Business Rule Guide for Artisan Network Portal Front-End Database Connection

Overview

The **Artisan Network Portal** is a dynamic online platform designed to bridge skilled artisans (e.g., carpenters, painters, tailors, or technicians) with clients seeking specialized services. This guide offers a comprehensive technical blueprint for developers, focusing on **database architecture**, **system integration**, and **front-end connectivity**.

User Types and Account Structure

User Categories

Clients/Customers

Post projects

Select artisans

Make payments

Provide ratings and feedback

Artisans

Register in specific trades (Electrical installation, Hardware maintenance, etc.)

Bid on projects

Complete work

Receive payments and ratings

Administrators

Moderate activities

Handle disputes

Process payments

Manage user accounts

Registration Requirements

Client Registration Fields

Full name

Contact information (email, phone)

Location/address

Username and password

Security question/answer

Payment information

Artisan Registration Fields

Full name

Trade specialty (from the 10 listed categories)

Contact information (email, phone)

Physical address

Certification/qualifications

Experience level

Portfolio/previous work

ID verification

Payment account details

Username and password

Database Schema

Core Tables

Users

```
sqlCREATE TABLE users (
  user_id INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(50) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  phone VARCHAR(20) NOT NULL,
  full_name VARCHAR(100) NOT NULL,
  user_type ENUM('client', 'artisan', 'admin') NOT NULL,
  registration_date DATETIME DEFAULT CURRENT_TIMESTAMP,
  last_login DATETIME,
  account_status ENUM('active', 'suspended', 'inactive') DEFAULT 'active',
  security_question VARCHAR(255),
  security_answer VARCHAR(255)
);
```

Client_Profiles

```
sqlCREATE TABLE client_profiles (
  client_id INT PRIMARY KEY,
  address VARCHAR(255),
  city VARCHAR(100),
  state VARCHAR(100),
```

```
  payment_info TEXT,
  FOREIGN KEY (client_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

Artisan_Profiles

```
sqlCREATE TABLE artisan_profiles (
  artisan_id INT PRIMARY KEY,
  trade_category ENUM('electrical_installation', 'hardware_maintenance', 'office_technology',
                'satellite_installation', 'networking', 'pc_maintenance',
                'solar_installation', 'cctv_installation', 'gsm_repairs',
                'electrical_construction') NOT NULL,
  years_experience INT,
  certification TEXT,
  bio TEXT,
  portfolio_url VARCHAR(255),
  id_verification_status ENUM('pending', 'verified', 'rejected') DEFAULT 'pending',
  payment_details TEXT,
  avg_rating DECIMAL(3,2) DEFAULT 0,
  FOREIGN KEY (artisan_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

Projects

```
sqlCREATE TABLE projects (
  project_id INT PRIMARY KEY AUTO_INCREMENT,
  client_id INT NOT NULL,
  title VARCHAR(255) NOT NULL,
  description TEXT NOT NULL,
```

```
  category ENUM('electrical_installation', 'hardware_maintenance', 'office_technology',
            'satellite_installation', 'networking', 'pc_maintenance',
            'solar_installation', 'cctv_installation', 'gsm_repairs',
            'electrical_construction') NOT NULL,
  budget DECIMAL(10,2),
  location VARCHAR(255),
  post_date DATETIME DEFAULT CURRENT_TIMESTAMP,
  deadline DATETIME,
  status ENUM('open', 'assigned', 'in_progress', 'completed', 'cancelled') DEFAULT 'open',
  requires_consultancy BOOLEAN DEFAULT FALSE,
  FOREIGN KEY (client_id) REFERENCES users(user_id)
);
```

Bids

```
sqlCREATE TABLE bids (
  bid_id INT PRIMARY KEY AUTO_INCREMENT,
  project_id INT NOT NULL,
  artisan_id INT NOT NULL,
  bid_amount DECIMAL(10,2) NOT NULL,
  proposal TEXT,
  estimated_completion DATETIME,
  bid_date DATETIME DEFAULT CURRENT_TIMESTAMP,
  status ENUM('pending', 'accepted', 'rejected') DEFAULT 'pending',
  FOREIGN KEY (project_id) REFERENCES projects(project_id),
  FOREIGN KEY (artisan_id) REFERENCES users(user_id)
);
```

