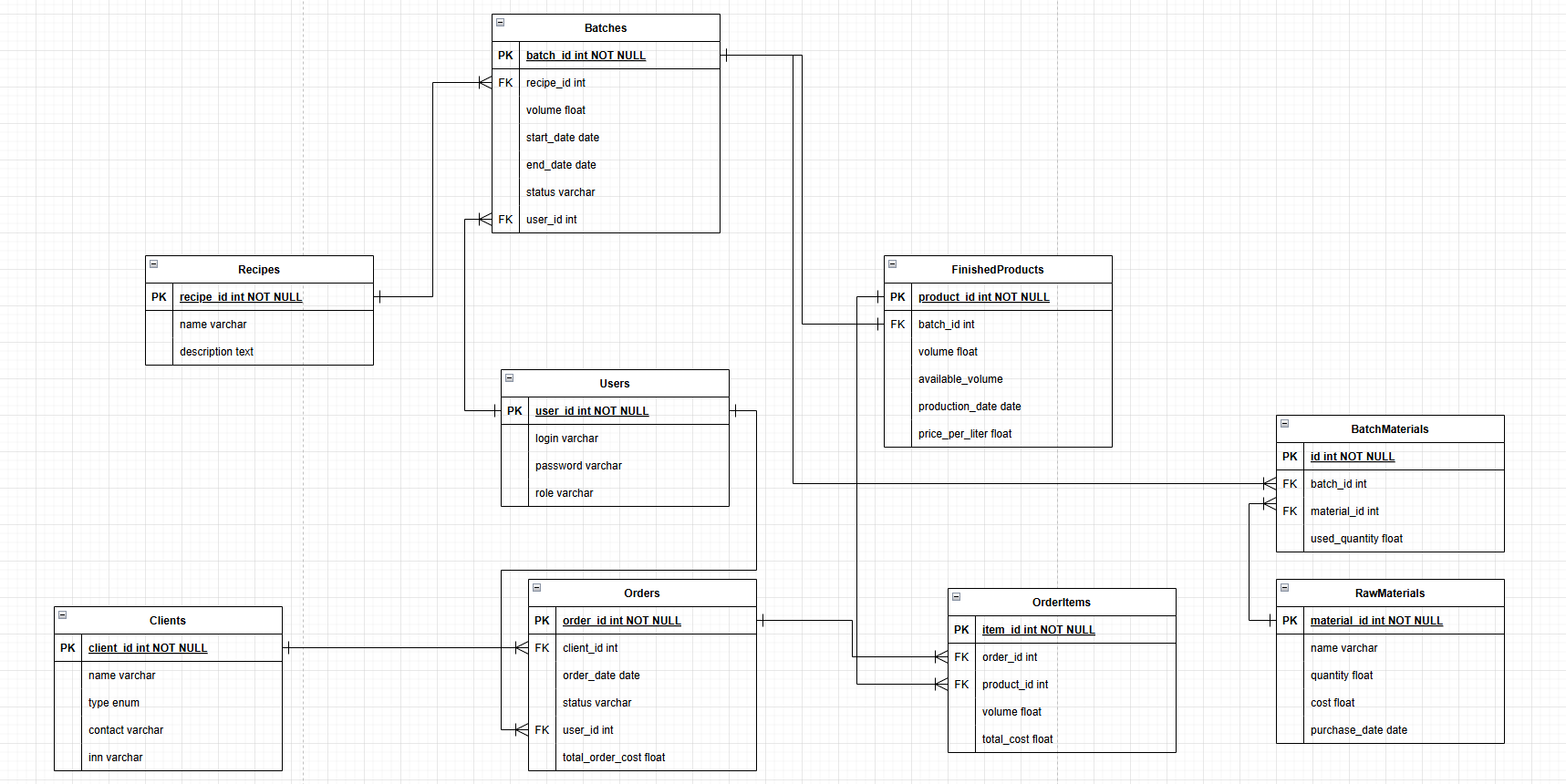


Рисунок 2- ER-диаграмма

Диаграмма классов представляет собой структурированное описание программных объектов, используемых в приложении. Она объединяет классы — программные сущности, содержащие атрибуты (данные) и методы (действия), в единую модель, где каждый класс отражает отдельный компонент системы. Такой подход позволяет определить, как объекты взаимодействуют друг с другом, и обеспечивает основу для реализации кода, упрощая разработку и поддержку приложения.

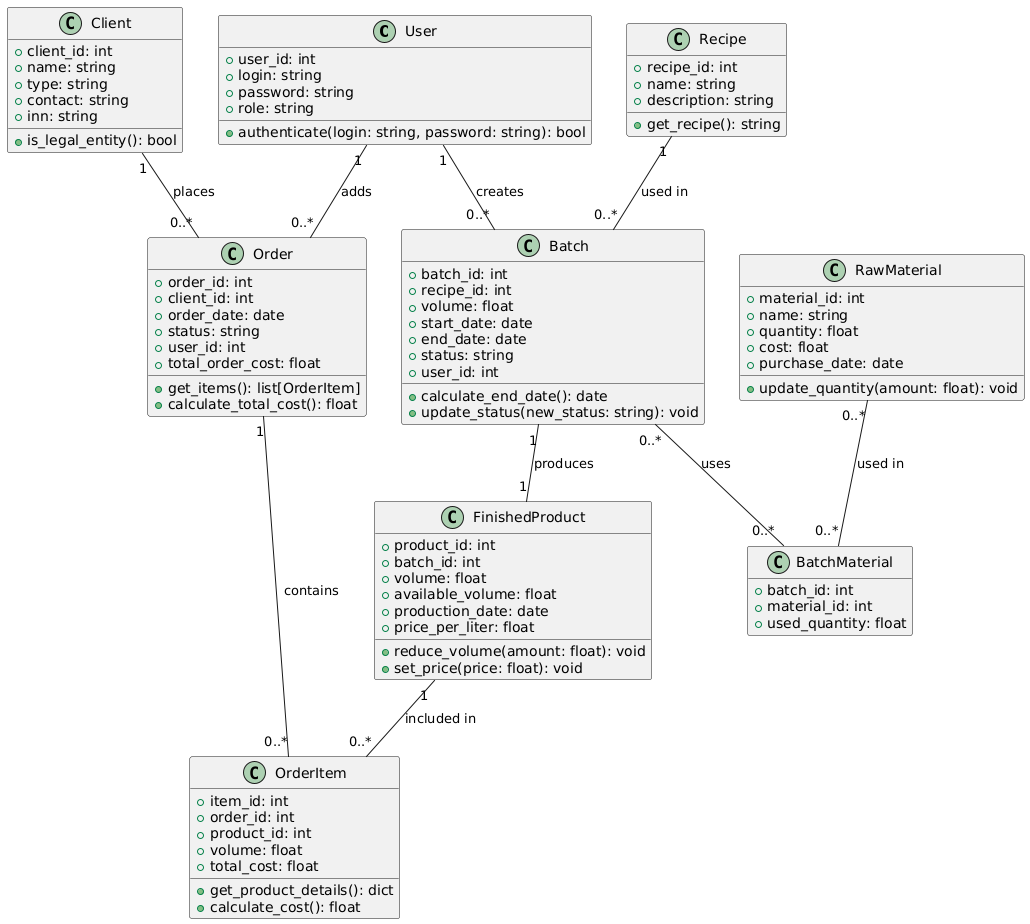


Рисунок 3 – Диаграмма классов

Диаграмма классов (Class Diagram) — это инструмент объектно-ориентированного проектирования, который показывает статическую структуру системы, включая классы, их свойства и взаимосвязи. Она применяется для проектирования программного обеспечения, помогая разработчикам визуализировать архитектуру приложения, определить ключевые элементы и их зависимости. В контексте разработки приложения для учета медовых напитков брожения диаграмма классов связывает функциональные требования с реальной программной реализацией на языке Python.

