Walkable MVP Task Tracker

This document tracks the progress of tasks for the Walkable MVP development.

Status Key:

- [ ] To Do
- [x] Complete
- [>] In Progress
- [!] Blocked

Phase 1 – User Authentication

Goal: Implement and thoroughly test user authentication functionalities, including sign-up, login, and logout, along with an authentication-aware navigation bar.

Tasks:

- [ ] **Task 1.1: Backend User Authentication API Development**
    - [ ] Subtask 1.1.1: Design and implement user model (username, email, password hash).
    - [ ] Subtask 1.1.2: Develop API endpoint for user registration (POST /api/register).
    - [ ] Subtask 1.1.3: Develop API endpoint for user login (POST /api/login) with JWT or session management.
    - [ ] Subtask 1.1.4: Develop API endpoint for user logout (POST /api/logout).
    - [ ] Subtask 1.1.5: Implement password hashing and salting (e.g., bcrypt).
    - [ ] Subtask 1.1.6: Implement JWT token generation and validation for authenticated requests.
    - [ ] Subtask 1.1.7: Write unit tests for all authentication API endpoints.
- [ ] **Task 1.2: Frontend User Authentication UI Development**
    - [ ] Subtask 1.2.1: Design and implement user registration form.
    - [ ] Subtask 1.2.2: Design and implement user login form.
    - [ ] Subtask 1.2.3: Integrate registration form with backend API.

    - [ ] Subtask 1.2.4: Integrate login form with backend API, storing JWT/session token securely.
    - [ ] Subtask 1.2.5: Implement client-side validation for forms.

- [ ] **Task 1.3: Auth-aware Navbar Implementation**

    - [ ] Subtask 1.3.1: Implement conditional rendering for navigation links (Login/Signup vs. Profile/Logout) based on authentication status.
    - [ ] Subtask 1.3.2: Implement logout functionality in the navbar, clearing client-side tokens/sessions.

- [ ] **Task 1.4: Integration Testing (Phase 1)**

    - [ ] Subtask 1.4.1: Test end-to-end user registration flow.
    - [ ] Subtask 1.4.2: Test end-to-end user login flow.
    - [ ] Subtask 1.4.3: Test end-to-end user logout flow.
    - [ ] Subtask 1.4.4: Verify navbar state changes correctly upon login/logout.
    - [ ] Subtask 1.4.5: Test authentication persistence across page refreshes.

Dependencies:

- Task 1.2 depends on Task 1.1 (Frontend UI needs Backend API).
- Task 1.3 depends on Task 1.2 (Navbar needs authentication status from frontend).
- Task 1.4 depends on Tasks 1.1, 1.2, and 1.3 (Integration testing requires all components to be in place).

Phase 2 – Discover Page + Map

Goal: Develop an interactive map-based discover page that displays nearby tours and the user's current location.

Tasks:

- [ ] **Task 2.1: Map Integration**

    - [ ] Subtask 2.1.1: Integrate Leaflet.js library into the frontend.
    - [ ] Subtask 2.1.2: Initialize and display a basic interactive map.

- [ ] **Task 2.2: Geolocation Implementation**

    - [ ] Subtask 2.2.1: Implement browser Geolocation API to get user's current location.
    - [ ] Subtask 2.2.2: Display user's location on the map with a custom marker.

    - [ ] Subtask 2.2.3: Handle geolocation permissions and errors gracefully.

- [ ] **Task 2.3: Tour Data Display on Map**

    - [ ] Subtask 2.3.1: Design backend API endpoint to fetch nearby tours (e.g., GET /api/tours/nearby?lat=&lon=&radius=).
    - [ ] Subtask 2.3.2: Implement frontend logic to call the nearby tours API based on user's location.
    - [ ] Subtask 2.3.3: Parse tour data and display tour pins/markers on the Leaflet map.
    - [ ] Subtask 2.3.4: Implement basic popup/tooltip for tour pins showing tour title.

- [ ] **Task 2.4: Integration Testing (Phase 2)**

    - [ ] Subtask 2.4.1: Verify map loads correctly.
    - [ ] Subtask 2.4.2: Test user geolocation accuracy and marker display.
    - [ ] Subtask 2.4.3: Test fetching and displaying nearby tours.
    - [ ] Subtask 2.4.4: Verify tour pin popups work as expected.

Dependencies:

- Task 2.3 depends on Task 2.1 and Task 2.2 (Displaying tours requires a map and user location).

Phase 3 – Create Tour Flow

Goal: Enable creators to create and upload new audio tours, including tour details, map placement, and audio files.

Tasks:

- [ ] **Task 3.1: Tour Creation Form Development**

    - [ ] Subtask 3.1.1: Design and implement a multi-step form for tour creation (title, description, category).
    - [ ] Subtask 3.1.2: Implement client-side validation for form fields.

- [ ] **Task 3.2: Map Integration for Tour Placement**

    - [ ] Subtask 3.2.1: Integrate map (Leaflet) into the tour creation form.
    - [ ] Subtask 3.2.2: Implement address search functionality (geocoding API integration).