Transactions

```
sqlCREATE TABLE transactions (
  transaction_id INT PRIMARY KEY AUTO_INCREMENT,
  project_id INT NOT NULL,
  client_id INT NOT NULL,
  artisan_id INT NOT NULL,
  amount DECIMAL(10,2) NOT NULL,
  transaction_type ENUM('project_payment', 'consultancy_fee', 'platform_fee') NOT NULL,
  status ENUM('pending', 'completed', 'refunded', 'disputed') DEFAULT 'pending',
  transaction_date DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (project_id) REFERENCES projects(project_id),
  FOREIGN KEY (client_id) REFERENCES users(user_id),
  FOREIGN KEY (artisan_id) REFERENCES users(user_id)
);
```

Reviews

```
sqlCREATE TABLE reviews (
  review_id INT PRIMARY KEY AUTO_INCREMENT,
  project_id INT NOT NULL,
  reviewer_id INT NOT NULL,
  reviewee_id INT NOT NULL,
  rating INT NOT NULL CHECK (rating BETWEEN 1 AND 5),
  comment TEXT,
  review_date DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (project_id) REFERENCES projects(project_id),
  FOREIGN KEY (reviewer_id) REFERENCES users(user_id),
  FOREIGN KEY (reviewee_id) REFERENCES users(user_id)
```

);

Consultancy_Services

sqlCREATE TABLE consultancy_services (

service_id INT PRIMARY KEY AUTO_INCREMENT,

project_id INT NOT NULL,

consultant_id INT NOT NULL,

service_description TEXT,

fee DECIMAL(10,2) NOT NULL,

status ENUM('requested', 'in_progress', 'completed', 'cancelled') DEFAULT 'requested',

FOREIGN KEY (project_id) REFERENCES projects(project_id),

FOREIGN KEY (consultant_id) REFERENCES users(user_id)

);

Disputes

sqlCREATE TABLE disputes (

dispute_id INT PRIMARY KEY AUTO_INCREMENT,

project_id INT NOT NULL,

complainant_id INT NOT NULL,

respondent_id INT NOT NULL,

issue TEXT NOT NULL,

status ENUM('open', 'under_review', 'resolved', 'closed') DEFAULT 'open',

resolution TEXT,

created_date DATETIME DEFAULT CURRENT_TIMESTAMP,

resolved_date DATETIME,

FOREIGN KEY (project_id) REFERENCES projects(project_id),

```
  FOREIGN KEY (complainant_id) REFERENCES users(user_id),
  FOREIGN KEY (respondent_id) REFERENCES users(user_id)
);
```

Database Connection Setup

Configuration File Structure

Create a config.php file that will store database connection parameters:

```
php<?php
// Database configuration
define('DB_SERVER', 'localhost');
define('DB_USERNAME', 'username');
define('DB_PASSWORD', 'password');
define('DB_NAME', 'artisan_network_portal');

// Attempt to connect to MySQL database
$conn = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD, DB_NAME);

// Check connection
if($conn === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}
?>
```

Connection Class (Using PDO for added security)

Create a Database.php class:

```
php<?php
class Database {
```

```
    private $host = "localhost";
    private $db_name = "artisan_network_portal";
    private $username = "username";
    private $password = "password";
    public $conn;

    // Get database connection
    public function getConnection() {
        $this->conn = null;

        try {
            $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->db_name, $this->username, $this->password);
            $this->conn->exec("set names utf8");
            $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        } catch(PDOException $exception) {
            echo "Connection error: " . $exception->getMessage();
        }

        return $this->conn;
    }
}
?>
```

Business Rules Implementation

User Registration and Authentication

Client Registration

```
phpfunction registerClient($userData) {
    // Validate required fields
    if(empty($userData['full_name']) || empty($userData['email']) || empty($userData['phone']) ||
       empty($userData['username']) || empty($userData['password'])) {
        return ["status" => "error", "message" => "All fields are required"];
    }

    // Hash password
    $hashedPassword = password_hash($userData['password'], PASSWORD_DEFAULT);

    // Insert user first
    $query = "INSERT INTO users (username, password, email, phone, full_name, user_type)
              VALUES (?, ?, ?, ?, ?, 'client')";

    // Execute query with prepared statements
    // Then insert into client_profiles

    return ["status" => "success", "message" => "Registration successful"];
}
```

