



# FLUXOGRAMA SIMPLIFICADO

Sistema de Controle Financeiro Finanza

IFSUL - Campus Venâncio Aires



## FLUXOGRAMA SIMPLIFICADO -

## Sistema Finanza



### Visão Geral do Sistema

#### SISTEMA FINANZA

App Mobile (Android) ↔ Servidor (Java) ↔ Desktop Admin |  
|  
- Usuários comuns      - MySQL Database      - Administradores  
- Gestão financeira      - Porta 12345      - Gerenciar  
- Funciona offline      - Sincronização      usuários



## MOBILE (Android)

### Estrutura de Arquivos

```
app/src/main/java/com/example/finanza/
    |
    +-- ui/                      # TELAS (8 arquivos)
        +-- LoginActivity      → Tela de login
        +-- RegisterActivity   → Cadastro de usuário
        +-- MenuActivity        → Dashboard principal
        +-- AccountsActivity    → Gerenciar contas
        +-- CategoriaActivity   → Gerenciar categorias
        +-- MovementsActivity    → Lançamentos financeiros
        +-- ProfileActivity      → Perfil do usuário
        +-- SettingsActivity     → Configurações
    |
    +-- model/                   # DADOS (4 arquivos)
        +-- Usuario            → id, nome, email, senha
        +-- Conta               → id, nome, tipo, saldo
        +-- Categoria           → id, nome, tipo (receita/despesa)
        +-- Lancamento          → id, valor, data, descrição
    |
    +-- db/                      # BANCO LOCAL SQLite (5 arquivos)
        +-- AppDatabase         → Configuração do banco
        +-- UsuarioDao          → Buscar/Salvar usuários
        +-- ContaDao             → Buscar/Salvar contas
        +-- CategoriaDao         → Buscar/Salvar categorias
        +-- LancamentoDao       → Buscar/Salvar lançamentos
    |
    +-- network/                 # COMUNICAÇÃO (6 arquivos)
        +-- ServerClient        → Conecta ao servidor (porta 12345)
        +-- Protocol             → Define comandos (LOGIN, ADD_CONTA, etc)
        +-- AuthManager           → Gerencia login/sessão
        +-- SyncService           → Sincroniza dados
        +-- EnhancedSyncService   → Sincronização avançada
        +-- ConflictResolutionManager → Resolve conflitos
```

## | FLUXO 1: Login no Mobile

USUÁRIO ABRE O APP

↓

LoginActivity.java

- Exibe tela com campos: email, senha, botão "Entrar"
- Layout: activity\_login.xml

↓

USUÁRIO DIGITA: joao@gmail.com / senha123

USUÁRIO CLICA: Botão "Entrar"

↓

LoginActivity.realizarLogin()

- Valida campos não vazios
- Chama → AuthManager.login(email, senha)

↓

AuthManager.java

- Criptografa senha (SHA-256)
- Monta comando: "LOGIN|joao@gmail.com|hash\_senha"
- Chama → ServerClient.sendCommand(comando)

↓

ServerClient.java

- Abre conexão TCP com servidor (porta 12345)
- Envia: "LOGIN|joao@gmail.com|hash"
- Aguarda resposta do servidor

↓

REDE/INTERNET

↓

SERVIDOR (ClientHandler.java)

- Recebe comando "LOGIN|..."
- Chama → UsuarioDAO.buscarPorEmail(email)

↓

UsuarioDAO.java (Servidor)

- Executa SQL: SELECT \* FROM usuario WHERE email = 'joao@...'
- Busca no MySQL
- Compara senha hash
- Retorna: "OK|{id:1,nome:João,email:joao@gmail.com}"

↓

REDE/INTERNET

↓

ServerClient.java (Mobile)

- Recebe resposta: "OK|{dados}"

- Retorna para AuthManager

↓

AuthManager.java

- Salva dados no banco local (SQLite)
- Chama → UsuarioDao.inserir(usuario)
- Salva sessão (SharedPreferences)

