Community Skill & Item Sharing Platform

Complete Project Roadmap & Reference Guide

Project Goal: Build a local community platform where people share skills and items based on trust and mutual help, not money.

Tech Stack: Node.js, Express.js, MySQL, PUG, Docker, GitHub Actions

Target Grade: Distinction ⭐

🎯 PROJECT OVERVIEW

What You're Building

A web application where community members can:

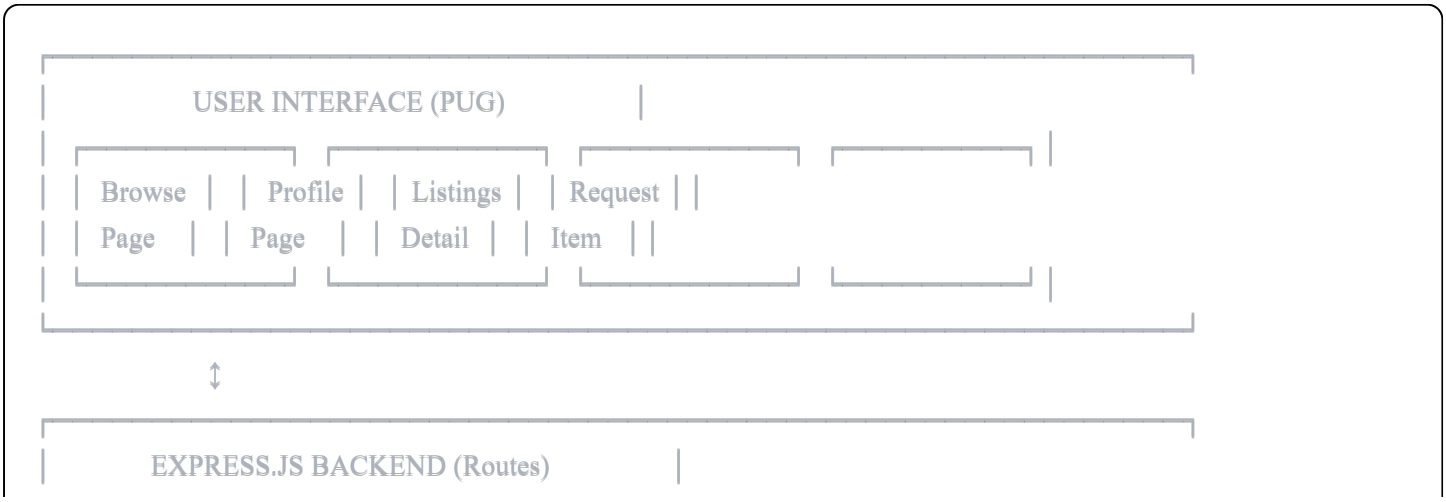
- Create profiles with their skills and available items
- Browse what others are offering
- Request skills/items from neighbors
- Build trust through a points/ratings system
- Find nearby resources using maps
- Exchange help without money

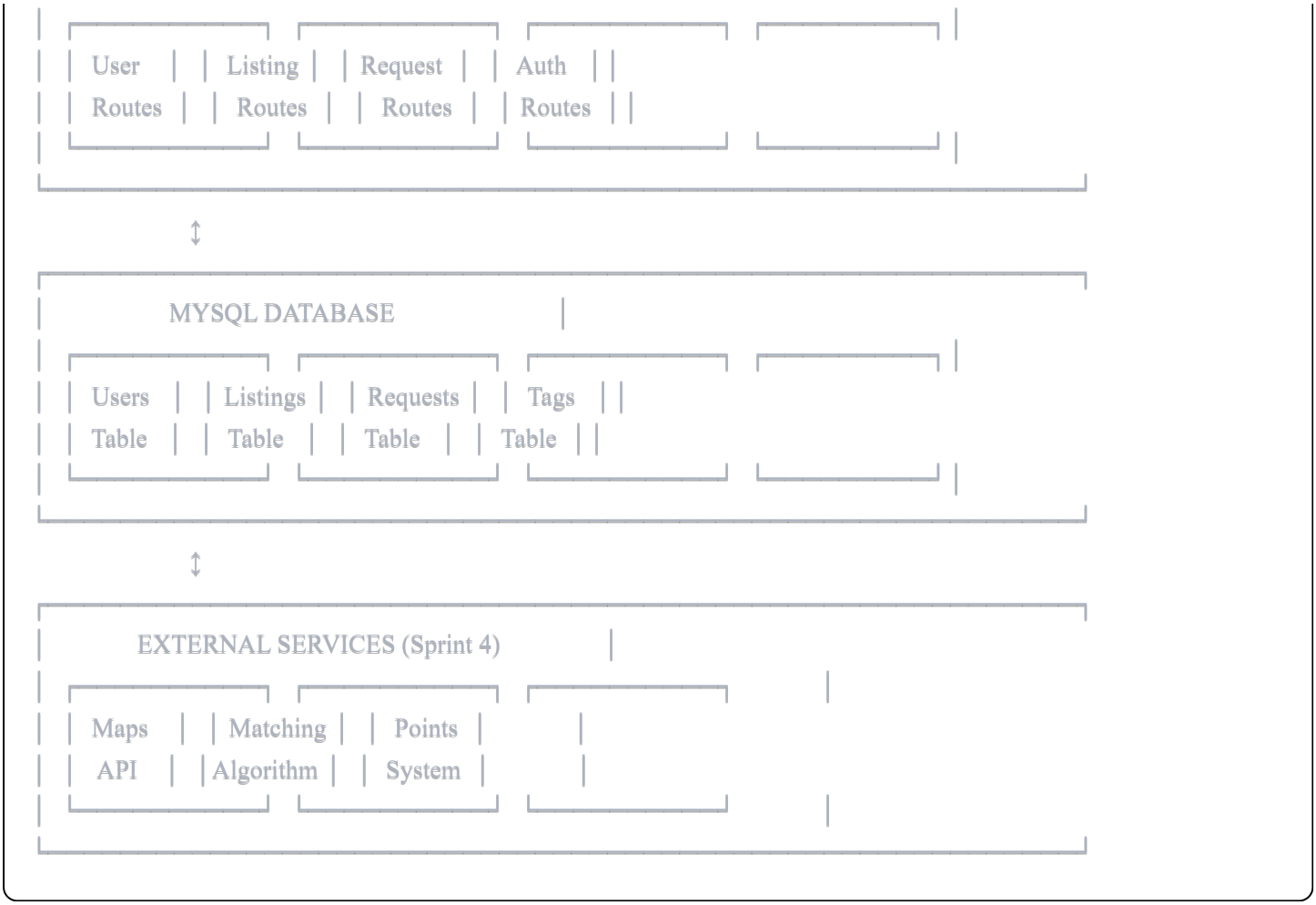
Core Concept

Simple Start (Sprint 3): Basic listings → Browse → Request
Enhanced (Sprint 4): + Smart matching + Location maps + Points system + (Messaging if time)

📊 PROJECT ARCHITECTURE

High-Level Structure





🗓️ SPRINT-BY-SPRINT BREAKDOWN

SPRINT 1: Setup & Planning (Weeks 1-4)

Due: Lab Week 4
Grade Weight: Foundation (not directly graded but essential)

