Database Schema & System Architecture

Community Skills Platform - Technical Reference

📊 COMPLETE DATABASE SCHEMA

Sprint 3 Tables (Core/Essential)

1. USERS TABLE

```sql
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    bio TEXT,
    location VARCHAR(255),
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    profile_pic VARCHAR(255) DEFAULT 'default-avatar.png',
    points INT DEFAULT 0,
    average_rating DECIMAL(3, 2) DEFAULT 0.00,
    is_active BOOLEAN DEFAULT 1,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    INDEX idx_location (latitude, longitude),
    INDEX idx_email (email)
);
```

Example Data:

```sql
```

```
INSERT INTO users (email, password_hash, first_name, last_name, bio, location, latitude, longitude) VALUES
('sarah.student@university.ac.uk', '$2b$10$...', 'Sarah', 'Johnson',
 'Second year CS student. Love helping with math and borrowing books!',
 'Roehampton, London', 51.4567, -0.2634),

('david.handy@email.com', '$2b$10$...', 'David', 'Wilson',
 'Retired carpenter with 40 years experience. Happy to help with repairs!',
 'Richmond, London', 51.4613, -0.3037),

('maya.parent@email.com', '$2b$10$...', 'Maya', 'Patel',
 'Working mum of two. Have lots of kids items to share!',
 'Kingston, London', 51.4123, -0.3007);
```

---

2. LISTINGS TABLE

sql

```
CREATE TABLE listings (
    listing_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    type ENUM('skill', 'item') NOT NULL,
    category VARCHAR(100) NOT NULL,
    condition_note VARCHAR(255),  -- For items: 'Like new', 'Good', 'Fair'
    photo_url VARCHAR(255) DEFAULT 'default-listing.png',
    is_available BOOLEAN DEFAULT 1,
    view_count INT DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    INDEX idx_type (type),
    INDEX idx_category (category),
    INDEX idx_user (user_id),
    INDEX idx_available (is_available)
);
```

Categories:

- Tools & Equipment
- Books & Media
- Skills & Teaching
- Household Items

- Sports & Hobbies
- Technology
- Childcare & Baby Items
- Other

Example Data:

```sql
INSERT INTO listings (user_id, title, description, type, category, photo_url) VALUES
(1, 'Math Tutoring - GCSE & A-Level',
 'I can help with math homework and exam prep. Got an A* in my A-levels!',
 'skill', 'Skills & Teaching', 'math-tutor.jpg'),

(2, 'Power Drill Set',
 'Professional cordless drill with bits. Perfect for DIY projects.',
 'item', 'Tools & Equipment', 'drill.jpg'),

(3, 'Baby Clothes Bundle 6-12 months',
 'Barely worn baby clothes, outgrown quickly! Includes 10 outfits.',
 'item', 'Childcare & Baby Items', 'baby-clothes.jpg'),

(2, 'Basic Carpentry Skills',
 'Can teach basic woodworking, furniture repair, or help with projects.',
 'skill', 'Skills & Teaching', 'carpentry.jpg');
```

---

3. TAGS TABLE

```sql
CREATE TABLE tags (
    tag_id INT AUTO_INCREMENT PRIMARY KEY,
    tag_name VARCHAR(50) UNIQUE NOT NULL,
    category VARCHAR(100),
    usage_count INT DEFAULT 0,

    INDEX idx_tag_name (tag_name)
);
```

Example Data:

```sql
```

```sql
INSERT INTO tags (tag_name, category) VALUES
('mathematics', 'Skills & Teaching'),
('tutoring', 'Skills & Teaching'),
('GCSE', 'Skills & Teaching'),
('tools', 'Tools & Equipment'),
('DIY', 'Tools & Equipment'),
('baby', 'Childcare & Baby Items'),
('clothes', 'Childcare & Baby Items'),
('woodworking', 'Skills & Teaching'),
('repairs', 'Skills & Teaching'),
('power-tools', 'Tools & Equipment');
```

---

4. LISTING_TAGS TABLE (Junction Table)

```sql
CREATE TABLE listing_tags (
    listing_id INT NOT NULL,
    tag_id INT NOT NULL,

    PRIMARY KEY (listing_id, tag_id),
    FOREIGN KEY (listing_id) REFERENCES listings(listing_id) ON DELETE CASCADE,
    FOREIGN KEY (tag_id) REFERENCES tags(tag_id) ON DELETE CASCADE,

    INDEX idx_listing (listing_id),
    INDEX idx_tag (tag_id)
);
```

