JetBrains

Факультет разработки

Специальность программирование

Выпускная дипломная работа на тему:

“Исследование особенностей разработки приложения с графическим интерфейсом на языке Java и работы с базами данных на примере программы по учету товаров”

Автор:
Арсентьева Виктория Сергеевна

Г. Санкт-Петербург, 2023

Оглавление

Введение

Тема проекта: Исследование особенностей разработки приложения с графическим интерфейсом на языке java и работы с базами данных на примере программы по учету товаров.

Цель: изучить особенности разработки приложения для бизнеса и рассмотреть все ее этапы.

Задачи:

1. Изучить литературу, касающуюся темы исследования.
2. Подробно узнать о всех этапах разработки приложения.
3. Рассмотреть основные виды применения графического интерфейса в java.
4. Научиться использовать JavaFx и Scene Builder
5. Узнать, что такое база данных, рассмотреть существующие типы и привести примеры.
6. Научиться пользоваться СУБД.
7. Рассмотреть основные понятия ACID.
8. Исследовать особенности MySQL.
9. Разработать предложения по учету товаров для магазина одежды
10. Выполнить ручное тестирование программы

Инструменты: VS Code, MySQL, Git

Основная часть

Этапы разработки приложения

Техническое задание

Техническое задание (ТЗ) — это часто используемый в IT документ для подготовки к реализации программного продукта. В нем описывается планируемый функционал, а также учитываются индивидуальные особенности разработки.

На этом этапе необходимо перевести идею приложения в вид конкретных технических указаний.

Техническое задание необходимо для избежание недопониманий между заказчиком и исполнителем, а также для более четкого понимания шагов выполнения работы.

Выделяют два наиболее популярных подхода к составлению ТЗ: водопадная модель и Agile-методология.

Водопадная модель — методология разработки ПО, построенная на строгом выполнении заранее определенной последовательности шагов, начиная от сбора и анализа требований, заканчивая реализацией и интеграцией разработанного продукта.

Такой подход считается устаревшим и неоптимальным для разработки масштабных проектов, но подходит для небольших проектов, не имеющих обширного функционала.

Agile — это итеративный подход к управлению проектами и разработке программного обеспечения, который помогает командам быстрее и с меньшими проблемами поставлять ценность клиентам.

Agile-манифест разработки программного обеспечения гласит:

- Люди и взаимодействие важнее процессов и инструментов
- Работающий продукт важнее исчерпывающей документации
- Сотрудничество с заказчиком важнее согласования условий контракта
- Готовность к изменениям важнее следования первоначальному плану.

Для более комфортного взаимодействия заказчика и команды разработки, независимо от выбранной методологии, стоит разрабатывать техническое задание.

Проектирование и разработка дизайна приложения

Проектирование дизайна является очень важной частью разработки любого проекта, имеющего графический интерфейс. Потому что это то, с чем непосредственно будет взаимодействовать пользователь нашего продукта.

Для начала, необходимо ознакомиться с ТЗ и понять, какие функции необходимы в приложении, и создать user flow&

Userflow – это визуальное представление последовательности действий, которые пользователь выполняет для достижения своей цели (блок-схема работы приложения).

Далее необходимо нарисовать прототипы всех (если их несколько) экранов вашего приложения(wireframes)

Wireframes – это эскизы, схемы того, где будут располагаться изображения, ярлыки, кнопки и прочее.

На следующем этапе создаются прототипы приложения(mockup)

Mockup – это низко детализированный прототип, чтобы определить приоритет и расположение элементов интерфейса на экране, предусмотреть для них удобное для доступа местоположение.

После, создаются детализированные, кликабельные прототипы с необходимой динамикой, анимацией и микровзаимодействиями для пользовательского тестирования.

После тестирования исправляются недочеты, дорабатывается дизайн и готовый интерфейс переходит на этап разработки.

Важно при проектировании учитывать особенности платформы, на которую рассчитан продукт; стремиться к интуитивно понятному дизайну, которым будет удобно пользоваться и понимать, чего хочет пользователь от продукта.

Разработка приложения (написание кода)

После готового ТЗ и отрисованного дизайна для приложения работа над проектом перемещается на стадию программирования. Во время этого этапа создается функционал самого приложения. Этот процесс подразделяется на две части: Front-end (клиентская) и Back-end (серверная) части.

Frontend – разработка пользовательского интерфейса, то есть той части сайта или приложения, которую видят посетители страницы. Главная задача фронтенд разработчика — перевести готовый дизайн-макет в код так, чтобы все работало правильно.

Наиболее популярные языки для работы с frontend: HTML, CSS, Javascript, React, Vue, TypeScript, Elm, JQuery.

Backend – это внутренняя часть продукта, которая находится на сервере и скрыта от пользователей. Для её разработки могут использоваться самые разные языки, например, Python, PHP, Go, JavaScript, Java, C#.

Тестирование

В процессе работы будет появляться не одна версия приложения, и каждая из них будет разделена процессом тестирования. Нельзя просто создать и выпустить продукт, нужно удостовериться, что он работает четко, не огорчая пользователя неполадками.

Тестировщики прописывают множество сценариев, по которым начинают работать приложением, проходя путь, который в дальнейшем совершат пользователи. Они включают самый распространенный набор действий, и важно, чтоб ни на одном из этапов не произошло сбоя.

Тестирование приложения проходит, обнаруживаются неисправности, которые разработчики исправляют. И снова тестирование уже новой, исправленной версии приложения. И так до тех пор, пока не будет получен удовлетворительный результат, способный привести к конверсии.

Как и при тестировании сайтов, в веб-приложении сначала тестируется верстка и ее соответствие дизайну, после этого проверяется функциональная часть проекта.

Техническая поддержка

Техническая поддержка помогает оперативно исправить недочеты и вернуть приложение в рабочее состояние, если была выявлена неисправность или необходимо что-либо поменять (цвет кнопок и текста, шрифты, расположение картинки на странице).

Также и сайты, и приложения необходимо обновлять и добавлять новый функционал.

Графический интерфейс в Java

Для реализации графического интерфейса в Java существует несколько библиотек, имеющих свои преимущества и недостатки. Рассмотрим каждую из них в хронологическом порядке, начиная с самой старой.

Abstract Window Toolkit (AWT)

Abstract Window Toolkit (AWT) впервые была выпущена в 1995 году компанией Sun Microsystems. Это была первая попытка создать графический интерфейс для Java. AWT выступал в качестве прослойки, вызывающей методы из библиотек, написанных на языке C. А эти методы, в свою очередь, использовали графические компоненты операционной системы. (рис. 1)

Сейчас AWT используется в основном для апплетов.

Достоинства:

- Скорость работы;
- Графические компоненты похожи на стандартные;
- Автоматическое освобождение использованных ресурсов;
- Простота освоения;

Недостатки:

- Использование нативных компонентов налагает ограничения на использование их свойств. Некоторые компоненты могут вообще не работать на «неродных» платформах;
- Некоторые свойства, такие как иконки и всплывающие подсказки, в AWT вообще отсутствуют;
- Стандартных компонентов AWT очень немного
- Программа выглядит по-разному на разных платформах (может быть кривоватой).

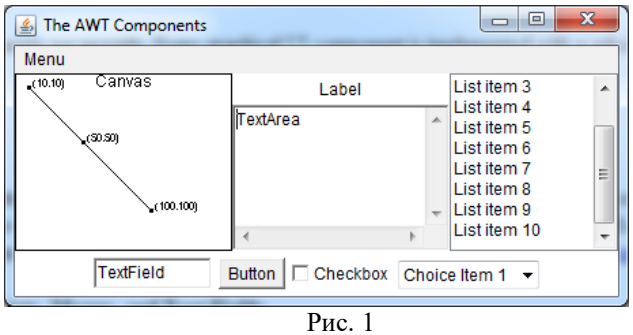


Рис. 1

Swing

После AWT, в 1998 году, Sun выпустила Swing. Он полностью написан на Java и для отрисовки использует 2D. Набор стандартных компонентов значительно превосходит AWT по разнообразию и функциональности. Однако, скорость работы ранних версий Swing была довольно низкой, а ошибки в написании программы могли и вовсе привести к зависанию операционной системы. (рис. 2)

На сегодняшний день Swing остается самым популярным фреймворком для создания пользовательских интерфейсов на Java.

Достоинства:

- Простота освоения
- Большое количество документации, помогающей решить все возникающие проблемы;
- Встроенный редактор форм почти во всех средах разработки;
- На базе Swing есть много расширений типа SwingX;
- Поддержка различных стилей (Look and feel).
- Принцип Lightweight (компоненты Swing обрисовываются самими компонентами на поверхности родительского окна, без использования компонентов операционной системы).

