COMS3009A- Software Design

Web Application Project 2025

Freelancing Platform – Software Architecture

Group Members:

Names	**Student Numbers**
Dolf Shungube	2683067
Faranani Nemutanzhela	2651532
Bokang Madondo	2681261
Mashego Mabeloane Gideon	2654606
Mutsawashe Njowo	2651487
Nelson	2681283

1. Executive Summary

This document describes the software architecture for a web-based freelancing platform that connects clients with freelancers. The system facilitates job posting, application management, project progress tracking, messaging, and document handling. The platform is built using a modern web stack with Supabase as the backend-as-a-service provider.

2. System Overview

The freelancing platform enables three types of users (Clients, Freelancers, and Admins) to interact through a web interface. The system supports the complete freelancing workflow from job creation to project completion, including real-time messaging and progress tracking.

2.1 Key Features:

- User authentication and role-based access control
- Job posting and application management
- Real-time messaging between clients and freelancers
- Project progress tracking with task management
- Document upload and management (CVs, contracts)
- Reporting and analytics
- PDF report generation

3. Architectural Style and Patterns

3.1 Primary Architecture: Client-Server Architecture with Three-Tier Pattern

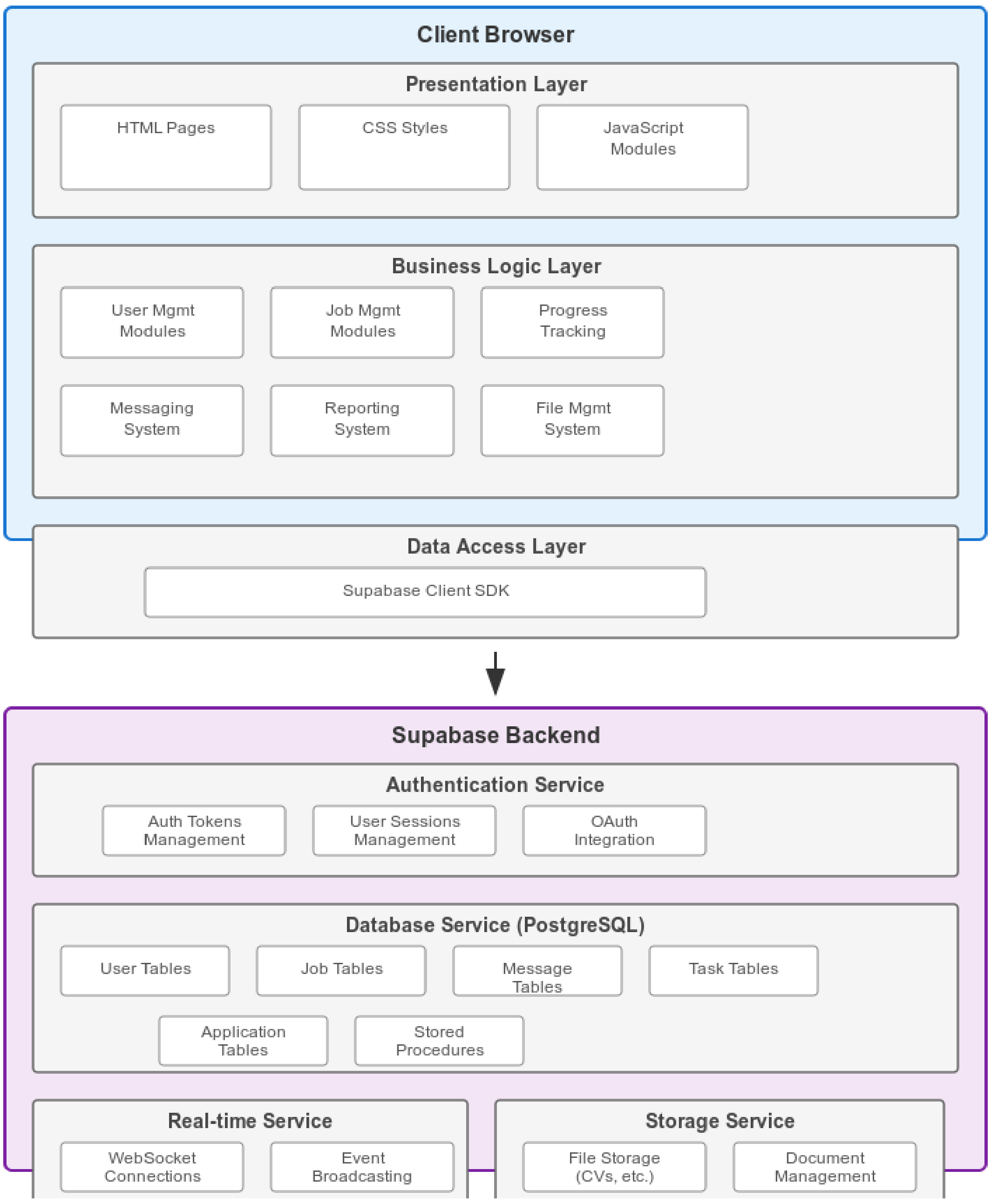
- **Presentation Tier:** Frontend web application (HTML, CSS, JavaScript)
- **Logic Tier:** Business logic implemented in client-side JavaScript modules
- **Data Tier:** Supabase (PostgreSQL database + authentication + real-time + storage)

