Tapin Full-Stack Development Starter Guide

Your Roadmap to Launching a Volunteer Connect Platform

Project: Tapin – Local Volunteer/Business Finder Connect App
Tech Stack: React.js + Flask (Python) + MySQL
Timeline: Final-Year Full-Stack Project
Goal: Build a mobile-first platform connecting volunteers with community organizations

Executive Summary

This comprehensive guide provides everything you need to successfully build and deploy **Tapin**, your volunteer platform connecting local volunteers with community businesses and organizations. You'll learn the essential technologies (React, Flask, MySQL), design patterns (REST APIs, JWT authentication), and deployment strategies (Heroku, Vercel) required for a production-ready full-stack application[80][81][^87].

The guide follows a structured learning path: **Front-End Fundamentals** → **Back-End APIs** → **Database Design** → **Integration** → **Deployment**, ensuring you build a solid foundation before tackling complex features[80][81][^90].

Part 1: Front-End Development with React

1.1 Core HTML, CSS, and JavaScript

Before diving into React, ensure you're comfortable with web fundamentals[80][87]:

HTML5:

- Semantic elements (header, nav, main, section, article, footer)
- Forms with validation (input types, required attributes)
- Accessibility attributes (ARIA labels, alt text, semantic structure)

CSS3:

- Flexbox and Grid for responsive layouts
- CSS Variables for theming (matching Tapin's color palette)
- Media queries for mobile-first design (320px → 768px → 1024px+)
- Transitions and animations for micro-interactions

JavaScript (ES6+):

- Arrow functions, destructuring, spread/rest operators

- Promises and async/await for API calls
- Array methods (map, filter, reduce) for data manipulation
- Template literals for dynamic content

Recommended Learning:

- MDN Web Docs (comprehensive reference)
- FreeCodeCamp Responsive Web Design Certification
- JavaScript30 by Wes Bos (practical exercises)

1.2 React Fundamentals

Master React concepts essential for building Tapin's user interface[81][87][^90]:

Core Concepts:

- **Components:** Functional components for all UI elements (ListingCard, FilterChips, NavigationBar)
- **Props:** Pass data from parent to child components (opportunity data, user info, organization details)
- **State Management:** useState hook for local state (form inputs, toggle states, filter selections)
- **useEffect Hook:** Handle side effects (API calls, subscriptions, DOM updates)
- **Conditional Rendering:** Show/hide elements based on state (loading spinners, error messages, empty states)

React Project Structure for Tapin:

```
tapin-frontend/
├── public/
│   ├── index.html
│   └── favicon.ico
├── src/
│   ├── components/
│   │   ├── common/
│   │   │   ├── Button.jsx
│   │   │   ├── Card.jsx
│   │   │   ├── Input.jsx
│   │   │   └── FilterChip.jsx
│   │   ├── layout/
│   │   │   ├── Header.jsx
│   │   │   ├── Footer.jsx
│   │   │   └── BottomNav.jsx
│   │   ├── listings/
│   │   │   ├── ListingCard.jsx
│   │   │   ├── ListingDetail.jsx
│   │   │   └── ListingGrid.jsx
│   │   └── map/
│   │       ├── MapView.jsx
│   │       └── MapMarker.jsx
│   ├── pages/
```

```
│   │   ├── Home.jsx
│   │   ├── Browse.jsx
│   │   ├── ListingDetail.jsx
│   │   ├── Dashboard.jsx
│   │   ├── Profile.jsx
│   │   ├── Login.jsx
│   │   └── Register.jsx
│   ├── context/
│   │   └── AuthContext.jsx
│   ├── api/
│   │   └── api.js
│   ├── utils/
│   │   └── helpers.js
│   ├── App.jsx
│   └── index.jsx
├── package.json
└── .env
```

Bootstrap Integration:

- Install React-Bootstrap: `npm install react-bootstrap bootstrap`
- Import Bootstrap CSS in `index.jsx`
- Use Bootstrap components (Container, Row, Col, Nav) for responsive grid
- Customize Bootstrap variables to match Tapin's bright blue brand color

1.3 React-Flask Communication

Learn how React communicates with your Flask backend via REST APIs[80][81][87][90]:

Axios Setup:

```
// src/api/api.js
import axios from 'axios';

const api = axios.create({
  baseURL: process.env.REACT_APP_API_URL || 'http://localhost:5000/api',
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json',
  },
});

export default api;
```

Making API Requests:

```
// Fetch volunteer opportunities
import api from '../api/api';

const fetchOpportunities = async (filters) =&gt; {
  try {
```

```
    const response = await api.get('/opportunities', { params: filters });
    return response.data;
  } catch (error) {
    console.error('Error fetching opportunities:', error);
    throw error;
  }
};

// Create new opportunity (organization owner)
const createOpportunity = async (opportunityData) =&gt; {
  try {
    const response = await api.post('/opportunities', opportunityData);
    return response.data;
  } catch (error) {
    console.error('Error creating opportunity:', error);
    throw error;
  }
};
```

Using API in React Components:

```
import React, { useState, useEffect } from 'react';
import { fetchOpportunities } from '../api/opportunities';

const Browse = () =&gt; {
  const [opportunities, setOpportunities] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() =&gt; {
    const loadOpportunities = async () =&gt; {
      try {
        setLoading(true);
        const data = await fetchOpportunities({ category: 'all' });
        setOpportunities(data);
      } catch (err) {
        setError('Failed to load opportunities');
      } finally {
        setLoading(false);
      }
    };

    loadOpportunities();
  }, []);

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error}</div>;

  return (
    <div>
      {opportunities.map(opp =&gt; (
        &lt;ListingCard key={opp.id} opportunity={opp} /&gt;
      ))}
    </div>
```

```
  );
};
```

1.4 Responsive Mobile-First Design

Implement responsive design principles for Tapin's mobile-first approach[81][87][^90]:

Media Query Strategy:

```
/* Mobile-first base styles (320px - 767px) */
.listing-card {
  width: 100%;
  padding: 16px;
  margin-bottom: 12px;
}

/* Tablet (768px - 1023px) */
@media (min-width: 768px) {
  .listing-grid {
    display: grid;
    grid-template-columns: repeat(2, 1fr);
    gap: 24px;
  }
}

/* Desktop (1024px+) */
@media (min-width: 1024px) {
  .listing-grid {
    grid-template-columns: repeat(3, 1fr);
    max-width: 1200px;
    margin: 0 auto;
  }
}
```

Touch-Friendly Components:

- Minimum 48x48px touch targets for all buttons and interactive elements
- 8px spacing between clickable elements to prevent accidental taps
- Large form inputs (48px height) to prevent iOS auto-zoom
- Swipeable cards for mobile browsing experience

Recommended Resources:

- React Documentation (official)
- React Tutorial by The Odin Project
- Full Modern React Tutorial by Net Ninja (YouTube)
- React Hooks Course by Web Dev Simplified