Example Data:

```sql
INSERT INTO listing_tags (listing_id, tag_id) VALUES
(1, 1), -- Math tutoring -> mathematics
(1, 2), -- Math tutoring -> tutoring
(1, 3), -- Math tutoring -> GCSE
(2, 4), -- Drill -> tools
(2, 5), -- Drill -> DIY
(2, 10), -- Drill -> power-tools
(3, 6), -- Baby clothes -> baby
(3, 7); -- Baby clothes -> clothes
```

---

5. REQUESTS TABLE

```sql
CREATE TABLE requests (
    request_id INT AUTO_INCREMENT PRIMARY KEY,
    listing_id INT NOT NULL,
    requester_id INT NOT NULL,
    status ENUM('pending', 'accepted', 'declined', 'completed', 'cancelled') DEFAULT 'pending',
    message TEXT,
    requested_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    responded_date TIMESTAMP NULL,
    completed_date TIMESTAMP NULL,

    FOREIGN KEY (listing_id) REFERENCES listings(listing_id) ON DELETE CASCADE,
    FOREIGN KEY (requester_id) REFERENCES users(user_id) ON DELETE CASCADE,

    INDEX idx_listing (listing_id),
    INDEX idx_requester (requester_id),
    INDEX idx_status (status),

    -- Prevent duplicate pending requests
    UNIQUE KEY unique_pending_request (listing_id, requester_id, status)
);
```

Example Data:

```sql
INSERT INTO requests (listing_id, requester_id, status, message) VALUES
(2, 1, 'accepted', 'Hi! I need to drill some holes for shelves. Can I borrow for the weekend?'),
(1, 3, 'pending', 'Hi Sarah! My daughter is struggling with GCSE math. Could you help?'),
(3, 2, 'completed', 'Perfect for my granddaughter! Thank you!');
```

Sprint 4 Tables (Advanced Features)

6. RATINGS TABLE

```sql
```

```sql
CREATE TABLE ratings (
    rating_id INT AUTO_INCREMENT PRIMARY KEY,
    request_id INT NOT NULL,
    rater_id INT NOT NULL,      -- Person giving the rating
    rated_id INT NOT NULL,      -- Person being rated
    score INT NOT NULL CHECK (score BETWEEN 1 AND 5),
    comment TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (request_id) REFERENCES requests(request_id) ON DELETE CASCADE,
    FOREIGN KEY (rater_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (rated_id) REFERENCES users(user_id) ON DELETE CASCADE,

    -- Each user can only rate once per request
    UNIQUE KEY unique_rating (request_id, rater_id),

    INDEX idx_rated_user (rated_id),
    INDEX idx_score (score)
);
```

Example Data:

sql

```sql
INSERT INTO ratings (request_id, rater_id, rated_id, score, comment) VALUES
(1, 1, 2, 5, 'David was super helpful and the drill was in perfect condition!'),
(1, 2, 1, 5, 'Sarah was respectful and returned everything on time. Great!'),
(3, 2, 3, 4, 'Nice clothes, exactly as described. Quick pickup too.');
```

---

7. MESSAGES TABLE (If implementing)

sql

```sql
CREATE TABLE messages (
    message_id INT AUTO_INCREMENT PRIMARY KEY,
    sender_id INT NOT NULL,
    receiver_id INT NOT NULL,
    request_id INT,  -- Optional: link to specific request
    content TEXT NOT NULL,
    is_read BOOLEAN DEFAULT 0,
    sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (sender_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (receiver_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (request_id) REFERENCES requests(request_id) ON DELETE SET NULL,

    INDEX idx_sender (sender_id),
    INDEX idx_receiver (receiver_id),
    INDEX idx_conversation (sender_id, receiver_id),
    INDEX idx_unread (receiver_id, is_read)
);
```

