Технический дизайн серверной архитектуры

Проект: FairSplit Mobile App
Платформа: Firebase (Firestore, Auth, Functions)
Версия документа: 1.0
Архитектурный паттерн: Offline-First / Cloud-Sync

1. Обзор архитектуры

Система строится на базе **Cloud Firestore** как основного хранилища данных. Ключевая особенность — поддержка работы в режиме Offline-First с синхронизацией по стратегии **Total Last Write Wins (LWW)**.

Компоненты Firebase

1. **Authentication:** Гибридная схема (Анонимный вход -> Конвертация в постоянный аккаунт).
2. **Cloud Firestore:** Документо-ориентированная БД. Используется денормализация для оптимизации чтения.
3. **Cloud Functions:** Выполнение серверной бизнес-логики (генерация инвайтов, аудит изменений).
4. **Security Rules:** Обеспечение доступа на основе принадлежности к группе и логики "Smart Linking" (слияние с призраками).

---

2. Схема базы данных (Firestore Schema)

База данных состоит из двух корневых коллекций: users и groups. Все финансовые операции инкапсулированы внутри документов групп для обеспечения изоляции данных.

2.1. Коллекция users (Профили)

Хранит глобальные данные пользователя и настройки слияния (Smart Linking).

- **Path:** /users/{userId}
- **Read/Write:** Владелец может читать/писать свой документ.

JSON

```
{
  "uid": "String (Auth ID)",
  "email": "String? (Optional)",
  "display_name": "String",
  "photo_url": "String? (URL)",
  "is_anonymous": "Boolean",

  // Ключевое поле для Smart Linking.
  // Содержит список UUID призраков, история которых теперь принадлежит этому пользователю.
  "linked_ghost_ids": [
    "ghost_uuid_1",
    "ghost_uuid_2"
  ],

  "fcm_token": "String? (Для Push-уведомлений)",
  "created_at": "Timestamp",
  "updated_at": "Timestamp"
}
```

2.2. Коллекция groups (Группы)

Корневой агрегат. Содержит метаданные и список участников (реальных и виртуальных).

- **Path:** /groups/{groupId}
- **Read:** Члены массива members.
- **Write:** Члены массива members (частичное обновление).

JSON

```
{
  "id": "String (UUID)",
  "name": "String",
  "currency": "String (ISO 4217, Immutable)",
  "owner_id": "String (User UID)",
  "invite_code": "String? (Generated by Server)",

  // Массив UID реальных пользователей для Security Rules
  "members": [
    "user_uid_1",
    "user_uid_2"
  ],
```

```
  // Карта призраков (виртуальных участников).
  // Хранится внутри документа группы для быстрой отрисовки UI.
  "ghosts": {
    "ghost_uuid_1": {
      "name": "Alex (Offline)",
      "is_merged": true,
      "merged_with_uid": "user_uid_2"
    },
    "ghost_uuid_2": {
      "name": "Maria",
      "is_merged": false
    }
  },

  "created_at": "Timestamp",
  "updated_at": "Timestamp" // Поле для LWW конфликтов
}
```

2.3. Подколлекция expenses (Траты)

Основная операционная сущность.

- **Path:** /groups/{groupId}/expenses/{expenseId}
- **Write:** Создатель записи ИЛИ пользователь, усыновивший призрака-создателя.

JSON

```
{
  "id": "String (UUID)",
  "description": "String",
  "amount": "Number (Double)",
  "currency": "String (Must match Group currency)",
  "date": "Timestamp",

  // Ссылка на создателя. Может быть UID юзера или UUID призрака.
  // НЕ меняется при Smart Linking (историческая правда).
  "creator_id": "String",

  // Кто платил. Map<ParticipantID, Amount>
  "payers": {
```

```
    "user_uid_1": 100.00,
    "ghost_uuid_1": 50.00
  },

  // За кого платили. Map<ParticipantID, Amount>
  "splits": {
    "user_uid_1": 75.00,
    "ghost_uuid_1": 75.00
  },

  "category": "String (Optional)",
  "is_deleted": "Boolean (Soft Delete)",
  "created_at": "Timestamp",
  "updated_at": "Timestamp" // Критично для синхронизации
}
```

2.4. Подколлекция settlements (Расчеты)

Транзакции погашения долгов.

- **Path:** /groups/{groupId}/settlements/{settlementId}

JSON

```
{
  "id": "String (UUID)",
  "from_id": "String (ParticipantID)",
  "to_id": "String (ParticipantID)",
  "amount": "Number",
  "currency": "String",
  "status": "String (PENDING | CONFIRMED | REJECTED)",
  "created_at": "Timestamp",
  "updated_at": "Timestamp"
}
```

2.5. Подколлекция history (Audit Log)

Системная коллекция для аудита. Заполняется **только** Cloud Functions. Клиент имеет доступ только на чтение.