↓

LoginActivity.java

- Exibe Toast: "Login realizado com sucesso!"
- Abre MenuActivity (dashboard)
- Fecha LoginActivity

↓

USUÁRIO VÊ: Dashboard com saldo, receitas, despesas

### Tempo total: 1-2 segundos

**Arquivos envolvidos:** - Mobile: LoginActivity → AuthManager → ServerClient → UsuarioDao → MenuActivity - Servidor: ClientHandler → UsuarioDAO → MySQL

---



## FLUXO 2: Adicionar uma Despesa

USUÁRIO ESTÁ EM: MovementsActivity (lista de lançamentos)  
USUÁRIO CLICA: Botão "+" (FloatingActionButton)

↓

MovementsActivity.java  
• Abre dialog (dialog\_add\_transaction.xml)  
• Carrega spinners:  
- Contas: ContaDao.listarPorUsuario()  
- Categorias: CategoriaDao.listarPorTipo("despesa")

↓

USUÁRIO VÊ: Formulário  
• Descrição: [Almoço]  
• Valor: [50.00]  
• Data: [15/01/2024]  
• Tipo: (•) Receita (○) Despesa  
• Conta: [Nubank ▼]  
• Categoria: [Alimentação ▼]  
• Botões: [Cancelar] [Salvar]

↓

USUÁRIO CLICA: Botão "Salvar"

↓

MovementsActivity.onSaveTransaction()  
• Valida campos (não vazios, valor > 0)  
• Cria objeto Lancamento:  
- uuid = UUID aleatório  
- descricao = "Almoço"  
- valor = 50.00  
- tipo = "despesa"  
- contaId = 3 (Nubank)  
- categoriaId = 5 (Alimentação)  
- syncStatus = 2 (NEEDS\_SYNC)

↓

DataIntegrityValidator.validarLancamento()  
• Valida valor positivo ✓  
• Valida conta existe: ContaDao.buscarPorId(3) ✓  
• Valida categoria existe: CategoriaDao.buscarPorId(5) ✓  
• Verifica duplicatas recentes: LancamentoDao.buscarSimilares()

↓

LancamentoDao.inserirSeguro()  
• INSERT INTO lancamento (...) VALUES (...)  
• Banco SQLite local  
• Retorna ID: 123

↓

```
MovementsActivity.atualizarSaldoConta()
• Busca conta: ContaDao.buscarPorId(3)
• Calcula: saldoAtual = 500 - 50 = 450
• Atualiza: ContaDao.atualizar(conta)
• UPDATE conta SET saldoAtual = 450 WHERE id = 3
```

↓

```
EnhancedSyncService.syncLancamento()
• Formata dados para envio
• Comando: "ADD_MOVIMENTACAO_ENHANCED|uuid|50.00|..."
• Chama → ServerClient.sendCommand(comando)
```

↓

REDE/INTERNET

↓

```
SERVIDOR (ClientHandler.processarAddMovimentacao)
• Parse dos dados recebidos
• Cria objeto Movimentacao
• Chama → MovimentacaoDAO.inserir(movimentacao)
```

↓

```
MovimentacaoDAO.java (Servidor)
• INSERT INTO movimentacao (...) VALUES (... )
• Banco MySQL
• UPDATE conta SET saldo_inicial = 450 WHERE id = 3
• Retorna ID gerado: 542
```

↓

REDE/INTERNET

↓

```
EnhancedSyncService (Mobile)
• Recebe: "OK|542"
• Atualiza: LancamentoDao.marcarComoSincronizado(123)
• syncStatus = 1 (SYNCHED)
```

↓

```
MovementsActivity.recarregarLancamentos()
• Busca: LancamentoDao.listarAtivosPorUsuario(usuarioId)
• Atualiza RecyclerView
• Exibe Toast: "Lançamento salvo com sucesso!"
```

↓

```
USUÁRIO VÊ: Nova despesa aparece na lista
• "Almoço - R$ 50,00 - 15/01/2024"
• Saldo da conta Nubank: R$ 450,00 (era R$ 500)
```