---

8. USER_PREFERENCES TABLE (Optional - for matching algorithm)

sql

```sql
CREATE TABLE user_preferences (
    preference_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL UNIQUE,
    preferred_categories JSON,  -- Store array of preferred categories
    max_distance_km INT DEFAULT 10,
    notification_enabled BOOLEAN DEFAULT 1,

    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

Example Data:

sql

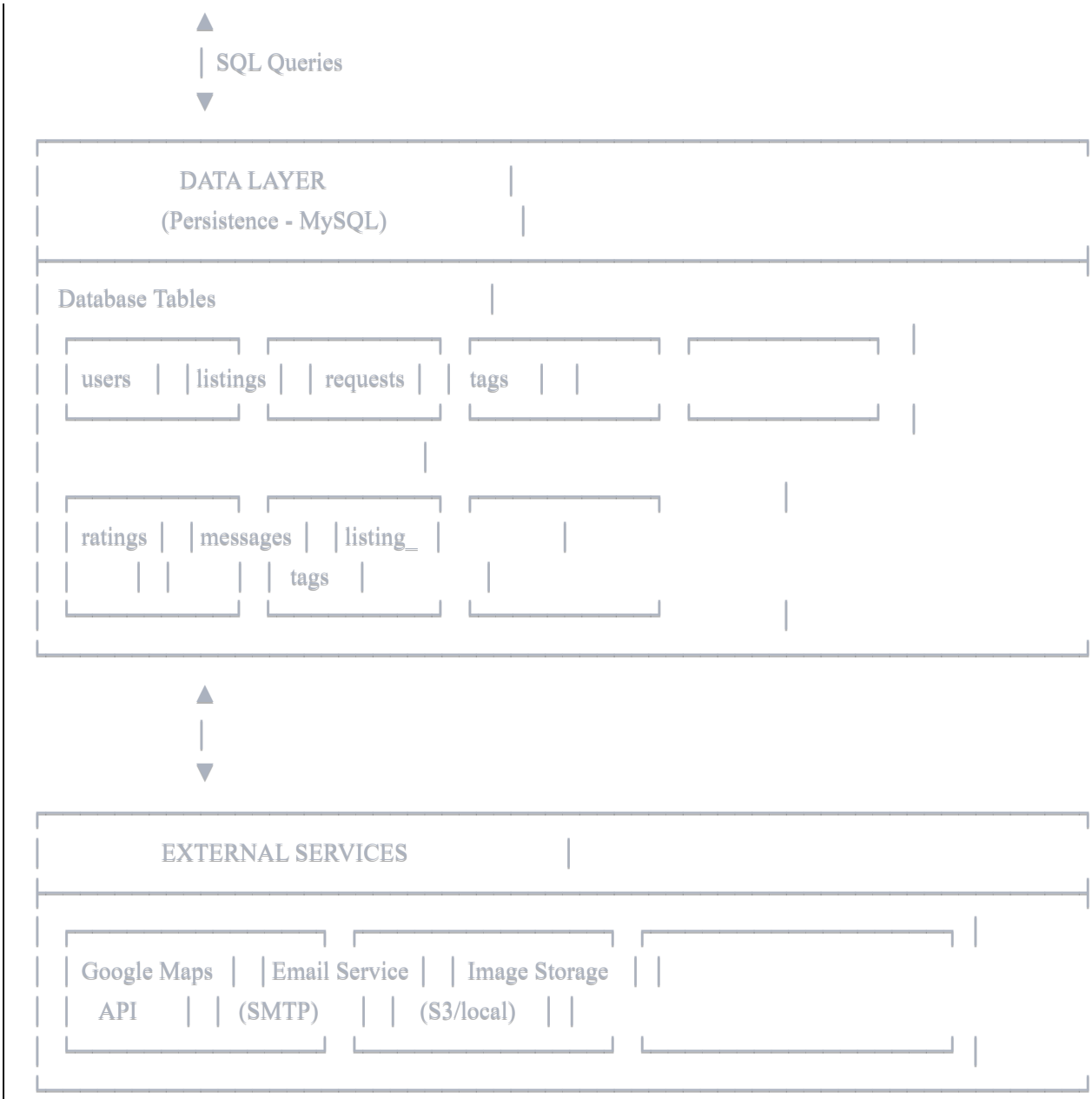
```sql
INSERT INTO user_preferences (user_id, preferred_categories, max_distance_km) VALUES
(1, '["Books & Media", "Skills & Teaching", "Technology"]', 5),
(2, '["Tools & Equipment", "DIY"]', 15),
(3, '["Childcare & Baby Items", "Household Items"]', 8);
```

🔗 DATABASE RELATIONSHIPS (ERD)


TAGS
tag_id (PK)
tag_name
category
REQUESTS
request_id (PK)
listing_id (FK)
requester_id(FK)
status
message
requested_date
ONE REQUEST
CAN HAVE
MULTIPLE RATINGS
(from both users)
ONE REQUEST
CAN HAVE
MESSAGES
MESSAGES
message_id (PK)
sender_id (FK)
receiver_id(FK)
request_id (FK)
content
is_read
sent_at
RATINGS
rating_id (PK)
request_id (FK)
rater_id (FK)
rated_id (FK)
score (1-5)
comment

SYSTEM ARCHITECTURE

Layer Architecture



📁 FILE STRUCTURE (Detailed)

