Multi-Channel Solution Architecture Blueprint

XYZ Bank Full Stack Banking Software

Task 6: Multi-Channel Solution Architecture Blueprint
Groups: 33 and 1
Course: FOC 322 Professional Software Development
Authors: [Your Names]
Date: [Current Date]

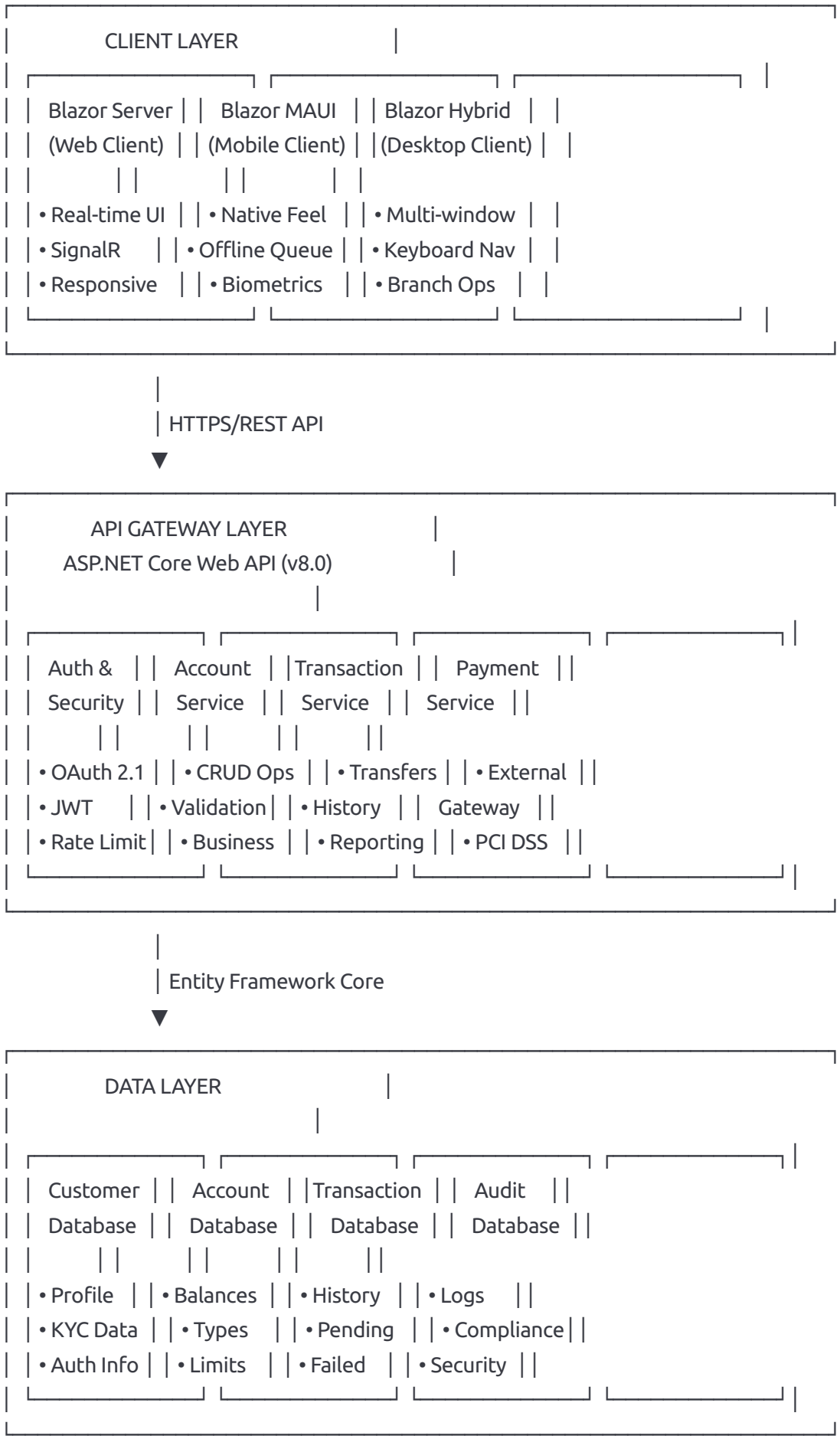
Executive Summary

This document presents a comprehensive multi-channel solution architecture for XYZ Bank's full stack banking software. The architecture supports Web (Blazor Server), Mobile (Blazor MAUI), and Desktop (Blazor Hybrid) clients built on a shared ASP.NET Core API tier, ensuring consistent user experience across all platforms while maintaining security, scalability, and regulatory compliance.

---

1. Architecture Overview

1.1 High-Level Architecture



1.2 Design Principles

1. **Shared Business Logic**: Common domain models and services across all platforms
2. **API-First Architecture**: All client interactions go through standardized REST APIs
3. **Security by Design**: OAuth 2.1, JWT tokens, and end-to-end encryption
4. **Responsive Design**: Adaptive UI that works across different screen sizes
5. **Offline Capability**: Mobile and desktop clients support offline operations
6. **Real-time Updates**: SignalR for live notifications and balance updates

---

2. Technology Stack Comparison Matrix

2.1 Web Platform Options

Criteria	Blazor Server	Blazor WebAssembly	Razor Pages	Weight	Score
Performance	9	7	8	25%	8.5
Security	10	6	9	30%	8.7
Real-time Features	10	7	5	20%	8.4
Offline Support	3	9	2	10%	4.6
Development Speed	9	8	9	15%	8.7
****Total Score**	**8.22**	**7.05**	**7.65**		

Recommendation: **Blazor Server** - Best for banking due to superior security and real-time capabilities

2.2 Mobile Platform Options

Criteria	Blazor MAUI	React Native	Flutter	Weight	Score
Native Performance	8	9	9	25%	8.5