Part 2: Back-End Development with Flask

2.1 Flask Server Setup

Learn to set up a Flask server with REST API endpoints for Tapin[80][81][85][87][88][90]:

Project Structure:

```
tapin-backend/
├── app.py                 # Main Flask application
├── config.py              # Configuration settings
├── requirements.txt       # Python dependencies
├── .env                   # Environment variables
├── models/
│   ├── __init__.py
│   ├── user.py
│   ├── organization.py
│   ├── opportunity.py
│   └── review.py
├── routes/
│   ├── __init__.py
│   ├── auth.py
│   ├── opportunities.py
│   ├── users.py
│   └── organizations.py
├── utils/
│   ├── __init__.py
│   ├── validators.py
│   └── helpers.py
└── migrations/            # Database migrations
```

Basic Flask App (app.py):

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_cors import CORS
from flask_migrate import Migrate
from config import Config

# Initialize extensions
db = SQLAlchemy()
migrate = Migrate()

def create_app():
    app = Flask(__name__)
    app.config.from_object(Config)

    # Initialize extensions with app
    db.init_app(app)
    migrate.init_app(app, db)
    CORS(app, origins=['http://localhost:3000'])  # React dev server

    # Register blueprints (routes)
```

```
    from routes.auth import auth_bp
    from routes.opportunities import opportunities_bp
    from routes.users import users_bp

    app.register_blueprint(auth_bp, url_prefix='/api/auth')
    app.register_blueprint(opportunities_bp, url_prefix='/api/opportunities')
    app.register_blueprint(users_bp, url_prefix='/api/users')

    return app

if __name__ == '__main__':
    app = create_app()
    app.run(debug=True, port=5000)
```

Configuration (config.py):

```
import os
from datetime import timedelta

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'dev-secret-key-change-in-production'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
        'mysql://username:password@localhost:3306/tapin_db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    JWT_SECRET_KEY = os.environ.get('JWT_SECRET_KEY') or 'jwt-secret-key'
    JWT_ACCESS_TOKEN_EXPIRES = timedelta(hours=1)
    JWT_REFRESH_TOKEN_EXPIRES = timedelta(days=30)
```

Dependencies (requirements.txt):

```
Flask==3.0.0
Flask-SQLAlchemy==3.1.1
Flask-CORS==4.0.0
Flask-Migrate==4.0.5
Flask-JWT-Extended==4.6.0
PyMySQL==1.1.0
python-dotenv==1.0.0
bcrypt==4.1.2
```

2.2 REST API Endpoints

Design RESTful endpoints following best practices[80][81][85][87][88][90]:

HTTP Methods and Status Codes:

- **GET:** Retrieve resources (200 OK, 404 Not Found)
- **POST:** Create new resources (201 Created, 400 Bad Request)
- **PUT/PATCH:** Update resources (200 OK, 404 Not Found)
- **DELETE:** Remove resources (204 No Content, 404 Not Found)

Tapin API Endpoint Design:

```
# routes/opportunities.py
from flask import Blueprint, request, jsonify
from models.opportunity import Opportunity
from app import db

opportunities_bp = Blueprint('opportunities', __name__)

# GET /api/opportunities - List all opportunities (with filters)
@opportunities_bp.route('/', methods=['GET'])
def get_opportunities():
    category = request.args.get('category')
    location = request.args.get('location')
    date = request.args.get('date')

    query = Opportunity.query

    if category:
        query = query.filter_by(category=category)
    if location:
        query = query.filter(Opportunity.location.like(f'%{location}%'))
    if date:
        query = query.filter_by(date=date)

    opportunities = query.all()
    return jsonify([opp.to_dict() for opp in opportunities]), 200

# GET /api/opportunities/&lt;id&gt; - Get single opportunity
@opportunities_bp.route('/&lt;int:id&gt;', methods=['GET'])
def get_opportunity(id):
    opportunity = Opportunity.query.get_or_404(id)
    return jsonify(opportunity.to_dict()), 200

# POST /api/opportunities - Create new opportunity (auth required)
@opportunities_bp.route('/', methods=['POST'])
@jwt_required()  # Requires valid JWT token
def create_opportunity():
    data = request.get_json()

    # Validate required fields
    required_fields = ['title', 'description', 'organization_id', 'date', 'location']
    for field in required_fields:
        if field not in data:
            return jsonify({'error': f'{field} is required'}), 400

    opportunity = Opportunity(
        title=data['title'],
        description=data['description'],
        organization_id=data['organization_id'],
        date=data['date'],
        location=data['location'],
        category=data.get('category', 'General'),
        capacity=data.get('capacity', 10)
    )

    db.session.add(opportunity)
    db.session.commit()
```

```
    return jsonify(opportunity.to_dict()), 201

# PUT /api/opportunities/&lt;id&gt; - Update opportunity
@opportunities_bp.route('/&lt;int:id&gt;', methods=['PUT'])
@jwt_required()
def update_opportunity(id):
    opportunity = Opportunity.query.get_or_404(id)
    data = request.get_json()

    # Update fields if provided
    for field in ['title', 'description', 'date', 'location', 'capacity']:
        if field in data:
            setattr(opportunity, field, data[field])

    db.session.commit()
    return jsonify(opportunity.to_dict()), 200

# DELETE /api/opportunities/&lt;id&gt; - Delete opportunity
@opportunities_bp.route('/&lt;int:id&gt;', methods=['DELETE'])
@jwt_required()
def delete_opportunity(id):
    opportunity = Opportunity.query.get_or_404(id)
    db.session.delete(opportunity)
    db.session.commit()
    return '', 204
```

2.3 Authentication with JWT

Implement user registration, login, and JWT token management[[86][89][92][95][^97]:

Authentication Routes (routes/auth.py):

```
from flask import Blueprint, request, jsonify
from flask_jwt_extended import (
    create_access_token, create_refresh_token,
    jwt_required, get_jwt_identity
)
from models.user import User
from app import db
import bcrypt

auth_bp = Blueprint('auth', __name__)

# POST /api/auth/register - User registration
@auth_bp.route('/register', methods=['POST'])
def register():
    data = request.get_json()

    # Validate required fields
    if not all(k in data for k in ['email', 'password', 'name']):
        return jsonify({'error': 'Missing required fields'}), 400

    # Check if user already exists
```

```
    if User.query.filter_by(email=data['email']).first():
        return jsonify({'error': 'Email already registered'}), 400

    # Hash password with bcrypt
    hashed_password = bcrypt.hashpw(
        data['password'].encode('utf-8'),
        bcrypt.gensalt()
    ).decode('utf-8')

    # Create new user
    user = User(
        email=data['email'],
        password=hashed_password,
        name=data['name'],
        role=data.get('role', 'volunteer')  # 'volunteer' or 'organization'
    )

    db.session.add(user)
    db.session.commit()

    return jsonify({
        'message': 'User registered successfully',
        'user': user.to_dict()
    }), 201