```
community-skills-platform/
│
├── 📁 .github/
│   └── workflows/
│       ├── lint.yml            # ESLint check on push
│       ├── test.yml            # Run tests automatically
│       └── docker-build.yml    # Build Docker image
│
├── 📁 config/
│   ├── database.js             # MySQL connection pool
│   └── session.js              # Session configuration
│
├── 📁 db/
```

```
│   ├── schema.sql              # CREATE TABLE statements
│   ├── seed.sql                # Test data INSERT statements
│   └── migrations/             # Version-controlled DB changes
│       ├── 001_initial_tables.sql
│       ├── 002_add_ratings.sql
│       └── 003_add_messages.sql
│
├── 📁 middleware/
│   ├── auth.js                 # isAuthenticated, isOwner checks
│   ├── validation.js           # Input validation middleware
│   ├── errorHandler.js         # Centralized error handling
│   └── upload.js               # File upload (multer) config
│
├── 📁 routes/
│   ├── index.js                # Homepage routes
│   ├── auth.js                 # Register, login, logout
│   ├── users.js                # User profile, list, update
│   ├── listings.js             # Browse, detail, create, edit
│   ├── requests.js             # Make, accept, decline requests
│   ├── ratings.js              # Submit, view ratings (Sprint 4)
│   ├── matching.js             # Recommendations (Sprint 4)
│   └── messages.js             # Send, view messages (Sprint 4)
│
├── 📁 views/
│   ├── layout.pug              # Master template (header, footer)
│   ├── index.pug               # Homepage
│   ├── error.pug               # Error page (404, 500)
│   │
│   ├── 📁 auth/
│   │   ├── register.pug        # Registration form
│   │   ├── login.pug           # Login form
│   │   └── forgot-password.pug # Password reset (optional)
│   │
│   ├── 📁 users/
│   │   ├── list.pug            # All users (Sprint 3 required)
│   │   ├── profile.pug         # User profile (Sprint 3 required)
│   │   └── edit.pug            # Edit own profile
│   │
│   ├── 📁 listings/
│   │   ├── list.pug            # Browse listings (Sprint 3 required)
│   │   ├── detail.pug          # Listing detail (Sprint 3 required)
│   │   ├── create.pug          # Create listing form
│   │   ├── edit.pug            # Edit listing form
│   │   └── map.pug             # Map view (Sprint 4)
│   │
│   ├── 📁 requests/
│   │   ├── make.pug            # Request form
```

```
│   │   ├── list.pug              # User's requests
│   │   └── detail.pug            # Request detail
│   │
│   ├── dashboard/
│   │   ├── index.pug             # User dashboard
│   │   ├── my-listings.pug       # Manage listings
│   │   └── my-requests.pug       # Manage requests
│   │
│   └── components/               # Reusable components
│       ├── listing-card.pug      # Listing preview card
│       ├── user-card.pug         # User preview card
│       ├── navigation.pug        # Nav bar
│       └── footer.pug            # Footer
│
├── public/
│   ├── css/
│   │   ├── style.css             # Main stylesheet
│   │   ├── components.css        # Component styles
│   │   └── responsive.css        # Mobile responsive
│   │
│   ├── js/
│   │   ├── main.js               # Main JavaScript
│   │   ├── validation.js         # Client-side validation
│   │   ├── map.js                # Google Maps integration
│   │   └── messaging.js          # Real-time messaging (Sprint 4)
│   │
│   └── images/
│       ├── logo.png
│       ├── default-avatar.png
│       ├── default-listing.png
│       └── uploads/              # User uploaded images
│
├── utils/
│   ├── geocode.js                # Address to lat/lng conversion
│   ├── distance.js               # Calculate distances
│   ├── points.js                 # Points calculation logic
│   └── email.js                  # Email sending functions
│
├── tests/                        # Automated tests (optional)
│   ├── routes.test.js
│   ├── database.test.js
│   └── matching.test.js
│
├── app.js                        # Main Express application
├── server.js                     # Server startup
├── package.json                  # Dependencies
├── package-lock.json
```