Goal: Get your team organized and project planned

Deliverables Checklist:

✅ GitHub Setup

- [ ] Create repository: `community-skills-platform`
- [ ] Add all team members as collaborators
- [ ] Each member makes at least 1 commit
- [ ] Set up branch protection (main branch)

✅ Project Management

- [ ] Create GitHub Project board (Kanban)
- [ ] Set up columns: Backlog, To Do, In Progress, Review, Done
- [ ] Create Product Backlog with user stories

✅ Development Environment

- [ ] Add scaffolding files (provided by module)
- [ ] Customize README.md with your project details
- [ ] Each team member can run Docker containers locally
- [ ] Test database connection

✅ **PDF Document** (submit to Moodle):

Section 1: Team Information

```
Group Name: [e.g., "Community Connectors"]
Members:
- Member 1 Name (Student ID) - Role: [e.g., Backend Lead]
- Member 2 Name (Student ID) - Role: [e.g., Frontend Lead]
- Member 3 Name (Student ID) - Role: [e.g., Database/DevOps]
- [Member 4 if applicable]
```

Section 2: Project Description

Title: Community Skill & Item Sharing Platform

Problem Statement:
Many people have skills and items they rarely use, while others need those same resources but can't afford them. This creates waste and missed opportunities for community connection.

Solution:
A web platform where neighbors can share skills (tutoring, cooking, repairs) and items (tools, books, equipment) based on trust and mutual help rather than money.

Target Users:
- Local residents seeking/offering help
- Students needing study support or items
- Families sharing children's items/equipment
- Anyone wanting to reduce costs and build community

Key Features:
Sprint 3 (Basic):
- User profiles with skills/items
- Browse listings by category
- Request skills/items
- Basic search and filtering

Sprint 4 (Advanced):
- Smart matching algorithm
- Location-based maps
- Points/ratings system
- [Messaging if time permits]

Section 3: Code of Conduct

Example Template:

OUR TEAM CODE OF CONDUCT

1. Communication
   - Respond to messages within 24 hours
   - Attend all scheduled meetings or give 24h notice
   - Use [Discord/WhatsApp/Slack] for daily communication
   - Weekly standup: [Day/Time]

2. Work Standards
   - Commit code regularly (minimum 2x per week)
   - Write clear commit messages
   - Test your code before pushing
   - Ask for help when stuck for >2 hours
   - Complete assigned tasks by agreed deadlines

3. Collaboration
   - Review pull requests within 48 hours
   - Give constructive feedback
   - Share knowledge and help teammates
   - No blame - we solve problems together

4. Meetings
   - Be on time (5-minute grace period)
   - Come prepared with updates
   - Stay focused - no phone scrolling
   - Everyone gets chance to speak

5. Conflict Resolution
   - Address issues directly and respectfully
   - If unresolved, escalate to module team
   - Document concerns with evidence
   - Follow university dismissal process if needed

Signed by all team members: [Names & Dates]

Section 4: Personas (Minimum 2)

PERSONA 1: Sarah the Student

Demographics:
- Age: 20
- Status: 2nd year university student
- Location: Student accommodation, 15-min walk to campus
- Tech comfort: High

Goals:
- Save money on textbooks
- Find study buddies for difficult modules
- Borrow items she needs occasionally (tools for flat repairs)

Frustrations:
- Can't afford to buy everything she needs
- Doesn't know neighbors in accommodation
- Existing platforms require money she doesn't have

How our platform helps:
- Browse books offered by other students
- Find tutoring help for difficult subjects
- Borrow tools from nearby residents
- Build connections with neighbors

PERSONA 2: David the Retired Handyman

Demographics:
- Age: 67
- Status: Retired carpenter
- Location: Local resident for 30 years
- Tech comfort: Medium (uses smartphone)

Goals:
- Stay active and useful in retirement
- Share his carpentry/repair skills
- Meet new people in changing neighborhood
- Lend tools he no longer uses daily

Frustrations:
- Feels isolated since retirement
- Has expensive tools sitting unused
- Wants to help but doesn't know who needs help

How our platform helps:

- Offer carpentry skills to neighbors
- Lend tools to people who need them
- Gain recognition through points/ratings
- Feel valued in the community

PERSONA 3: Maya the Working Parent

Demographics:
- Age: 35
- Status: Working mother of two (ages 3 & 6)
- Location: Family home in local area
- Tech comfort: High

Goals:
- Save money on children's items they outgrow quickly
- Find childcare swaps with other parents
- Access items needed temporarily (camping gear, party supplies)

Frustrations:
- Buying items used only once is wasteful and expensive
- Doesn't know other parents in area
- Time-poor but wants to be part of community

How our platform helps:
- Share children's clothes/toys as kids outgrow them
- Find other parents for playdates/childcare swaps
- Borrow items needed occasionally
- Build local parent network

Section 5: Ethical Issues

IDENTIFIED ETHICAL ISSUES & MITIGATIONS

1. Privacy & Data Protection

Issue: Users share personal information (location, skills, items)

Risks:

- Data breaches exposing sensitive information
- Stalking or harassment using location data

Mitigations:

- Store only essential data (GDPR compliance)
- Hash passwords with bcrypt
- Allow users to control visibility of location (show area, not exact address)
- Implement reporting system for suspicious behavior
- Regular security audits

2. Safety & Trust

Issue: Strangers meeting to exchange items/services

Risks:

- Theft or damage to items
- Personal safety concerns
- Scams or no-shows

Mitigations:

- User ratings/reputation system
- Community guidelines clearly displayed
- Encourage public meeting places for exchanges
- Report and block functionality
- Verified user badges (email verification)

3. Equality & Accessibility

Issue: Not everyone has equal access or ability to participate

Risks:

- Excluding people without tech skills
- Favoring users with more to offer
- Accessibility barriers for disabled users

Mitigations:

- Simple, intuitive interface design
- Accessibility standards (WCAG 2.1)
- Points system rewards participation, not just offering
- Help documentation and tutorials
- Responsive design for various devices

4. Liability & Responsibility

Issue: Who is responsible if something goes wrong?

Risks:

- Damaged or lost items
- Poor quality services
- Platform liability for user actions

Mitigations:
- Clear Terms of Service
- User agreement on responsibilities
- Platform as facilitator, not guarantor
- Insurance recommendations for high-value items
- Dispute resolution guidelines

5. Fair Use & Exploitation
Issue: System could be exploited unfairly
Risks:
- Users taking without giving back
- "Professional" users treating it like a business
- Unequal distribution of benefits

Mitigations:
- Points system encourages reciprocity
- Limits on request frequency
- Community moderation
- Flag suspicious patterns (taking only, never giving)
- Ban repeat offenders

Section 6: Meeting Records Template

SPRINT 1 MEETING RECORDS

Meeting 1 - [Date]
Attendees: [All/Names of absent]
Duration: [Time]
Agenda:
1. Introductions and role assignment
2. Review project brief
3. Set up communication channels
4. Agree on meeting schedule

Key Decisions:
- Roles assigned: [list]
- Communication tool: [Discord/WhatsApp]
- Meeting times: [Weekly schedule]
- Code of Conduct agreed