Разработка макетов интерфейса программного модуля

Дизайн страниц приложения разработан для использования на настольных компьютерах, обеспечивая удобство и функциональность для пользователей. Интерфейс спроектирован с акцентом на интуитивность и эффективность работы в desktop-среде.

Цветовая палитра построена на минималистичных и практичных оттенках. Основной фон выполнен в светло-сером цвете, создавая спокойную и ненавязчивую основу, комфортную для длительного взаимодействия. Текст оформлен в темно-сером оттенке, что гарантирует высокую читаемость и четкость на экране. Кнопки выделены светло-голубым фоном, добавляя визуальный акцент и обеспечивая интерактивность без излишней яркости. Поля ввода имеют белый фон с темно-серым текстом, что подчеркивает их функциональность и улучшает контраст для удобства заполнения. Такое сочетание цветов придает интерфейсу чистоту, объем и профессиональный вид, поддерживая фокус на содержимом приложения.

Окно входа, представленное на рисунке 4, представляет собой компактный диалог с минималистичным дизайном.

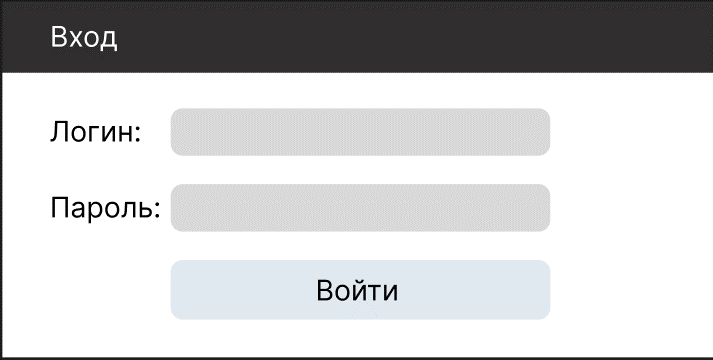


Рисунок 4 – Макет окна входа

Главное окно приложения представляет собой рабочую область с вкладочной навигацией. В верхней части расположена панель вкладок. В нижней части окна предусмотрены функциональные кнопки, а также область для уведомлений.

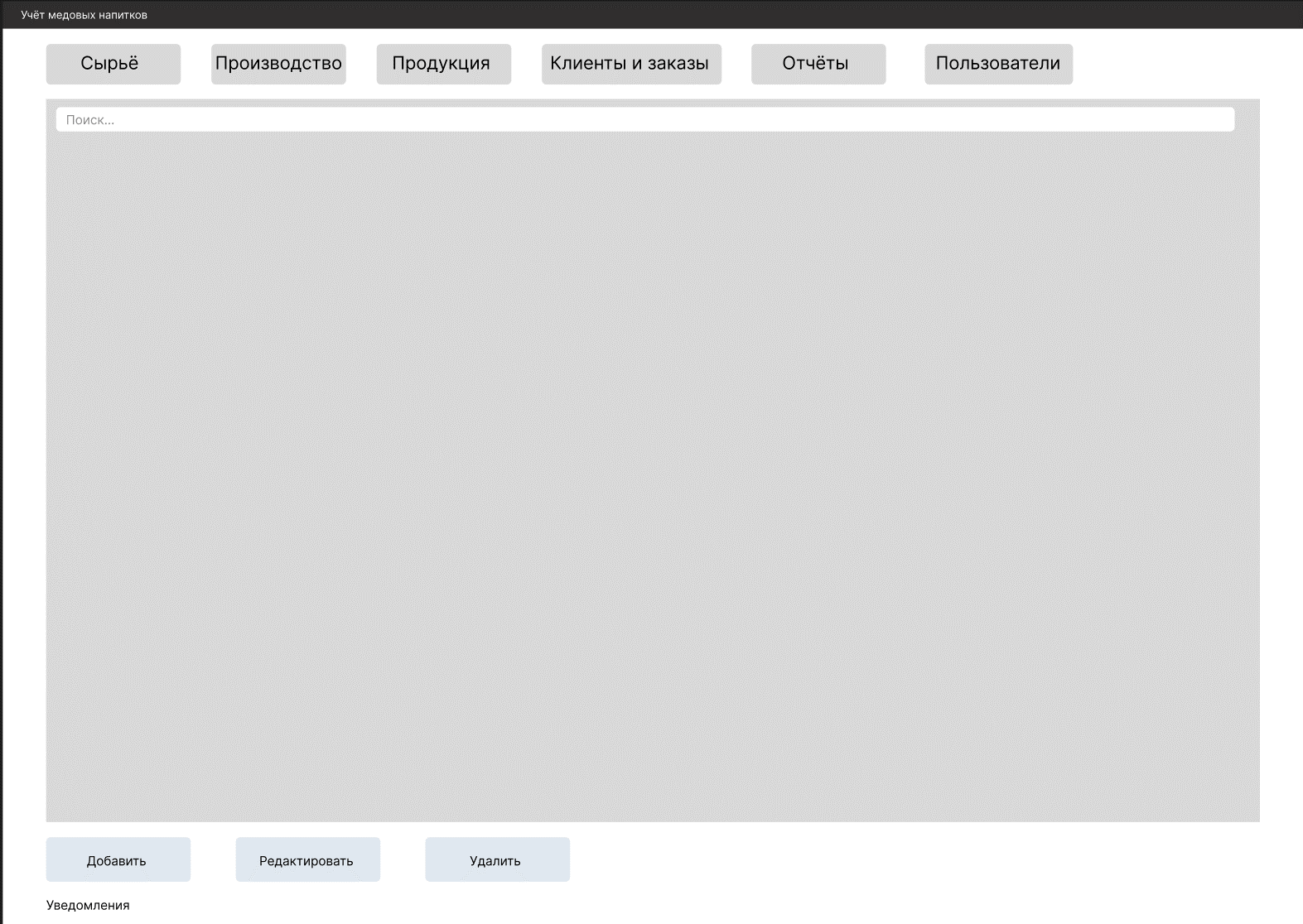


Рисунок 5 – Макет главного окна

Разработка программного модуля

Разработка интерфейсов программного модуля

В рамках проекта были реализованы ключевые формы приложения, обеспечивающие удобное взаимодействие пользователя с системой учета медовых напитков. Каждая форма разработана с учетом функциональности и минималистичного дизайна, что позволяет сосредоточиться на выполнении задач.

QSS – это механизм, который позволяет задавать стили для виджетов, аналогично CSS. Он используется для настройки внешнего вида интерфейса.

Листинг 1 – Таблица стилей для приложения

APP\_STYLE = """

    QWidget {

        background-color: #f5f5f5;

        color: #333333;

        font-family: Arial;

        font-size: 12px;

    }

    QPushButton {

        background-color: #e0e8f0;

        color: #333333;

        border: 1px solid #b0b0b0;

        padding: 6px 12px;

        border-radius: 3px;

    }

    QPushButton:hover {

        background-color: #d0d8e0;

    }

    QPushButton:pressed {

        background-color: #c0c8d0;

    }

    QLineEdit, QDoubleSpinBox, QDateEdit, QComboBox {

        background-color: #ffffff;

        color: #333333;

        border: 1px solid #b0b0b0;

        padding: 4px;

        border-radius: 2px;

    }

    QTableWidget {

        background-color: #ffffff;

        border: 1px solid #b0b0b0;

    }

    QTableWidget::item {

        background-color: #ffffff;

    }

    QTableWidget::item:nth-child(even) {

        background-color: #f0f0f0;

    }

Продолжение листинга 1.