```
├── .env                    # Environment variables (git ignored)
├── .env.example            # Template for environment vars
├── .gitignore
├── .eslintrc.json          # Linting configuration
├── docker-compose.yml      # Docker services
├── Dockerfile              # Node.js container
└── README.md               # Project documentation
```

KEY ROUTES & ENDPOINTS

Public Routes (No login required)

```
GET  /                      Homepage
GET  /about                 About page
GET  /how-it-works          How it works page

GET  /auth/register         Registration form
POST /auth/register         Create new account
GET  /auth/login            Login form
POST /auth/login            Authenticate user
GET  /auth/logout           Log out user

GET  /users                 List all users (Sprint 3 required)
GET  /users/:id             View user profile (Sprint 3 required)

GET  /listings              Browse all listings (Sprint 3 required)
GET  /listings/:id          View listing detail (Sprint 3 required)
GET  /listings/category/:cat  Filter by category (Sprint 3 required)
GET  /listings/search       Search listings
```

Protected Routes (Login required)

```
GET  /dashboard             User dashboard

GET  /profile/edit          Edit own profile
POST /profile/update        Update profile

GET  /listings/create       Create listing form
POST /listings/create       Save new listing
GET  /listings/:id/edit     Edit listing form
POST /listings/:id/update   Update listing
POST /listings/:id/delete   Delete listing

POST /requests/create       Make a request
```

```
GET  /requests/received       Requests on my listings
GET  /requests/sent           Requests I've made
POST /requests/:id/accept     Accept a request
POST /requests/:id/decline    Decline a request
POST /requests/:id/complete   Mark as completed
POST /requests/:id/cancel     Cancel my request
```

Sprint 4 Routes (Advanced)

```
GET  /recommendations         Personalized recommendations
GET  /listings/map            Map view of listings
GET  /listings/nearby         Nearby listings (with distance)

POST /ratings/submit          Submit a rating
GET  /ratings/user/:id        View user's ratings

GET  /messages                Inbox/message list
GET  /messages/:userId        Conversation with specific user
POST /messages/send           Send a message
```

🔍 SAMPLE SQL QUERIES

For Sprint 3 Pages

1. Users List Page

sql

```sql
-- Get all users with their listing count and average rating
SELECT
    u.user_id,
    u.first_name,
    u.last_name,
    u.location,
    u.profile_pic,
    u.points,
    u.average_rating,
    u.created_at,
    COUNT(DISTINCT l.listing_id) as listing_count
FROM users u
LEFT JOIN listings l ON u.user_id = l.user_id AND l.is_available = 1
GROUP BY u.user_id
ORDER BY u.points DESC, u.created_at DESC;
```

2. User Profile Page

sql

```sql
-- Get specific user's details with all their listings
SELECT
    u.*,
    COUNT(DISTINCT l.listing_id) as total_listings,
    COUNT(DISTINCT r.request_id) as completed_exchanges
FROM users u
LEFT JOIN listings l ON u.user_id = l.user_id
LEFT JOIN requests r ON u.user_id = r.requester_id AND r.status = 'completed'
WHERE u.user_id = ?
GROUP BY u.user_id;

-- Get user's active listings
SELECT * FROM listings
WHERE user_id = ? AND is_available = 1
ORDER BY created_at DESC;

-- Get user's ratings (Sprint 4)
SELECT r.*, u.first_name, u.last_name
FROM ratings r
JOIN users u ON r.rater_id = u.user_id
WHERE r.rated_id = ?
ORDER BY r.created_at DESC
LIMIT 10;
```

3. Listings Page

sql