**Tempo total: 1-3 segundos**

**Banco de dados modificado:** - SQLite (Mobile): 2 tabelas (lancamento, conta) - MySQL (Servidor): 2 tabelas (movimentacao, conta)

---

## FLUXO 3: Dashboard (MenuActivity)

```
MenuActivity.onCreate()
• Recebe usuarioId do Intent
• Inicializa banco: AppDatabase.getInstance()
```

↓

```
BUSCAR CONTAS
• ContaDao.listarPorUsuario(usuarioId)
• SELECT * FROM conta WHERE usuarioId = 1
• Retorna lista: [Nubank, Caixa, Itaú]
```

↓

```
CALCULAR SALDO TOTAL
• Loop nas contas
• saldoTotal = Nubank(450) + Caixa(1000) + Itaú(2000) = 3450
```

↓

```
BUSCAR RECEITAS
• LancamentoDao.somaPorTipo("receita", usuarioId)
• SELECT SUM(valor) FROM lancamento WHERE tipo='receita'
• Retorna: 5000.00
```

↓

```
BUSCAR DESPESAS
• LancamentoDao.somaPorTipo("despesa", usuarioId)
• SELECT SUM(valor) FROM lancamento WHERE tipo='despesa'
• Retorna: 3500.00
```

↓

```
ATUALIZAR INTERFACE
• tvSaldoTotal.setText("R$ 3.450,00")
• tvTotalReceitas.setText("R$ 5.000,00") (verde)
• tvTotalDespesas.setText("R$ 3.500,00") (vermelho)
• tvSaldoMensal.setText("R$ 1.500,00")
```

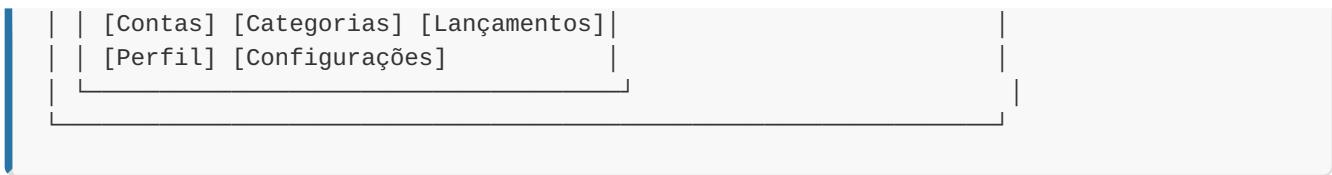
↓

```
INICIAR SINCRONIZAÇÃO (background)
• new Thread() → SyncService.startSync(usuarioId)
• Sincroniza dados pendentes com servidor
• Busca atualizações do servidor
```

↓

```
USUÁRIO VÊ: Dashboard
```

	Saldo Total: R\$ 3.450,00
	Receitas: R\$ 5.000,00
	Despesas: R\$ 3.500,00
	Saldo Mensal: R\$ 1.500,00



**Queries executadas:** 1. SELECT \* FROM conta WHERE usuarioid = 1 2. SELECT SUM(valor) FROM lancamento WHERE tipo='receita' AND usuarioid=1 3. SELECT SUM(valor) FROM lancamento WHERE tipo='despesa' AND usuarioid=1



## SERVIDOR (Java)

### Estrutura de Arquivos

```
DESKTOP VERSION/ServidorFinanza/src/
├── MainServidor.java          # Inicia o servidor
├── server/
│   ├── FinanzaServer.java     # Servidor TCP (porta 12345)
│   ├── ClientHandler.java     # Processa cada cliente
│   └── Protocol.java          # Define comandos
├── dao/                      # Acesso ao MySQL
│   ├── UsuarioDAO.java
│   ├── ContaDAO.java
│   ├── CategoriaDAO.java
│   └── MovimentacaoDAO.java
├── model/
│   ├── Usuario.java
│   ├── Conta.java
│   ├── Categoria.java
│   └── Movimentacao.java
└── util/
    ├── DatabaseUtil.java       # Conexão MySQL
    └── SecurityUtil.java       # Criptografia SHA-256
```