# POST /api/auth/login - User login
@auth_bp.route('/login', methods=['POST'])
def login():
    data = request.get_json()

    # Find user by email
    user = User.query.filter_by(email=data.get('email')).first()

    if not user or not bcrypt.checkpw(
        data.get('password').encode('utf-8'),
        user.password.encode('utf-8')
    ):
        return jsonify({'error': 'Invalid credentials'}), 401

    # Create JWT tokens
    access_token = create_access_token(identity=user.id)
    refresh_token = create_refresh_token(identity=user.id)

    return jsonify({
        'access_token': access_token,
        'refresh_token': refresh_token,
        'user': user.to_dict()
    }), 200

# POST /api/auth/refresh - Refresh access token
@auth_bp.route('/refresh', methods=['POST'])
@jwt_required(refresh=True)
def refresh():
    current_user_id = get_jwt_identity()
    access_token = create_access_token(identity=current_user_id)
    return jsonify({'access_token': access_token}), 200
```

```
# GET /api/auth/me - Get current user
@auth_bp.route('/me', methods=['GET'])
@jwt_required()
def get_current_user():
    current_user_id = get_jwt_identity()
    user = User.query.get_or_404(current_user_id)
    return jsonify(user.to_dict()), 200
```

2.4 Security Best Practices

Implement essential security measures for production readiness[86][87][^89]:

Input Validation:

```
# utils/validators.py
import re

def validate_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None

def validate_password(password):
    # At least 8 characters, 1 uppercase, 1 lowercase, 1 number
    if len(password) &lt; 8:
        return False
    if not re.search(r'[A-Z]', password):
        return False
    if not re.search(r'[a-z]', password):
        return False
    if not re.search(r'\d', password):
        return False
    return True

def sanitize_input(data):
    # Remove potentially harmful characters
    if isinstance(data, str):
        return data.strip().replace('&lt;', '&lt;').replace('&gt;', '&gt;')
    return data
```

Password Security:

- **Never** store passwords in plain text
- Use bcrypt or Argon2 for password hashing
- Implement password strength requirements (8+ chars, mixed case, numbers)
- Add account lockout after failed login attempts (5 attempts = 15 min lockout)

Session Management:

- Use JWT tokens with short expiration (1 hour for access, 30 days for refresh)
- Store refresh tokens securely (HTTP-only cookies in production)

- Implement token refresh mechanism using axios interceptors
- Add logout functionality that invalidates tokens

Recommended Resources:

- Flask Mega-Tutorial by Miguel Grinberg
- Flask RESTful API Course by Tech With Tim (YouTube)
- Flask Documentation (official)
- Flask-JWT-Extended Documentation

Part 3: Database Design with MySQL

3.1 MySQL Database Setup

Configure MySQL database connection with Flask-SQLAlchemy[85][88][90][91][94][96]:

Install MySQL:

- **Windows:** Download MySQL Installer from mysql.com
- **Mac:** `brew install mysql`
- **Linux:** `sudo apt-get install mysql-server`

Create Tapin Database:

```
-- Connect to MySQL
mysql -u root -p

-- Create database
CREATE DATABASE tapin_db CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- Create user for application
CREATE USER 'tapin_user'@'localhost' IDENTIFIED BY 'secure_password';
GRANT ALL PRIVILEGES ON tapin_db.* TO 'tapin_user'@'localhost';
FLUSH PRIVILEGES;
```

Flask-SQLAlchemy Configuration:

```
# config.py
class Config:
    SQLALCHEMY_DATABASE_URI = 'mysql+pymysql://tapin_user:secure_password@localhost:3306/
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    SQLALCHEMY_ECHO = True  # Log SQL queries (disable in production)
```

3.2 Database Schema Design

Design normalized database schema for Tapin following best practices[106][109][112][115][^117]:

Core Tables:

1. Users Table:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    name VARCHAR(100) NOT NULL,
    role ENUM('volunteer', 'organization') NOT NULL,
    phone VARCHAR(20),
    location VARCHAR(255),
    bio TEXT,
    profile_picture VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_email (email),
    INDEX idx_role (role)
);
```

2. Organizations Table:

```
CREATE TABLE organizations (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    organization_name VARCHAR(255) NOT NULL,
    description TEXT,
    address VARCHAR(255),
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    website VARCHAR(255),
    verified BOOLEAN DEFAULT FALSE,
    rating DECIMAL(3, 2) DEFAULT 0.00,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    INDEX idx_location (latitude, longitude),
    INDEX idx_verified (verified)
);
```

3. Opportunities Table:

```
CREATE TABLE opportunities (
    id INT AUTO_INCREMENT PRIMARY KEY,
    organization_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    category ENUM('Community', 'Education', 'Environment', 'Healthcare',
                  'Animals', 'Arts', 'Technology', 'Food') NOT NULL,
    location VARCHAR(255) NOT NULL,
```

```
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    date DATE NOT NULL,
    start_time TIME,
    end_time TIME,
    capacity INT DEFAULT 10,
    current_signups INT DEFAULT 0,
    requirements TEXT,
    skills_needed VARCHAR(255),
    age_minimum INT,
    status ENUM('draft', 'published', 'full', 'completed', 'cancelled') DEFAULT 'draft',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (organization_id) REFERENCES organizations(id) ON DELETE CASCADE,
    INDEX idx_category (category),
    INDEX idx_date (date),
    INDEX idx_status (status),
    INDEX idx_location (latitude, longitude)
);
```

4. Signups Table (Many-to-Many Relationship):

```
CREATE TABLE signups (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    opportunity_id INT NOT NULL,
    status ENUM('registered', 'confirmed', 'completed', 'cancelled') DEFAULT 'registered'
    signed_up_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    checked_in_at TIMESTAMP NULL,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (opportunity_id) REFERENCES opportunities(id) ON DELETE CASCADE,
    UNIQUE KEY unique_signup (user_id, opportunity_id),
    INDEX idx_user (user_id),
    INDEX idx_opportunity (opportunity_id),
    INDEX idx_status (status)
);
```

5. Reviews Table:

```
CREATE TABLE reviews (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    opportunity_id INT NOT NULL,
    rating INT CHECK (rating &gt;= 1 AND rating &lt;= 5),
    comment TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (opportunity_id) REFERENCES opportunities(id) ON DELETE CASCADE,
    INDEX idx_opportunity (opportunity_id),
    INDEX idx_rating (rating)
);
```

6. Saved Opportunities Table:

```
CREATE TABLE saved_opportunities (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    opportunity_id INT NOT NULL,
    saved_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (opportunity_id) REFERENCES opportunities(id) ON DELETE CASCADE,
    UNIQUE KEY unique_save (user_id, opportunity_id),
    INDEX idx_user (user_id)
);
```