Code Sharing	10	6	7	30%	8.2
Offline Support	9	8	8	20%	8.4
Device Integration	9	9	8	15%	8.7
Team Skill Fit	10	5	6	10%	7.5
****Total Score**	**8.57**	**7.4**	**7.7**		

Recommendation: **Blazor MAUI** - Maximum code reuse with existing C# expertise

2.3 Desktop Platform Options

Criteria	Blazor Hybrid	WPF	Electron	Weight	Score
Performance	8	10	6	25%	8.0
Cross-Platform	9	3	10	20%	7.2
Code Sharing	10	7	8	30%	8.7
Native Integration	9	10	7	15%	8.7
Resource Usage	8	9	5	10%	7.5
****Total Score**	**8.31**	**7.5**	**7.2**		

Recommendation: **Blazor Hybrid in MAUI** - Best balance of performance and code sharing

---

3. Detailed Architecture Components

3.1 Shared Component Library Structure

```
XYZBank.Shared/
├── Components/
│   ├── Common/
│   │   ├── LoadingSpinner.razor
│   │   ├── ErrorBoundary.razor
│   │   └── NavigationMenu.razor
│   ├── Account/
│   │   ├── AccountSummary.razor
│   │   ├── AccountDetails.razor
│   │   └── AccountSelector.razor
│   ├── Transaction/
│   │   ├── TransactionList.razor
│   │   ├── TransactionDetails.razor
│   │   └── TransferForm.razor
│   └── Security/
│       ├── LoginForm.razor
│       ├── BiometricAuth.razor
│       └── TwoFactorAuth.razor
├── Services/
│   ├── Interfaces/
│   │   ├── IAccountService.cs
│   │   ├── ITransactionService.cs
│   │   ├── IAuthService.cs
│   │   └── INotificationService.cs
│   └── Implementations/
│       ├── AccountService.cs
│       ├── TransactionService.cs
│       └── AuthService.cs
├── Models/
│   ├── DTOs/
│   │   ├── AccountDto.cs
│   │   ├── TransactionDto.cs
│   │   └── UserDto.cs
│   ├── Requests/
│   │   ├── TransferRequest.cs
│   │   └── LoginRequest.cs
│   └── Responses/
│       ├── ApiResponse.cs
│       └── AuthResponse.cs
└── Utilities/
    ├── Extensions/
    ├── Helpers/
    └── Constants/
```