## Fluxo do Servidor

```
INICIAR: java MainServidor
```

↓

```
MainServidor.main()
• Exibe banner
• Cria: FinanzaServer server = new FinanzaServer()
• Chama: server.start()
```

↓

```
FinanzaServer.start()
• Testa conexão MySQL: DatabaseUtil.testConnection()
• Inicializa banco: DatabaseUtil.initializeDatabase()
• Abre servidor: ServerSocket(12345)
• Loop infinito: while(true) { accept() }
```

↓

```
SERVIDOR RODANDO
• Escutando na porta 12345
• Aguardando conexões...
```

↓

```
CLIENTE CONECTA (Mobile ou Desktop)
• Socket clientSocket = serverSocket.accept()
• Cria thread: ClientHandler handler = new ClientHandler(socket)
• Inicia: handler.start()
```

↓

```
ClientHandler.run()
• Cria streams de entrada/saída
• Loop: while((comando = input.readLine()) != null)
  • String resposta = processarComando(comando)
  • output.println(resposta)
```

↓

```
ClientHandler.processarComando()
• Parse: String[] partes = Protocol.parseCommand(cmd)
• Switch por tipo de comando:
  - "LOGIN" → processarLogin()
  - "REGISTER" → processarRegistro()
  - "ADD_CONTA" → processarAddConta()
  - "ADD_MOVIMENTACAO" → processarAddMovimentacao()
  - 40+ outros comandos...
```

↓

```
Método processar<Comando>()
• Extrai parâmetros
• Chama DAO apropriado
• DAO executa SQL no MySQL
```

- Formata resposta
- Retorna: "OK|dados" ou "ERROR|mensagem"



## DESKTOP ADMIN (Java Swing)

### Estrutura de Arquivos

```
DESKTOP VERSION/ClienteFinanza/src/  
|   └── MainCliente.java          # Inicia a aplicação  
|  
|   └── view/                      # Interface Swing  
|       ├── LoginView.java         # Tela de login  
|       ├── AdminDashboardView.java # Tabela de usuários  
|       └── EditarUsuarioDialog.java # Editar usuário  
|  
|   └── controller/  
|       ├── AuthController.java    # Controle de login  
|       └── FinanceController.java # Controle de operações  
|  
|   └── model/  
|       ├── Usuario.java  
|       ├── Conta.java  
|       ├── Categoria.java  
|       └── Movimentacao.java  
|  
└── util/  
    └── NetworkClient.java        # Cliente TCP para servidor
```

## Fluxo Desktop Admin

INICIAR: java MainCliente

↓

MainCliente.main()  
• Configura Look and Feel  
• Cria: LoginView loginView = new LoginView()  
• Exibe: loginView.setVisible(true)

↓

ADMIN VÊ: Tela de Login

🔒 Login Administrativo  
Email: [admin@finanza.com ]  
Senha: [\*\*\*\*\*]  
[Entrar]

↓

ADMIN DIGITA credenciais e clica "Entrar"

↓

LoginView.btnEntrarActionPerformed()  
• Valida campos  
• Chama: AuthController.login(email, senha)

↓

AuthController.login()  
• Criptografa senha: SecurityUtil.hashPassword()  
• Monta comando: "LOGIN|email|hash|admin"  
• Chama: NetworkClient.sendCommand(comando)

↓

NetworkClient.sendCommand()  
• Socket socket = new Socket("localhost", 12345)  
• Envia comando via PrintWriter  
• Aguarda resposta via BufferedReader  
• Fecha socket  
• Retorna resposta