Action Items:
- [ ] Member 1: Set up GitHub repo by [date]
- [ ] Member 2: Draft personas by [date]
- [ ] Member 3: Research ethical issues by [date]
- [ ] All: Set up Docker environment by [date]

Next Meeting: [Date/Time]

Meeting 2 - [Date]
[Same format]

[Document at least 2-3 meetings for Sprint 1]

SPRINT 2: Requirements & Design (Weeks 5-6)

Due: Lab Week 6
Grade Weight: ~20%

Goal: Define exactly what you're building before you code

Deliverables Checklist:

✅ **PDF Document:**

Section 1: User Stories

Format: "As a [user type], I want to [action], so that [benefit]"

MUST HAVE (Sprint 3 - Basic Features):

Epic: User Management
- As a new user, I want to create an account with email and password, so that I can access the platform
- As a user, I want to create my profile with name, bio, and location, so that others know about me
- As a user, I want to view other users' profiles, so that I can learn about community members

Epic: Listing Skills/Items
- As a user, I want to create a listing for a skill I can offer, so that others can find my services
- As a user, I want to create a listing for an item I can share, so that others can borrow it
- As a user, I want to add photos to my listings, so that people can see what I'm offering
- As a user, I want to categorize my listings with tags, so that they're easy to find
- As a user, I want to mark listings as available/unavailable, so that people know current status

Epic: Browsing & Discovery
- As a user, I want to see all listings on one page, so that I can browse what's available
- As a user, I want to filter listings by category (skills/items), so that I find relevant offers
- As a user, I want to search listings by keyword, so that I can find specific things quickly
- As a user, I want to view detailed information about a listing, so that I can decide if I want it
- As a user, I want to see who posted a listing, so that I know who I'll interact with

Epic: Making Requests
- As a user, I want to request a skill/item from another user, so that I can receive help
- As a user, I want to see all requests I've made, so that I can track them
- As a user, I want to see requests others have made for my listings, so that I can respond
- As a user, I want to accept or decline requests, so that I control my availability

SHOULD HAVE (Sprint 4 - Intermediate):

Epic: User Authentication
- As a user, I want to log in securely, so that my account is protected
- As a user, I want to log out, so that others can't access my account
- As a user, I want to reset my password if I forget it, so that I can regain access

Epic: Points/Ratings System
- As a user, I want to earn points when I help others, so that I'm recognized for contributions
- As a user, I want to rate other users after exchanges, so that reputation is visible
- As a user, I want to see my total points, so that I track my contributions
- As a user, I want to see others' ratings, so that I know who to trust

COULD HAVE (Sprint 4 - Advanced):

Epic: Smart Matching
- As a user, I want to see suggestions for skills/items I might need, so that I discover relevant offers
- As a user, I want matches based on my location, so that I find nearby help
- As a user, I want matches based on my previous requests, so that recommendations are personalized

Epic: Location Features
- As a user, I want to see listings on a map, so that I know what's nearby
- As a user, I want to filter by distance, so that I find local resources
- As a user, I want to set my service radius, so that I only help people within reasonable distance

Epic: Messaging (stretch goal)
- As a user, I want to message other users directly, so that I can coordinate exchanges
- As a user, I want to receive notifications, so that I don't miss messages

Section 2: Use Case Diagram

[You'll need to draw this - here's what to include]

Actors:
- Unregistered User (Visitor)
- Registered User (Community Member)
- System

Use Cases:

Unregistered User can:
- View homepage
- Browse public listings
- Register account

Registered User can:
- All of above, plus:
- Log in / Log out
- Create profile
- Create listing (skill or item)
- Edit/Delete own listings
- Browse all listings
- Search listings
- View listing details
- Request skill/item
- View own requests
- Accept/Decline requests on own listings
- Rate other users
- View own points
- [Sprint 4] View map of listings
- [Sprint 4] Receive recommendations

System:
- Sends email verification
- Calculates matching scores
- Updates points
- Generates recommendations

Section 3: Sprint 3 Specification

This is your detailed blueprint for Sprint 3:

SPRINT 3 DETAILED SPECIFICATION

Objective: Build core functionality allowing users to create listings and make requests

Timeline: Weeks 7-10 (4 weeks)

Required Pages (as per brief):

1. USERS LIST PAGE (/users)
   Purpose: Show all registered community members
   Data from database:
   - User ID
   - Full name
   - Profile picture (or default)
   - Location (area, not full address)
   - Number of listings they have
   - Member since date
   - Points/rating (if Sprint 4 done)

   Features:
   - Click on user to view profile
   - Search users by name
   - Sort by join date or points

   Wireframe: [You'll draw this - simple grid of user cards]

2. USER PROFILE PAGE (/users/:id)
   Purpose: Show detailed information about a specific user
   Data from database:
   - User's full details
   - Bio/description
   - Skills they offer
   - Items they share
   - All their listings
   - Points/ratings (Sprint 4)
   - Reviews from others (Sprint 4)

   Features:
   - View all user's listings
   - Click listing to see detail
   - Contact button (Sprint 4 messaging)

   Wireframe: [You'll draw - profile header + listings grid]

3. LISTINGS PAGE (/listings)
   Purpose: Browse all available skills and items

Data from database:
- Listing ID
- Title
- Description (truncated)
- Category/Type (skill or item)
- Tags
- Owner name
- Photo
- Availability status
- Posted date

Features:
- Filter by: All / Skills / Items
- Filter by tags/categories
- Search by keyword
- Sort by: Newest, Most Requested, Closest (Sprint 4)

Wireframe: [Draw - filters sidebar + card grid]

4. LISTING DETAIL PAGE (/listings/:id)
Purpose: Show full information about a specific listing
Data from database:
- All listing details
- Owner information
- Full description
- Multiple photos
- Availability calendar (Sprint 4)
- Reviews/ratings (Sprint 4)
- Similar listings (Sprint 4)

Features:
- "Request this" button
- Contact owner (Sprint 4)
- Report listing
- Share listing

Wireframe: [Draw - hero image + details + owner card]

5. TAGS/CATEGORIES PAGE
Purpose: Browse listings by category
Options:
- Show as filter on listings page, OR
- Separate page showing category tiles

Categories:
- Tools & Equipment
- Books & Media

- Skills & Teaching
- Household Items
- Sports & Hobbies
- Technology
- Other

Wireframe: [Draw - category cards or tag cloud]

Additional Required Pages (not in brief but necessary):

6. HOMEPAGE (/)
Purpose: Welcome users and explain platform
Content:
- Hero section with tagline
- How it works (3 steps)
- Featured listings
- Categories overview
- Sign up call-to-action

Wireframe: [Draw]

7. CREATE LISTING PAGE (/listings/new)
Purpose: Form for creating new listings
Form fields:
- Title *
- Type (skill or item) *
- Description *
- Category/Tags *
- Upload photo
- Availability (always / specific times)
- Condition (for items)

Wireframe: [Draw - form layout]

8. MY DASHBOARD (/dashboard)
Purpose: User's personal hub
Shows:
- My listings (with edit/delete)
- Requests I've made (status)
- Requests on my listings (to accept/decline)
- My points (Sprint 4)
- Quick stats

Wireframe: [Draw - tabs or cards]

Section 4: Activity Diagrams

Create these for key user flows:

1. ACTIVITY DIAGRAM: User Registration Flow

Start
↓
[User visits homepage]
↓
[Clicks "Sign Up"]
↓
[Fills registration form]
↓
Decision: Valid data?
No → [Show errors] → [Back to form]
Yes ↓
[Create account in database]
↓
[Send verification email]
↓
[Redirect to profile setup]
↓
[User completes profile]
↓
[Save profile to database]
↓
[Redirect to dashboard]
↓
End

---

2. ACTIVITY DIAGRAM: Creating a Listing

Start
↓
[User logged in]
↓
[Click "Create Listing"]
↓
[Fill listing form]
↓
Decision: All required fields?
No → [Show validation errors] → [Back to form]
Yes ↓
[Upload photo (optional)]
↓
[Select category/tags]
↓

[Click "Submit"]
↓
[Save listing to database]
↓
[Show success message]
↓
[Redirect to listing detail page]
↓
End

---

3. ACTIVITY DIAGRAM: Requesting an Item/Skill

Start
↓
[User browsing listings]
↓
[Finds interesting listing]
↓
[Clicks to view detail]
↓
Decision: User logged in?
No → [Redirect to login] → [After login, return to listing]
Yes ↓
[Click "Request This"]
↓
[Fill request form (message, dates needed)]
↓
[Submit request]
↓
[Create request record in database]
↓
[Notify listing owner (Sprint 4)]
↓
[Show confirmation to requester]
↓
End

---

4. ACTIVITY DIAGRAM: Responding to Request (Owner)

Start
↓
[Owner logs in]
↓

[Goes to dashboard]
↓
[Sees pending requests]
↓
[Clicks on request]
↓
[Reviews request details]
↓
Decision: Accept or Decline?
Decline → [Click "Decline"] → [Update database] → [Notify requester] → End
Accept ↓
[Click "Accept"]
↓
[Update request status in database]
↓
[Award points to owner (Sprint 4)]
↓
[Notify requester]
↓
[Show exchange details]
↓
End

Section 5: Additional Diagrams (Optional but impressive)

5. SEQUENCE DIAGRAM: Request Flow
(Shows interaction between User, Frontend, Backend, Database)

6. ENTITY RELATIONSHIP DIAGRAM (ERD)
(Shows database tables and relationships - see database section below)

7. CLASS DIAGRAM (if using MVC pattern)
(Shows how your code is organized)

8. WIREFRAMES
(Hand-drawn or digital mockups of all pages)
For distinction, make these detailed with:
- Actual content examples
- Navigation elements
- Responsive layout notes

9. COLOR SCHEME & DESIGN
Primary: [e.g., #2E7D32 - Community green]
Secondary: [e.g., #FFA726 - Warm orange]
Accent: [e.g., #1976D2 - Trust blue]

Typography:
- Headings: [e.g., Poppins, bold]
- Body: [e.g., Open Sans, regular]

Design principles:
- Warm, friendly, welcoming
- Clear visual hierarchy
- Mobile-first responsive
- Accessibility: WCAG 2.1 AA

Section 7: Kanban Board Screenshot

Take screenshot showing:

- Columns: Backlog, To Do, In Progress, Review, Done
- User stories as cards
- Sprint 3 tasks ready to start

Section 8: GitHub Link

Repository: https://github.com/[your-username]/community-skills-platform

Section 9: Lab Progress Confirmation

All team members confirm:

- ✅ Docker environment running
- ✅ Can connect to MySQL database
- ✅ Scaffolding files understood
- ✅ Git workflow practiced
- ✅ Ready to begin Sprint 3 development

Section 10: Meeting Records Continue from Sprint 1, document all Sprint 2 meetings

---

SPRINT 3: Core Development (Weeks 7-10)

Due: Lab Week 10
Grade Weight: ~40%

Goal: Build the MVP - basic but fully functional

This is where you start coding!

Development Strategy:

Week 7: Database & Backend Setup

- Design database schema
- Create all tables
- Write seed data (test users, listings)
- Set up Express routes structure
- Test database connections

Week 8: Basic Pages

- Users list page
- User profile page
- Listing page
- Listing detail page
- Home page

Week 9: Create/Edit Functionality

- Registration/login (basic)
- Create listing form
- Request system
- Dashboard

Week 10: Polish & Testing

- CSS styling
- Bug fixes
- Test all features
- Prepare demo
- Write documentation

Code Organization:

```
community-skills-platform/
├── docker-compose.yml
├── Dockerfile
├── package.json
├── README.md
├── .gitignore
│
├── app.js (main Express app)
│
├── routes/
│   ├── index.js (homepage)
│   ├── users.js (user routes)
│   ├── listings.js (listing routes)
│   ├── requests.js (request routes)
│   └── auth.js (login/register - Sprint 4)
│
├── views/ (PUG templates)
│   ├── layout.pug (master template)
│   ├── index.pug (homepage)
│   ├── users/
│   │   ├── list.pug (users list)
│   │   └── profile.pug (user profile)
│   ├── listings/
│   │   ├── list.pug (all listings)
│   │   ├── detail.pug (single listing)
│   │   └── create.pug (create form)
│   └── dashboard/
│       └── index.pug (user dashboard)
│
├── public/ (static files)
│   ├── css/
│   │   └── style.css
│   ├── js/
│   │   └── main.js
│   └── images/
│       └── (user uploads)
│
├── db/
│   ├── schema.sql (create tables)
│   └── seed.sql (test data)
│
└── config/
    └── database.js (MySQL connection)
```

Deliverables Checklist:

✅ **Working Application** (runs in Docker)

Required pages working:

- [ ] Users list page (pulls from database)
- [ ] User profile page (pulls from database)
- [ ] Listings page (pulls from database)
- [ ] Listing detail page (pulls from database)
- [ ] Tags/categories (pulls from database)
- [ ] Homepage
- [ ] Create listing form
- [ ] Dashboard

✅ **Database**

- [ ] MySQL running in Docker
- [ ] All tables created
- [ ] Test data populated
- [ ] Relationships working

✅ **PDF Document:**

Section 1: Implemented User Stories

> Copy the user stories from Sprint 2 that you actually built in Sprint 3
> Mark each as: ✅ Completed / ⏳ Partially / ❌ Not started
>
> Example:
> ✅ As a user, I want to view all listings, so that I can browse available skills/items
> Implementation: /listings route, displays all listings from database with filters
>
> ⏳ As a user, I want to search listings by keyword
> Implementation: Search box added but search logic needs optimization

Section 2: Database Design

Show your ERD (Entity Relationship Diagram) and table schemas

EXAMPLE DATABASE SCHEMA (you'll refine this):

TABLE: users

Column	Type	Notes
user_id	INT (PK)	Auto
email	VARCHAR(255)	Unique
password_hash	VARCHAR(255)	
first_name	VARCHAR(100)	
last_name	VARCHAR(100)	
bio	TEXT	Null OK
location	VARCHAR(255)	
latitude	DECIMAL(10,8)	Sprint4
longitude	DECIMAL(11,8)	Sprint4
profile_pic	VARCHAR(255)	Null OK
points	INT	Default 0
created_at	TIMESTAMP	Auto