3.3 SQLAlchemy ORM Models

Create Python models mapping to database tables[85][88][90][91][94][96]:

User Model (models/user.py):

```
from app import db
from datetime import datetime

class User(db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(255), unique=True, nullable=False, index=True)
    password = db.Column(db.String(255), nullable=False)
    name = db.Column(db.String(100), nullable=False)
    role = db.Column(db.Enum('volunteer', 'organization'), nullable=False)
    phone = db.Column(db.String(20))
    location = db.Column(db.String(255))
    bio = db.Column(db.Text)
    profile_picture = db.Column(db.String(255))
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow

    # Relationships
    organization = db.relationship('Organization', backref='user', uselist=False, cascade
    signups = db.relationship('Signup', backref='user', cascade='all, delete-orphan')
    reviews = db.relationship('Review', backref='user', cascade='all, delete-orphan')
    saved_opportunities = db.relationship('SavedOpportunity', backref='user', cascade='al

    def to_dict(self):
        return {
            'id': self.id,
            'email': self.email,
            'name': self.name,
            'role': self.role,
            'phone': self.phone,
            'location': self.location,
            'bio': self.bio,
            'profile_picture': self.profile_picture,
            'created_at': self.created_at.isoformat() if self.created_at else None
        }
```

Opportunity Model (models/opportunity.py):

```
from app import db
from datetime import datetime

class Opportunity(db.Model):
    __tablename__ = 'opportunities'

    id = db.Column(db.Integer, primary_key=True)
    organization_id = db.Column(db.Integer, db.ForeignKey('organizations.id'), nullable=F
    title = db.Column(db.String(255), nullable=False)
    description = db.Column(db.Text, nullable=False)
    category = db.Column(db.Enum('Community', 'Education', 'Environment', 'Healthcare',
                                 'Animals', 'Arts', 'Technology', 'Food'), nullable=Fals
    location = db.Column(db.String(255), nullable=False)
    latitude = db.Column(db.Numeric(10, 8))
    longitude = db.Column(db.Numeric(11, 8))
    date = db.Column(db.Date, nullable=False)
    start_time = db.Column(db.Time)
    end_time = db.Column(db.Time)
    capacity = db.Column(db.Integer, default=10)
    current_signups = db.Column(db.Integer, default=0)
    requirements = db.Column(db.Text)
    skills_needed = db.Column(db.String(255))
    age_minimum = db.Column(db.Integer)
    status = db.Column(db.Enum('draft', 'published', 'full', 'completed', 'cancelled'), d
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow

    # Relationships
    organization = db.relationship('Organization', backref='opportunities')
    signups = db.relationship('Signup', backref='opportunity', cascade='all, delete-orpha
    reviews = db.relationship('Review', backref='opportunity', cascade='all, delete-orpha

    def to_dict(self):
        return {
            'id': self.id,
            'organization_id': self.organization_id,
            'organization_name': self.organization.organization_name if self.organizatior
            'title': self.title,
            'description': self.description,
            'category': self.category,
            'location': self.location,
            'latitude': float(self.latitude) if self.latitude else None,
            'longitude': float(self.longitude) if self.longitude else None,
            'date': self.date.isoformat() if self.date else None,
            'start_time': self.start_time.isoformat() if self.start_time else None,
            'end_time': self.end_time.isoformat() if self.end_time else None,
            'capacity': self.capacity,
            'current_signups': self.current_signups,
            'spots_available': self.capacity - self.current_signups,
            'requirements': self.requirements,
            'skills_needed': self.skills_needed,
            'age_minimum': self.age_minimum,
            'status': self.status,
```

```
            'created_at': self.created_at.isoformat() if self.created_at else None
        }
```

3.4 Database Migrations

Use Flask-Migrate to manage database schema changes[85][88][90][91]:

Initialize Migrations:

```
# Initialize migration repository
flask db init

# Create initial migration
flask db migrate -m "Initial database schema"

# Apply migration to database
flask db upgrade
```

Creating New Migrations:

```
# After modifying models, create new migration
flask db migrate -m "Add profile_picture to users"

# Review migration file in migrations/versions/
# Edit if necessary

# Apply migration
flask db upgrade

# Rollback migration if needed
flask db downgrade
```

Migration Best Practices:

- Always review generated migration files before applying
- Test migrations on development database first
- Create backup before running migrations on production
- Use descriptive migration messages
- Never edit applied migrations—create new ones instead

Recommended Resources:

- Flask-SQLAlchemy Documentation
- Miguel Grinberg's Flask Database Tutorial
- MySQL Tutorial by Programming with Mosh (YouTube)
- Database Design Course by freeCodeCamp

Part 4: Third-Party API Integration

4.1 Google Maps API Integration

Integrate Google Maps for location-based features[100][101][102][103][104][107]:

Step 1: Get API Key

1. Go to Google Cloud Console (console.cloud.google.com)
2. Create new project: "Tapin"
3. Enable Maps JavaScript API and Places API
4. Create API key under "Credentials"
5. Restrict API key to your domains (localhost for development)

Step 2: Install React Library

```
npm install @vis.gl/react-google-maps
```

Step 3: Create Map Component

```
// src/components/map/MapView.jsx
import React from 'react';
import { APIProvider, Map, AdvancedMarker } from '@vis.gl/react-google-maps';

const MapView = ({ opportunities, center, zoom = 12 }) =&gt; {
  return (
    &lt;APIProvider apiKey={process.env.REACT_APP_GOOGLE_MAPS_API_KEY}&gt;
      &lt;Map
        defaultCenter={center}
        defaultZoom={zoom}
        mapId={process.env.REACT_APP_MAP_ID}  // Required for advanced markers
        style={{ width: '100%', height: '500px' }}
      &gt;
        {opportunities.map(opp =&gt; (
          &lt;AdvancedMarker
            key={opp.id}
            position={{ lat: opp.latitude, lng: opp.longitude }}
            onClick={() =&gt; handleMarkerClick(opp)}
          /&gt;
        ))}
      &lt;/Map&gt;
    &lt;/APIProvider&gt;
  );
};

const handleMarkerClick = (opportunity) =&gt; {
  // Show opportunity details in bottom sheet or modal
  console.log('Selected opportunity:', opportunity);
};
```

```
export default MapView;
```

Step 4: Geocoding Addresses

```
// Convert address to lat/lng when creating opportunity
const geocodeAddress = async (address) =&gt; {
  const geocoder = new window.google.maps.Geocoder();

  return new Promise((resolve, reject) =&gt; {
    geocoder.geocode({ address }, (results, status) =&gt; {
      if (status === 'OK' &amp;&amp; results[^0]) {
        const location = results[^0].geometry.location;
        resolve({
          latitude: location.lat(),
          longitude: location.lng()
        });
      } else {
        reject(new Error('Geocoding failed'));
      }
    });
  });
};

// Usage in CreateOpportunity form
const handleSubmit = async (formData) =&gt; {
  const { latitude, longitude } = await geocodeAddress(formData.location);

  const opportunityData = {
    ...formData,
    latitude,
    longitude
  };

  await createOpportunity(opportunityData);
};
```