3.2 API Layer Architecture

csharp

```
// API Controller Structure
XYZBank.API/
├── Controllers/
│   ├── AccountController.cs
│   ├── TransactionController.cs
│   ├── AuthController.cs
│   └── NotificationController.cs
├── Middleware/
│   ├── AuthenticationMiddleware.cs
│   ├── ExceptionMiddleware.cs
│   └── RateLimitMiddleware.cs
├── Services/
│   ├── Business/
│   │   ├── AccountBusinessService.cs
│   │   └── TransactionBusinessService.cs
│   └── Infrastructure/
│       ├── EmailService.cs
│       └── SmsService.cs
└── Data/
    ├── Context/
    │   └── BankingDbContext.cs
    ├── Entities/
    │   ├── Account.cs
    │   ├── Transaction.cs
    │   └── User.cs
    └── Repositories/
        ├── IAccountRepository.cs
        └── AccountRepository.cs
```

3.3 Platform-Specific Implementations

Web Application (Blazor Server)

csharp

```csharp
// Program.cs configuration
builder.Services.AddServerSideBlazor()
    .AddSignalR(options =>
    {
        options.MaximumReceiveMessageSize = 1024 * 1024; // 1MB
        options.EnableDetailedErrors = true;
    });

// Real-time notifications
builder.Services.AddScoped<INotificationHub, NotificationHub>();
```

Mobile Application (Blazor MAUI)

csharp

```csharp
// MauiProgram.cs
public static MauiApp CreateMauiApp()
{
    var builder = MauiApp.CreateBuilder();
    builder
        .UseMauiApp<App>()
        .ConfigureFonts(fonts =>
        {
            fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
        });

    // Platform-specific services
    builder.Services.AddSingleton<IBiometricService, BiometricService>();
    builder.Services.AddSingleton<IOfflineStorageService, OfflineStorageService>();

    return builder.Build();
}
```

Desktop Application (Blazor Hybrid)