3.2 Architectural Patterns Used:

- **Module Pattern:** Code organized into ES6 modules for separation of concerns
- **MVC Pattern:** Implicit separation of data handling, business logic, and UI
- **Observer Pattern:** Real-time updates using Supabase subscriptions
- **Repository Pattern:** Data access abstraction through Supabase client

4. System Architecture

4.1 High-Level Architecture



4.2 Component Architecture

Frontend Components:

1. **Authentication Module**
   - User registration and login
   - OAuth integration (Google)
   - Password reset functionality
   - Session management

2. **User Management Module**
    - Profile management for all user types
    - Settings and preferences
    - Role-based access control
3. **Job Management Module**
    - Job creation and posting
    - Job browsing and searching
    - Application submission and management
    - Job assignment workflow
4. **Progress Tracking Module**
    - Task creation and management
    - Progress visualization
    - Status reporting
    - Timeline tracking
5. **Messaging Module**
    - Real-time chat functionality
    - Message history
    - Unread message notifications
    - User-to-user communication
6. **Reporting Module**
    - Progress reports
    - PDF generation
    - Analytics and insights
    - Data visualization
7. **File Management Module**
    - CV upload and management
    - Contract document handling
    - File storage and retrieval

4.3 Data Architecture

1. Database Schema (Conceptual)

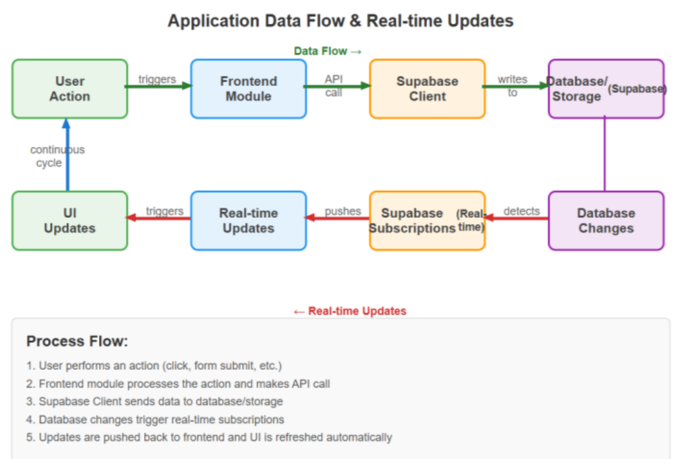
Core Entities:

- **Users:** Authentication and basic user information
- **Clients:** Client-specific profile information
- **Freelancers:** Freelancer-specific profile and CV information
- **Admins:** Administrative user information
- **Jobs:** Project postings and details
- **Applications:** Job applications from freelancers
- **Tasks:** Individual tasks within jobs
- **Messages:** Communication between users
- **Contracts:** Legal documents and agreements

Key Relationships:

- Users (1:1) → Clients/Freelancers/Admins
- Clients (1:N) → Jobs
- Jobs (1:N) → Applications
- Freelancers (1:N) → Applications
- Jobs (1:N) → Tasks
- Users (N:N) → Messages
- Jobs (1:1) → Contracts