Недостатки:

- Работать с менеджерами компоновки непросто в сложных интерфейсах.

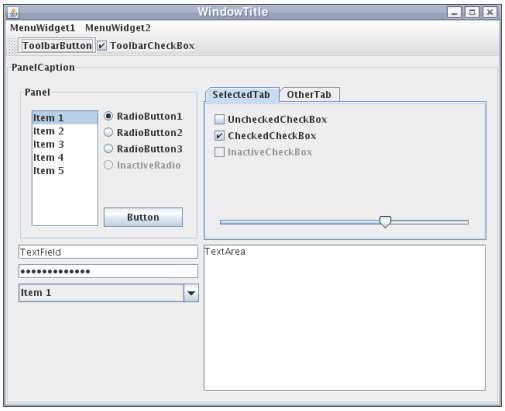


Рис. 2

Standard Widget Toolkit (SWT)

SWT был выпущен компанией IBM во времена, когда Swing был ещё медленным, и в основном для продвижения среды программирования Eclipse.

Использование SWT делает Java-приложение более эффективным, но снижает независимость от операционной системы и оборудования, требует ручного освобождения ресурсов (рис. 3)

Достоинства:

- Использует компоненты операционной системы — скорость выше;
- Eclipse предоставляет визуальный редактор форм;
- Обширная документация и множество примеров;
- Возможно использование AWT и Swing компонентов.

Недостатки:

- Для каждой платформы необходимо поставлять отдельную библиотеку;
- Нужно всегда следить за использованием ресурсов и вовремя их освобождать;
- Сложная архитектура

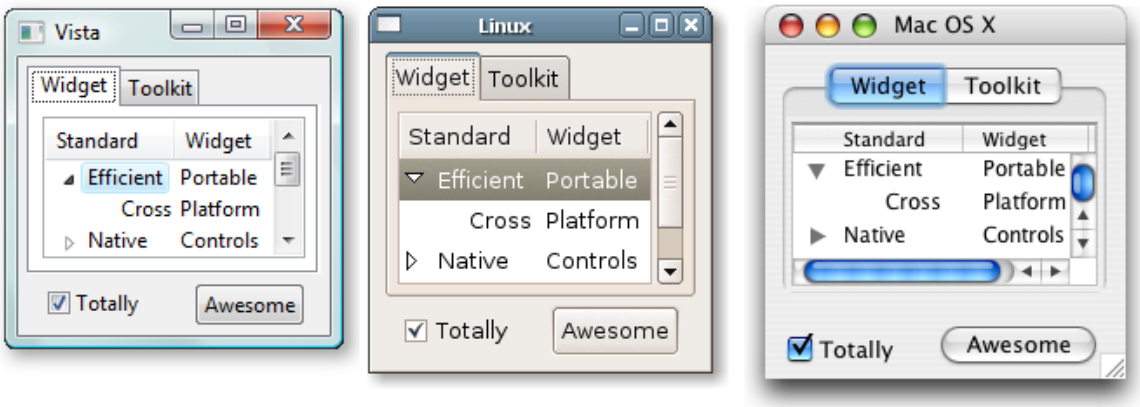


Рис. 3

JavaFX

JavaFX была выпущена в 2008 году компанией Oracle. Она позиционируется как платформа для создания насыщенного интернет-приложения.

Может использоваться как для создания настольных приложений, запускаемых непосредственно из-под операционных систем, так и для интернет-приложений, работающих в браузерах, и для приложений на мобильных устройствах.

Достоинства:

- Быстрая работа за счет графического конвейера;
- Множество различных компонентов;
- Поддержка стилей;
- Утилиты для создания установщика программы;
- Приложение можно запускать как десктопное и в браузере как часть страницы.
- Подробная документация и большое число готовых примеров
- Поддержка мультимедийного контента, анимации
- Генерирование разметки FXML, с помощью графического редактора Scene Builder

Недостатки:

- Фреймворк еще разрабатывается, поэтому могут случаться падения;

Рис. 4

База данных и СУБД

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД).

СУБД – интерфейс между базой данных и пользователями или программами, предоставляющий пользователям возможность получать и обновлять информацию, а также управлять ее упорядочением и оптимизацией.

В качестве примеров популярного программного обеспечения для управления базами данных, или СУБД, можно назвать MySQL, Microsoft Access, Microsoft SQL Server, FileMaker Pro, СУБД Oracle Database и dBASE.

Рассмотрим основные типы баз данных и их особенности:

Резидентные

Резидентная база данных (*in-memory database, IMDB*) — база данных, размещаемая в оперативной памяти.

Данные обрабатываются быстро, поэтому резидентные БД популярны там, где нужно обеспечить максимально короткое время отклика. Они помогают управлять телекоммуникационным оборудованием, проводить торги в онлайн-режиме или Real-Time обслуживание. Базы in-memory поддерживают и быстрое написание, и быстрое чтение. В основном они работают с записями «ключ-значение», но также могут работать со столбцами.

Чтобы при неожиданной перезагрузке не потерять данные, нужно сделать запись с предварительным журналированием на энергонезависимом устройстве. Это можно отнести к минусам базы in-memory — приходится вкладываться в дорогостоящие инфраструктурные решения, чтобы обеспечить бесперебойное питание. Также нужно постоянно копировать информацию на твёрдые носители. Ещё один недостаток БД — дорогое масштабирование.

Примеры: Redis, Apache Ignite, Tarantool

Поисковые

Поисковая база данных – организованный массив информации, в котором хранятся данные, собранные модулями индексирования поисковой системы (поисковыми ботами).

Этот тип БД нужен для получения сведений через фильтр. Искать можно по любому введённому значению, в том числе по отдельным словам. Можно пользоваться полнотекстовым поиском. Поисковые базы данных хорошо масштабируются и удобны для хранения журналов, объёмных текстовых значений.

Можно использовать поисковые БД для мониторинга оптимизации цен, обнаружения ошибок в приложении по бронированию билетов и решения множества других задач. В базе могут хранится миллиарды документов. Поиск осуществляется быстро. Минусы системы — плохая аналитическая поддержка и ограниченная возможность применения БД (можно использовать только для пакетных вставок).

Пример: Elastic.

Столбчатые

Столбчатая база данных – нереляционная БД, в которой данные группируются по столбцам. В ней «соседними» являются данные из одного и того же столбца, но из разных строк.

В БД такого типа данные хранятся в столбцах, а не в строках. Получение доступа к содержимому осуществляется без помощи ключей. При использовании столбчатых баз данных используют пакетную вставку. Есть поддержка аналитики и возможность удобного масштабирования.

Такие базы данных используют там, где нужно запрашивать информацию по определённым столбцам, — в системах розничных продаж и финансовых транзакций. Основной минус у БД только один: она подходит только для пакетных вставок.

Примеры: Clickhouse, Vertica.

Документные

Документные базы данных – БД, предназначенные для хранения, извлечения и обработки документоориентированной информации и предоставляют современный способ хранения данных в формате JSON, а не в виде строк и столбцов.

В таких базах отлично хранится несвязанная информация в больших объёмах. Они поддерживают JSON. Для любого ключа можно создать сложное значение и сразу включить всю структуру данных в одну запись. Выборка по запросу может содержать части множества документов без их полной загрузки в оперативную память.

В документоориентированных базах нет привязки к схеме. Они подходят для OLTP и поддерживают сложные типы. Такие БД предпочитают использовать в системах управления содержимым, для поиска документов, в издательском деле. Три недостатка базы данных — отсутствие хорошей аналитической поддержки и поддержки транзакций, а также сложности с масштабированием.

Примеры: CouchDB, Couchbase, MongoDB

Графовые

Графовые базы данных – нереляционные БД, хранящие данные в контексте сущностей и связей между сущностями.

Данные хранятся в виде графов, то есть моделей с узлами и связями. Они достаточно гибкие, с логичной структурой. Узлы служат для хранения сущностей данных, а рёбра — для хранения взаимосвязей между сущностями, которыми можно управлять.

Графовые БД применяют для решения задач в биоинформатике, а также для моделирования социальных сетей, чтобы хранить взаимосвязанную информацию о людях. Базы данных такого типа плохо поддаются масштабированию, а второй их недостаток — необходимость использовать особый язык запросов SPARQL, который отличается от SQL.

Примеры: OrientDB, Neo4j

Объектно-ориентированные

Объектно-ориентированная база данных (ООБД) — база данных, в которой данные моделируются в виде объектов, их атрибутов, методов и классов.