Map Features for Tapin:

- Display opportunity markers clustered by location
- Show user's current location (with permission)
- Filter visible opportunities by map bounds
- "Get Directions" button linking to Google Maps
- Map/List toggle for different browsing modes

4.2 Email API Integration (Optional)

Integrate email service for notifications using SendGrid or Mailgun:

SendGrid Setup:

```
pip install sendgrid
```

```
# utils/email.py
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail

def send_verification_email(user_email, verification_link):
    message = Mail(
        from_email='noreply@tapin.com',
        to_emails=user_email,
        subject='Verify Your Tapin Account',
        html_content=f'''
        <h1>Welcome to Tapin!</h1>
        <p>Click the link below to verify your email:</p>
        <a href="{verification_link}">Verify Email</a>
        '''
    )

    try:
        sg = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
        response = sg.send(message)
        return True
    except Exception as e:
        print(f'Error sending email: {e}')
        return False

def send_signup_confirmation(user_email, opportunity_title, organization_name):
    # Email confirming volunteer signup
    pass

def send_reminder_email(user_email, opportunity_details):
    # Reminder 24 hours before opportunity
    pass
```

Recommended Resources:

- Google Maps Platform Documentation
- React Google Maps Tutorial by Google (YouTube)
- SendGrid Python Quickstart Guide

Part 5: Version Control with Git & GitHub

5.1 Git Setup and Best Practices

Master Git for collaborative development and version control:

Initialize Repository:

```
# Navigate to project folder
cd tapin

# Initialize git repository
git init

# Add all files to staging
git add .

# Create initial commit
git commit -m "Initial commit: Project setup with React and Flask"

# Create GitHub repository (via GitHub website)
# Connect local repo to GitHub
git remote add origin https://github.com/yourusername/tapin.git

# Push to GitHub
git branch -M main
git push -u origin main
```

Git Workflow Best Practices:

1. **Feature Branch Strategy:**

```
# Create new branch for feature
git checkout -b feature/user-authentication

# Make changes, commit frequently
git add .
git commit -m "Add JWT authentication to Flask backend"

# Push feature branch to GitHub
git push origin feature/user-authentication

# Create Pull Request on GitHub for review
# Merge to main after approval
```

2. **Meaningful Commit Messages:**

```
# Bad commits
git commit -m "fix"
git commit -m "updated files"

# Good commits
git commit -m "Fix: Resolve signup button not triggering API call"
```

```
git commit -m "Feature: Add map view toggle to browse page"
git commit -m "Refactor: Extract opportunity card into reusable component"
```

3. **Commit Frequency:**

- Commit after completing a logical unit of work
- Commit before switching tasks or branches
- Commit at end of each work session
- Don't commit broken code to main branch

5.2 GitHub Project Management

Use GitHub's project management tools for organization:

GitHub Projects (Kanban Board):

1. Create Project Board:
   - Go to repository → Projects → New Project
   - Choose "Board" template
   - Create columns: Backlog, To Do, In Progress, Review, Done
2. Create Issues for Tasks:
   - Issues → New Issue
   - Title: "Implement user registration form"
   - Description: Detailed requirements
   - Labels: frontend, backend, bug, enhancement, documentation
   - Assign to team members
   - Link to milestone (e.g., "MVP Launch")
3. Organize with Milestones:
   - Milestone 1: Basic Setup (Week 1-2)
   - Milestone 2: User Authentication (Week 3-4)
   - Milestone 3: Opportunity CRUD (Week 5-6)
   - Milestone 4: Map Integration (Week 7-8)
   - Milestone 5: Deployment (Week 9-10)

Sample Issue Template:

```
## User Story
As a volunteer, I want to browse opportunities by category so that I can find causes I ca

## Acceptance Criteria
- [ ] Display category filter chips horizontally
- [ ] Filter opportunities when category selected
- [ ] Show active filter in bright blue
```

```
- [ ] Clear filter button removes selection
- [ ] Update URL with active filter for sharing

## Technical Notes
- Use React state for active filters
- API endpoint: GET /api/opportunities?category=Education
- Bootstrap filter chip component
```

5.3 README Documentation

Create comprehensive README for your repository:

README.md **Structure:**

```
# Tapin - Local Volunteer Platform

Connect volunteers with community organizations for meaningful service opportunities.

## Features

- 🔍 Browse volunteer opportunities by category and location
- 🗺️ Interactive map view with location markers
- 👤 User profiles for volunteers and organizations
- 📊 Organization dashboards with analytics
- ⭐ Review and rating system
- 📧 Email notifications for signups and reminders

## Tech Stack

**Frontend:**
- React.js 18
- React-Bootstrap
- Axios
- Google Maps API (@vis.gl/react-google-maps)

**Backend:**
- Flask 3.0
- Flask-SQLAlchemy
- Flask-JWT-Extended
- MySQL

## Getting Started

### Prerequisites

- Node.js 18+ and npm
- Python 3.10+
- MySQL 8.0+

### Installation

**1. Clone repository:**
\```bash
git clone https://github.com/yourusername/tapin.git
```

```
cd tapin
```

2. Backend Setup:
```bash
cd backend
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate
pip install -r requirements.txt

# Create .env file
cp .env.example .env
# Edit .env with your database credentials

# Initialize database
flask db upgrade
flask run
```

3. Frontend Setup:
```bash
cd frontend
npm install

# Create .env file
cp .env.example .env
# Add Google Maps API key

npm start
```

Project Structure

```
tapin/
├── backend/
│   ├── app.py
│   ├── config.py
│   ├── models/
│   ├── routes/
│   └── requirements.txt
├── frontend/
│   ├── src/
│   │   ├── components/
│   │   ├── pages/
│   │   ├── context/
│   │   └── api/
│   └── package.json
└── README.md
```

API Documentation

Authentication Endpoints

- `POST /api/auth/register` - Register new user

```
- `POST /api/auth/login` - Login user
- `POST /api/auth/refresh` - Refresh access token
- `GET /api/auth/me` - Get current user

### Opportunity Endpoints

- `GET /api/opportunities` - List opportunities (with filters)
- `GET /api/opportunities/:id` - Get single opportunity
- `POST /api/opportunities` - Create opportunity (auth required)
- `PUT /api/opportunities/:id` - Update opportunity
- `DELETE /api/opportunities/:id` - Delete opportunity

## Contributing

1. Fork the repository
2. Create feature branch (`git checkout -b feature/AmazingFeature`)
3. Commit changes (`git commit -m 'Add AmazingFeature'`)
4. Push to branch (`git push origin feature/AmazingFeature`)
5. Open Pull Request

## License

MIT License

## Contact

Your Name - youremail@example.com
Project Link: https://github.com/yourusername/tapin
\```
```