TABLE: listings

Column	Type	Notes
listing_id	INT (PK)	Auto
user_id	INT (FK)	→ users
title	VARCHAR(255)	
description	TEXT	
type	ENUM	skill/item
category	VARCHAR(100)	
photo_url	VARCHAR(255)	Null OK
is_available	BOOLEAN	Default 1
created_at	TIMESTAMP	
updated_at	TIMESTAMP	

TABLE: requests

Column	Type	Notes
request_id	INT (PK)	Auto
listing_id	INT (FK)	→ listings
requester_id	INT (FK)	→ users
status	ENUM	pending/accepted/declined
message	TEXT	Null OK
requested_date	TIMESTAMP	

| responded_date | TIMESTAMP | Null OK|
+----------------+--------------+--------+

TABLE: tags
+----------------+--------------+--------+
| Column | Type | Notes |
+----------------+--------------+--------+
| tag_id | INT (PK) | Auto |
| tag_name | VARCHAR(50) | Unique |
+----------------+--------------+--------+

TABLE: listing_tags (junction table)
+----------------+--------------+--------+
| listing_id | INT (FK) | → listings |
| tag_id | INT (FK) | → tags |
+----------------+--------------+--------+
PRIMARY KEY (listing_id, tag_id)

[Sprint 4 additions]

TABLE: ratings
+----------------+--------------+--------+
rating_id	INT (PK)	Auto
request_id	INT (FK)	→ requests
rater_id	INT (FK)	→ users
rated_id	INT (FK)	→ users
score	INT	1-5
comment	TEXT	Null OK
created_at	TIMESTAMP	
+----------------+--------------+--------+

TABLE: messages (if implementing)
+----------------+--------------+--------+
message_id	INT (PK)	Auto
sender_id	INT (FK)	→ users
receiver_id	INT (FK)	→ users
request_id	INT (FK)	→ requests (optional)
content	TEXT	
is_read	BOOLEAN	Default 0
sent_at	TIMESTAMP	
+----------------+--------------+--------+

Section 3: Task Breakdown

Copy from GitHub Project:

Example format:

Developer: Sarah
Tasks completed:
- Set up database schema
- Created users table and seed data
- Built /users list route
- Designed user profile page
Total commits: 23

Developer: Mike
Tasks completed:
- Set up Express routing structure
- Built /listings routes
- Created listing detail page
- Implemented search functionality
Total commits: 19

[Continue for all team members]

Section 4: GitHub Links

Repository: [URL]
GitHub Project: [URL]

Section 5: Participation Metrics Screenshot

From GitHub, show:

- Commit history graph
- Contributions by member
- Pull requests created/reviewed

Section 6: Kanban Board Screenshot

Show your progress with cards moved to "Done"

Section 7: Meeting Records

All Sprint 3 meetings documented

SPRINT 4: Advanced Features (Weeks 11-13)

Due: Weeks 13-14 (Presentation)

Grade Weight: ~40%

Goal: Add sophisticated features that demonstrate advanced skills

Your Chosen Features (Priority Order):

1. Points/Ratings System (Week 11 - Priority 1)

- Easiest to implement
- Builds confidence
- Core to community trust

2. Matching Algorithm (Week 11-12 - Priority 2)

- Shows technical depth
- Central to project concept
- Good for demonstration

3. Maps API Integration (Week 12-13 - Priority 3)

- Visual impact for presentation
- Demonstrates API integration skills
- Location-based features

4. Messaging System (Week 13 - STRETCH GOAL)

- Only if ahead of schedule
- Most complex feature
- Great bonus but not essential

Feature Implementation Details:

FEATURE 1: Points & Ratings System

How it works:

- Users earn points by helping others
- Users can rate each other after exchanges
- Points/ratings displayed on profiles
- Builds reputation and trust

Implementation Steps:

1. **Database changes:**

sql

```sql
-- Already in users table:
ALTER TABLE users ADD COLUMN points INT DEFAULT 0;
ALTER TABLE users ADD COLUMN average_rating DECIMAL(3,2) DEFAULT 0;

-- Create ratings table (shown above in database section)
```

2. **Award points logic:**

javascript

```javascript
// When request is accepted:
// Owner earns points
// Add this to your request acceptance route

async function acceptRequest(requestId) {
  // Update request status
  await db.query('UPDATE requests SET status = "accepted" WHERE request_id = ?', [requestId]);

  // Award points to listing owner
  const points = 10; // Award 10 points per accepted request
  await db.query('UPDATE users SET points = points + ? WHERE user_id = ?', [points, ownerId]);
}
```

3. **Rating system:**

javascript

```
// After exchange is complete:
// Both users can rate each other

async function submitRating(requestId, raterId, ratedId, score, comment) {
    // Insert rating
    await db.query(
        'INSERT INTO ratings (request_id, rater_id, rated_id, score, comment) VALUES (?, ?, ?, ?, ?)',
        [requestId, raterId, ratedId, score, comment]
    );

    // Recalculate average rating
    const [result] = await db.query(
        'SELECT AVG(score) as avg_rating FROM ratings WHERE rated_id = ?',
        [ratedId]
    );

    await db.query('UPDATE users SET average_rating = ? WHERE user_id = ?',
        [result[0].avg_rating, ratedId]
    );
}
```

4. **Display on profile:**

```
pug

// In user profile page (profile.pug)
.user-stats
    .stat
        h3 #{user.points}
        p Community Points
    .stat
        h3
            i.star-icon ★
            | #{user.average_rating.toFixed(1)}
        p Average Rating
```

Testing:

- Create test exchanges
- Award points
- Submit ratings
- Verify calculations correct

FEATURE 2: Matching Algorithm

How it works:

- System suggests listings user might be interested in
- Based on: location, previous requests, popular items, user's offerings

Algorithm Logic:

```javascript
```

```
// Basic matching algorithm

async function getRecommendations(userId) {
    const recommendations = [];

    // 1. Get user's location
    const [user] = await db.query('SELECT * FROM users WHERE user_id = ?', [userId]);

    // 2. Get user's previous request categories
    const [history] = await db.query(`
        SELECT DISTINCT l.category
        FROM requests r
        JOIN listings l ON r.listing_id = l.listing_id
        WHERE r.requester_id = ?
    `, [userId]);

    const interestedCategories = history.map(h => h.category);

    // 3. Find nearby listings in those categories
    if (interestedCategories.length > 0) {
        const [categoryMatches] = await db.query(`
            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
            FROM listings l
            JOIN users u ON l.user_id = u.user_id
            WHERE l.category IN (?)
            AND l.user_id != ?
            AND l.is_available = 1
            HAVING distance < 10
            ORDER BY distance
            LIMIT 5
        `, [user.latitude, user.longitude, user.latitude, interestedCategories, userId]);

        recommendations.push(...categoryMatches);
    }

    // 4. Fill remaining with popular listings nearby
    if (recommendations.length < 5) {
        const [popular] = await db.query(`
            SELECT l.*, u.first_name, u.last_name,
                COUNT(r.request_id) as request_count
            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.user_id != ?
        AND l.is_available = 1
        GROUP BY l.listing_id
        ORDER BY request_count DESC
        LIMIT ?
      `, [userId, 5 - recommendations.length]);

      recommendations.push(...popular);
    }

    return recommendations;
}
```