В манифесте ООБД предлагаются обязательные характеристики, которым должна отвечать любая ООБД:

- Поддержка сложных объектов.
- Поддержка индивидуальности объектов
- Поддержка инкапсуляции.
- Поддержка типов и классов.
- Поддержка наследования типов и классов от их предков.
- Перегрузка в сочетании с полным связыванием
- Вычислительная полнота.
- Набор типов данных должен быть расширяемым.

Пример: Oracle RDBMS

Распределенные

Распределённая база данных (*distributed database, DDB*) — база данных, составные части которой размещаются в различных узлах компьютерной сети в соответствии с каким-либо критерием.

DDB — это именно единая база данных, а не произвольный набор файлов, индивидуально хранимых на разных узлах сети и являющейся распределенной файловой системой. Данные представляют собой DDB, только если они связаны в соответствии с некоторым структурным формализмом, реляционной моделью, а доступ к ним обеспечивается единым высокоуровневым интерфейсом.

Распределённые базы могут иметь разный уровень реплицированности — от полного отсутствия дублирования информации, до полного дублирования всей информации во всех распределённых копиях (например, блокчейн).

Пример: DB2

Реляционные

Реляционные базы данных – БД, в которых Данные организованы в виде таблиц, состоящих из столбцов и строк.

Это самый популярный тип БД, в которых информация хранится в виде таблиц. В строках находится описание каждого отдельного свойства объекта, а столбцы нужны для извлечения определённых свойств из строки. Таблицы могут быть взаимосвязаны.

Реляционная модель проста, но позволяет выполнить множество разных задач. Ею удобно пользоваться, если нужно связать элементы данных между собой и безопасно и надёжно управлять ими. Такие таблицы можно создать для хранения и обработки телефонных номеров пациентов, логинов и паролей пользователей, для отслеживания товарных запасов. При этом БД обеспечивает целостность данных в различных экземплярах базы в одно и то же время.

В реляционных БД есть поддержка SQL, а также индексация, которая позволяет быстрее находить нужные данные. Особый плюс таких баз — нормализация данных: они делятся на разные таблицы, поэтому исключены повторяющиеся или пустые ячейки. Транзакции реляционных БД соответствуют ACID — набору свойств, который гарантирует их надёжную обработку. Из минусов баз можно отметить относительно низкую скорость доступа к данным, плохую поддержку неструктурированных данных, сложность масштабирования и образование большого количества таблиц, из-за чего бывает трудно понять структуру данных.

Примеры: MySQL, Oracle DB, PostgreSQL

Требования ACID и транзакции

Говоря о базах данных, необходимо рассмотреть понятие ACID требований и транзакций. От корректного функционирования БД может зависеть не только скорость, но и надежность приложения.

Для начала, узнаем, что такое транзакция и какие она имеет виды.

Транзакция – это набор последовательных операций с базой данных, соединенных в одну логическую единицу.

Виды:

- **Неявная транзакция** (задает любую отдельную инструкцию INSERT, UPDATE или DELETE как единицу транзакции).
- **Явная транзакция** (обычно это группа инструкций языка Transact-SQL, начало и конец которой обозначаются такими инструкциями, как BEGIN TRANSACTION, COMMIT и ROLLBACK).

Свойства транзакции обозначаются аббревиатурой “ACID”, каждая буква которой обозначает свойство.

ACID (*atomicity, consistency, isolation, durability*) — набор требований к транзакционной системе, обеспечивающий наиболее надёжную и предсказуемую её работу.

Рассмотрим каждое из свойств подробно:

Атомарность (Atomicity)

Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной. Поскольку на практике невозможно одновременно и атомарно выполнить всю последовательность операций внутри транзакции, вводится понятие «отката» (rollback): если транзакцию не удаётся полностью завершить, результаты всех её до сих пор произведённых действий будут отменены, и система вернётся во «внешне исходное» состояние — со стороны будет казаться, что транзакции и не было.

Согласованность (Consistency)

Транзакция, достигающая своего нормального завершения (*EOT — end of transaction*, завершение транзакции) и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных. Другими словами, каждая успешная транзакция по определению фиксирует только допустимые результаты

Изолированность (Isolation)

Во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат.

Долговечность (Durability)

Независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу. Другими словами, если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

Система контроля версий

Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии.

Преимущества использования систем контроля версий:

1. **Полная история изменений каждого файла**

Наличие полной истории позволяет возвращаться к предыдущим версиям, чтобы проводить анализ основных причин возникновения ошибок и устранять проблемы в старых версиях программного обеспечения.

2. **Ветвление и слияние**

Создание «веток» в инструментах VCS позволяет иметь несколько независимых друг от друга направлений разработки, а также выполнять их слияние, чтобы разработчики могли проверить, что изменения, внесенные в каждую из веток, не конфликтуют друг с другом.

3. **Отслеживаемость**

Благодаря этому разработчики могут вносить корректные и совместимые изменения в соответствии с долгосрочным планом разработки системы. Это особенно важно для эффективной работы с унаследованным кодом, поскольку дает разработчикам возможность точнее оценить объем дальнейшей работы.

Разрабатывать программное обеспечение можно и без управления версиями, но такой подход подвергает проект огромному риску. Таким образом, вопрос заключается не в том, использовать ли управление версиями, а в том, какую систему управления версиями выбрать.

Выделяют следующие виды СКВ:

1. **Локальные (rcs)**

Локальные системы контроля версий — это самый простой вид подобных систем. Они используются в основном индивидуальными разработчиками, а не командами. В локальной системе контроля версий все данные проекта хранятся на одном компьютере, а изменения файлов проекта хранятся в виде патчей. (рис.5)

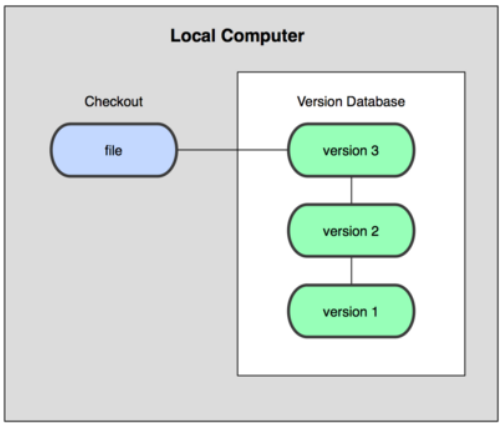


Рис. 5

2. **Централизованные (**CVS, Subversion, Perforce)

Централизованные системы контроля версий используют рабочий процесс на основе выгрузки для подключения к главному серверу. Все изменения или обновления исходного кода автоматически сохраняются в репозитории как новая версия. Централизованные системы контроля версий поддерживают мощные возможности ветвления и слияния, не требующие клонирования репозитория на несколько компьютеров. В этом смысле она более безопасна.

Для работы централизованным системам контроля версий требуется подключение к сети. (рис. 6)

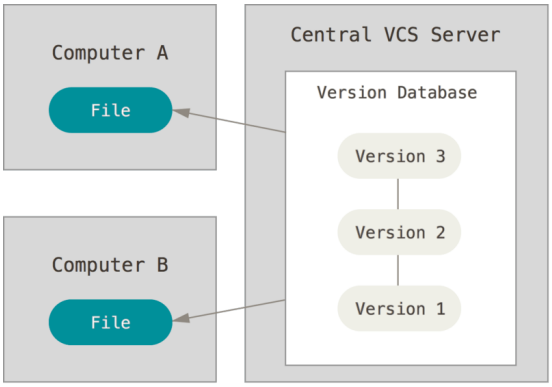


Рис. 6

3. Распределенные (Git, Mercurial, Bazaar, Darcs)

Распределенные системы контроля версий позволяют добавлять код, создавать ветвления и объединять код без подключения к главному серверу. Каждый участник команды работает с клонированным репозиторием, который хранится в облаке.

Основное преимущество таких систем состоит в том, что участники команды могут быстро работать независимо, не волнуясь о медленном сетевом подключении или VPN. Можно даже работать с проектом в автономном режиме, хотя для отправки или извлечения обновлений все же требуется подключение к Интернету. Систему контроля версий Git рассмотрим отдельно более подробно.