Recommended Resources:

- Pro Git Book (free online)
- GitHub Guides (guides.github.com)
- Git Tutorial by The Net Ninja (YouTube)

Part 6: Deployment Strategies

6.1 Heroku Deployment (Backend)

Deploy Flask backend to Heroku's cloud platform[[105][108][111][116][^118]:

Step 1: Prepare Flask App for Heroku

Create `Procfile` in backend root:

```
web: gunicorn app:app
```

Create `runtime.txt`:

```
python-3.11.5
```

Update `requirements.txt`:

```
# Generate requirements
pip freeze &gt; requirements.txt

# Add gunicorn for production server
pip install gunicorn
pip freeze &gt; requirements.txt
```

Step 2: Heroku Setup

```
# Install Heroku CLI
# Windows: Download from heroku.com
# Mac: brew install heroku/brew/heroku
# Linux: curl https://cli-assets.heroku.com/install.sh | sh

# Login to Heroku
heroku login

# Create Heroku app
heroku create tapin-backend

# Add MySQL add-on (ClearDB or JawsDB)
heroku addons:create jawsdb:kitefin

# Get database URL
heroku config:get JAWSDB_URL

# Set environment variables
heroku config:set SECRET_KEY=your-secret-key-here
heroku config:set JWT_SECRET_KEY=your-jwt-secret-here

# Deploy to Heroku
git push heroku main

# Run database migrations
heroku run flask db upgrade

# Open app
heroku open
```

Step 3: GitHub Integration

1. Go to Heroku Dashboard
2. Select your app → Deploy tab
3. Connect to GitHub repository
4. Enable "Automatic Deploys" from main branch
5. Every push to main will auto-deploy

GitHub Student Developer Pack:

- Apply at education.github.com/pack
- Receive $13/month Heroku credits for 24 months ($312 total)[^111]
- Covers Eco Dynos ($5/month) + Mini Postgres ($5/month) + Mini Key-Value Store ($3/month)

6.2 Vercel Deployment (Frontend)

Deploy React frontend to Vercel for optimal performance[108][114]:

Step 1: Prepare React App

Update `.env` for production:

```
REACT_APP_API_URL=https://tapin-backend.herokuapp.com/api
REACT_APP_GOOGLE_MAPS_API_KEY=your-api-key
```

Step 2: Vercel Setup

```
# Install Vercel CLI
npm install -g vercel

# Login
vercel login

# Deploy from project directory
cd frontend
vercel

# Answer prompts:
# - Set up and deploy? Yes
# - Which scope? Your account
# - Link to existing project? No
# - Project name: tapin-frontend
# - Directory: ./
# - Override settings? No

# Production deployment
vercel --prod
```

Step 3: Automatic Deployments

1. Go to vercel.com/dashboard
2. Import GitHub repository
3. Configure:
    - Framework Preset: Create React App
    - Root Directory: frontend
    - Build Command: npm run build

- Output Directory: build
- Environment Variables: Add REACT_APP_* variables

4. Deploy

Every push to main branch auto-deploys to production.

6.3 Alternative: Render.com (Free Tier)

Render offers free hosting for both frontend and backend:

Backend (Flask):

1. Create account at render.com
2. New → Web Service
3. Connect GitHub repository
4. Configure:
   - Environment: Python 3
   - Build Command: `pip install -r requirements.txt`
   - Start Command: `gunicorn app:app`
   - Add environment variables
5. Create free MySQL database on Render
6. Deploy

Frontend (React):

1. New → Static Site
2. Connect GitHub repository
3. Configure:
   - Build Command: `npm run build`
   - Publish Directory: `build`
   - Add environment variables
4. Deploy

Recommended Resources:

- Heroku Dev Center Documentation
- Deploying Full-Stack Apps by Codecademy
- Vercel Documentation
- Render Deploy Guides

Part 7: Development Workflow & Timeline

7.1 Recommended Development Order

Follow this sequential approach to build Tapin systematically:

Phase 1: Foundation (Weeks 1-2)

- Set up Git repository and GitHub project board
- Initialize React frontend project with Bootstrap
- Initialize Flask backend project
- Set up MySQL database and test connection
- Create basic README and project documentation

Phase 2: Authentication (Weeks 3-4)

- Design database schema (users, organizations tables)
- Implement user registration backend (Flask route + SQLAlchemy model)
- Implement login backend with JWT tokens
- Create registration form in React
- Create login form in React
- Set up Axios with JWT interceptors
- Test authentication flow end-to-end

Phase 3: Core Features - Opportunities (Weeks 5-7)

- Design opportunities table schema
- Create Opportunity model in Flask
- Implement CRUD API endpoints for opportunities
- Build Browse page in React (list view)
- Build Opportunity Detail page
- Implement filters (category, date, location)
- Add search functionality

Phase 4: Map Integration (Week 8)

- Get Google Maps API key
- Install and configure React Google Maps library
- Add latitude/longitude to opportunities table
- Implement geocoding for addresses
- Create MapView component
- Add map/list toggle to Browse page

- Test marker clustering

Phase 5: User Profiles & Dashboard (Week 9)

- Create signups and reviews tables
- Implement signup/registration for opportunities
- Build volunteer profile page
- Build organization dashboard
- Display volunteer statistics
- Show organization analytics

Phase 6: Polish & Deployment (Week 10)

- Add loading states and error handling
- Implement responsive design for mobile
- Write unit tests for critical functions
- Deploy backend to Heroku
- Deploy frontend to Vercel
- Configure environment variables
- Test production deployment
- Write deployment documentation

7.2 Daily Development Checklist

Morning Routine:

- [ ] Review GitHub project board
- [ ] Select 1-2 issues to work on today
- [ ] Create feature branch for today's work
- [ ] Pull latest changes from main

During Development:

- [ ] Write code in small, testable chunks
- [ ] Commit after each completed unit of work
- [ ] Test functionality in browser/Postman
- [ ] Document any issues or blockers

End of Day:

- [ ] Commit all work with meaningful messages
- [ ] Push feature branch to GitHub
- [ ] Update issue status on project board
- [ ] Review tomorrow's priorities

7.3 Testing Strategies

Backend Testing:

```
# tests/test_auth.py
import pytest
from app import create_app, db
from models.user import User

@pytest.fixture
def client():
    app = create_app('testing')
    with app.test_client() as client:
        with app.app_context():
            db.create_all()
        yield client
        with app.app_context():
            db.drop_all()

def test_register_user(client):
    response = client.post('/api/auth/register', json={
        'email': 'test@example.com',
        'password': 'TestPass123',
        'name': 'Test User',
        'role': 'volunteer'
    })
    assert response.status_code == 201
    assert 'user' in response.json

def test_login_user(client):
    # Create user first
    client.post('/api/auth/register', json={
        'email': 'test@example.com',
        'password': 'TestPass123',
        'name': 'Test User',
        'role': 'volunteer'
    })

    # Test login
    response = client.post('/api/auth/login', json={
        'email': 'test@example.com',
        'password': 'TestPass123'
    })
    assert response.status_code == 200
    assert 'access_token' in response.json
```

