WPF Cloud Storage Client — Архитектура и первые шаги

Документ покрывает: архитектуру клиента, ожидаемые API-контракты, что нужно от серверной части сейчас, план разработки и список задач для старта.

---

1. Можно ли начать без SignalR?

Да — **SignalR можно добавить позже**. Пока что реализуем классический HTTP-клиент (HttpClient/Refit). SignalR подключим как отдельный сервис и включим в ViewModel, когда сервер добавит hub.

---

2. Рекомендуемая архитектура клиента (высокоуровнево)

- MVVM (CommunityToolkit.Mvvm)
- Слои:
- **UI (Views)** — WPF XAML
- **ViewModels** — CommunityToolkit.ObservableObject / RelayCommand
- **Services** — сетевые клиенты, аутентификация, хранилище токена, файловый менеджер, диалоговые сервисы
- **Models / DTOs** — типы, соответствующие API (FileDto, FolderDto, AuthTokens и т.д.)
- **Repositories (опционально)** — если нужно кеширование локально
- **DI / Composition Root** — Microsoft.Extensions.DependencyInjection

---

3. Базовая структура проекта (предложение)

```
/ClientSolution
  /CloudClient (WPF)
    /Views
    /ViewModels
    /Services
    /Models
    /Helpers
    App.xaml
  /CloudClient.Core (class lib)
    /Interfaces
    /DTOs
    /Services
  /CloudClient.Tests
  README.md
```

---

4. Важные NuGet-пакеты

- CommunityToolkit.Mvvm
- Microsoft.Extensions.DependencyInjection
- Microsoft.Extensions.Http
- Refit (опционально, если хотите декларативные HTTP-клиенты)
- Flurl.Http (альтернатива)
- Newtonsoft.Json или System.Text.Json
- SQLite / EF Core (если планируете локальный кэш)
- Polcy (resilience/retry)

---

5. Что нужно знать о сервере сейчас (чтобы писать клиент без сервера)

Чтобы начать, минимальный набор от сервера или дизайн-решения, которые нужно утвердить:

1. **OpenAPI/Swagger** — идеальный вариант: URL с описанием API. Позволит автоматом генерировать DTO и клиенты.
2. **Эндпоинты (минимум):**
3. `POST /api/auth/login` — возвращает JWT (и refresh токен?)
4. `POST /api/auth/register` — регистрация
5. `GET /api/files` — список файлов пользователя (пагинация?)
6. `POST /api/files/upload` — multipart/form-data для загрузки
7. `GET /api/files/{id}` — скачивание
8. `DELETE /api/files/{id}` — удаление
9. (опц.) `GET /api/folders`, `POST /api/folders` — для папок
10. (опц.) `POST /api/files/{id}/share` — создание публичной ссылки
11. **Аутентификация:** формат JWT (claims), куда передавать (Authorization: Bearer {token}), срок жизни токена, есть ли refresh token. CORS настройки (для dev). Максимальный размер загрузки.
12. **Коды ответов и ошибки:** стандартные HTTP 200/201/400/401/403/404/413 (payload too large) — полезно знать политику ошибок.
13. **Ограничения и политики:** лимит по объёму на пользователя, ограничения по типу файлов, одновременные загрузки.
14. **Файловый download:** отдаётся напрямую как `application/octet-stream` или через pre-signed URL?
15. **SignalR (если позже):** hub route и схема сообщений (например: `FileUploaded` с payload `{fileId, fileName, uploaderId}`)

Если у сервера будет Swagger — можно почти не договариваться заранее и начать сразу.

---

6. Что я могу сделать без сервера (practical)

1. **Скелет UI** — окна: логин/регистрация, главный экран со списком файлов, диалог загрузки.
2. **Мок-API** — сделать интерфейсы служб и локальный мок, который возвращает тестовые DTO. Позволит строить ViewModels и UI.

3. **Аутентификация на клиенте** — UI для ввода логина/пароля, хранение токена в защищённом хранилище (ProtectedData / DPAPI или Windows Credential Vault).
4. **Загрузка файлов UI** — прогресс-бар, выбор файлов, отмена (токен отмены).
5. **Unit-тесты для ViewModels** — с моками сервисов.

---

7. Ожидаемые интерфейсы сервисов (пример)

```
public interface IAuthService
{
    Task<AuthResult> LoginAsync(string email, string password);
    Task LogoutAsync();
    string? AccessToken { get; }
}

public interface IFileService
{
    Task<IEnumerable<FileDto>> GetFilesAsync(CancellationToken ct);
    Task<UploadResult> UploadFileAsync(Stream content, string filename,
IProgress<long> progress, CancellationToken ct);
    Task<Stream> DownloadFileAsync(Guid fileId, CancellationToken ct);
    Task DeleteFileAsync(Guid fileId, CancellationToken ct);
}
```

---

8. Как хранить JWT на клиенте

- Для desktop-приложения: DPAPI (ProtectedData) или Windows Credential Manager — не храните токен в plaintext. При старте приложения десериализуйте токен и проверьте срок действия.
- Если есть refresh token — безопасно хранить его отдельно и реализовать автоматический refresh в `HttpClient` через DelegatingHandler.

---

9. UI/UX рекомендации (коротко)

- Список файлов — иконка, имя, размер, дата, кнопки «скачать», «удалить», «поделиться»
- Drag & drop для загрузки
- Отображение прогресса по файлу и общий прогресс
- Подтверждение при удалении
- Уведомления внутри приложения (Toast) — потом можно заменить или дополнить SignalR

---

10. План работ на ближайшие спринты (пример)

Sprint 0 — подготовка (1–2 дня) * Настройка решения, DI, базовая структура * Реализация IAuthService и IFileService интерфейсов * Моки для сервисов

Sprint 1 — базовый функционал (3–5 дней) * Login/Register UI + мок аутентификации * Экран списка файлов (mock data) * Загрузка файла (UI + mock) с прогрессом * Скачивание/удаление (mock)

Sprint 2 — подключение реального сервера * Подключение к реальному API/Swagger * Обработка реальных ошибок, тестирование CORS, размер загрузки * Работа с JWT/refresh

Sprint 3 — доп. функции * Папки / структура * Поделиться ссылкой * Поддержка SignalR

---

11. Что мне нужно от тебя прямо сейчас (коротко)

1. Подтверди приоритеты фич (обязательное vs опциональное). Например: `Login, Upload, List, Download, Delete` — обязательно; `Folders, Share, Quota` — опционально.
2. Реши, нужно ли локальное кэширование / офлайн-режим (опционально).
3. Хочешь ли thunk/рефакторинг с Refit для автоматической генерации клиентов или ручной `HttpClient`?

---

12. Checklist — стартовые файлы/таски (можно прямо начать)

- [ ] Создать решение и проекты
- [ ] Добавить NuGet
- [ ] Настроить DI
- [ ] Создать модели (AuthResult, FileDto)
- [ ] Реализовать MockAuthService и MockFileService
- [ ] Сделать LoginView + MainView skeleton
- [ ] Протестировать локально с моками

Если хочешь, могу:
сгенерировать начальный проект/скелет файлов (csproj + классы) и дать пошаговую инструкцию по сборке;
или прислать готовые примеры кода для `IAuthService` и `IFileService` (реализация мок + real HttpClient handler).

Скажи, как предпочитаешь — **я готов начать с генерации скелета проекта** или сначала сделать подробный список ожидаемых API-эндпоинтов (OpenAPI-стайл).