    - [ ] Subtask 3.2.3: Allow clickable pin placement on the map to set tour location.
    - [ ] Subtask 3.2.4: Store selected coordinates with the tour data.

- [ ] **Task 3.3: Audio File Upload**
    - [ ] Subtask 3.3.1: Implement file input for audio file uploads.
    - [ ] Subtask 3.3.2: Develop backend API endpoint for secure audio file storage (e.g., POST /api/tours/upload-audio).
    - [ ] Subtask 3.3.3: Implement frontend logic to upload audio files to the backend.
    - [ ] Subtask 3.3.4: Handle file size limits and accepted audio formats.

- [ ] **Task 3.4: Tour Data Submission**
    - [ ] Subtask 3.4.1: Develop backend API endpoint to receive and save complete tour data (POST /api/tours).
    - [ ] Subtask 3.4.2: Implement frontend logic to submit all tour data (text, coordinates, audio file references) to the backend.
    - [ ] Subtask 3.4.3: Associate the created tour with the authenticated creator's user ID.

- [ ] **Task 3.5: Integration Testing (Phase 3)**
    - [ ] Subtask 3.5.1: Test end-to-end tour creation flow, including form submission.
    - [ ] Subtask 3.5.2: Verify map address search and pin placement accuracy.
    - [ ] Subtask 3.5.3: Test audio file upload functionality and storage.
    - [ ] Subtask 3.5.4: Confirm tour data is correctly saved and associated with the creator.

Dependencies:

- Task 3.2 depends on Task 3.1 (Map for placement is part of the form).
- Task 3.3 depends on Task 3.1 (Audio upload is part of the form).
- Task 3.4 depends on Tasks 3.1, 3.2, and 3.3 (Submission requires all data to be collected).

Phase 4 – Tour Discovery Experience

Goal: Enhance the discover page to fetch and display real tour data, implement filtering/searching, and create a detailed tour page with playable audio.

Tasks:

- [ ] **Task 4.1: Discover Page Data Fetching**
    - [ ] Subtask 4.1.1: Modify existing discover page to fetch actual tour data from the backend (GET /api/tours).
    - [ ] Subtask 4.1.2: Display tour listings with relevant information (title, description snippet, creator).
- [ ] **Task 4.2: Filter and Search Functionality**
    - [ ] Subtask 4.2.1: Implement search bar for tour titles and descriptions.
    - [ ] Subtask 4.2.2: Implement filtering options (e.g., by category, duration).
    - [ ] Subtask 4.2.3: Develop backend API to support filtered and searched queries (e.g., GET /api/tours?search=&category=).
    - [ ] Subtask 4.2.4: Integrate frontend search/filter with backend API.
- [ ] **Task 4.3: Tour Detail Page Development**
    - [ ] Subtask 4.3.1: Design and implement a dedicated tour detail page (e.g., /tours/:id).
    - [ ] Subtask 4.3.2: Fetch complete tour details from backend (GET /api/tours/:id).
    - [ ] Subtask 4.3.3: Display tour title, full description, creator info, and map with tour pin.
    - [ ] Subtask 4.3.4: Integrate an audio player to play the tour's audio file.
- [ ] **Task 4.4: Integration Testing (Phase 4)**
    - [ ] Subtask 4.4.1: Test discover page displaying real tour data.
    - [ ] Subtask 4.4.2: Verify search and filter functionality works correctly.
    - [ ] Subtask 4.4.3: Test navigation to tour detail page.
    - [ ] Subtask 4.4.4: Verify all tour details are displayed accurately on the detail page.
    - [ ] Subtask 4.4.5: Test audio playback on the tour detail page.

Dependencies:

- Task 4.1 depends on Task 3.4 (Need existing tour data to fetch).
- Task 4.2 depends on Task 4.1 (Filtering/searching applies to fetched data).
- Task 4.3 depends on Task 3.4 (Tour detail page needs complete tour data).

Phase 5 – User & Creator Profiles

Goal: Create profile pages for users and creators, displaying relevant information such as uploaded tours or completed tours.

Tasks:

- [ ] **Task 5.1: Profile Page Backend API**
    - [ ] Subtask 5.1.1: Develop backend API endpoint to fetch user/creator profile data (e.g., GET /api/users/:id/profile).
    - [ ] Subtask 5.1.2: Develop backend API endpoint to fetch tours uploaded by a specific creator (e.g., GET /api/users/:id/tours).
    - [ ] Subtask 5.1.3: Develop backend API endpoint to fetch tours completed/listened to by a user (e.g., GET /api/users/:id/completed-tours).
- [ ] **Task 5.2: Profile Page Frontend Development**
    - [ ] Subtask 5.2.1: Design and implement a generic profile page template.
    - [ ] Subtask 5.2.2: Display basic editable user information (e.g., username, email).
    - [ ] Subtask 5.2.3: Implement logic to display uploaded tours for creators.
    - [ ] Subtask 5.2.4: Implement logic to display completed/listened tours for users.
    - [ ] Subtask 5.2.5: Add a simple form for editing basic profile information.
- [ ] **Task 5.3: Profile Editing Functionality**
    - [ ] Subtask 5.3.1: Develop backend API endpoint to update user profile information (e.g., PUT /api/users/:id/profile).
    - [ ] Subtask 5.3.2: Integrate frontend editing form with the backend update API.
- [ ] **Task 5.4: Integration Testing (Phase 5)**
    - [ ] Subtask 5.4.1: Test profile page display for both users and creators.
    - [ ] Subtask 5.4.2: Verify uploaded tours are correctly listed on creator profiles.