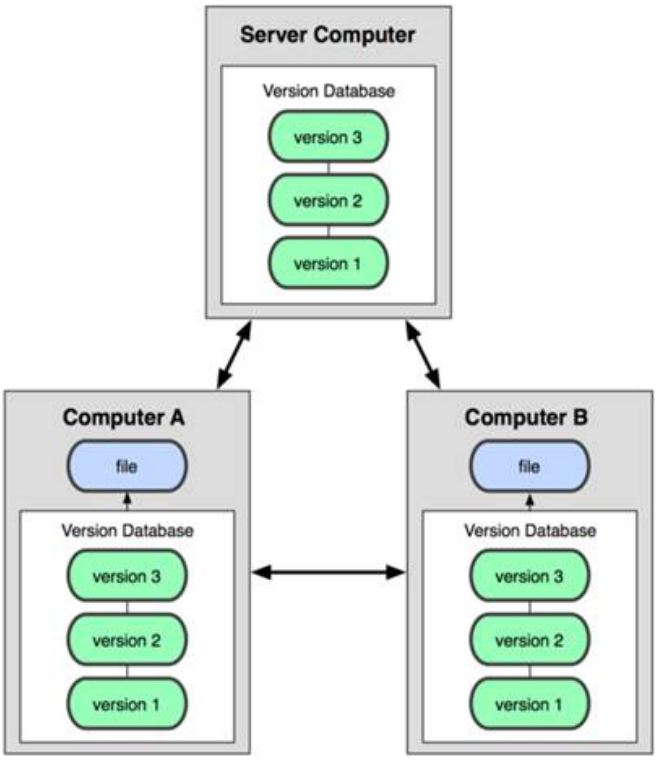


Рис. 7

Понятие Git и его преимущества

Git – распределенная система управления версиями.

Преимущества:

1. Производительность

Git показывает очень высокую производительность в сравнении со множеством альтернатив. Это возможно благодаря оптимизации процедур фиксации коммитов, создания веток, слияния и сравнения предыдущих версий. Алгоритмы Git разработаны с учетом глубокого знания атрибутов, характерных для реальных деревьев файлов исходного кода, а также типичной динамики их изменений и последовательностей доступа.

Вместе с тем распределенная архитектура системы сама по себе обеспечивает существенный прирост производительности.

2. Безопасность

При разработке в Git прежде всего обеспечивается целостность исходного кода под управлением системы. Содержимое файлов, а также объекты репозитория, фиксирующие взаимосвязи между файлами, каталогами, версиями, тегами и коммитами, защищены при помощи криптографически стойкого алгоритма хеширования SHA1. Он защищает код и историю изменений от случайных и злонамеренных модификаций, а также позволяет проследить историю в полном объеме.

Использование Git гарантирует подлинность истории изменений исходного кода.

3. Гибкость

Гибкость — одна из основных характеристик Git. Она проявляется в поддержке различных нелинейных циклов разработки, эффективности использования с малыми и крупными проектами, а также совместимости со многими системами и протоколами.

Инструкция по работе с Git

Проверка наличия установленного GIT

В терминале выполнить команду *git version*.

Если GIT установлен, то появится сообщение с информацией о версии программы, иначе появится сообщение об ошибке.

Установка GIT

Загружаем последнюю версию git с сайта https://git-scm.com/downloads. Устанавливаем с настройками по умолчанию

Настройка GIT

При первом использовании git необходимо представиться. Для этого нужно ввести в терминале две команды:

git config --global user.name "Ваше имя"

git config --global user.email "почта@example.com"

Для проверки ваших данных можно использовать следующую команду:

git config --global --list

Инициализация репозитория

Получить репозиторий можно двумя способами:

- **Клонирование уже существующего репозитория**

Чтобы клонировать репозиторий необходимо узнать, где он расположен и скопировать ссылку на него. Далее использовать команду *git clone*:

git clone https://github.com/ViktoriaArsenteva/git-course.git

- **Создание нового репозитория**

Для создания нового репозитория необходимо открыть терминал, зайти в папку проекта и выполнить команду *git init*

Если вы работаете через вспомогательные программы (например, VS Code), то просто создайте и откройте папку в программе.

Чтобы открыть папку через терминал, нужно создать папку (если ее еще нет) через терминал (указывайте реальный путь к месту, в котором будет хранится папка):

mkdir /Users/UserName/Desktop/Directoryname

Далее ее нужно открыть:

cd /Users/UserName/Desktop/Directoryname

Когда папка открыта выполняем команду *git init*

Запись изменений в репозиторий

Рассмотрим несколько важнейших команд:

- **git status**

Команда *git status* показывает информацию о текущем состоянии репозитория: есть ли новые, удаленные или измененные файлы.

Пример работы данной команды:
vikulik@Air-Vikulik GitInsruction % git status

On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
modified: GIT_INSTRUCTION.md
no changes added to commit (use "git add" and/or "git commit -a")

- **git add**

Чтобы git начал отслеживать новый или измененный файл необходимо использовать команду *git add*.

Чтобы добавить конкретный файл нужно к команде добавить параметр, например:

git add GIT_INSTRUCTION.md

Чтобы отслеживать все изменения и файлы нужно в качестве параметра добавить точку: *git add .*

- **git commit**

Для того, чтобы зафиксировать изменения используем команду *git commit* и в качестве параметра добавим *-m* (message):

git commit -m "create new file"

Также можно совместить команды *git add* и *git commit* (но в этом случае зафиксируются ВСЕ изменения!):

git commit -a -m "create new file"

- **git diff**

Команда *git diff* показывает все изменения, которые не были зафиксированы (т.е. разницу между последним коммитом и текущей версией).

Просмотр истории коммитов

Чтобы посмотреть все выполненные фиксации необходимо использовать команду *git log*. Она показывает всю информацию о каждом коммите (хеш, автор, список изменений и дата).

Для более удобного вывода можно использовать команду с параметром *--one line*:

git log --oneline

В результате будет выведен список коммитов с указанием хеша и комментария.

Перемещение между изменениями

Для того, чтобы откатиться к определенному комиту используетя команда *git checkout* и в качестве параметра принимает хеш необходимого коммита (узнать его можно через команду *git log*).

git checkout de106dd

Для возвращения к актуальной версии можно использовать команду *git checkout master* или команду *git switch* и в качестве параметра добавить “ – “ :

git switch -

git checkout master

Игнорирование файлов

Чтобы игнорировать файлы или папки необходимо создать отдельный файл в репозитории с названием “*.gitignore”*. (рис. 8)

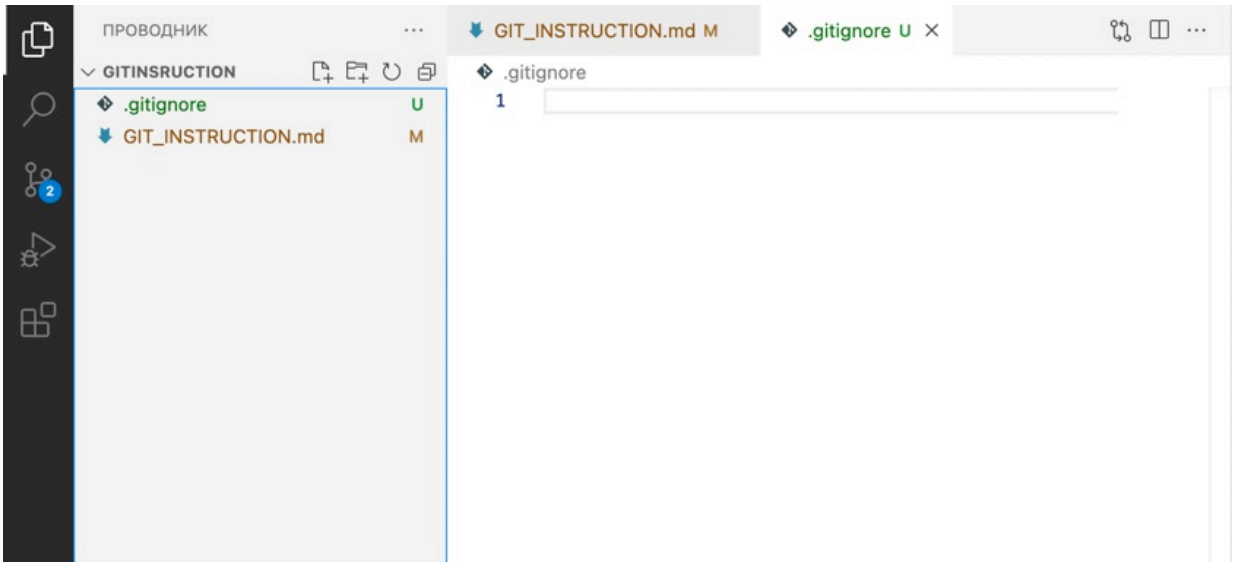


Рис. 8

Работа с ветками

- **Создание веток**

Чтобы создать ветку необходимо выполнить команду *git branch* и указать в качестве параметра название новой ветки:

git branch branch_name

Команда создает новую ветку, но не переходит на нее, поэтому, чтобы переключиться на новую ветку необходимо выполнить следующую команду:

git checkout branch_name

Чтобы посмотреть все ветки нужно выплнить команду *git branch* без параметра:

vikulik@Air-Vikulik GitInsruction % git branch

** createbranch*

ignore

master

Ветка на которой сейчас находится пользователь отмечена “ * ”.