    QTableWidget::item:selected {

        background-color: #d0e0f0;

        color: #333333;

    }

    QHeaderView::section {

        background-color: #e0e0e0;

        color: #333333;

        padding: 4px;

        border: 1px solid #b0b0b0;

    }

    QTabWidget::pane {

        border: 1px solid #b0b0b0;

    }

    QTabBar::tab {

        background-color: #e8e8e8;

        color: #333333;

        padding: 6px 12px;

        border: 1px solid #b0b0b0;

        border-bottom: none;

    }

    QTabBar::tab:selected {

        background-color: #ffffff;

        color: #333333;

        border-bottom: 2px solid #4682b4;

    }

    QLabel#notificationLabel {

        color: #008000;

        font-weight: bold;

    }

"""

Форма входа, представленная на рисунке 6, предназначена для авторизации пользователей. Она представляет собой компактное окно с двумя полями ввода: "Логин" и "Пароль". Поля выполнены с белым фоном и темно-серым текстом, обрамлены тонкой серой границей, что обеспечивает высокую читаемость и визуальную четкость. Кнопка "Войти" выделена светло-голубым фоном, который при наведении слегка темнеет, а при нажатии становится еще темнее (#c0c8d0), предоставляя пользователю обратную связь. Фон формы — светло-серый, что поддерживает общий стиль приложения и создает чистый, профессиональный вид.

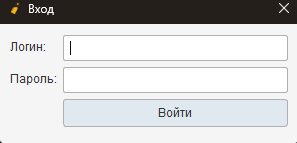


Рисунок 6 – Окно входа

Листинг 2 – Форма входа пользователя

class LoginWindow(QDialog):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Учет медовых напитков")

        self.setFixedSize(300, 120)

        layout = QFormLayout()

        self.login\_input = QLineEdit(self)

        self.login\_input.setMaxLength(50)

        self.password\_input = QLineEdit(self)

        self.password\_input.setMaxLength(255)

        self.password\_input.setEchoMode(QLineEdit.Password)

        self.login\_button = QPushButton("Войти", self)

        layout.addRow("Логин:", self.login\_input)

        layout.addRow("Пароль:", self.password\_input)

        layout.addWidget(self.login\_button)

        self.login\_button.clicked.connect(self.validate\_and\_accept)

        self.setLayout(layout)

        self.setStyleSheet(APP\_STYLE)

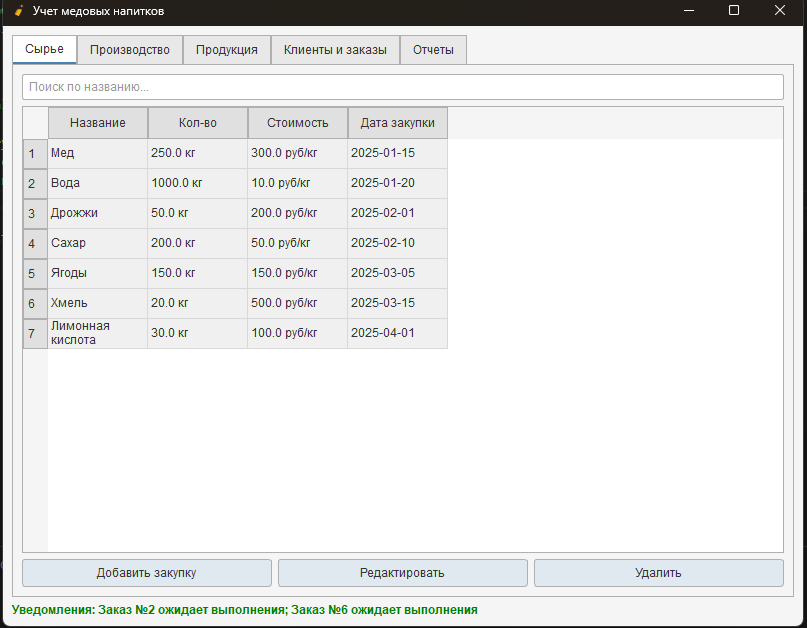


Рисунок 7 – Главное окно

Листинг 3 – Форма главного окна

class MainWindow(QMainWindow):

    def \_\_init\_\_(self, role="Технолог", user\_id=None):

        super().\_\_init\_\_()

        self.role = UserRole(role)

        self.user\_id = user\_id

        self.session = Session()

        self.setWindowTitle("Учет медовых напитков")

        self.setGeometry(100, 100, 800, 600)

        central\_widget = QWidget()

        self.setCentralWidget(central\_widget)

        layout = QVBoxLayout(central\_widget)

        self.tabs = QTabWidget()

        layout.addWidget(self.tabs)

        if self.role == UserRole.TECHNOLOGIST:

            self.init\_raw\_materials()

            self.init\_production()

        elif self.role == UserRole.ASSISTANT:

            self.init\_products()

            self.init\_orders()

        elif self.role == UserRole.ENTREPRENEUR:

            self.init\_raw\_materials()

            self.init\_production()

            self.init\_products()

            self.init\_orders()

            self.init\_reports()

        elif self.role == UserRole.ADMIN:

            self.init\_users()

        self.notification\_label = QLabel("Уведомления: проверка...")

        self.notification\_label.setObjectName("notificationLabel")

        layout.addWidget(self.notification\_label)

        self.update\_notifications()

        self.setStyleSheet(APP\_STYLE)

Разработка кода программного модуля

Создание партии – важный процесс, который необходимо учитывать. Метод create\_batch() создаёт партию, проверяет наличие необходимого сырья, списывает его и сохраняет данные в базу.

Листинг 4 – Метод для создания партии

def create\_batch(self):

recipe\_name = self.recipe\_combo.currentText()

recipe = self.session.query(Recipe).filter\_by(name=recipe\_name).first()

volume = self.volume\_spin.value()

Продолжение листинга 4

price = self.price\_spin.value()

if not recipe or volume <= 0 or price < 0:

QMessageBox.warning(self, "Ошибка", "Выберите рецепт, объем > 0, цена >= 0")

return

honey = self.session.query(RawMaterial).filter\_by(name="Мед").first()

if not honey or honey.quantity < volume:

QMessageBox.warning(self, "Ошибка", f"Недостаточно меда (нужно {volume} кг, есть {honey.quantity if honey else 0} кг)")

return

batch = Batch(

recipe\_id=recipe.recipe\_id,

volume=volume,

start\_date=self.start\_date.date().toString("yyyy-MM-dd"),

end\_date=self.start\_date.date().addDays(14).toString("yyyy-MM-dd"),

status=BatchStatus.FERMENTING.value,

user\_id=self.current\_user\_id,

price\_per\_liter=price

)

self.session.add(batch)

self.session.commit()

honey.quantity -= volume

self.session.commit()

self.accept()

Метод update\_batch\_status() управляет жизненным циклом партии и создаёт запись о готовой продукции, когда партия достигает статуса «Готово».

Листинг 6 – Обновление статуса партии

def update\_batch\_status(self, batch\_id, status):

batch = self.session.query(Batch).filter\_by(batch\_id=batch\_id).first()

if batch.status != status:

batch.status = status

if status == BatchStatus.READY.value:

existing\_product = self.session.query(FinishedProduct).filter\_by(batch\_id=batch.batch\_id).first()

if not existing\_product:

finished\_product = FinishedProduct(

batch\_id=batch.batch\_id,

volume=batch.volume,

available\_volume=batch.volume,

production\_date=batch.end\_date,

price\_per\_liter=batch.price\_per\_liter

)

Продолжение листинга 6

self.session.add(finished\_product)

self.session.commit()

self.update\_product\_table()

self.update\_notifications()

Метод generate\_report() создаёт PDF-отчёты, которые разбиваются на страницы при переполнении. Может содержать различные данные.