    - [ ] Subtask 5.4.3: Verify completed tours are correctly listed on user profiles (if tracking is implemented).
    - [ ] Subtask 5.4.4: Test profile information editing functionality.

Dependencies:

- Task 5.2 depends on Task 5.1 (Frontend needs backend data).
- Task 5.3 depends on Task 5.1 and 5.2 (Editing requires existing profile data and UI).

Phase 6 – Monetization (Mock Only)

Goal: Implement a mock Stripe integration for pay-per-tour or tipping functionality to demonstrate the flow, without actual payment processing.

Tasks:

- [ ] **Task 6.1: Mock Stripe Flow Design**
    - [ ] Subtask 6.1.1: Define the user flow for pay-per-tour (e.g., selecting a tour, proceeding to a mock payment page).
    - [ ] Subtask 6.1.2: Define the user flow for tipping a creator (e.g., a tip button on the tour detail page or creator profile).
- [ ] **Task 6.2: Mock Payment UI Development**
    - [ ] Subtask 6.2.1: Create a mock payment page that simulates Stripe Checkout or Elements.
    - [ ] Subtask 6.2.2: Include fields for mock payment details (e.g., dummy card number).
    - [ ] Subtask 6.2.3: Implement buttons for successful and failed mock payments.
- [ ] **Task 6.3: Backend Mock Payment Handling**
    - [ ] Subtask 6.3.1: Develop a mock backend API endpoint to simulate payment processing (e.g., POST /api/mock-payment).
    - [ ] Subtask 6.3.2: This endpoint should return a success or failure response based on mock payment details.
    - [ ] Subtask 6.3.3: For pay-per-tour, update the user's access to the tour upon mock payment success.

- [ ] **Task 6.4: Integration Testing (Phase 6)**
    - [ ] Subtask 6.4.1: Test the complete mock payment flow for pay-per-tour.
    - [ ] Subtask 6.4.2: Test the complete mock payment flow for tipping.
    - [ ] Subtask 6.4.3: Verify that tour access is granted/denied based on mock payment outcome.

Dependencies:

- Task 6.2 depends on Task 6.1 (UI needs flow definition).
- Task 6.3 depends on Task 6.2 (Backend needs UI to send mock payment data).

Phase 7 – Final Testing & Deployment

Goal: Conduct comprehensive testing across various devices and deploy the MVP publicly.

Tasks:

- [ ] **Task 7.1: Cross-Browser and Device Testing**
    - [ ] Subtask 7.1.1: Test the web application on major desktop browsers (Chrome, Firefox, Edge, Safari).
    - [ ] Subtask 7.1.2: Test the web application on various mobile devices and screen sizes (iOS, Android).
    - [ ] Subtask 7.1.3: Ensure responsive design and touch interactions work correctly.
- [ ] **Task 7.2: Bug Fix Tracking and Resolution**
    - [ ] Subtask 7.2.1: Set up a bug tracking system (e.g., simple spreadsheet, GitHub Issues).
    - [ ] Subtask 7.2.2: Log all identified bugs with clear descriptions, reproduction steps, and severity.
    - [ ] Subtask 7.2.3: Prioritize and resolve bugs, retesting after fixes.
- [ ] **Task 7.3: Performance Optimization**
    - [ ] Subtask 7.3.1: Optimize image and audio assets for faster loading times.
    - [ ] Subtask 7.3.2: Implement code splitting and lazy loading for frontend assets.
    - [ ] Subtask 7.3.3: Optimize backend queries and database performance.

- [ ] **Task 7.4: Deployment Preparation**
    - [ ] Subtask 7.4.1: Configure environment variables for production.
    - [ ] Subtask 7.4.2: Set up continuous integration/continuous deployment (CI/CD) pipeline (optional for MVP, but recommended).
    - [ ] Subtask 7.4.3: Prepare deployment scripts or configurations for the chosen hosting platform (e.g., Replit deployment, Vercel, Netlify, Heroku).
- [ ] **Task 7.5: Public MVP Deployment**
    - [ ] Subtask 7.5.1: Deploy the frontend application to a public URL.
    - [ ] Subtask 7.5.2: Deploy the backend API to a public URL.
    - [ ] Subtask 7.5.3: Perform final smoke tests on the live environment.

Dependencies:

- Task 7.2 depends on Task 7.1 (Bugs are found during testing).
- Task 7.3 depends on all previous phases (Optimization applies to the entire application).
- Task 7.5 depends on Task 7.4 (Deployment requires preparation).