More Advanced Algorithm (if time):

javascript

```
// Collaborative filtering approach
// "Users who requested X also requested Y"

async function getAdvancedRecommendations(userId) {
    // Find similar users (users who requested similar things)
    const [similarUsers] = await db.query(`
        SELECT r2.requester_id, COUNT(*) as common_interests
        FROM requests r1
        JOIN listings l1 ON r1.listing_id = l1.listing_id
        JOIN listings l2 ON l1.category = l2.category
        JOIN requests r2 ON l2.listing_id = r2.listing_id
        WHERE r1.requester_id = ?
        AND r2.requester_id != ?
        GROUP BY r2.requester_id
        ORDER BY common_interests DESC
        LIMIT 10
    `, [userId, userId]);

    if (similarUsers.length === 0) return [];

    const similarUserIds = similarUsers.map(u => u.requester_id);

    // Get what those similar users requested that current user hasn't
    const [recommendations] = await db.query(`
        SELECT DISTINCT l.*, u.first_name, u.last_name
        FROM requests r
        JOIN listings l ON r.listing_id = l.listing_id
        JOIN users u ON l.user_id = u.user_id
        WHERE r.requester_id IN (?)
        AND l.listing_id NOT IN (
            SELECT listing_id FROM requests WHERE requester_id = ?
        )
        AND l.user_id != ?
        AND l.is_available = 1
        LIMIT 10
    `, [similarUserIds, userId, userId]);

    return recommendations;
}
```

Display recommendations:

```
pug
```

```
// On dashboard or homepage
section.recommendations
  h2 Recommended for You
  .listings-grid
    each listing in recommendations
      .listing-card
        img(src=listing.photo_url)
        h3= listing.title
        p= listing.description
        span.distance #{listing.distance}km away
        a(href=`/listings/${listing.listing_id}`) View Details
```

FEATURE 3: Maps API Integration

Using Google Maps JavaScript API

Setup:

1. Get API key from Google Cloud Console (free tier available)
2. Enable Maps JavaScript API
3. Restrict key to your domain

Implementation:

1. **Store user locations:**

javascript

```
// When user registers/updates profile
// Use geocoding to convert address to coordinates

const axios = require('axios');

async function geocodeAddress(address) {
    const apiKey = process.env.GOOGLE_MAPS_API_KEY;
    const url = `https://maps.googleapis.com/maps/api/geocode/json?address=${encodeURIComponent(address)}&key=${apil

    const response = await axios.get(url);

    if (response.data.results.length > 0) {
        const location = response.data.results[0].geometry.location;
        return {
            latitude: location.lat,
            longitude: location.lng
        };
    }

    throw new Error('Address not found');
}

// In registration route:
const coords = await geocodeAddress(userAddress);
await db.query(
    'UPDATE users SET latitude = ?, longitude = ? WHERE user_id = ?',
    [coords.latitude, coords.longitude, userId]
);
```

2. **Display map with listings:**

pug

```
// listings-map.pug
extends layout

block content
  #map(style="height: 500px; width: 100%;")

  script(src=`https://maps.googleapis.com/maps/api/js?key=${process.env.GOOGLE_MAPS_API_KEY}&callback=initMap
  script.
    function initMap() {
      // Center map on user's location or default
      const center = { lat: #{userLat || 51.505}, lng: #{userLng || -0.09} };

      const map = new google.maps.Map(document.getElementById('map'), {
        zoom: 12,
        center: center
      });

      // Add markers for listings
      const listings = !{JSON.stringify(listings)};

      listings.forEach(listing => {
        if (listing.latitude && listing.longitude) {
          const marker = new google.maps.Marker({
            position: { lat: listing.latitude, lng: listing.longitude },
            map: map,
            title: listing.title
          });

          // Info window on click
          const infoWindow = new google.maps.InfoWindow({
            content: `
              <div>
                <h3>${listing.title}</h3>
                <p>${listing.description}</p>
                <a href="/listings/${listing.listing_id}">View Details</a>
              </div>
            `
          });

          marker.addListener('click', () => {
            infoWindow.open(map, marker);
          });
        }
      });
    }
```

3. **Distance filter:**

```javascript
// Add to listings route
router.get('/listings/nearby', async (req, res) => {
  const { latitude, longitude, radius } = req.query;

  const [listings] = await db.query(`
    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
    HAVING distance < ?
    ORDER BY distance
  `, [latitude, longitude, latitude, radius || 10]);

  res.render('listings/map', { listings });
});
```

Privacy note for documentation:

- Don't show exact addresses
- Show approximate area (e.g., "2.3 km away")
- Users can reveal exact location after accepting request

FEATURE 4: In-App Messaging (STRETCH GOAL)

Only implement if features 1-3 are completed and working well!

This is the most complex feature. Basic implementation:

1. **Database table** (already shown above)
2. **Send message route:**

```javascript
```

```
router.post('/messages/send', isAuthenticated, async (req, res) => {
    const { receiverId, requestId, content } = req.body;
    const senderId = req.session.userId;

    await db.query(
        'INSERT INTO messages (sender_id, receiver_id, request_id, content) VALUES (?, ?, ?, ?)',
        [senderId, receiverId, requestId, content]
    );

    res.redirect(`/messages/${receiverId}`);
});
```

3. **View conversation:**

javascript

```
router.get('/messages/:userId', isAuthenticated, async (req, res) => {
    const currentUser = req.session.userId;
    const otherUser = req.params.userId;

    // Get conversation between two users
    const [messages] = await db.query(`
        SELECT m.*,
               u1.first_name as sender_name,
               u2.first_name as receiver_name
        FROM messages m
        JOIN users u1 ON m.sender_id = u1.user_id
        JOIN users u2 ON m.receiver_id = u2.user_id
        WHERE (m.sender_id = ? AND m.receiver_id = ?)
           OR (m.sender_id = ? AND m.receiver_id = ?)
        ORDER BY m.sent_at ASC
    `, [currentUser, otherUser, otherUser, currentUser]);

    // Mark as read
    await db.query(
        'UPDATE messages SET is_read = 1 WHERE receiver_id = ? AND sender_id = ?',
        [currentUser, otherUser]
    );

    res.render('messages/conversation', { messages, otherUser });
});
```

4. **Simple polling for new messages** (refresh every 5 seconds):

javascript

```
// In conversation page
setInterval(async () => {
    const response = await fetch(`/api/messages/${otherUserId}/new`);
    const newMessages = await response.json();
    if (newMessages.length > 0) {
        // Append to conversation
        newMessages.forEach(msg => appendMessage(msg));
    }
}, 5000);
```

For distinction, could add:

- Real-time updates (Socket.io)
- Unread message count
- Message notifications
- Message history pagination

CI/CD with GitHub Actions

Requirement: At least one GitHub Action

Recommended Actions:

1. **Automated Testing** (Basic - recommended)

```yaml
```

```yaml
# .github/workflows/test.yml
name: Run Tests

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '18'

    - name: Install dependencies
      run: npm install

    - name: Run tests
      run: npm test
```

2. **Docker Build** (Intermediate)

yaml

```yaml
# .github/workflows/docker-build.yml
name: Docker Build

on:
  push:
    branches: [ main ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Build Docker image
      run: docker-compose build

    - name: Test Docker containers
      run: |
        docker-compose up -d
        sleep 10
        docker-compose ps
        docker-compose down
```

3. **Linting** (Easy, good for beginners)

```yaml
```

```yaml
# .github/workflows/lint.yml
name: Lint Code

on: [push, pull_request]

jobs:
  lint:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Set up Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '18'

    - name: Install dependencies
      run: npm install

    - name: Run ESLint
      run: npm run lint
```

Choose ONE to start - linting is easiest for beginners!