Листинг 7 – Генерация отчёта

def generate\_report(self):

report\_type = self.report\_type.currentText()

start\_date = self.start\_date.date().toString("yyyy-MM-dd")

end\_date = self.end\_date.date().toString("yyyy-MM-dd")

font\_path = "DejaVuSans.ttf"

if not os.path.exists(font\_path):

QMessageBox.critical(self, "Ошибка", "Шрифт DejaVuSans.ttf не найден.")

return

default\_filename = f"report\_{report\_type.lower().replace(' ', '\_')}\_{start\_date}\_to\_{end\_date}.pdf"

output\_file, \_ = QFileDialog.getSaveFileName(self, "Сохранить отчет как", default\_filename, "PDF Files (\*.pdf)")

if not output\_file:

return

pdfmetrics.registerFont(TTFont("DejaVuSans", font\_path))

c = canvas.Canvas(output\_file, pagesize=A4)

c.setFont("DejaVuSans", 12)

width, height = A4

c.setFont("DejaVuSans", 16)

c.drawCentredString(width / 2, height - 40, f"Отчет: {report\_type}")

c.setFont("DejaVuSans", 12)

c.drawCentredString(width / 2, height - 60, f"Период: {start\_date} - {end\_date}")

y = height - 100

line\_height = 20

if report\_type == "Остатки сырья":

c.drawString(50, y, "Название Кол-во Стоимость Дата закупки")

y -= line\_height

c.line(50, y, width - 50, y)

y -= 10

materials = self.session.query(RawMaterial).filter(RawMaterial.purchase\_date.between(start\_date, end\_date)).all()

Продолжение листинга 7

for m in materials:

c.drawString(50, y, f"{m.name}")

c.drawString(150, y, f"{m.quantity} кг")

c.drawString(250, y, f"{m.cost} руб/кг")

c.drawString(350, y, f"{m.purchase\_date}")

y -= line\_height

if y < 50:

c.showPage()

c.setFont("DejaVuSans", 12)

y = height - 50

# ... (остальная логика для других типов отчетов аналогична) ...

c.showPage()

c.save()

QMessageBox.information(self, "Успех", f"Отчет сохранен как {output\_file}")

Метод create\_order() управляет созданием заказа с несколькими позициями, проверяет наличие продукции и списывает её.

Листинг 8 – Создание заказа

def create\_order(self):

client\_name = self.client\_combo.currentText()

client = self.session.query(Client).filter\_by(name=client\_name).first()

if not client:

QMessageBox.warning(self, "Ошибка", "Выберите клиента")

return

for row in range(self.items\_table.rowCount()):

product\_text = self.items\_table.cellWidget(row, 0).currentText()

product\_id = int(product\_text.split(" - ")[0].replace("Продукт ", ""))

volume = self.items\_table.cellWidget(row, 1).value()

product = self.session.query(FinishedProduct).filter\_by(product\_id=product\_id).first()

if product.available\_volume < volume:

QMessageBox.warning(self, "Ошибка", f"Недостаточно продукции {product\_id} (нужно {volume} л, есть {product.available\_volume} л)")

return

order = Order(

client\_id=client.client\_id,

order\_date=self.date\_edit.date().toString("yyyy-MM-dd"),

status=OrderStatus.PENDING.value,

user\_id=self.current\_user\_id,

total\_order\_cost=0

)

Продолжение листинга 8

self.session.add(order)

self.session.commit()

for row in range(self.items\_table.rowCount()):

product\_text = self.items\_table.cellWidget(row, 0).currentText()

product\_id = int(product\_text.split(" - ")[0].replace("Продукт ", ""))

volume = self.items\_table.cellWidget(row, 1).value()

total\_cost = float(self.items\_table.item(row, 2).text().replace(" руб", ""))

if volume <= 0 or total\_cost < 0:

QMessageBox.warning(self, "Ошибка", "Объем > 0, стоимость >= 0")

return

item = OrderItem(

order\_id=order.order\_id,

product\_id=product\_id,

volume=volume,

total\_cost=total\_cost

)

self.session.add(item)

product = self.session.query(FinishedProduct).filter\_by(product\_id=product\_id).first()

product.available\_volume -= volume

self.session.commit()

order.total\_order\_cost = sum(i.total\_cost for i in order.order\_items)

self.session.commit()

self.accept()

Тестирование программного модуля

Разработка тест-кейсов

Целью тест-кейса, представленного на рисунке 8 является проверка, что пользователь с правильными учетными данными может войти в систему.

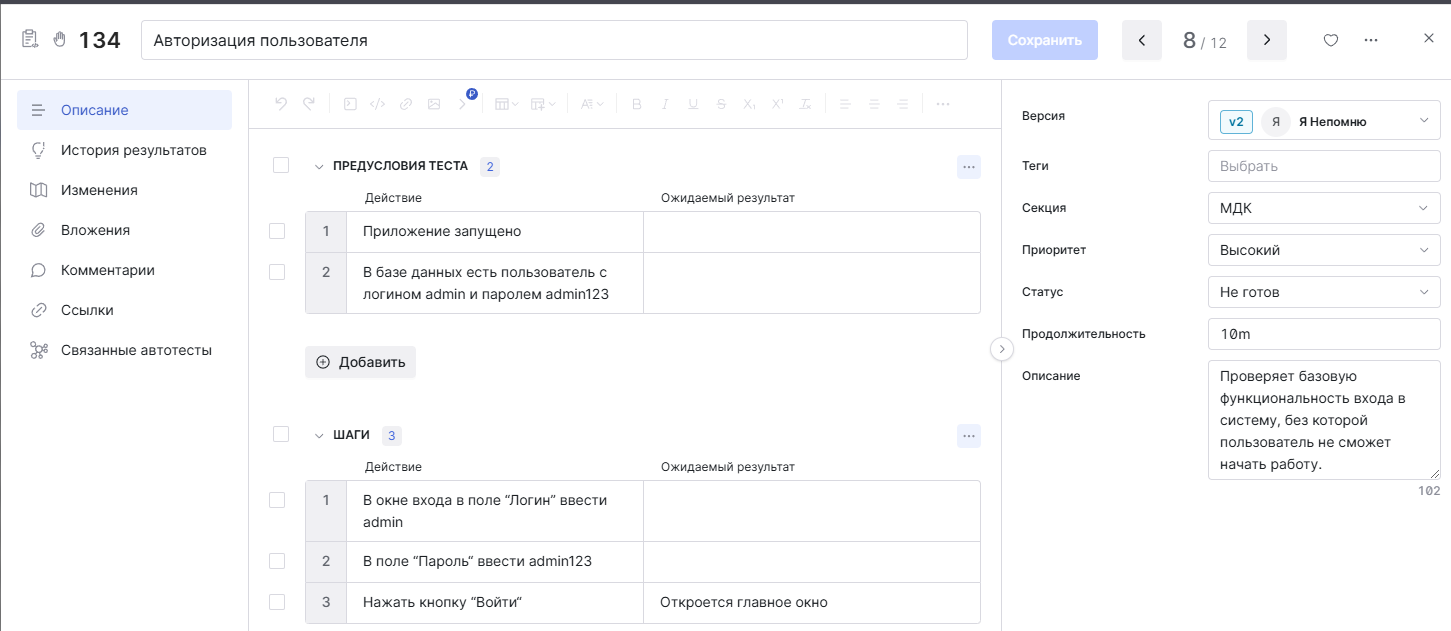


Рисунок 8 – Тест-кейс 1

Целью тест-кейса, представленного на рисунке 9 является проверка, что пользователь может добавить новое сырье в систему.