- **Path:** /groups/{groupId}/expenses/{expenseId}/history/{historyId}

JSON

```
{
  "action": "String (CREATE | UPDATE | DELETE)",
  "actor_uid": "String",
  "diff_payload": "Map (OldValue -> NewValue)",
  "timestamp": "Timestamp"
}
```

---

3. Серверная логика (Cloud Functions)

Для обеспечения безопасности и целостности данных используются триггеры и callable-функции.

3.1. Управление приглашениями (Callable)

- **createInviteCode**:
  - **Input:** groupId.
  - **Logic:** Проверяет права администратора -> Генерирует короткий 6-значный код -> Сохраняет маппинг code -> groupId в системную коллекцию -> Возвращает код клиенту.
- **joinByInviteCode**:
  - **Input:** code.
  - **Logic:** Находит groupId -> Добавляет request.auth.uid в массив members группы -> Возвращает данные группы клиенту.

3.2. Аудит изменений (Triggers)

- **onExpenseWrite** (Firestore Trigger):
  - Срабатывает на onCreate, onUpdate, onDelete в коллекции expenses.
  - Сравнивает before.data и after.data.
  - Формирует документ Diff.
  - Записывает его в подколлекцию history с правами администратора.
  - Гарантирует, что историю невозможно подделать с клиента.

---

4. Правила безопасности (Security Rules Strategy)

Логика правил обеспечивает выполнение требований ТЗ по доступу к историческим данным через Smart Linking.

Концепция проверки прав

Пользователь имеет право редактировать документ (update, delete), если выполняется одно из условий:

1. **Прямое авторство:** resource.data.creator_id == request.auth.uid
2. **Усыновление (Smart Linking):** ID создателя (который является призраком) содержится в массиве linked_ghost_ids профиля текущего пользователя.

Псевдокод правил (Logic Flow)

JavaScript

```
// Функция проверки членства в группе
function isMember(groupId) {
  return request.auth.uid in
get(/databases/$(database)/documents/groups/$(groupId)).data.members;
}

// Функция проверки прав на редактирование траты
function canEditExpense(resourceData) {
  let isDirectCreator = resourceData.creator_id == request.auth.uid;

  // Дополнительный запрос для Smart Linking (стоит 1 чтение)
  let userProfile = get(/databases/$(database)/documents/users/$(request.auth.uid));
  let isGhostAdopter = resourceData.creator_id in userProfile.data.linked_ghost_ids;

  return isDirectCreator || isGhostAdopter;
}

// Правила для Expenses
match /groups/{groupId}/expenses/{expenseId} {
  allow read: if isMember(groupId);
  allow create: if isMember(groupId) && request.resource.data.creator_id == request.auth.uid;
  allow update: if isMember(groupId) && canEditExpense(resource.data);
  allow delete: if isMember(groupId) && canEditExpense(resource.data);
}
```

---

5. Аутентификация и Конвертация

Сценарий: Первый вход (Anonymous)

1. Клиент проверяет наличие сети.
2. Вызов firebase.auth().signInAnonymously().
3. Создание документа в /users/{uid} с флагом is_anonymous: true.

Сценарий: Привязка (Link)

1. Пользователь выбирает "Войти через Google".
2. Клиент вызывает currentUser.linkWithCredential(credential).
3. При успехе Firebase обновляет Auth Record (UID сохраняется).
4. Клиент обновляет документ /users/{uid}, устанавливая email, photo_url и is_anonymous: false.
5. **Важно:** Данные групп и трат мигрировать не нужно, так как UID остается неизменным.

---

6. Индексы и Оптимизация

Для поддержки сортировок и фильтрации в Offline-First режиме и запросах потребуются следующие композитные индексы (Composite Indexes) в firestore.indexes.json:

1. **Expenses Listing:**
   - Collection ID: expenses
   - Fields: group_id (Asc) + date (Desc)
   - *Назначение:* Отображение списка трат в группе по хронологии.
2. **History Listing:**
   - Collection ID: history
   - Fields: timestamp (Desc)
   - *Назначение:* Отображение хронологии изменений конкретной траты.

Резюме для разработки

Данная структура полностью покрывает требования Offline-First (через локальный кэш SDK), поддерживает конфликты через LWW (поле updated_at) и реализует сложную логику слияния пользователей через массив linked_ghost_ids без необходимости массовой перезаписи исторических данных.