Общий подход и функционал приложения:

Файл находится в PullRequest

**Сервер**

1. При старте сервера ищется файл с настройками srv.properties создается объект класса SrvProperties(singleton), в котором находятся настройки подключения к базе данных
2. Далее стартует liquibase (система контроля версий базы данных). Настройки liquibase реализованы через java код, без xml файла.
3. Настройки log4j реализованы через java код, без отдельных файлов настроек. В плане дописать хранение логов в базе данных
4. Проверяется подключение к базе данных, если подключение невозможно приложение завершается.
5. Выдача для нужд приложения connection реализовано через класс DBConnection, который реализован через потокобезопасный паттерн singleton (хотя нужно было сделать, наверное, factory). Статья на хабре про синглтон «https://habr.com/ru/post/129494/ Правильный Singleton в Java»
6. При аутентификации пароль передается в зашифрованном виде используя алгоритм «PBKDF2WithHmacSHA1» и соль. Использовал статью <https://www.baeldung.com/java-password-hashing>
7. После успешной аутентификации хэндл удаляется из pipline и добавляется хэндл MessageHandle.
8. При синхронизации (односторонней) файлы на сервер приходят по частям. Размер одного пакета определяется в настройках клиента. Сначала приходит сообщение заголовок, в котором указан размер файла, на какое количество частей он будет разбит, и его относительный путь на клиенте (относительно корневой директории хранилища). При получении пакетов сначала все пишется во временный файл и только в случае успешного получения последнего пакета и совпадении размера файла исходного и файла результата переименовываем. (надо было использовать не размер, а контрольную сумму CRC32 из пакета commons-io apache)
9. Хранение файлов реализовано без директорий. Имя файла = Логин пользователя и относительный путь хэшируютя алгоритмом MD5 и к хэшу в конце добавляется логин пользователя. При принятии решения о хранении без директорий исходил из следующего, что запросы к БД выполняются быстрее, чем обход всех директорий при вложенности более 25 директорий и больше. (Замерял из интереса через LocalTime). Вероятность одинакового MD5 хэша у одного пользователя при разных файлах крайне мала.(но если будет коллизия затрет существующий файл).

**Клиент**

1. При разработке использовал видео «GeekBrains: "Разработка простого файлового менеджера на Java»
2. При старте приложения ищется файл с настройками client.properties создается объект класса ClientProperties (singleton), в котором находятся путь до корневой директории хранилища, логин и пароль для подключения к серверу. При отсутствии файла с настройками (при первом старте) сначала стартует окно с просьбой указать настройки, который тут же сохраняются в файл.
3. JavaFX использует по умолчанию log4j, а все остальное логгируется через slf4j и log4j. Добавил к логгеру JavaFX slf4j.
4. При синхронизации решение об отправке файла на сервер принимает клиент исходя из того, что файла есть или нет на сервере и времени последней модификации (где файл свежее)
5. Реализовано drag on drop – перетаскивание файлов или директорий на клиента.
6. Режим вставки и копирования через системный буфер. (Class Clipboard javafx), также как группы файлов, так и директорий
7. В зависимости от расширения файла или если это директория подставляется картинка в табличной форме. Иконки честно взяты из интернета, автор неизвестен.
8. Вывод сообщений об успешной аутентификации и синхронизации на форме клиента.
9. При двойном клике по файлу файл открывается в приложении, ассоциированном с данным файлом в операционной системе (word, excel, блокнот и т д)
10. При навигации нельзя подняться выше корневой директории хранилища
11. Иконки на кнопки взяты из Bootstrap v5 (формат svg перегнан в png, так как не разобрался как JavaFX работает с svg). Подстановку иконок сделал через css файлы.

**При разработке возникли следующие затруднения:**

1. ChunkedFile не совместим с нахождением в pipline ObjectDecoder ObjectEncoder.
2. FileChanel nio использует ByteBuffer (не сериализуемый). Даже если привести к byte[] запихать в сообщение, потом обратно на сервере к ByteBuffer и использовать FileChanel при записи в файл, при нахождении в pipline ObjectDecoder ObjectEncoder приходил полный бред.
3. Всё-таки у netty мало документации с примерами. Приходилось копаться иногда в исходном коде.