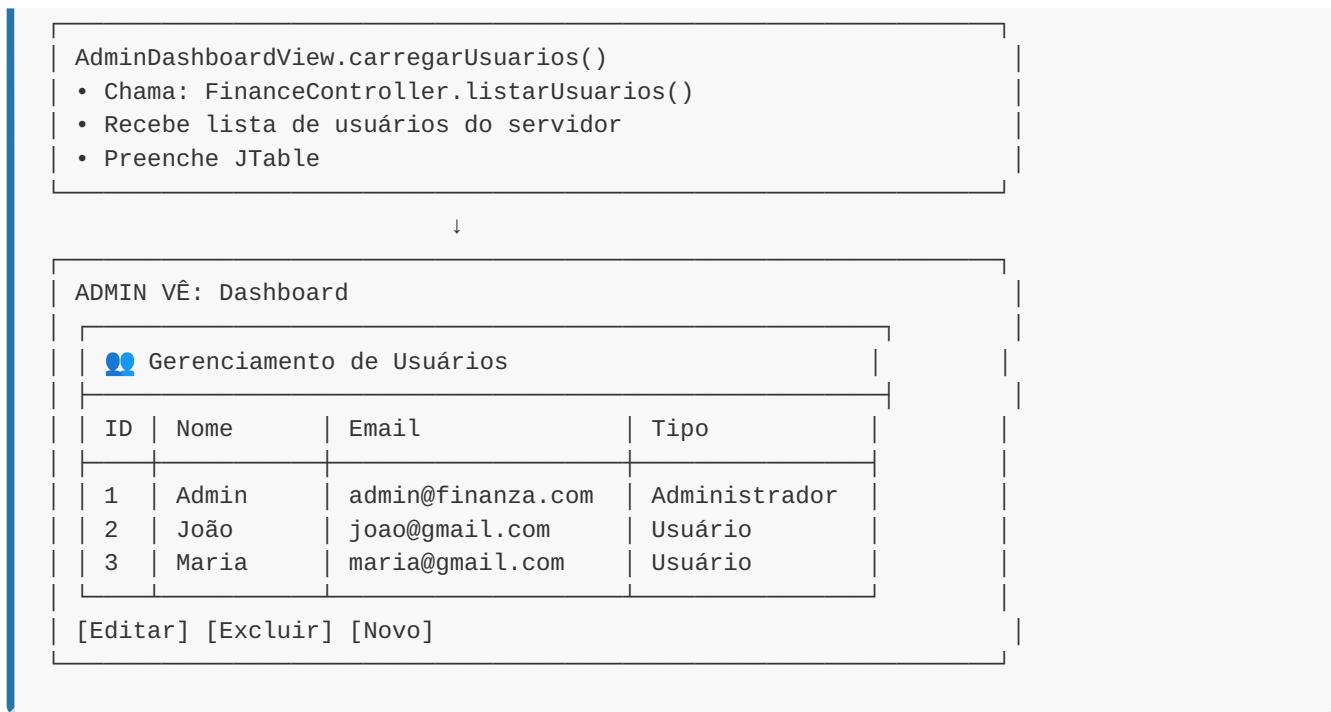
↓

SERVIDOR PROCESSA

↓

LoginView [recebe resposta]  
• Se OK: Fecha LoginView  
• Abre: AdminDashboardView dashboard = new ...  
• dashboard.setVisible(true)

↓



## PROTOCOLO DE COMUNICAÇÃO

### Comandos Principais

Comando	Formato	Resposta	Onde Processa
<b>LOGIN</b>	LOGIN email senha_hash	OK dados_usuario	ClientHandler.processarLogin()
<b>REGISTER</b>	REGISTER nome email senha uuid	OK id	ClientHandler.processarRegistro()
<b>ADD_CONTA</b>	ADD_CONTA nome tipo saldo userId	OK id_conta	ClientHandler.processarAddConta()
<b>LIST_CONTAS</b>	LIST_CONTAS userId	OK lista	ClientHandler.processarListContas()
<b>ADD_MOVIMENTACAO</b>	ADD_MOVIMENTACAO valor data desc ...	OK id	ClientHandler.processarAddMovimentacao()
<b>LIST_MOVIMENTACOES</b>	LIST_MOVIMENTACOES userId	OK lista	ClientHandler.processarListMovimentacoes()

## Formato de Mensagens

ENVIO: COMANDO|PARAM1|PARAM2|PARAM3

RESPOSTA: OK|dados ou ERROR|mensagem

Exemplo:

Envio: LOGIN|joao@gmail.com|abc123hash

Resposta: OK|{"id":2,"nome":"João","email":"joao@gmail.com"}

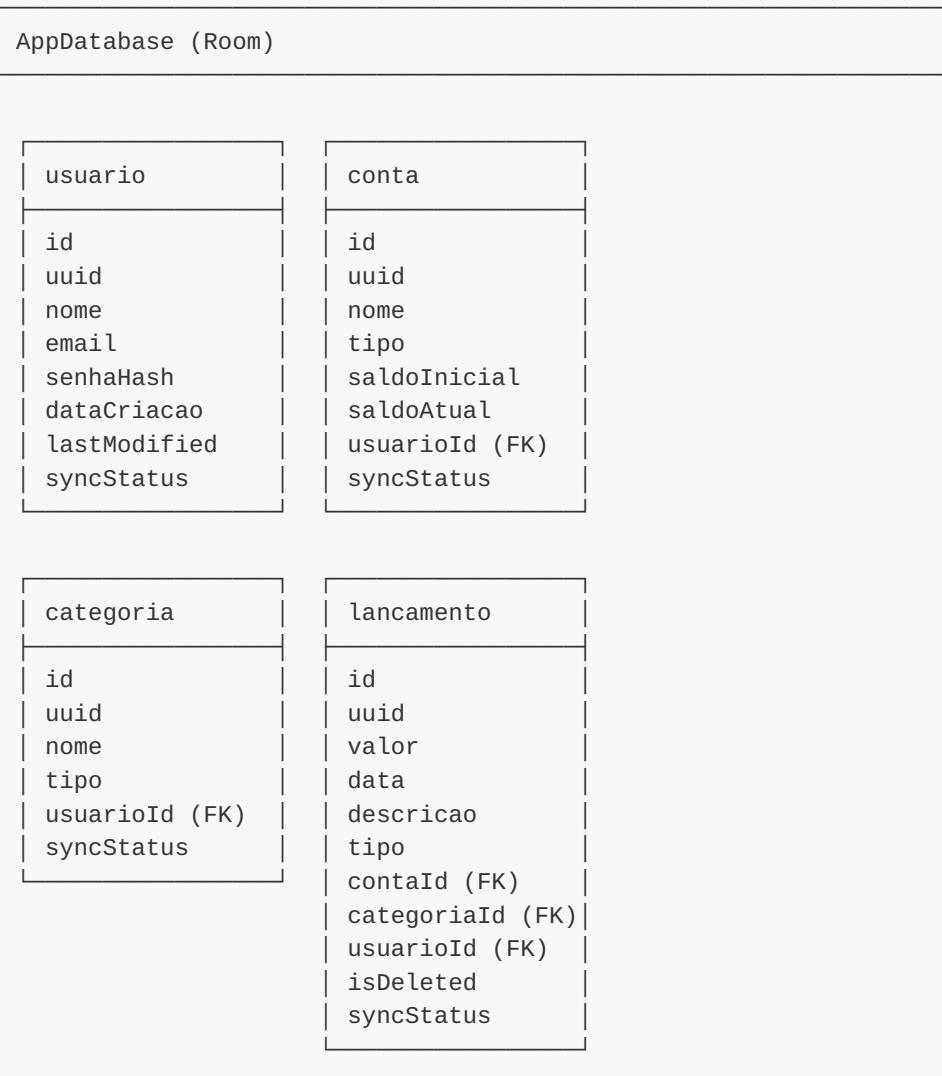
Envio: ADD\_CONTA|Nubank|corrente|500.00|2

Resposta: OK|15



## BANCO DE DADOS

### Mobile (SQLLite)



## Servidor (MySQL)



# SÍNCRONIZAÇÃO

## Como Funciona

MOBILE cria/edita dados localmente (SQLite)

- syncStatus = 2 (NEEDS\_SYNC)

↓

SyncService (Background)

- Busca registros com syncStatus = 2
- Para cada registro:
  - Envia ao servidor via ServerClient
  - Aguarda confirmação
  - Marca como sincronizado (syncStatus = 1)