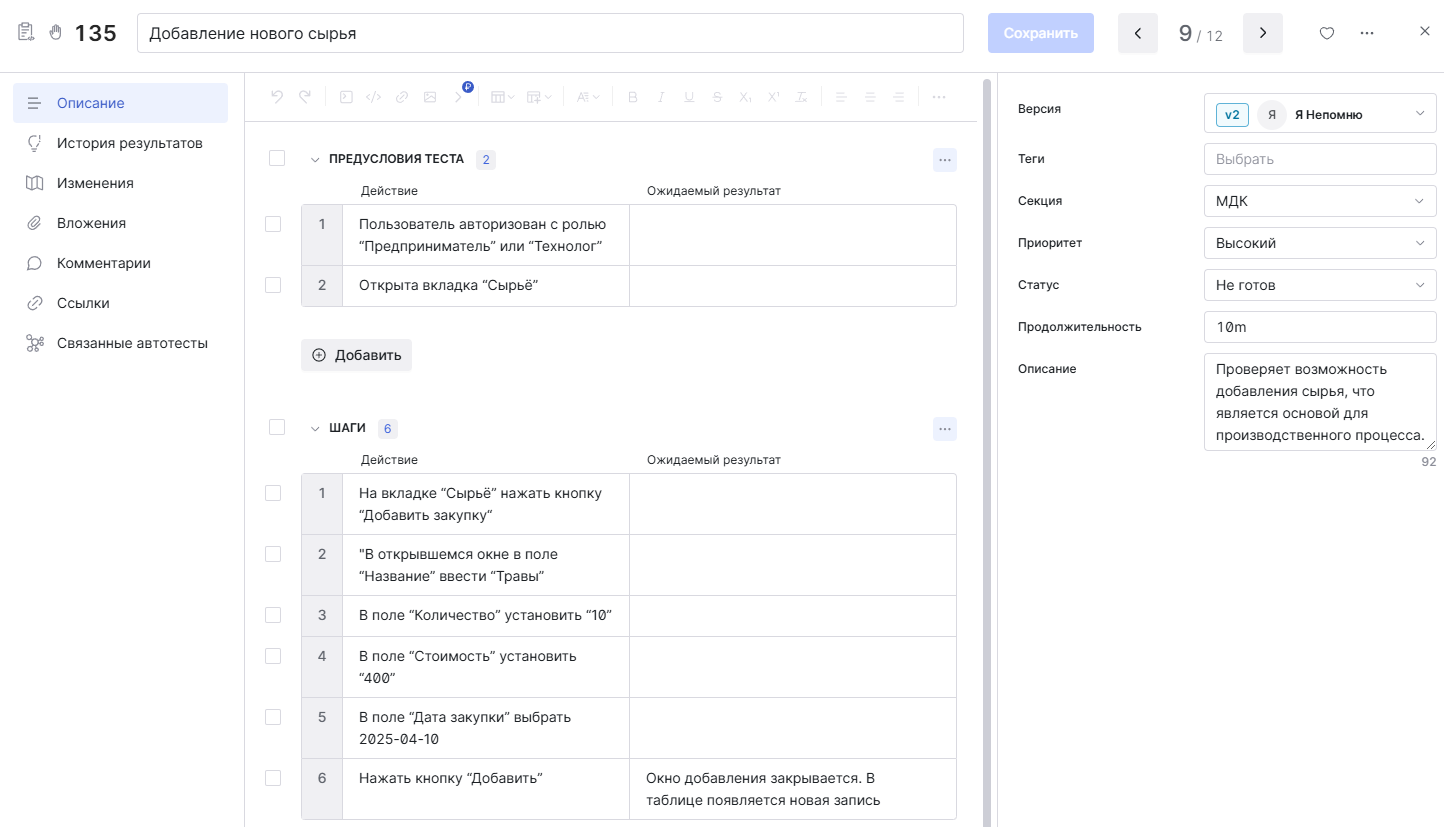


Рисунок 9 – Тест-кейс 2

Целью тест-кейса, представленного на рисунке 10 является проверка, что пользователь может создать заказ, и доступный объем продукции корректно уменьшается.

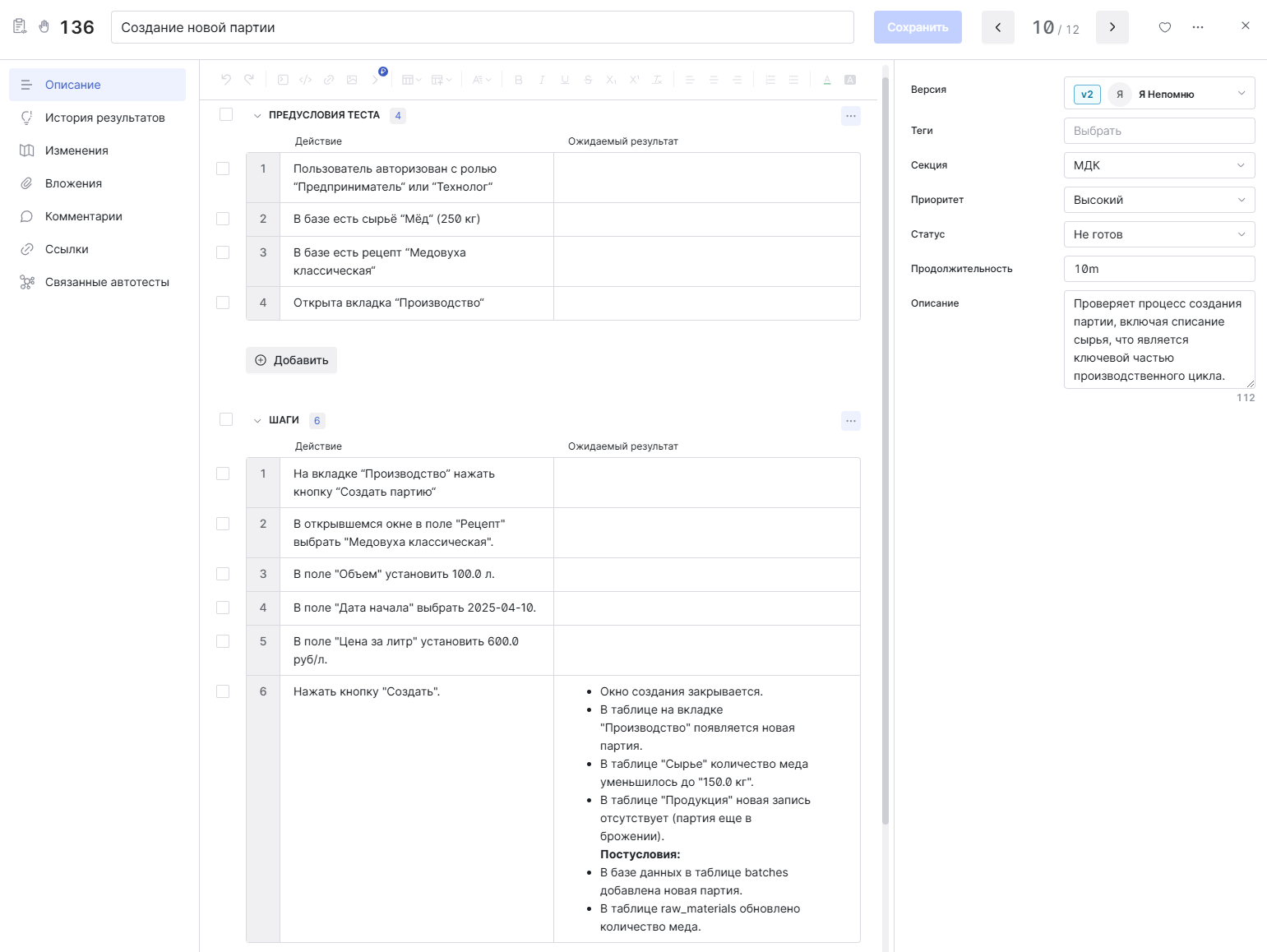


Рисунок 10 – Тест-кейс 3

Целью тест-кейса, представленного на рисунке 11 является проверка, что пользователь может добавить новое сырье в систему.