csharp

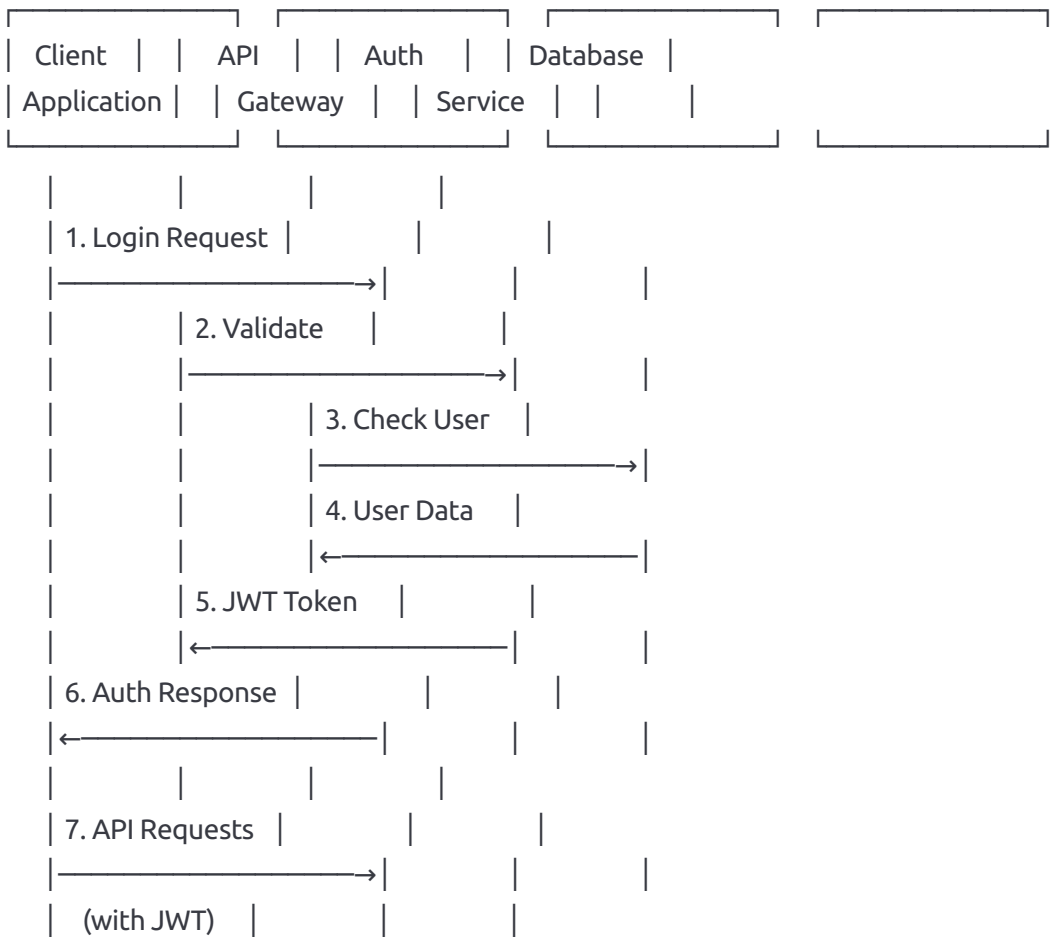
```csharp
// Program.cs for Desktop
var builder = MauiApp.CreateBuilder();
builder
    .UseMauiApp<App>()
    .ConfigureFonts(fonts =>
    {
        fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
    });

// Desktop-specific services
builder.Services.AddSingleton<IKeyboardShortcutService, KeyboardShortcutService>();
builder.Services.AddSingleton<IMultiWindowService, MultiWindowService>();
```

---

4. Security Architecture

4.1 Authentication Flow



4.2 Data Protection Strategy

- **Data at Rest**: AES-256 encryption for sensitive data
- **Data in Transit**: TLS 1.3 for all communications
- **API Security**: OAuth 2.1 with PKCE for mobile clients
- **Token Management**: Short-lived access tokens (15 minutes) with refresh tokens
- **Rate Limiting**: IP-based and user-based rate limiting

---

5. Performance Optimization

5.1 Caching Strategy

```csharp
// Multi-level caching approach
public class CacheService
{
    private readonly IMemoryCache _memoryCache;
    private readonly IDistributedCache _distributedCache;

    // L1 Cache: In-memory (fast, limited)
    // L2 Cache: Redis (shared, persistent)
    // L3 Cache: Database (source of truth)
}
```

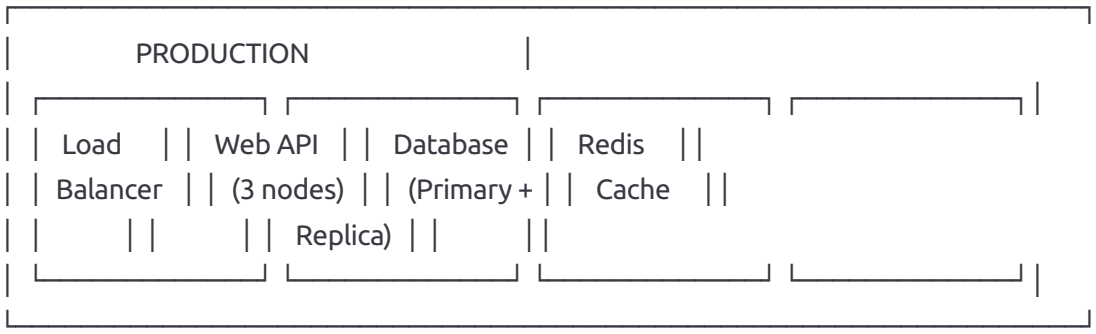
5.2 Database Optimization

- **Connection Pooling**: Configured for high concurrency
- **Query Optimization**: Entity Framework Core with compiled queries
- **Indexing Strategy**: Optimized for transaction queries
- **Partitioning**: Historical data partitioned by date