↓

SERVIDOR salva no MySQL

- INSERT ou UPDATE conforme necessário
- Retorna ID gerado ou confirmação

↓

SyncService busca atualizações do servidor

- Envia timestamp do último sync
- Recebe lista de mudanças desde então
- Aplica mudanças localmente

## Estados de Síncronização

syncStatus	Significado	Cor
0	LOCAL_ONLY - Apenas local	 Branco
1	SYNCED - Síncronizado	 Verde
2	NEEDS_SYNC - Precisa sincronizar	 Amarelo
3	CONFLICT - Conflito detectado	 Laranja
4	DELETED - Deletado	 Cinza



## RESUMO DE ARQUIVOS POR FUNÇÃO

### Quando o usuário FAZ LOGIN:

**Mobile:** 1. LoginActivity.java → Exibe formulário 2. AuthManager.java → Criptografa e valida 3. ServerClient.java → Envia ao servidor 4. UsuarioDao.java → Salva localmente após sucesso 5. MenuActivity.java → Abre dashboard

**Servidor:** 1. ClientHandler.java → Recebe comando LOGIN 2. UsuarioDAO.java → Busca no MySQL 3. SecurityUtil.java → Valida senha 4. Retorna dados do usuário

---

### Quando o usuário ADICIONA DESPESA:

**Mobile:** 1. MovementsActivity.java → Exibe formulário 2. DataIntegrityValidator.java → Valida dados 3. LancamentoDao.java → Insere no SQLite 4. ContaDao.java → Atualiza saldo 5. EnhancedSyncService.java → Sincroniza com servidor 6. ServerClient.java → Envia dados

**Servidor:** 1. ClientHandler.java → Recebe comando ADD\_MOVIMENTACAO 2. MovimentacaoDAO.java → Insere no MySQL 3. ContaDAO.java → Atualiza saldo no servidor 4. Retorna ID gerado

---

### Quando ADMIN EDITA USUÁRIO:

**Desktop:** 1. AdminDashboardView.java → Lista usuários 2. EditarUsuarioDialog.java → Formulário de edição 3. FinanceController.java → Processa alterações 4. NetworkClient.java → Envia ao servidor

**Servidor:** 1. ClientHandler.java → Recebe comando UPDATE\_USER 2. UsuarioDAO.java → UPDATE no MySQL 3. Retorna confirmação

---



## CONCLUSÃO

### Principais Arquivos por Camada

**INTERFACE (o que o usuário vê):** - Mobile: LoginActivity, MenuActivity, MovementsActivity, AccountsActivity - Desktop: LoginView, AdminDashboardView, EditarUsuarioDialog

**LÓGICA (processamento):** - Mobile: AuthManager, SyncService, EnhancedSyncService, DataIntegrityValidator - Desktop: AuthController, FinanceController - Servidor: ClientHandler, Protocol

**DADOS (banco de dados):** - Mobile: UsuarioDao, ContaDao, CategoriaDao, LancamentoDao (SQLite) - Servidor: UsuarioDAO, ContaDAO, CategoriaDAO, MovimentacaoDAO (MySQL)

**COMUNICAÇÃO (rede):** - Mobile: ServerClient, Protocol - Desktop: NetworkClient - Servidor: FinanzaServer, ClientHandler

## Fluxo Geral de Dados

```
USUÁRIO INTERAGE
↓
ACTIVITY (UI)
↓
MANAGER/VALIDATOR (Lógica)
↓
DAO LOCAL (SQLite)
↓
SYNC SERVICE (Sincronização)
↓
SERVER CLIENT (Rede)
↓
INTERNET
↓
CLIENT HANDLER (Servidor)
↓
DAO SERVIDOR (MySQL)
↓
RESPOSTA VOLTA
```

**Documento criado para:** Sistema Finanza

**Objetivo:** Entender rapidamente o fluxo do software

**Autor:** Kalleby Schultz - IFSUL Campus Venâncio Aires

**Data:** 2024