```sql
-- Get all listings with owner info and tags
SELECT
    l.*,
    u.first_name,
    u.last_name,
    u.average_rating,
    GROUP_CONCAT(DISTINCT t.tag_name) as tags,
    COUNT(DISTINCT r.request_id) as request_count
FROM listings l
JOIN users u ON l.user_id = u.user_id
LEFT JOIN listing_tags lt ON l.listing_id = lt.listing_id
LEFT JOIN tags t ON lt.tag_id = t.tag_id
LEFT JOIN requests r ON l.listing_id = r.listing_id
WHERE l.is_available = 1
GROUP BY l.listing_id
ORDER BY l.created_at DESC;
```

4. Listing Detail Page

sql

```sql
-- Get specific listing with all details
SELECT
    l.*,
    u.user_id,
    u.first_name,
    u.last_name,
    u.location,
    u.profile_pic,
    u.average_rating,
    u.points
FROM listings l
JOIN users u ON l.user_id = u.user_id
WHERE l.listing_id = ?;

-- Get listing's tags
SELECT t.* FROM tags t
JOIN listing_tags lt ON t.tag_id = lt.tag_id
WHERE lt.listing_id = ?;

-- Get request count
SELECT COUNT(*) as request_count
FROM requests
WHERE listing_id = ?;
```

5. Search Query

sql

```sql
-- Search listings by keyword
SELECT DISTINCT
    l.*,
    u.first_name,
    u.last_name
FROM listings l
JOIN users u ON l.user_id = u.user_id
LEFT JOIN listing_tags lt ON l.listing_id = lt.listing_id
LEFT JOIN tags t ON lt.tag_id = t.tag_id
WHERE l.is_available = 1
AND (
    l.title LIKE CONCAT('%', ?, '%')
    OR l.description LIKE CONCAT('%', ?, '%')
    OR l.category LIKE CONCAT('%', ?, '%')
    OR t.tag_name LIKE CONCAT('%', ?, '%')
)
ORDER BY l.created_at DESC;
```

For Sprint 4 Features

6. Matching Algorithm Query

sql

```sql
-- Get recommendations based on user's request history
SELECT
    l.*,
    u.first_name,
    u.last_name,
    ( 6371 * acos( cos( radians(?) )
    * cos( radians( u.latitude ) )
    * cos( radians( u.longitude ) - radians(?) )
    + sin( radians(?) )
    * sin( radians( u.latitude ) ) ) ) AS distance,
    COUNT(DISTINCT r.request_id) as popularity
FROM listings l
JOIN users u ON l.user_id = u.user_id
LEFT JOIN requests r ON l.listing_id = r.listing_id
WHERE l.is_available = 1
AND l.user_id != ?
AND l.category IN (
    SELECT DISTINCT l2.category
    FROM requests r2
    JOIN listings l2 ON r2.listing_id = l2.listing_id
    WHERE r2.requester_id = ?
)
GROUP BY l.listing_id
HAVING distance < 10
ORDER BY popularity DESC, distance ASC
LIMIT 10;
```

7. Nearby Listings with Distance

sql

```sql
-- Find listings within X km of user
SELECT
    l.*,
    u.first_name,
    u.last_name,
    u.latitude,
    u.longitude,
    ( 6371 * acos( cos( radians(?) )
    * cos( radians( u.latitude ) )
    * cos( radians( u.longitude ) - radians(?) )
    + sin( radians(?) )
    * sin( radians( u.latitude ) ) ) ) AS distance
FROM listings l
JOIN users u ON l.user_id = u.user_id
WHERE l.is_available = 1
AND u.latitude IS NOT NULL
AND u.longitude IS NOT NULL
HAVING distance < ?
ORDER BY distance ASC;
```

8. Calculate User's Average Rating

sql

```sql
-- Recalculate after new rating submitted
SELECT AVG(score) as avg_rating, COUNT(*) as rating_count
FROM ratings
WHERE rated_id = ?;

-- Update user's average rating
UPDATE users
SET average_rating = ?
WHERE user_id = ?;
```

9. Award Points

sql