Frontend Testing (React Testing Library):

```
// src/components/__tests__/ListingCard.test.js
import { render, screen } from '@testing-library/react';
import ListingCard from '../ListingCard';

const mockOpportunity = {
  id: 1,
```

```
  title: 'Beach Cleanup',
  organization_name: 'Ocean Conservation',
  location: 'Santa Monica Beach',
  date: '2025-11-15',
  category: 'Environment'
};

test('renders opportunity details', () =&gt; {
  render(&lt;ListingCard opportunity={mockOpportunity} /&gt;);

  expect(screen.getByText('Beach Cleanup')).toBeInTheDocument();
  expect(screen.getByText('Ocean Conservation')).toBeInTheDocument();
  expect(screen.getByText(/Santa Monica Beach/i)).toBeInTheDocument();
});
```

Manual Testing Checklist:

- [ ] User registration creates account in database
- [ ] Login returns valid JWT token
- [ ] Protected routes redirect unauthenticated users
- [ ] Create opportunity form validates required fields
- [ ] Browse page displays opportunities correctly
- [ ] Filters update displayed opportunities
- [ ] Map markers match opportunity locations
- [ ] Signup button updates capacity counter
- [ ] Review submission saves to database
- [ ] Mobile responsive layout works on phone screen

Part 8: Common Challenges & Solutions

8.1 CORS Issues

Problem: React frontend can't connect to Flask backend

Solution:

```
# app.py
from flask_cors import CORS

app = Flask(__name__)
CORS(app, resources={
    r"/api/*": {
        "origins": ["http://localhost:3000", "https://tapin-frontend.vercel.app"],
        "methods": ["GET", "POST", "PUT", "DELETE"],
        "allow_headers": ["Content-Type", "Authorization"]
```

```
    }
})
```

8.2 Database Connection Errors

Problem: "Can't connect to MySQL server"

Solutions:

- Verify MySQL service is running: `mysql.server start` (Mac) or check Services (Windows)
- Check credentials in `.env` file
- Ensure database exists: `CREATE DATABASE tapin_db;`
- Test connection: `mysql -u tapin_user -p tapin_db`

8.3 JWT Token Expiration

Problem: Users logged out unexpectedly

Solution: Implement token refresh with Axios interceptors

```
// src/api/api.js
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:5000/api'
});

// Add token to every request
api.interceptors.request.use(
  (config) =&gt; {
    const token = localStorage.getItem('access_token');
    if (token) {
      config.headers['Authorization'] = `Bearer ${token}`;
    }
    return config;
  },
  (error) =&gt; Promise.reject(error)
);

// Refresh token on 401 errors
api.interceptors.response.use(
  (response) =&gt; response,
  async (error) =&gt; {
    const originalRequest = error.config;

    if (error.response?.status === 401 &amp;&amp; !originalRequest._retry) {
      originalRequest._retry = true;

      try {
        const refreshToken = localStorage.getItem('refresh_token');
        const response = await axios.post('http://localhost:5000/api/auth/refresh', {
          refresh_token: refreshToken
```

```
        });

        const newAccessToken = response.data.access_token;
        localStorage.setItem('access_token', newAccessToken);

        originalRequest.headers['Authorization'] = `Bearer ${newAccessToken}`;
        return api(originalRequest);
      } catch (err) {
        // Refresh failed, redirect to login
        localStorage.removeItem('access_token');
        localStorage.removeItem('refresh_token');
        window.location.href = '/login';
      }
    }

    return Promise.reject(error);
  }
);

export default api;
```

8.4 Deployment Issues

Problem: App works locally but fails in production

Common Causes:

- Environment variables not set (add to Heroku/Vercel dashboard)
- Database migrations not run (`heroku run flask db upgrade`)
- Build errors due to missing dependencies (check logs)
- API URL hardcoded to localhost (use environment variables)

Debugging:

```
# Check Heroku logs
heroku logs --tail

# Check Vercel logs
vercel logs

# Test production API
curl https://tapin-backend.herokuapp.com/api/opportunities
```

Part 9: Additional Resources

9.1 Official Documentation

- **React:** reactjs.org/docs
- **Flask:** flask.palletsprojects.com
- **Flask-SQLAlchemy:** flask-sqlalchemy.palletsprojects.com
- **MySQL:** dev.mysql.com/doc
- **Google Maps Platform:** developers.google.com/maps

9.2 Video Tutorials

- **Full-Stack React + Flask:** "Create a React + Flask application" by Miguel Grinberg (YouTube)
- **MySQL for Python:** "MySQL for Python Developers" by PlanetScale
- **Flask REST API:** "Flask REST API Tutorial" by Tech With Tim
- **React Authentication:** "React Login Authentication with JWT" by Dave Gray

9.3 Community Support

- **Stack Overflow:** Use tags `reactjs`, `flask`, `mysql`, `sqlalchemy`
- **Reddit:** r/flask, r/reactjs, r/learnprogramming
- **Discord:** Reactiflux (React community), Python Discord
- **GitHub Discussions:** Ask in your repository discussions

9.4 Tools & Extensions

VS Code Extensions:

- ESLint (JavaScript linting)
- Prettier (code formatting)
- Python (IntelliSense)
- SQLTools (database management)
- GitLens (Git integration)
- Thunder Client (API testing)

Development Tools:

- Postman or Insomnia (API testing)
- MySQL Workbench (database GUI)
- Chrome DevTools (frontend debugging)
- React Developer Tools (browser extension)

Part 10: Project Success Checklist

10.1 MVP Features (Minimum Viable Product)

Core features required for final project submission:

- [x] **User Authentication**
  - [ ] Registration with email validation
  - [ ] Login with JWT tokens
  - [ ] Password hashing with bcrypt
  - [ ] Protected routes
- [x] **Opportunity Management**
  - [ ] Create opportunity (organization role)
  - [ ] View all opportunities (browse page)
  - [ ] View single opportunity (detail page)
  - [ ] Filter by category and location
  - [ ] Search by keyword
- [x] **Volunteer Features**
  - [ ] Sign up for opportunities
  - [ ] View profile with statistics
  - [ ] Save opportunities for later
  - [ ] Leave reviews and ratings
- [x] **Organization Features**
  - [ ] Dashboard with analytics
  - [ ] Manage posted opportunities
  - [ ] View volunteer sign-ups
  - [ ] Edit/delete opportunities
- [x] **Map Integration**
  - [ ] Display opportunities on map
  - [ ] Custom markers by category
  - [ ] Map/list toggle view
  - [ ] Geocode addresses to coordinates
- [x] **Responsive Design**
  - [ ] Mobile-first layout
  - [ ] Tablet breakpoint (768px)
  - [ ] Desktop breakpoint (1024px)