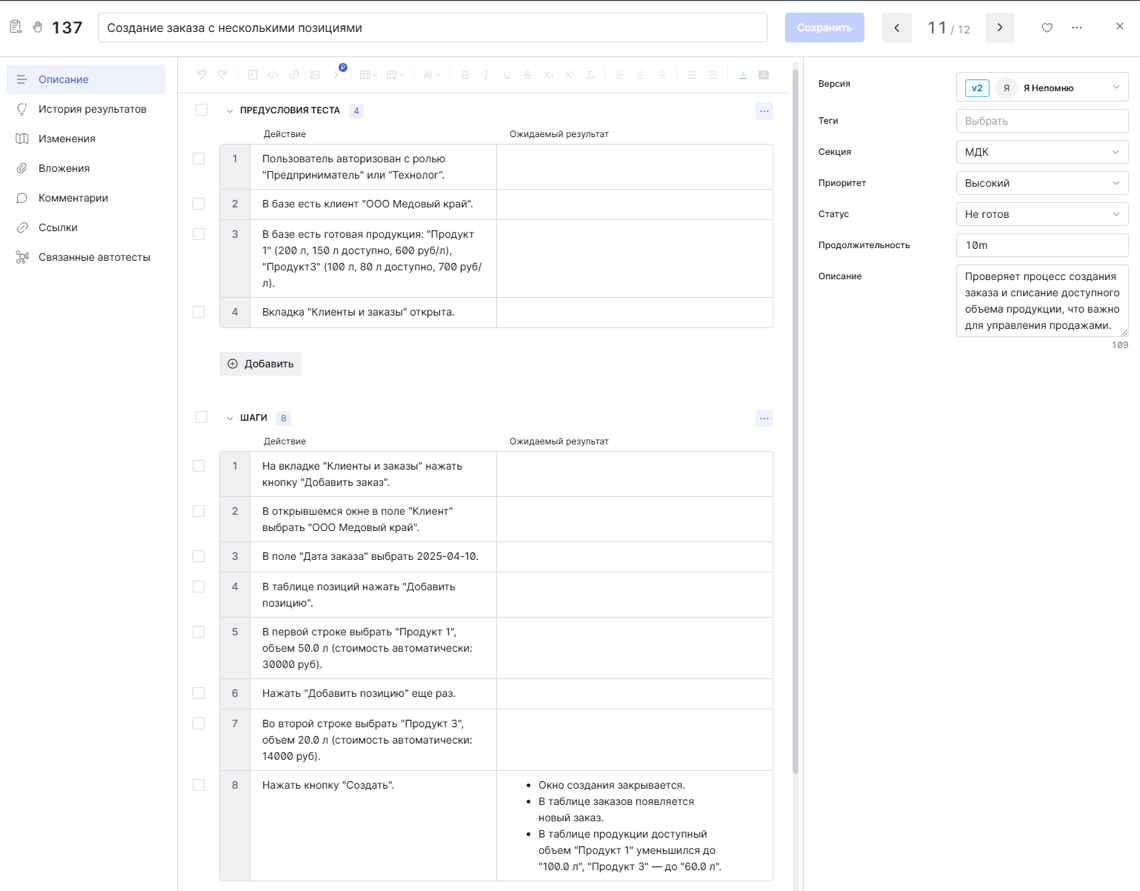


Рисунок 11 – Тест-кейс 4

Целью тест-кейса, представленного на рисунке 12 является проверка, что пользователь может сгенерировать PDF-отчет по остаткам сырья за выбранный период.

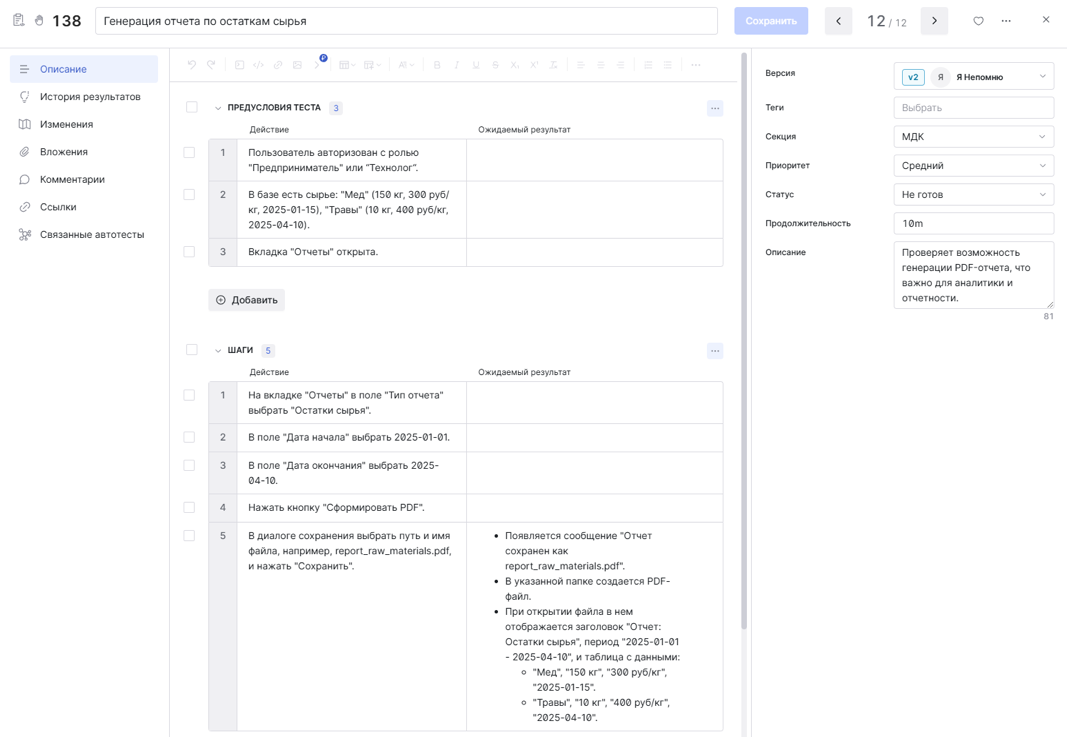


Рисунок 12 – Тест-кейс 5

Модульное тестирование

Модульное тестирование (unit testing) — это процесс проверки отдельных модулей или компонентов приложения на корректность их работы.

Pytest — один из самых популярных фреймворков для тестирования в Python. Он проще в использовании, чем unittest, благодаря более лаконичному синтаксису, автоматическому обнаружению тестов и мощным возможностям, таким как фикстуры (для подготовки данных) и плагины.

Было написано 6 тестов для ключевых диалогов приложения: LoginWindow (авторизация), CreateBatchDialog (создание партии), AddOrderDialog (создание заказа) и EditRawMaterialDialog (редактирование сырья).

Листинг кода 9 – Модульные тест

import pytest

from unittest.mock import patch, MagicMock

from PySide6.QtWidgets import QApplication, QLineEdit, QComboBox, QDoubleSpinBox, QTableWidgetItem

from PySide6.QtCore import QDate

from sqlalchemy import create\_engine

from sqlalchemy.orm import Session

from ui import LoginWindow, CreateBatchDialog, AddOrderDialog, MainWindow, EditRawMaterialDialog

from models import Base, User, RawMaterial, Recipe, Batch, Client, FinishedProduct, Order, OrderItem

# Фикстура для создания QApplication

@pytest.fixture(scope="session")

def app():

app = QApplication([])

yield app

app.quit()

# Фикстура для создания базы данных

@pytest.fixture

def db\_session():

engine = create\_engine('sqlite:///:memory:')

Base.metadata.create\_all(engine)

session = Session(engine)

yield session

session.close()

engine.dispose()

# Фикстура для очистки базы перед каждым тестом

@pytest.fixture

def clean\_db(db\_session):

db\_session.query(User).delete()

db\_session.query(RawMaterial).delete()

db\_session.query(Recipe).delete()

db\_session.query(Batch).delete()

Продолжение листинга 9

db\_session.query(Client).delete()

db\_session.query(FinishedProduct).delete()

db\_session.query(Order).delete()

db\_session.query(OrderItem).delete()

db\_session.commit()

return db\_session

def test\_login\_success(clean\_db, app):