- **Слияние веток**

Чтобы объединить ветки необходимо использовать команду *git merge* и в качестве параметра указать название ветки, которую необходимо слить в текущую:

git merge branch_name

- **Разрешение конфликтов**

Если при слиянии git обнаружит, что строки в ветках различаются, то возникнет конфликт, который будет предложено разрешить.

При конфликте git предложит выбрать какие изменения записать или оставить все.

- **Удаление веток**

Если изменения из ветки были перенесены в основную и ветка больше не нужна, то ее можно удалить с помощью команды *git branch* с параметром *-d* и именем ветки:

git branch -d branch_name

Работа с удаленными репозиториями

Рассмотрим работу с сервисом GitHub.

Для начала нужно зайти на сайт https://github.com и зарегистрироваться.

- **git clone**

Для того, чтобы клонировать уже существующий репозиторий к себе на компьютер вам необходимо получить адрес этого репо и выполнить команду *git clone*:

git clone https://github.com/ViktoriaArsenteva/GitInstruction.git

- **fork**

На сайте GitHub необходимо найти кнопку Fork и таким образом вы создадите копию репозитория у себя в аккаунте и сможете выгружать туда изменения.

- **git push**

Эта команда позволяет отправить свою версию репозитория во внешний репозиторий. При первом использовании команды *git push* понадобится авторизация.

- **git pull**

Эта команда позволяет скачать все изменения с удаленного репозитория на наш компьютер.

- **pull request**

Это команда для предложения изменений.

Создание приложения по учету товаров

Техническое задание

Итоговой целью разработки приложения по учету товаров в магазине одежды является упорядочивание хранимой продукции, облегчение поиска нужного товара, отслеживание уже имеющихся позиций.

Требования к интерфейсу:

- Понятность
- Минималистичность

Требования к работе приложение:

- Точность и актуальность хранимых данных
- Высокая скорость работы

Необходимые функции программы:

- Добавление новых товаров
- Просмотр всех товаров
- Изменение количества уже добавленной позиции
- Удаление товара
- Проверка уникальности при добавлении нового товара

Проектирование интерфейса приложения

Для разработки макета интерфейса моего проекта я выбрала приложение Balsamiq Wireframes (рис.9), потому что оно имеет все необходимые инструменты и интуитивно понятно для начинающего специалиста.

После изучения всех возможностей данного приложения были созданы несколько шаблонов, того, как будет выглядеть будущий проект. (рис.10)

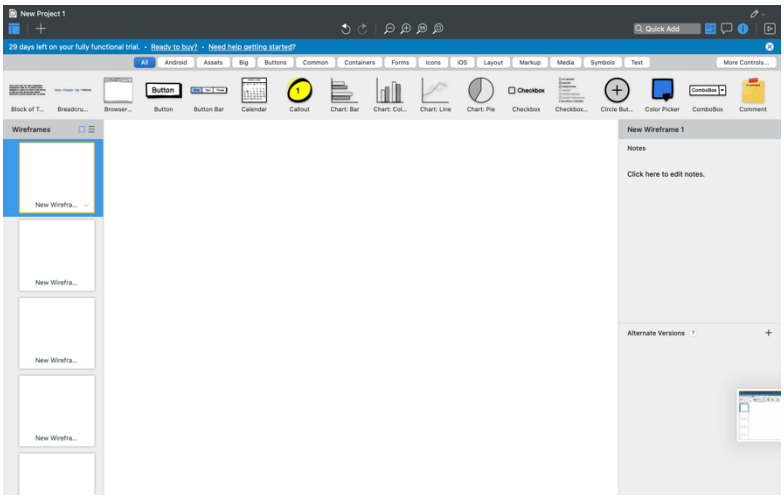


Рис. 9

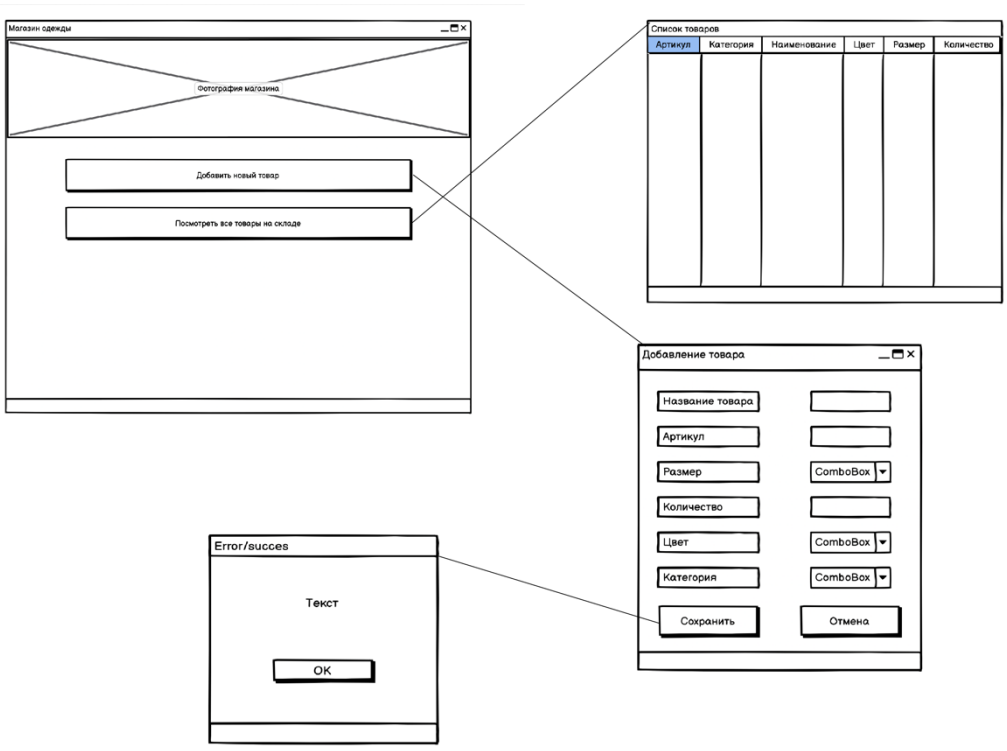


Рис. 10

Установка JavaFX

Для создания графического окружения в моем проекте я выбрала именно JavaFX, потому что это наиболее современный фреймворк. Поскольку до этого с этой библиотекой я не работала, то необходимо было прочитать всю документацию и разобраться в ее работе.

Рассмотрим этапы установки JavaFX через Visual Studio Code:

1. Необходимо зайти на сайт https://openjfx.io/index.htmlю (рис.11) и нажать на кнопку download, после чего мы попадаем на сайт https://gluonhq.com/products/javafx/. Ниже можем увидеть ссылки (рис.12) на всю необходимую документацию по установке и работе с фреймворком. Ей мы и будем пользоваться в дальнейшем.

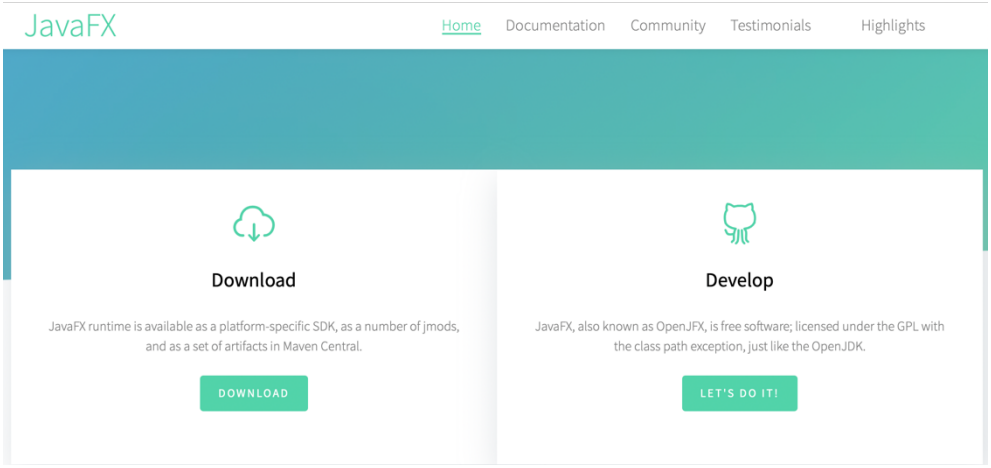


Рис. 11

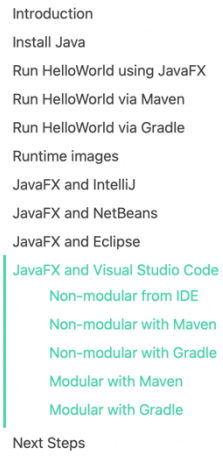


Рис. 12

Здесь нужно выбрать подходящие версию JavaFX, операционную систему, архитектуру и тип. Далее скачиваем нужный файл. (рис.13)

Рис. 13

2. Открываем VS Code и нажимаем Create Java Project (рис.14) и выбираем No build tools (рис.15). После необходимо будет выбрать расположение проекта и его название.