    - [ ] Touch-friendly 48x48px targets
- [x] **Deployment**
    - [ ] Backend deployed to Heroku
    - [ ] Frontend deployed to Vercel
    - [ ] Database hosted on Heroku/Render
    - [ ] Environment variables configured
    - [ ] HTTPS enabled

10.2 Documentation Requirements

- [ ] README.md with setup instructions
- [ ] **API Documentation** (endpoints, request/response examples)
- [ ] **Database Schema Diagram** (ERD)
- [ ] **User Guide** (how to use the application)
- [ ] **Developer Notes** (architecture decisions)

10.3 Code Quality

- [ ] Meaningful variable and function names
- [ ] Consistent code formatting (Prettier/Black)
- [ ] Comments explaining complex logic
- [ ] No hardcoded credentials (use .env)
- [ ] Error handling for API calls
- [ ] Input validation on frontend and backend
- [ ] Git history with meaningful commits

Conclusion

Congratulations on embarking on your full-stack development journey with Tapin! This guide has provided you with:

✅ **Front-end skills** (React, Bootstrap, responsive design)
✅ **Back-end expertise** (Flask, REST APIs, JWT authentication)
✅ **Database knowledge** (MySQL, SQLAlchemy ORM, schema design)
✅ **Third-party integrations** (Google Maps API, email services)
✅ **Version control** (Git, GitHub, project management)
✅ **Deployment strategies** (Heroku, Vercel, production-ready apps)
✅ **Best practices** (security, testing, documentation)

Next Steps:

1. **Start Small:** Begin with Phase 1 (Foundation setup)

2. **Learn by Doing:** Build each feature incrementally
3. **Test Frequently:** Verify functionality at every step
4. **Commit Often:** Maintain detailed Git history
5. **Ask for Help:** Use community resources when stuck
6. **Stay Organized:** Update your GitHub project board daily

Remember: Every experienced developer was once a beginner. The key to success is **consistent practice, curiosity, and persistence**. Your Tapin project will not only demonstrate your technical skills but also showcase your ability to build a real-world application that makes a positive impact on communities.

You've got this! Happy coding!

Quick Reference Commands

React Development:

```
npm install                    # Install dependencies
npm start                      # Start development server
npm run build                  # Build for production
```

Flask Development:

```
python -m venv venv            # Create virtual environment
source venv/bin/activate       # Activate (Mac/Linux)
venv\Scripts\activate          # Activate (Windows)
pip install -r requirements.txt # Install dependencies
flask run                      # Start development server
flask db upgrade               # Run database migrations
```

Git Commands:

```
git status                     # Check repository status
git add .                      # Stage all changes
git commit -m "message"        # Commit with message
git push origin main           # Push to GitHub
git pull origin main           # Pull latest changes
git checkout -b feature/name   # Create feature branch
```

Deployment:

```
heroku login                   # Login to Heroku
heroku create app-name         # Create Heroku app
git push heroku main           # Deploy to Heroku
heroku logs --tail             # View logs
vercel                         # Deploy to Vercel
```

Document Version: 1.0

Last Updated: October 28, 2025

For: Tapin Full-Stack Project Development

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39]

⁂

1. https://stackoverflow.com/questions/76117698/build-a-app-with-flask-reactjs-and-mysql-workbench
2. https://reference.nirajankhatiwada.com.np/posts/pages/react/auth/
3. https://blog.stackademic.com/full-stack-application-using-react-and-python-with-flask-0e3a768a0fa5
4. https://www.digitalocean.com/community/tutorials/how-to-use-flask-sqlalchemy-to-interact-with-databases-in-a-flask-application
5. https://permify.co/post/jwt-authentication-in-react/
6. https://blog.appseed.us/flask-react-full-stack-seed-projects/
7. https://www.j-labs.pl/en/tech-blog/flask-restful-with-sqlalchemy/
8. https://dev.to/sanjayttg/jwt-authentication-in-react-with-react-router-1d03
9. https://realpython.com/flask-connexion-rest-api-part-2/
10. https://mihai-andrei.com/blog/jwt-authentication-using-axios-interceptors/
11. https://www.geeksforgeeks.org/python/connect-flask-to-a-database-with-flask-sqlalchemy/
12. https://4geeks.com/lesson/react-flask-template
13. https://www.youtube.com/watch?v=nI8PYZNFtac
14. https://developers.google.com/codelabs/maps-platform/maps-platform-101-react-js
15. https://www.youtube.com/watch?v=PfZ4oLftItk
16. https://blog.logrocket.com/integrating-google-maps-react/
17. https://visgl.github.io/react-google-maps/docs
18. https://dev.to/aneeqakhan/how-to-add-google-maps-to-your-web-app-using-react-3c3e
19. https://www.heroku.com/students/
20. https://planetscale.com/learn/courses/mysql-for-python-developers/building-a-flask-app-with-mysql/creating-the-schema
21. https://mapsplatform.google.com/resources/blog/google-maps-platform-graduates-react-integration-library-to-1-0/
22. https://www.codecademy.com/article/deploying-a-full-stack-app-with-heroku
23. https://www.youtube.com/watch?v=yrk1RiuH8t4
24. https://www.nucamp.co/blog/coding-bootcamp-back-end-with-python-and-sql-database-integration-in-flask-a-howto-guide
25. https://www.emgoto.com/react-map-component/
26. https://www.heroku.com/github-students/
27. https://realpython.com/flask-database/
28. https://developers.google.com/maps/documentation/javascript/tutorials
29. https://www.youtube.com/watch?v=fUYvDBWEELU

30. https://www.reddit.com/r/learnpython/comments/191kcca/good_practices_with_database_and_flask/
31. https://www.reddit.com/r/webdev/comments/1buq30g/where_to_deploy_my_fullstack_project_im_lost/
32. https://stackoverflow.com/questions/18268196/best-practices-while-designing-databases-in-mysql
33. https://stackoverflow.com/questions/58907865/deploy-full-stack-app-with-heroku-express-back-end-react-front-end
34. https://www.reddit.com/r/react/comments/1i9lfvn/what_backenddatabase_stack_you_would_recommend/
35. https://github.com/jmreddy2106/React-Flask-Python-WebApp
36. https://flask-sqlalchemy.readthedocs.io/en/stable/quickstart/
37. https://codevoweb.com/react-query-context-api-axios-interceptors-jwt-auth/
38. https://www.esparkinfo.com/software-development/technologies/reactjs/react-with-flask
39. https://dev.to/francescoxx/build-a-crud-rest-api-in-python-using-flask-sqlalchemy-postgres-docker-28lo