"""Тест успешной авторизации с реальными QLineEdit"""

session = clean\_db

user = User(login="admin", password="admin123", role="Администратор")

session.add(user)

session.commit()

dialog = LoginWindow()

dialog.session = session

dialog.login\_input = QLineEdit()

dialog.login\_input.setText("admin")

dialog.password\_input = QLineEdit()

dialog.password\_input.setText("admin123")

with patch.object(dialog, 'accept') as mock\_accept:

dialog.validate\_and\_accept()

mock\_accept.assert\_called\_once()

def test\_login\_invalid\_password(clean\_db):

"""Тест авторизации с коротким паролем"""

session = clean\_db

user = User(login="admin", password="admin123", role="Администратор")

session.add(user)

session.commit()

dialog = LoginWindow()

dialog.session = session

dialog.login\_input = MagicMock()

dialog.login\_input.text.return\_value = "admin"

dialog.password\_input = MagicMock()

dialog.password\_input.text.return\_value = "pass" # Пароль короче 6 символов

with patch('PySide6.QtWidgets.QMessageBox.warning') as mock\_warning:

dialog.validate\_and\_accept()

mock\_warning.assert\_called\_once()

assert "Пароль должен содержать минимум 6 символов" in mock\_warning.call\_args[0][2]

def test\_create\_batch\_success(clean\_db):

"""Тест создания партии с достаточным количеством сырья"""

session = clean\_db

Продолжение листинга 9

honey = RawMaterial(name="Мед", quantity=250.0, cost=300.0, purchase\_date="2025-01-15")

recipe = Recipe(name="Медовуха классическая", description="Мед, вода, дрожжи")

session.add(honey)

session.add(recipe)

session.commit()

dialog = CreateBatchDialog(current\_user\_id=1)

dialog.session = session

dialog.recipe\_combo = MagicMock()

dialog.recipe\_combo.currentText.return\_value = "Медовуха классическая"

dialog.volume\_spin = MagicMock()

dialog.volume\_spin.value.return\_value = 100.0

dialog.start\_date = MagicMock()

dialog.start\_date.date.return\_value = QDate(2025, 4, 10)

dialog.price\_spin = MagicMock()

dialog.price\_spin.value.return\_value = 600.0

with patch.object(dialog, 'accept') as mock\_accept:

dialog.create\_batch()

mock\_accept.assert\_called\_once()

batch = session.query(Batch).first()

assert batch is not None

assert batch.volume == 100.0

assert batch.status == "В брожении"

assert batch.price\_per\_liter == 600.0

updated\_honey = session.query(RawMaterial).filter\_by(name="Мед").first()

assert updated\_honey.quantity == 150.0 # Списано 100 кг

def test\_create\_batch\_insufficient\_honey(clean\_db):

"""Тест создания партии с недостаточным количеством меда"""

session = clean\_db

honey = RawMaterial(name="Мед", quantity=50.0, cost=300.0, purchase\_date="2025-01-15")

recipe = Recipe(name="Медовуха классическая", description="Мед, вода, дрожжи")

session.add(honey)

session.add(recipe)

session.commit()

dialog = CreateBatchDialog(current\_user\_id=1)

dialog.session = session

dialog.recipe\_combo = MagicMock()

dialog.recipe\_combo.currentText.return\_value = "Медовуха классическая"

dialog.volume\_spin = MagicMock()

dialog.volume\_spin.value.return\_value = 100.0

dialog.start\_date = MagicMock()

dialog.start\_date.date.return\_value = QDate(2025, 4, 10)

Продолжение листинга 9

dialog.price\_spin = MagicMock()

dialog.price\_spin.value.return\_value = 600.0

with patch('PySide6.QtWidgets.QMessageBox.warning') as mock\_warning:

dialog.create\_batch()

mock\_warning.assert\_called\_once()

assert "Недостаточно меда" in mock\_warning.call\_args[0][2]

batch = session.query(Batch).first()

assert batch is None

def test\_create\_order\_success(clean\_db):

"""Тест создания заказа с несколькими позициями"""

session = clean\_db

client = Client(name="ООО Медовый край", type="Юрлицо", contact="info@medkray.ru", inn="123456789012")

product1 = FinishedProduct(product\_id=1, batch\_id=1, volume=200.0, available\_volume=150.0, production\_date="2025-02-15", price\_per\_liter=600.0)

product3 = FinishedProduct(product\_id=3, batch\_id=3, volume=100.0, available\_volume=80.0, production\_date="2025-03-02", price\_per\_liter=700.0)

session.add(client)

session.add(product1)

session.add(product3)

session.commit()

dialog = AddOrderDialog(current\_user\_id=1)

dialog.session = session

dialog.client\_combo = MagicMock()

dialog.client\_combo.currentText.return\_value = "ООО Медовый край"

dialog.date\_edit = MagicMock()

dialog.date\_edit.date.return\_value = QDate(2025, 4, 10)

dialog.items\_table = MagicMock()

dialog.items\_table.rowCount.return\_value = 2

combo1 = MagicMock()

combo1.currentText.return\_value = "Продукт 1 - Медовуха классическая"

volume\_spin1 = QDoubleSpinBox()

volume\_spin1.setValue(50.0)

item1 = MagicMock()

item1.text.return\_value = "30000 руб"

combo2 = MagicMock()

combo2.currentText.return\_value = "Продукт 3 - Медовый эль"

volume\_spin2 = QDoubleSpinBox()

volume\_spin2.setValue(20.0)

item2 = MagicMock()

item2.text.return\_value = "14000 руб"

Продолжение листинга 9

dialog.items\_table.cellWidget.side\_effect = [combo1, volume\_spin1, combo2, volume\_spin2, combo1, volume\_spin1, combo2, volume\_spin2]

dialog.items\_table.item.side\_effect = [item1, item2]

with patch.object(dialog, 'accept') as mock\_accept:

dialog.create\_order()

mock\_accept.assert\_called\_once()

order = session.query(Order).first()

assert order is not None

assert order.total\_order\_cost == 44000.0

assert len(order.order\_items) == 2

updated\_product1 = session.query(FinishedProduct).filter\_by(product\_id=1).first()

updated\_product3 = session.query(FinishedProduct).filter\_by(product\_id=3).first()

assert updated\_product1.available\_volume == 100.0

assert updated\_product3.available\_volume == 60.0

def test\_edit\_raw\_material(clean\_db):

"""Тест редактирования сырья"""

session = clean\_db

raw\_material = RawMaterial(name="Мед", quantity=250.0, cost=300.0, purchase\_date="2025-01-15")

session.add(raw\_material)

session.commit()

dialog = EditRawMaterialDialog(material\_id=raw\_material.material\_id)

dialog.session = session

dialog.material = session.query(RawMaterial).filter\_by(material\_id=raw\_material.material\_id).first()

print(f"Material before update: {dialog.material.\_\_dict\_\_}")

dialog.name\_input = MagicMock()

dialog.name\_input.text.return\_value = "Мед обновленный"

dialog.quantity\_spin = MagicMock()

dialog.quantity\_spin.value.return\_value = 200.0

dialog.cost\_spin = MagicMock()

dialog.cost\_spin.value.return\_value = 350.0

dialog.purchase\_date = MagicMock()

dialog.purchase\_date.date.return\_value = QDate(2025, 1, 20)

with patch.object(dialog, 'accept') as mock\_accept:

with patch('PySide6.QtWidgets.QMessageBox.warning') as mock\_warning:

dialog.save\_material()

mock\_accept.assert\_called\_once()

mock\_warning.assert\_not\_called()

updated\_material = session.query(RawMaterial).filter\_by(material\_id=raw\_material.material\_id).first()

Продолжение листинга 9

print(f"Updated material: {updated\_material.\_\_dict\_\_}")

assert updated\_material.name == "Мед обновленный"

assert updated\_material.quantity == 200.0

assert updated\_material.cost == 350.0

assert updated\_material.purchase\_date == "2025-01-20"

На рисунке 13 представлен успешный итог проведения модульного тестирования.