Sprint 4 Deliverables Checklist:

✅ **Working Advanced Features:**

- [ ] Points system implemented
- [ ] Ratings system working
- [ ] Matching algorithm suggesting relevant listings
- [ ] Maps showing listing locations
- [ ] Distance filtering working
- [ ] [Optional] Basic messaging

✅ **DevOps:**

- [ ] Application runs in Docker containers
- [ ] At least one GitHub Action implemented and working

✅ **Presentation (Week 13/14)**

Create PowerPoint with:

Slide 1: Title

- Project name
- Team members
- Tagline

Slide 2: The Problem

- What community problem does this solve?
- Who experiences this problem?
- Why existing solutions don't work

Slide 3: Our Solution

- High-level overview of your platform
- How it addresses the problem
- Key differentiators

Slide 4: How It Works

- User journey (3-4 steps with screenshots)
- Visual flow diagram

Slide 5: Key Features (Sprint 3)

- List core features with screenshots
- Emphasize what works

Slide 6: Advanced Features (Sprint 4)

- Matching algorithm (show example)
- Maps integration (screenshot of map)
- Points system (show user profile)

Slide 7: Technical Stack

- Technologies used
- Architecture diagram
- Why these choices?

Slide 8: Ethical Considerations

- Privacy measures
- Safety features
- Accessibility

- How you addressed concerns from Sprint 1

Slide 9: Development Process

- Sprint methodology
- Team roles
- GitHub workflow
- CI/CD pipeline

Slide 10: Challenges & Solutions

- Technical challenges faced
- How you solved them
- What you learned

Slide 11: Demo

- (Live demo or video recording)

Slide 12: Future Enhancements

- What would you add with more time?
- Scalability plans
- Additional features

Slide 13: Thank You & Questions

- Contact info
- GitHub repository link

Presentation Tips:

- 10-12 minutes total
- Every team member speaks
- Practice transitions
- Have backup (video) if live demo fails
- Show enthusiasm!

🎓 INDIVIDUAL ASSESSMENT (20%)

Due: Week 13-14 with group presentation

Part 1: STAR Story

Choose ONE question:

1. Tell me about a time when there was a disagreement in your team
2. Tell me about a time when you were stuck on a task
3. Tell me about a time when your project seemed stuck
4. Tell me about a time you were concerned about a deadline
5. Tell me about a time when you had to correct a colleague's work
6. Tell me about a time when you had to challenge someone in your team
7. Tell me about a time when you made a wrong decision
8. Tell me about a time when you had to shift priorities

STAR Structure:

Situation (2-3 sentences):

- Set the scene
- When did this happen? (which sprint/week)
- What was the context?

Example:
"During Sprint 3, around week 8, our team was building the listings page. We had split the work so that two members focused on the frontend (PUG templates and CSS) while I worked on the backend routes and database queries. We were behind schedule because some of the initial database design wasn't working as planned."

Task (2-3 sentences):

- What was your specific responsibility?
- What needed to be achieved?
- What was at stake?

Example:
"My task was to refactor the database queries for the listings page to make them more efficient. The page was timing out because we were making too many individual queries instead of using JOIN statements. If I didn't fix this by the end of the week, our group would fail the Sprint 3 code review."

Action (4-5 sentences - most important part):

- What specifically did YOU do?
- Why did you choose this approach?
- Who did you involve?
- What steps did you take?

Example:
"First, I analyzed the existing queries to identify the bottlenecks. I found we were making 5 separate queries per listing to get owner details, tags, and request counts. I researched MySQL JOIN operations and found examples of similar problems online. I then rewrote the main query to use LEFT JOINs to combine all the data in a single database call. I tested it locally with our test data and found it was 10 times faster. I then created a pull request with detailed comments explaining the changes and asked my teammate who built the frontend to review it. After she approved, I merged it and verified the listings page was now loading quickly."

Result (2-3 sentences):

- What happened because of your actions?
- Quantify if possible (time saved, bugs fixed, etc.)
- What did you learn?

Example:
"The listings page load time decreased from 3 seconds to 0.3 seconds with the refactored queries. Our team successfully passed the Sprint 3 code review and the lecturer specifically complimented our efficient database design. I learned the importance of planning database queries carefully from the start and how much performance can be gained through proper use of JOIN statements. This experience also taught me to speak up early when I see potential problems rather than hoping they'll resolve themselves."

Part 2: Module Reflection (150-200 words)

Reflect on what you learned and how you'll apply it to FYP.

Structure:

Paragraph 1: Technical Skills

- What technical skills did you develop?
- What surprised you?
- What do you want to explore more?

Paragraph 2: Project Management

- What did you learn about planning?
- Time management lessons?
- Team coordination insights?

Paragraph 3: Application to FYP

- What will you do differently?
- What approaches will you keep?
- What mistakes will you avoid?

Example:

"This module taught me that proper planning saves enormous amounts of time during development. Our team spent two full weeks on Sprint 2 creating detailed specifications, and initially this felt like wasted time when we were eager to start coding. However, once we began Sprint 3, having those wireframes and database designs meant we rarely had to stop and figure out what to build next - we just followed the plan. This will definitely influence my FYP approach; I'll invest more time upfront in detailed planning rather than jumping straight into code.

I also discovered I really enjoy backend development and database design more than I expected. Working with MySQL and figuring out efficient query structures was like solving puzzles, and seeing the performance improvements from good database design was really satisfying. For my FYP, I'm now considering projects that involve complex data relationships or recommendation systems.

The biggest lesson about time management was that everything takes longer than you expect, especially debugging and integration. Our Sprint 4 matching algorithm looked simple on paper but took two full weeks to get working properly. For my FYP, I'll build in buffer time and have a very clear MVP so I'm not trying to implement ambitious features in the final weeks. I'll also commit to Git daily like we did in this project - having that commit history and being able to roll back when things broke was invaluable."