2. Data Flow Architecture



Process Flow:

1. User performs an action (click, form submit, etc.)
2. Frontend module processes the action and makes API call
3. Supabase Client sends data to database/storage
4. Database changes trigger real-time subscriptions
5. Updates are pushed back to frontend and UI is refreshed automatically

4.3 Security Architecture

1. Authentication and Authorization

Authentication Methods:

- Email/password authentication

- OAuth 2.0 (Google integration)
- JWT token-based sessions
- Password recovery mechanisms

Authorization Strategy:

- Role-based access control (Client, Freelancer, Admin)
- Resource-level permissions
- Session-based security
- API endpoint protection

2. Data Security

Security Measures:

- Encrypted data transmission (HTTPS)
- Secure file storage with access controls
- Input validation and sanitization
- SQL injection prevention through parameterized queries
- XSS prevention through content sanitization

4.4 Integration Architecture

1. External Integrations

Supabase Services:

- Authentication API
- Database API (PostgreSQL)
- Storage API
- Real-time API
- Edge Functions (if used)

Third-party Libraries:

- SweetAlert2 for user notifications
- Chart.js for data visualization
- html2pdf for report generation
- PDF.js for document viewing

4.5 API Architecture

Data Access Pattern:

- Direct Supabase client SDK calls

- RPC (Remote Procedure Call) for complex queries
- Real-time subscriptions for live updates
- RESTful API patterns through Supabase

4.6 Deployment Architecture

1. Frontend Deployment

Static Web Hosting:

- Azure Static Web Apps
- CDN for asset delivery
- HTTPS enforcement
- Environment-specific configurations

2. Backend Services

Supabase Cloud:

- Managed PostgreSQL database
- Authentication service
- File storage service
- Real-time WebSocket service
- Edge functions (if applicable)

4.7 Performance Architecture

1. Optimization Strategies

Frontend Optimization:

- Code splitting through ES6 modules
- Lazy loading of non-critical resources
- Client-side caching
- Minimized HTTP requests

Backend Optimization:

- Database indexing
- Query optimization through RPC functions
- Connection pooling (managed by Supabase)
- CDN for static assets

5. **Scalability Considerations**

Horizontal Scaling:

- Stateless frontend architecture
- Managed scaling through Supabase
- Load balancing (handled by cloud provider)

Vertical Scaling:

- Database performance tuning
- Storage optimization
- Memory management

6. Monitoring and Maintenance

6.1 Monitoring Strategy

Application Monitoring:

- Client-side error tracking
- Performance monitoring
- User analytics
- System health checks

Database Monitoring:

- Query performance analysis
- Connection monitoring
- Storage utilization tracking

6.2 Maintenance Procedures

Regular Maintenance:

- Database backup and recovery
- Security updates
- Performance optimization
- Bug fixes and feature updates

7. Risk Assessment and Mitigation

7.1 Technical Risks

Risk Categories:

- Single point of failure (Supabase dependency)
- Data loss risks
- Security vulnerabilities
- Performance bottlenecks

Mitigation Strategies:

- Regular backups
- Security audits
- Performance monitoring
- Disaster recovery planning

7.2 Business Risks

Risk Categories:

- User data privacy
- Service availability
- Scalability limitations
- Integration dependencies

Mitigation Strategies:

- Privacy compliance measures
- Service level agreements
- Capacity planning
- Vendor diversification considerations

8. Future Architecture Considerations

8.1 Scalability Enhancements

Potential Improvements:

- Microservices architecture migration
- API gateway implementation
- Caching layer addition
- Database sharding considerations

8.2 Technology Evolution

Future Technologies:

- Progressive Web App (PWA) features
- Mobile application development
- Advanced analytics implementation
- AI/ML integration possibilities

9. Conclusion

The current architecture provides a solid foundation for a freelancing platform with good separation of concerns, scalability potential, and modern web development practices. The use of Supabase as a backend-as-a-service provides rapid development capabilities while maintaining professional-grade features for authentication, real-time updates, and data management.

The modular frontend architecture allows for maintainable code and future enhancements, while the cloud-based backend ensures reliability and scalability. The architecture successfully addresses the core requirements of a freelancing platform while providing room for future growth and feature additions.