Artisan Registration

```
phpfunction registerArtisan($userData, $profileData) {
    // Validate required fields
    // Hash password
    // Insert into users table
    // Insert into artisan_profiles table
```

```
    return ["status" => "success", "message" => "Registration successful, pending verification"];
}
```

Authentication

```
phpfunction authenticateUser($username, $password) {
    // Query to find user
    $query = "SELECT user_id, username, password, user_type, account_status FROM users
WHERE username = ?";

    // Execute query with prepared statements
    // Verify password with password_verify()
    // Check account status
    // Return user data or error
}
```

Project Management

Creating a Project

```
phpfunction createProject($projectData, $clientId) {
    // Validate project data
    // Insert into projects table
    // Return project ID
}
```

Submitting a Bid

```
phpfunction submitBid($bidData, $artisanId) {
    // Check if artisan is verified
```

```
    // Check if project is still open
    // Insert bid into bids table
}
```

Accepting a Bid

```
phpfunction acceptBid($bidId, $clientId) {
    // Verify client owns the project
    // Update bid status to accepted
    // Update project status to assigned
    // Reject all other bids
}
```

Payment Processing

Processing Project Payments

```
phpfunction processPayment($projectId, $clientId, $artisanId, $amount) {
    // Validate payment information
    // Create transaction record
    // Handle platform fees (10% of the transaction)
    // Update project status
}
```

Consultancy Fee Processing

```
phpfunction processConsultancyFee($projectId, $clientId, $consultantId, $amount) {
    // Similar to processPayment but for consultancy services
}
```

Review and Rating System

```
phpfunction submitReview($projectId, $reviewerId, $revieweeId, $rating, $comment) {
    // Verify project is completed
    // Verify reviewer was involved in the project
    // Insert review
    // Update average rating for the reviewee
}
```

Front-end Integration Guidelines

Authentication Integration

Login Form Connection

```
javascript// Example using fetch API
document.getElementById('loginForm').addEventListener('submit', async (e) => {
    e.preventDefault();

    const formData = new FormData(e.target);

    try {
        const response = await fetch('api/auth/login.php', {
            method: 'POST',
            body: formData
        });

        const data = await response.json();

        if (data.status === 'success') {
            // Store token and redirect
            localStorage.setItem('authToken', data.token);
```

```
            window.location.href = data.redirect;
        } else {
            // Show error message
            document.getElementById('errorMsg').textContent = data.message;
        }
    } catch (error) {
        console.error('Login error:', error);
    }
});
```

Registration Form Connection

javascript// Similar implementation to login but with more fields and validation

Project Management Integration

Creating Project Form

```
javascript// Example for submitting a new project
async function submitNewProject(projectData) {
    try {
        const token = localStorage.getItem('authToken');

        const response = await fetch('api/projects/create.php', {
            method: 'POST',
            headers: {
                'Authorization': `Bearer ${token}`,
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(projectData)
```

```
        });

        const result = await response.json();
        return result;
    } catch (error) {
        console.error('Error creating project:', error);
        throw error;
    }
}
```

Bid Submission Form

javascript// Similar implementation for bid submission

Security Best Practices

Input Validation

Validate all user inputs on both client and server side

Use prepared statements for all database queries

Implement CSRF protection

Authentication Security

Use JWT (JSON Web Tokens) for authentication

Implement token expiration and refresh mechanisms

Store passwords with strong hashing (PHP's password_hash)

API Security

Implement rate limiting

Validate request origins

Use HTTPS for all connections

Implementation Checklist

Database Setup

- [ ] Create database schema
- [ ] Set up user permissions
- [ ] Implement foreign key constraints

Backend API Development

- [ ] Set up connection classes
- [ ] Implement user authentication endpoints
- [ ] Create project management endpoints
- [ ] Develop payment processing system
- [ ] Build review and rating system

Front-end Integration

- [ ] Implement authentication forms
- [ ] Create project management interfaces
- [ ] Design bidding system UI
- [ ] Develop payment processing forms
- [ ] Build review submission components

Testing Procedures

Database Connection Testing

Test connection from development environment

Verify proper transaction handling

Validate error handling for failed connections

User Management Testing

Test registration process for all user types

Verify login functionality

Test password reset flow

Project Management Testing

Test project creation

Verify bid submission and acceptance

Test project completion flow

Payment Processing Testing

Test payment transactions

Verify fee calculations

Test refund processes