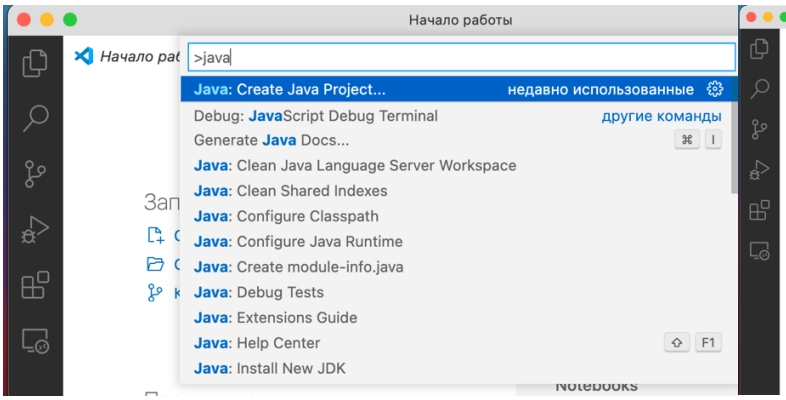


Рис. 14

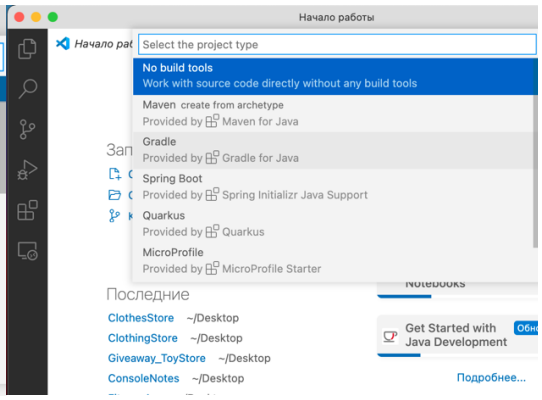


Рис. 15

Далее открываем файл App.java в папке /src для активации расширений java.

3. Импортируем библиотеки из скачанного ранее файла. Для этого необходимо развернуть вкладку Java Projects и нажать на “+” напротив строки Referenced Libraries (рис.16). После нужно найти нашу папку Javafx-sdk (рис.17), в ней выбрать каталог lib и импортировать все файлы с расширением jar

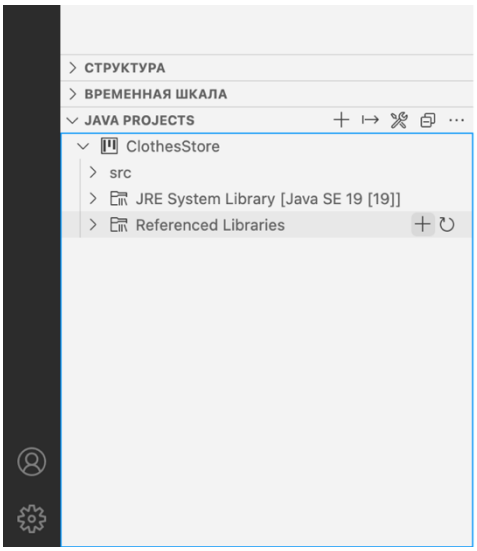


Рис. 16

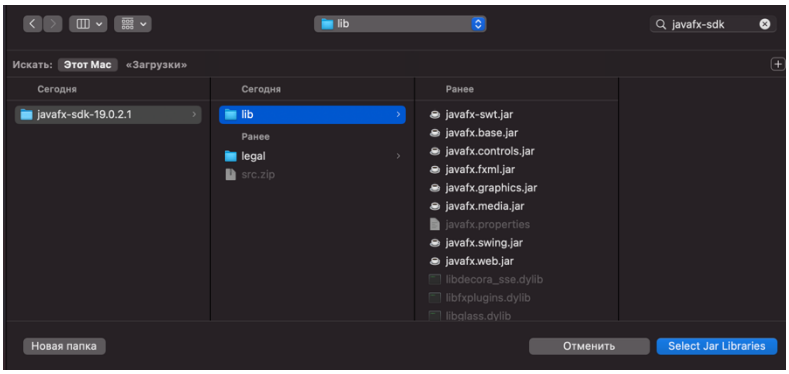


Рис. 17

4. Создаем новую папку в /src и в ней файлы Controller.java; Main.java; Fx.fxml. App.java можно удалить (рис.18).

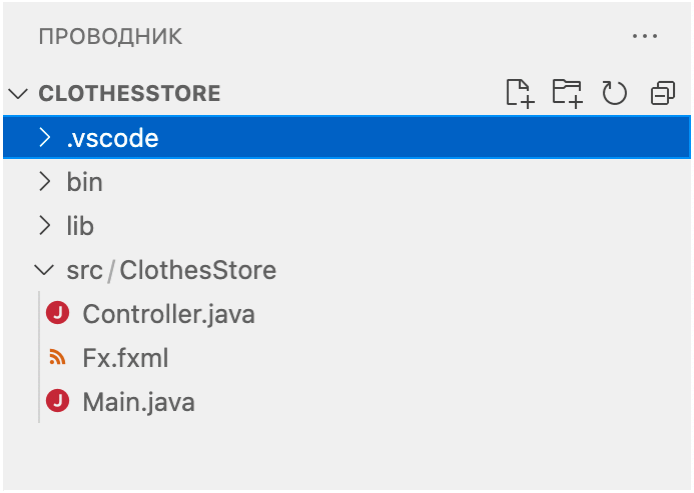


Рис. 18

5. Переходим во вкладку запуск и откладка в меню слева и создаем файл launch.json (рис.19). В этом файле добавляем конфигурацию “vmArgs”, в которой необходимо указать путь к папке lib (рис.20).

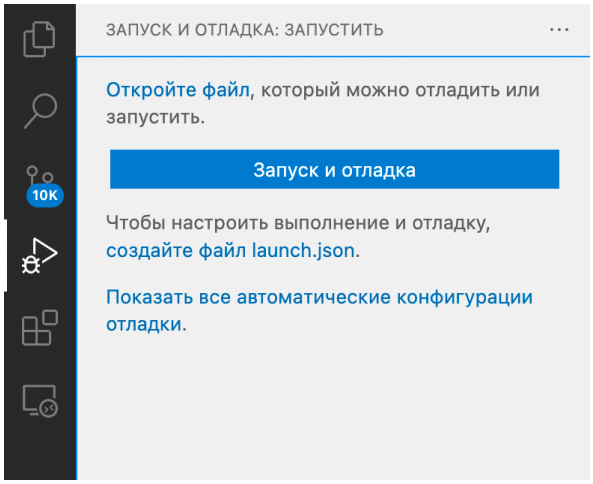


Рис. 19

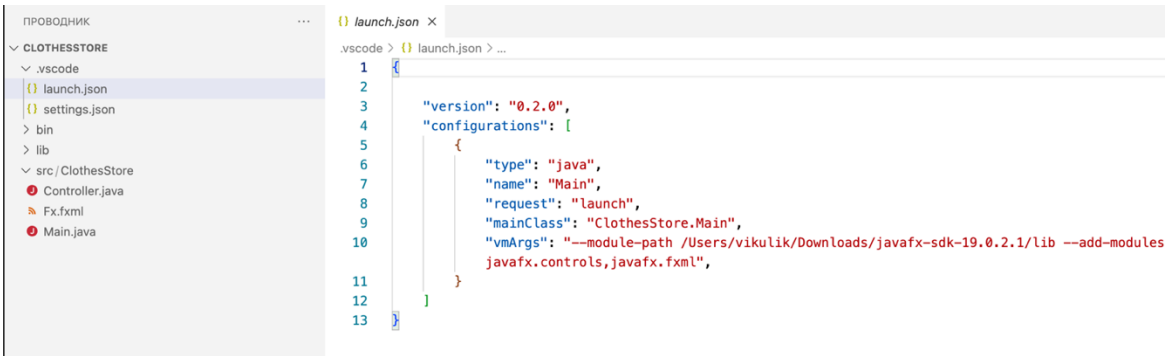


Рис. 20

6. Для проверки работоспособности добавленных библиотек выполним следующий код, который предлагается в документации (рис.21, рис.22, рис. 23).