---

6. Deployment Architecture

6.1 Environment Strategy



6.2 CI/CD Pipeline

yaml

```yaml
# Azure DevOps Pipeline
trigger:
  branches:
    include:
      - main
      - develop

stages:
  - stage: Build
    jobs:
      - job: BuildWeb
        displayName: 'Build Web Application'
      - job: BuildMobile
        displayName: 'Build Mobile Apps'
      - job: BuildDesktop
        displayName: 'Build Desktop Apps'

  - stage: Test
    jobs:
      - job: UnitTests
      - job: IntegrationTests
      - job: SecurityTests

  - stage: Deploy
    jobs:
      - job: DeployToStaging
      - job: DeployToProduction
```

---

7. Architecture Decision Records (ADRs)

ADR-001: Choose Blazor Server for Web Platform

Status: Accepted
Date: [Current Date]

Context: Need to select web technology for banking application requiring real-time updates and high security.

Decision: Use Blazor Server over Blazor WebAssembly or traditional MVC.

Rationale:

- **Security**: Code executes on server, reducing client-side vulnerabilities
- **Performance**: No need to download large WebAssembly files
- **Real-time**: Native SignalR integration for live updates
- **SEO**: Server-side rendering benefits
- **Bandwidth**: Minimal data transfer after initial load

Consequences:

- Requires constant server connection
- Higher server resource usage
- Excellent security posture
- Optimal for banking use case

ADR-002: Choose Blazor MAUI for Mobile Platform

Status: Accepted
Date: [Current Date]

Context: Need mobile solution that maximizes code reuse while providing native performance.

Decision: Use Blazor MAUI over React Native or Flutter.

Rationale:

- **Code Reuse**: 95% code sharing with web application
- **Team Expertise**: Leverages existing C# skills
- **Native Performance**: Compiles to native code
- **Microsoft Ecosystem**: Consistent tooling and support
- **Offline Capability**: Easy to implement offline features

Consequences:

- Learning curve for MAUI-specific concepts
- Excellent code reuse and maintenance
- Strong performance characteristics
- Seamless integration with existing architecture

ADR-003: Use ASP.NET Core for API Layer

Status: Accepted
Date: [Current Date]

Context: Need robust, scalable API platform for banking services.

Decision: Use ASP.NET Core Web API with Entity Framework Core.

Rationale:

- **Performance**: High-performance runtime
- **Security**: Built-in security features
- **Scalability**: Excellent scaling characteristics
- **Ecosystem**: Rich middleware ecosystem
- **Compliance**: Enterprise-grade security features

Consequences:

- Excellent performance and security
- Rich feature set for banking requirements
- Strong community and Microsoft support
- Ideal for financial services applications

---

8. Implementation Roadmap

Phase 1: Foundation (Weeks 1-2)

- [ ] Set up shared component library
- [ ] Implement base API structure
- [ ] Create authentication system
- [ ] Establish database schema

Phase 2: Core Features (Weeks 3-4)

- [ ] Account management APIs
- [ ] Transaction processing
- [ ] Basic UI components
- [ ] Authentication integration

Phase 3: Platform Implementation (Weeks 5-6)

- [ ] Web application (Blazor Server)
- [ ] Mobile application (Blazor MAUI)
- [ ] Desktop application (Blazor Hybrid)
- [ ] Cross-platform testing

Phase 4: Advanced Features (Weeks 7-8)

- [ ] Real-time notifications
- [ ] Offline capabilities
- [ ] Performance optimization
- [ ] Security hardening