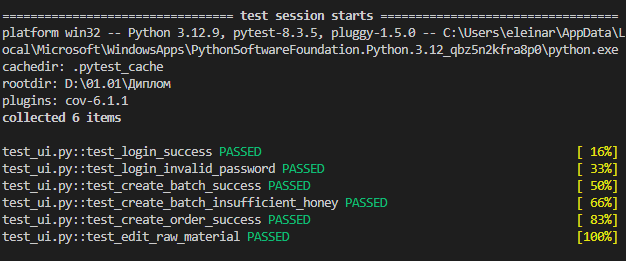


Рисунок 13 – Результат модульного тестирования

Приложения

Приложение А – Спецификация требований

Термины и определения

В данном документе используются следующие термины: сырье — материалы, используемые для производства медовухи (например, мед); партия — единица производства медовухи, создаваемая на основе выбранного рецепта; рецепт — описание состава и технологии производства (например, "Медовуха классическая" требует 1 кг меда на 1 литр продукции); заказ — запрос клиента на покупку готовой продукции, включающий одну или несколько позиций с указанием продукта и объема; продукт — готовая медовуха, произведенная из партии и доступная для продажи (например, "Медовуха классическая" или "Медовый эль").

Общие положения

Данный документ описывает требования к программному приложению для управления производством медовухи. Приложение предназначено для автоматизации процессов учета сырья, управления производством партий, оформления заказов и контроля доступа пользователей. Оно должно обеспечивать удобное взаимодействие с базой данных SQLite, где хранятся данные о сырье, партиях, заказах и пользователях. Приложение создано для сотрудников предприятия по производству медовухи, включая администраторов, сотрудников производства и менеджеров по продажам. Администраторы получают полный доступ ко всем функциям, сотрудники производства могут создавать и управлять партиями, а менеджеры по продажам оформляют заказы и просматривают доступную продукцию. Приложение должно быть простым в использовании, надежным и обеспечивать базовую автоматизацию производственных процессов.

Типичный сценарий использования

Администратор запускает приложение, вводит логин "admin" и пароль "admin123" в окне авторизации, после чего переходит в главное окно. В главном окне он видит таблицы с сырьем, партиями и заказами. Администратор открывает диалог создания партии, выбирает рецепт "Медовуха классическая", указывает объем 100 литров, дату начала 2025-04-10 и цену за литр 600 рублей. Система проверяет наличие меда (250 кг в базе), создает партию, уменьшает количество меда до 150 кг и закрывает диалог. Затем администратор открывает диалог оформления заказа, выбирает клиента "ООО Медовый край", указывает дату 2025-04-10, добавляет позицию с продуктом "Медовуха классическая" объемом 50 литров (доступно 150 литров) и стоимостью 30000 рублей. Заказ создается, доступный объем продукта уменьшается до 100 литров, и диалог закрывается.

Функциональные требования

Авторизация (LoginWindow) Приложение предоставляет окно авторизации с полями для ввода логина и пароля, а также кнопкой "Войти". Система проверяет корректность введенных данных: логин должен содержать минимум 3 символа, пароль — минимум 6 символов. Если поля пустые, отображается сообщение об ошибке "Заполните все поля". Если логин или пароль не соответствуют минимальной длине, отображаются сообщения "Логин должен содержать минимум 3 символа" или "Пароль должен содержать минимум 6 символов". При успешной авторизации пользователь переходит в главное окно приложения в соответствии с его ролью (администратор, сотрудник производства или менеджер по продажам). В случае некорректных данных авторизация не выполняется, и пользователь остается на экране входа.  
Создание партии (CreateBatchDialog) Приложение позволяет создавать партии медовухи через диалоговое окно, содержащее выпадающий список рецептов, поле для ввода объема, поле для выбора даты начала производства, поле для ввода цены за литр и кнопки "Создать" и "Отмена". Пользователь выбирает рецепт, указывает объем, дату и цену. Система проверяет наличие достаточного количества сырья: для рецепта "Медовуха классическая" требуется 1 кг меда на 1 литр продукции. Если сырья недостаточно, отображается сообщение "Недостаточно меда", и партия не создается. При успешном создании партия добавляется в базу данных со статусом "В брожении", количество сырья уменьшается, и диалог закрывается.  
Оформление заказа (AddOrderDialog) Приложение позволяет оформлять заказы через диалоговое окно с выпадающим списком клиентов, полем для выбора даты, таблицей для добавления позиций (продукт, объем, стоимость) и кнопками "Создать" и "Отмена". Пользователь выбирает клиента, указывает дату и добавляет позиции, выбирая продукт и объем. Система проверяет доступный объем: если запрошенный объем превышает доступный, отображается сообщение "Недостаточно продукции [ID продукта] (нужно [объем] л, есть [доступный объем] л)", и заказ не создается. Также проверяется, что объем больше 0, а стоимость не отрицательная; в противном случае отображается сообщение "Объем > 0, стоимость >= 0". При успешном создании заказ добавляется в базу, доступный объем уменьшается, общая стоимость рассчитывается, и диалог закрывается.  
Редактирование сырья (EditRawMaterialDialog) Приложение позволяет редактировать сырье через диалоговое окно с полями для названия, количества, стоимости, даты закупки и кнопками "Сохранить" и "Отмена". Пользователь изменяет данные, система проверяет их: название не должно быть пустым, количество и стоимость — не отрицательными. Если данные некорректны, отображается сообщение "Заполните название, количество и стоимость корректно", и изменения не сохраняются. При успешном редактировании данные обновляются в базе, и диалог закрывается.  
Основной интерфейс (MainWindow) После авторизации пользователь переходит в главное окно с таблицами сырья, партий и заказов, а также кнопками для вызова диалогов в зависимости от роли: администраторы могут добавлять, редактировать и удалять записи, сотрудники производства — создавать и просматривать партии, менеджеры по продажам — оформлять и просматривать заказы.

Нефункциональные требования

Приложение должно быть реализовано на языке Python версии 3.8 или выше с использованием библиотек PySide6 версии 6.0 или выше для графического интерфейса и SQLAlchemy версии 1.4 или выше для работы с базой данных SQLite. Время отклика интерфейса на действия пользователя не должно превышать 1 секунды при нормальной нагрузке. Приложение должно быть совместимо с операционной системой Windows 10 и выше. Данные в базе должны сохраняться надежно, без потери при корректном завершении работы приложения. Интерфейс должен быть интуитивно понятным, с использованием стандартных элементов управления: кнопки, таблицы, поля ввода, выпадающие списки. Все сообщения об ошибках должны быть на русском языке и четко объяснять причину проблемы.

Требования к тестированию

Для проверки корректности работы приложения разработаны модульные тесты с использованием библиотеки pytest версии 7.0 или выше. Тесты должны покрывать основные сценарии: успешная и неуспешная авторизация, создание партии при достаточном и недостаточном количестве сырья, оформление заказа и редактирование сырья. Каждый тест должен проверять ожидаемое поведение системы, включая обработку ошибок и обновление данных в базе. Тесты должны быть независимыми, используя временную базу данных в памяти (SQLite :memory:). Критерии приемки: приложение считается готовым, если все тесты проходят, и администратор может успешно авторизоваться, создать партию, оформить заказ и отредактировать сырье.

Требования к документации

К приложению должна быть приложена пользовательская документация, описывающая процесс установки, запуска и использования приложения, включая описание ролей пользователей, их возможностей и инструкции по устранению типичных ошибок (например, сбой подключения к базе данных). Также должна быть подготовлена техническая документация, описывающая структуру базы данных, зависимости проекта (например, требуемые версии Python, PySide6, SQLAlchemy) и инструкции по запуску тестов.

Приложение Б – Репозиторий с документами в системе контроля версий

https://github.com/Eleinar/TheEnd