📝 GRADING CRITERIA & HOW TO GET DISTINCTION

What Gets You High Marks:

Sprint 1 (Foundation):

- ✅ Professional documentation
- ✅ Detailed personas with research
- ✅ Thoughtful ethical analysis
- ✅ Strong code of conduct
- ✅ Clear project description
- ✅ All team members contributing equally

Sprint 2 (Planning):

- ✅ Comprehensive user stories
- ✅ Detailed wireframes for all pages
- ✅ Activity diagrams showing key flows
- ✅ Professional ERD with normalized database
- ✅ Clear Sprint 3 specification
- ✅ Evidence of research (color psychology, UX principles)

Sprint 3 (Core Development):

- ✅ All required pages working perfectly

- ✅ Clean, well-commented code
- ✅ Efficient database queries
- ✅ Professional UI design
- ✅ No major bugs
- ✅ Good Git practices (meaningful commits, branches)
- ✅ Equal contributions from all team members

Sprint 4 (Advanced Features):

- ✅ 2-3 advanced features fully working
- ✅ Features genuinely add value
- ✅ Complex algorithm explained clearly
- ✅ External API integrated properly
- ✅ CI/CD pipeline working
- ✅ Polished presentation
- ✅ Strong demonstration

Individual Assessment:

- ✅ Genuine STAR story (not generic)
- ✅ Specific details and examples
- ✅ Shows reflection and learning
- ✅ Professional writing
- ✅ Clear growth mindset

Common Mistakes to Avoid:

❌ Starting coding before planning
❌ Unequal team contributions
❌ Copy-pasting code you don't understand
❌ Adding features that don't work
❌ Poor Git hygiene (huge commits, no messages)
❌ Ignoring the brief requirements
❌ Generic STAR stories (sounds like AI)
❌ Leaving everything to the last minute
❌ Not testing thoroughly
❌ Bad UI/UX (ugly = lower marks)

🚀 WEEK-BY-WEEK ACTION PLAN

Week 1-2:

- [ ] Form team
- [ ] Set up communication (Discord/WhatsApp)
- [ ] First meeting: assign roles
- [ ] Read brief thoroughly
- [ ] Brainstorm project ideas
- [ ] Choose project concept
- [ ] Validate with module team

Week 3:

- [ ] Set up GitHub repository
- [ ] All members make first commit
- [ ] Set up Docker environment
- [ ] Create GitHub Project board
- [ ] Write Code of Conduct
- [ ] Start personas

Week 4:

- [ ] Complete Sprint 1 documentation
- [ ] Submit Sprint 1 PDF
- [ ] Attend code review
- [ ] Fix any issues raised

Week 5:

- [ ] Write all user stories
- [ ] Create wireframes for all pages
- [ ] Draw activity diagrams
- [ ] Design database schema
- [ ] Plan Sprint 3 in detail

Week 6:

- [ ] Complete Sprint 2 documentation
- [ ] Submit Sprint 2 PDF
- [ ] Attend code review
- [ ] Start early Sprint 3 prep

Week 7 (Sprint 3 starts):

- [ ] Create database tables

- [ ] Write seed data
- [ ] Set up Express routes structure
- [ ] Test database connection
- [ ] Build homepage

Week 8:

- [ ] Build users list page
- [ ] Build user profile page
- [ ] Build listings page
- [ ] Build listing detail page
- [ ] Implement basic CSS

Week 9:

- [ ] Build registration/login (basic)
- [ ] Build create listing form
- [ ] Build request system
- [ ] Build dashboard
- [ ] More CSS styling

Week 10:

- [ ] Bug fixing
- [ ] Testing all features
- [ ] Polish UI
- [ ] Complete Sprint 3 documentation
- [ ] Submit Sprint 3 PDF
- [ ] Attend code review
- [ ] Plan Sprint 4

Week 11 (Sprint 4 starts):

- [ ] Implement points system
- [ ] Implement ratings system
- [ ] Start matching algorithm
- [ ] Test advanced features

Week 12:

- [ ] Complete matching algorithm
- [ ] Integrate Maps API
- [ ] Set up GitHub Action
- [ ] [Optional] Start messaging if ahead
- [ ] Create presentation slides

Week 13:

- [ ] Final testing
- [ ] Deploy application
- [ ] Practice presentation
- [ ] Record demo backup
- [ ] Complete individual reflection

Week 13-14:

- [ ] Deliver presentation
- [ ] Submit all final documents
- [ ] Celebrate! 🎉

💡 TIPS FOR SUCCESS

For Beginners with Node.js/Express:

Start with tutorials:

- Express.js crash course (YouTube)
- MySQL basics (W3Schools)
- PUG templating (official docs)
- Docker for beginners

Practice before Sprint 3:

- Build a simple blog in Express
- Connect to MySQL
- Create basic CRUD operations
- Understand routing

Use these resources:

- Stack Overflow (when stuck)
- Express documentation
- MySQL documentation
- GitHub examples

Team Communication:

Daily:

- Quick messages on progress
- Ask for help when stuck >2 hours
- Share useful resources

Weekly:

- Standup meeting (15 min)
- What did you do?
- What will you do?
- Any blockers?

Before deadlines:

- Extra check-in meetings
- Code review together
- Test everything

Git Workflow:

Good commits: ✅ "Add user registration route with validation" ✅ "Fix listings query to include user details" ✅ "Update CSS for responsive mobile layout"

Bad commits: ❌ "Updated stuff" ❌ "Fixed bug" ❌ "Changes"

Branch strategy:

- `main` - stable, working code
- `develop` - integration branch
- `feature/user-auth` - individual features
- `bugfix/listings-query` - bug fixes

Code Quality:

Must haves:

- Comments explaining complex logic
- Consistent indentation
- Meaningful variable names
- Error handling
- Input validation

File organization:

- One route per file

- Separate concerns (routes, models, views)
- Reusable functions
- Clear folder structure

🎯 YOUR NEXT STEPS (RIGHT NOW)

Since you're starting Sprint 1:

This Week (Week 1-2):

1. **TODAY:**
   - [ ] Read this entire document
   - [ ] Share with team members
   - [ ] Schedule first team meeting
2. **First Meeting (this week):**
   - [ ] Introductions
   - [ ] Discuss roles (who's good at what?)
   - [ ] Set up Discord/WhatsApp group
   - [ ] Agree on meeting times
   - [ ] Divide Sprint 1 tasks
3. **By End of Week 2:**
   - [ ] GitHub repo created
   - [ ] Everyone makes first commit
   - [ ] Docker running on everyone's computer
   - [ ] Start writing Code of Conduct
   - [ ] Start writing personas
4. **By Week 3:**
   - [ ] Code of Conduct finalized
   - [ ] 2-3 personas completed
   - [ ] Ethical issues identified
   - [ ] Project description written
   - [ ] Meeting records documented
5. **By Week 4 Lab:**
   - [ ] Sprint 1 PDF complete
   - [ ] All team members ready
   - [ ] Application scaffolding running
   - [ ] Attend code review

❓ QUESTIONS TO DISCUSS WITH YOUR TEAM

Before your first meeting, think about:

1. **Roles:**
    - Who wants to focus on frontend (PUG/CSS)?
    - Who wants backend (Express/routes)?
    - Who wants database (MySQL)?
    - Who wants DevOps (Docker/GitHub Actions)?
2. **Schedule:**
    - When can everyone meet?
    - What days/times work for everyone?
    - How will we communicate daily?
3. **Working Style:**
    - Pair programming or individual tasks?
    - How often do we commit to Git?
    - How do we handle disagreements?
    - What if someone isn't pulling their weight?
4. **Goals:**
    - What grade is everyone aiming for?
    - How much effort can everyone commit?
    - Any constraints (jobs, other modules)?

This roadmap is your complete guide. Reference it constantly. Update it as you go. Make it your own.

You've got this! 🚀

Questions? Stuck? Check:

1. This document first
2. Module materials on Moodle
3. Office hours with lecturers
4. Your team members

Remember: Distinction = Planning + Execution + Polish

Now go have that first team meeting! 💪