Main.java

```java
package ClothesStore;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception{
        Parent root = FXMLLoader.load(getClass().getResource(name: "Fx.fxml"));
        primaryStage.setTitle("Hello World");
        primaryStage.setScene(new Scene(root, 400, 300));
        primaryStage.show();
    }


    Run | Debug
    public static void main(String[] args) {
        launch(args);
    }
}
```

Рис. 21

Fx.fxml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.StackPane?>


<StackPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/
1" fx:controller="ClothesStore.Controller">
    <children>
        <Label fx:id="label" text="Label" />
    </children>
</StackPane>
```

Рис. 22

Controller.java

```java
package ClothesStore;

import javafx.fxml.FXML;
import javafx.scene.control.Label;

public class Controller {

    @FXML
    private Label label;

    public void initialize() {
        String javaVersion = System.getProperty(key: "java.version");
        String javafxVersion = System.getProperty(key: "javafx.version");
        label.setText("Hello, JavaFX " + javafxVersion + "\nRunning on Java " + javaVersion + ".");
    }
}
```

Рис. 23

7. Запустим программу и получим следующий результат.

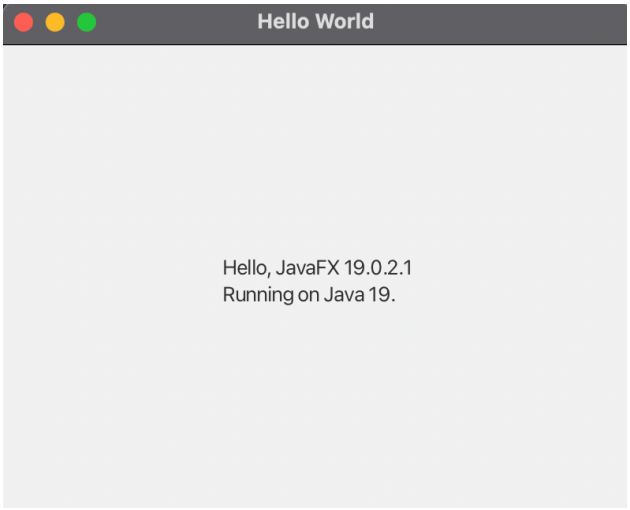


Рис. 24

8. Для облегчения создания графического интерфейса будем использовать приложение Scene Builder, которое можно скачать с официального сайта Gluon.

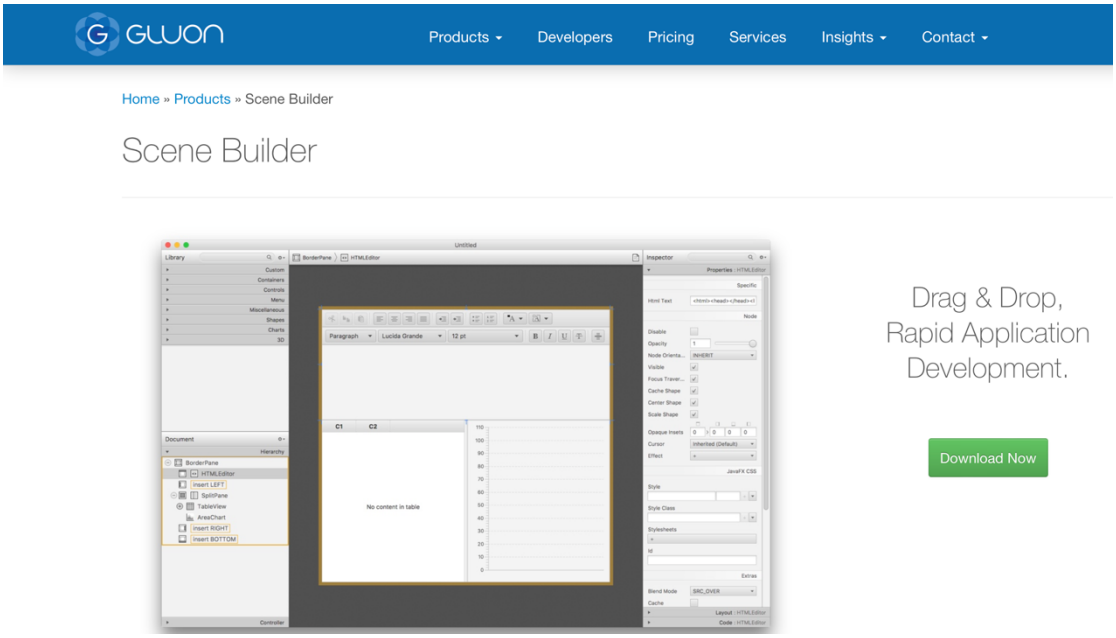


Рис. 25

Создание графической оболочки (SceneBuilder)

Для каждого нового окна нашей программы необходимо создать отдельный файл с расширением fxml, который будет генерироваться с помощью SceneBuilder и контроллера, в котором будет реализовано любое взаимодействие с активным окном.

Далее запускаем SceneBuilder, нажимаем Open project (рис.25) и выбираем нужный файл в проекте. После файлы, с которыми мы ранее уже работали будут отображаться списком в панели слева.

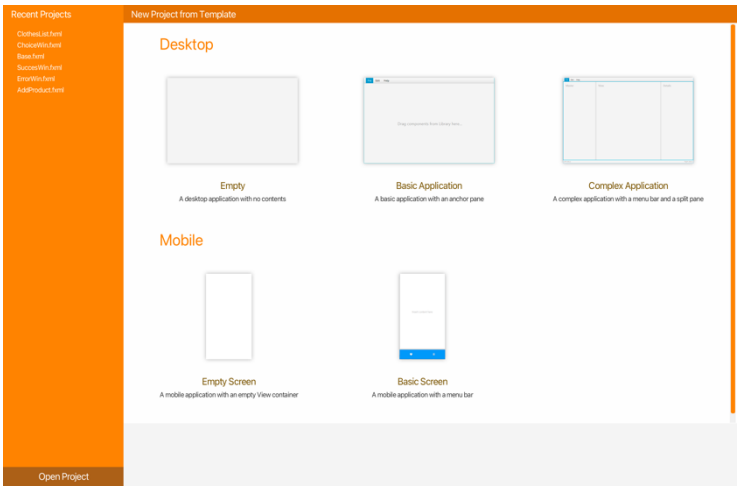


Рис. 26

Рассмотрим основные элементы SceneBuilder, которые будут использованы в проекте (Для добавления любого элемента необходимо перетянуть его из панели слева):

- Pane

Pane – это панель компоновки, на которой впоследствии будут располагаться остальные компоненты. Для установления размера окна можно растянуть панель в редакторе или ввести значения в панели справа в строках Pref Width и Pref Height (рис.27).

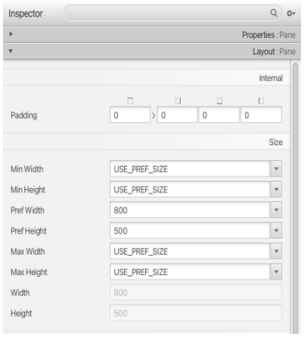


Рис. 27

- Button

  Ключевой возможностью кнопки является способность реагировать на нажатия пользователей и по нажтию выполнять некоторое действие.

Для изменения текста на кнопке, его шрифта размера и т.д. используем вкладку Properties (рис.28).

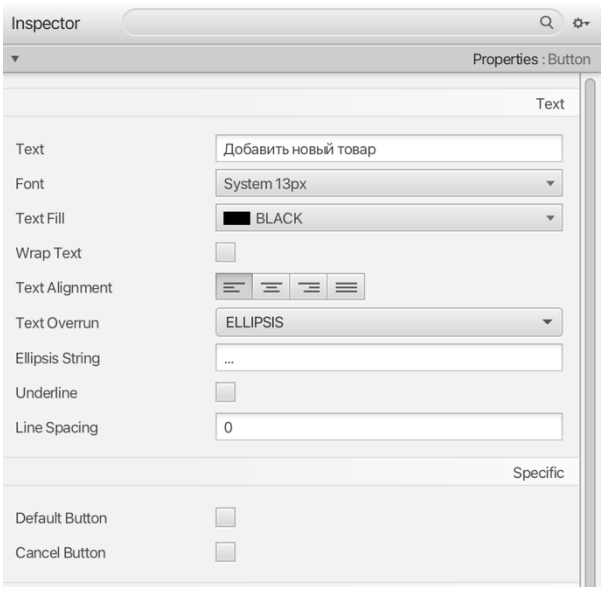


Рис. 28

- Label

Label – это текстовая метка.

- TextField

TextField – это поле для ввода однострочного текста. Можно задать задать начальный текст или оставить пустым. С помощью него мы будем получать данные от пользователя.

- ChoiceBox

ChoiceBox – выпадающий список, в котором напротив выбранного элемента ставится метка.

Для элементов также можно во вкладке Code установить значения fx:id и On Action (рис.29).

Id необходим для того чтобы мы могли обращаться именно к этому элементу (например, изменить текст на кнопке).

On Action привязывает действие с кнопкой (нажатие) к определенному методу в контроллере.

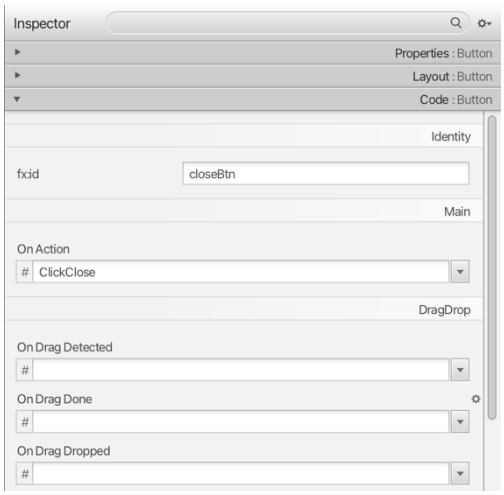


Рис. 29

Для присваивания окну определенного контроллера необходимо зайти на вкладку Controller на панели слева и ввести название файла (рис.30).

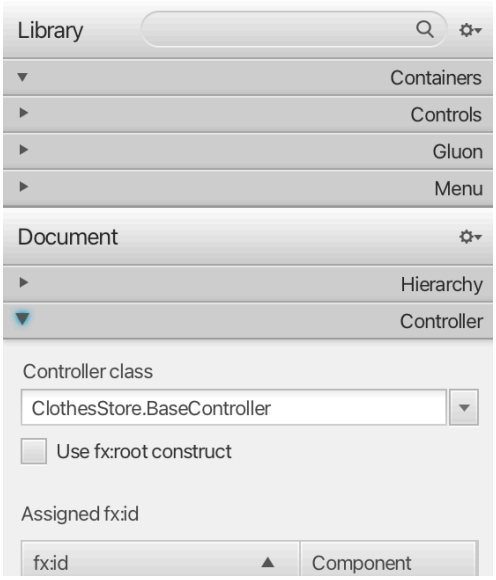


Рис. 30

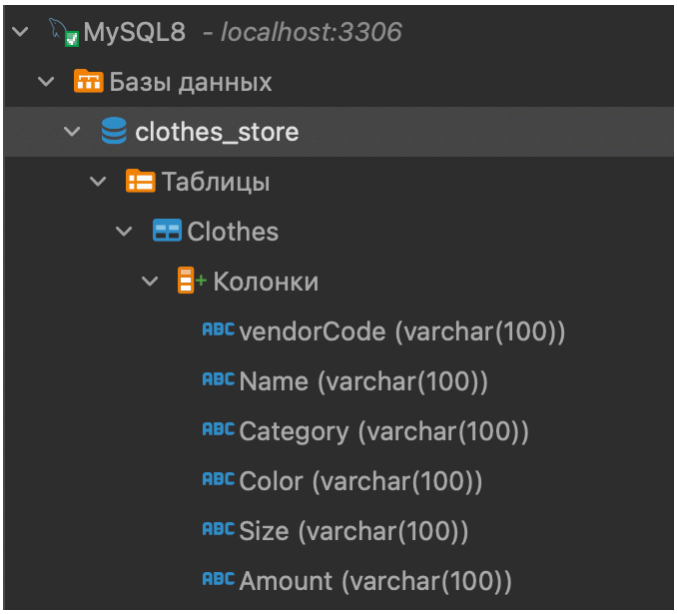
Далее в контроллере объявляем кнопку, чтобы к ней можно было обращаться и создаем метод, который будет вызываться при нажатии на кнопку:

```
@FXML
private Button closeBtn; // Button – название, тип элемента; closeBtn – fx:id

public void ClickClose(ActionEvent actionEvent) {
    Stage stage = (Stage) closeBtn.getScene().getWindow();
    stage.close();
 }

// Метод ClickClose вызывается при нажатии на кнопку closeBtn и закрывает окно, в котором она расположена.
```

Подключение к MySQL серверу

Для подключения к базе данных MySQL необходимо выполнить следующие шаги:

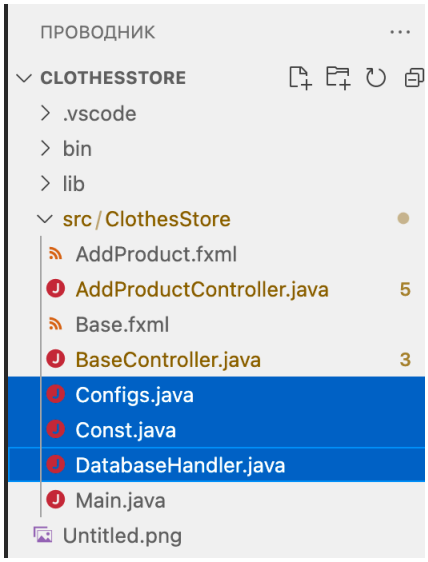
1. Заходим в любую СУБД (в моем случае DBeaver), создаем при помощи редактора новую базу данных *clothes_store* и в ней таблицу *clothes* со следующими столбцами:



Либо можно воспользоваться SQL запросом:

```
CREATE DATABASE `clothes_store`
CREATE TABLE `Clothes` (
        `vendorCode` varchar(100) NOT NULL,
        `Name` varchar(100) NOT NULL,
        `Category` varchar(100) NOT NULL,
        `Color` varchar(100) NOT NULL,
        `Size` varchar(100) NOT NULL,
        `Amount` varchar(100) NOT NULL
       ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
      COLLATE=utf8mb4_0900_ai_ci;
```

2. В редакторе кода создаем несколько новых файлов (Configs.java, Const.java, DatabaseHandler.java)



Configs.java содержит сведения о базе данных, к которой необходимо подключиться.

Configs.java

```
src > ClothesStore > Configs.java > ...
1   package ClothesStore;
2
3   public class Configs {
4       protected String dbHost = "localhost";
5       protected String dbPort = "3306";
6       protected String dbUser = "root";
7       protected String dbPass = "89244895363";
8       protected String dbName = "clothes_store";
9   }
10
```

Const.java хранить константы названий таблиц и колонок (необязательный файл, но во избежание ошибок он необходим, т.к. при изменении значений в базе данных будет удобнее изменить всего один файл в программе)

Const.java

```
src > ClothesStore > Const.java > ...
1   package ClothesStore;
2
3   public class Const {
4       public static final String CLOTHES_TABLE = "Clothes";
5
6       public static final String CLOTHES_VENDORCODE = "vendorCode";
7       public static final String CLOTHES_CATEGORY = "Category";
8       public static final String CLOTHES_NAME = "Name";
9       public static final String CLOTHES_COLOR = "Color";
10      public static final String CLOTHES_SIZE= "Size";
11      public static final String CLOTHES_AMOUNT = "Amount";
12  }
```

DatabaseHandler.java является основным файлом, через который происходит подключение. В нем будут реализованы методы записи и чтения из БД.

Очень важно проверить корректность введенных данных, потому что любая неточность приведет к ошибке.

Если все выполнено верно, то соединение будет установлено

Используемая литература:

1. https://cmsmagazine.ru/journal/items-razrabotka-tz-dlja-it-proekta-chto-stoit-znat/
2. https://www.atlassian.com/ru/agile
3. http://agilemanifesto.org/iso/ru/manifesto.html
4. https://turumburum.ua/blog/dizayn-mobilnykh-prilozheniy-protsess-razrabotki-i-etapy-proektirovaniya/
5. https://webcase.com.ua/blog/etapy-razrabotki-mobilnogo-prilozheniya/#f5
6. https://www.aisol.ru/articles/sozdanie_mobilnyh_prilozheniy_etapy_razrabotki#2_etap
7. https://xakep.ru/2014/09/10/java-gui/
8. Рыженко А. В. Объектно-ориентированное программирование: Учебно-методический комплекс по дисциплине для специальности 010501 – "Прикладная математика и информатика". – 2007.
9. Хабибуллин И. Ш. Java 7 (4-е изд.). – БХВ-Петербург, 2012.
10. https://www.oracle.com
11. https://openjfx.io/index.html
12. https://unity.com/ru/solutions/what-is-version-control
13. https://www.atlassian.com
14.