---

9. Risk Assessment and Mitigation

9.1 Technical Risks

Risk	Probability	Impact	Mitigation
Blazor Server connection issues	Medium	High	Implement connection retry logic and offline fallback
MAUI platform limitations	Low	Medium	Prototype critical features early
Performance under load	Medium	High	Implement comprehensive load testing
Security vulnerabilities	Low	Very High	Regular security audits and penetration testing

9.2 Business Risks

Risk	Probability	Impact	Mitigation
Regulatory compliance issues	Low	Very High	Engage compliance team early
User adoption challenges	Medium	High	Comprehensive user testing and training
Integration complexity	Medium	Medium	Phased rollout with fallback plans

---

10. Conclusion

This multi-channel solution architecture provides XYZ Bank with a robust, scalable, and secure platform that maximizes code reuse while delivering native experiences across web, mobile, and desktop platforms. The Blazor ecosystem choice ensures consistency, reduces development time, and leverages the team's existing C# expertise.

The architecture supports all banking requirements including real-time updates, offline capabilities, multi-platform deployment, and strict security standards required for financial services. The phased implementation approach reduces risk while delivering value incrementally.

Appendices

Appendix A: Technology Stack Details

Frontend Technologies:

- Blazor Server 8.0
- Blazor MAUI 8.0
- SignalR for real-time communication
- Bootstrap 5 for responsive design

Backend Technologies:

- ASP.NET Core 8.0 Web API
- Entity Framework Core 8.0
- SQL Server 2022
- Redis for caching

Security:

- OAuth 2.1 with PKCE
- JWT tokens with refresh mechanism
- HTTPS/TLS 1.3
- AES-256 encryption

DevOps:

- Azure DevOps for CI/CD
- Docker containers
- Azure App Service
- Application Insights for monitoring

Appendix B: API Endpoints Summary

Authentication:

- `POST /api/auth/login`
- `POST /api/auth/refresh`
- `POST /api/auth/logout`

Account Management:

- `GET /api/accounts`
- `GET /api/accounts/{id}`
- `POST /api/accounts`
- `PUT /api/accounts/{id}`

Transactions:

- `GET /api/transactions`
- `POST /api/transactions/transfer`
- `GET /api/transactions/{id}`
- `GET /api/transactions/history`

Appendix C: Database Schema Overview

```sql
-- Key database tables
CREATE TABLE Users (
    Id UNIQUEIDENTIFIER PRIMARY KEY,
    Email NVARCHAR(255) UNIQUE NOT NULL,
    PasswordHash NVARCHAR(255) NOT NULL,
    CreatedAt DATETIME2 NOT NULL,
    LastLoginAt DATETIME2
);

CREATE TABLE Accounts (
    Id UNIQUEIDENTIFIER PRIMARY KEY,
    UserId UNIQUEIDENTIFIER NOT NULL,
    AccountNumber NVARCHAR(20) UNIQUE NOT NULL,
    Balance DECIMAL(18,2) NOT NULL,
    AccountType NVARCHAR(50) NOT NULL,
    CreatedAt DATETIME2 NOT NULL,
    FOREIGN KEY (UserId) REFERENCES Users(Id)
);

CREATE TABLE Transactions (
    Id UNIQUEIDENTIFIER PRIMARY KEY,
    FromAccountId UNIQUEIDENTIFIER,
    ToAccountId UNIQUEIDENTIFIER,
    Amount DECIMAL(18,2) NOT NULL,
    TransactionType NVARCHAR(50) NOT NULL,
    Status NVARCHAR(20) NOT NULL,
    CreatedAt DATETIME2 NOT NULL,
    FOREIGN KEY (FromAccountId) REFERENCES Accounts(Id),
    FOREIGN KEY (ToAccountId) REFERENCES Accounts(Id)
);
```

Document Version: 1.0

Last Updated: [Current Date]

Next Review: [Date + 3 months]