```sql
-- When request is accepted, award points to listing owner
UPDATE users
SET points = points + 10
WHERE user_id = ?;

-- When request is completed, award bonus points
UPDATE users
SET points = points + 5
WHERE user_id IN (?, ?);  -- Both users get bonus
```

🎨 CSS FRAMEWORK & STYLING

Color Scheme (Suggested)

```css
```

```css
:root {
  /* Primary Colors */
  --primary-green: #2E7D32;       /* Trust, community, nature */
  --primary-light: #66BB6A;
  --primary-dark: #1B5E20;

  /* Secondary Colors */
  --secondary-orange: #FFA726;    /* Warmth, sharing */
  --secondary-light: #FFD54F;
  --secondary-dark: #F57C00;

  /* Accent */
  --accent-blue: #1976D2;         /* Action, links */
  --accent-light: #42A5F5;

  /* Neutral */
  --gray-100: #F5F5F5;
  --gray-300: #E0E0E0;
  --gray-500: #9E9E9E;
  --gray-700: #616161;
  --gray-900: #212121;

  /* Semantic Colors */
  --success: #4CAF50;
  --warning: #FF9800;
  --error: #F44336;
  --info: #2196F3;

  /* Typography */
  --font-heading: 'Poppins', sans-serif;
  --font-body: 'Open Sans', sans-serif;
}
```

Key Component Styles

```
css
```

```css
/* Listing Card */
.listing-card {
    background: white;
    border-radius: 12px;
    box-shadow: 0 2px 8px rgba(0,0,0,0.1);
    transition: transform 0.2s, box-shadow 0.2s;
}

.listing-card:hover {
    transform: translateY(-4px);
    box-shadow: 0 4px 16px rgba(0,0,0,0.15);
}

/* User Badge */
.user-rating {
    display: inline-flex;
    align-items: center;
    padding: 4px 12px;
    background: var(--gray-100);
    border-radius: 20px;
}

/* Category Tag */
.tag {
    display: inline-block;
    padding: 4px 12px;
    background: var(--primary-light);
    color: white;
    border-radius: 16px;
    font-size: 0.875rem;
}

/* Request Button */
.btn-request {
    background: var(--secondary-orange);
    color: white;
    padding: 12px 24px;
    border: none;
    border-radius: 8px;
    font-weight: 600;
    cursor: pointer;
    transition: background 0.2s;
}

.btn-request:hover {
```

```
    background: var(--secondary-dark);
}
```

🐳 DOCKER CONFIGURATION

docker-compose.yml

yaml

```yaml
version: '3.8'

services:
  app:
    build: .
    ports:
      - "3000:3000"
    volumes:
      - .:/app
      - /app/node_modules
    environment:
      - NODE_ENV=development
      - DB_HOST=db
      - DB_USER=root
      - DB_PASSWORD=password
      - DB_NAME=community_skills
      - SESSION_SECRET=your_secret_key_here
    depends_on:
      - db
    command: npm run dev

  db:
    image: mysql:8.0
    ports:
      - "3306:3306"
    environment:
      - MYSQL_ROOT_PASSWORD=password
      - MYSQL_DATABASE=community_skills
    volumes:
      - mysql_data:/var/lib/mysql
      - ./db/schema.sql:/docker-entrypoint-initdb.d/1-schema.sql
      - ./db/seed.sql:/docker-entrypoint-initdb.d/2-seed.sql

volumes:
  mysql_data:
```

Dockerfile

dockerfile

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

📋 ENVIRONMENT VARIABLES (.env)

env

```
# Server
NODE_ENV=development
PORT=3000

# Database
DB_HOST=db
DB_USER=root
DB_PASSWORD=password
DB_NAME=community_skills
DB_PORT=3306

# Session
SESSION_SECRET=your_very_secure_random_string_here

# Sprint 4 - External APIs
GOOGLE_MAPS_API_KEY=your_google_maps_api_key

# Sprint 4 - Email (optional)
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USER=your_email@gmail.com
EMAIL_PASSWORD=your_app_password

# Sprint 4 - File Upload
UPLOAD_PATH=./public/images/uploads
MAX_FILE_SIZE=5242880  # 5MB in bytes
```

This document is your technical reference for the entire project. Keep it alongside your main roadmap!