

Feature Sprint Planning: Heonix MVP Core Implementation (Two-Week Sprint)

Sprint Goal

The goal of the sprint is to implement the core features of the Heonix MVP to allow a user to sign up, record and organize their professional achievements easily, and trust that their data is secure. By the end of this 2-week sprint, the team will deliver a web-based prototype where a user can frictionlessly onboard and log their first achievement, categorize entries with tags, and export their accomplishments in a PDF format, all with robust data privacy.

This will validate the MVP solution of helping users capture achievements in real time and begin forming a logging habit. The sprint’s success is defined by having a working web app on Vercel with these fundamental capabilities implemented and tested, laying the groundwork for habit-forming features like reminders and streaks in future sprints.

The initial prototype for Heonix was built using Figma and tested with a small group of users through both Figma’s interactive preview and Lovable’s user testing platform. This early round of testing provided valuable feedback on the app’s usability, onboarding flow, and core achievement logging experience. Insights from these sessions helped refine key interactions and informed the prioritization of features for the MVP sprint.

Epics and User Stories

We identified one primary epic for this sprint: “Core Achievement Logging & Management.” This epic encompasses all functionality needed for a user to begin using Heonix and see immediate value. It is broken down into several user stories, each representing a slice of functionality that can be completed within the sprint:

Functionality	User Story	Goal
User Onboarding & First Use	“As a new user, I want a frictionless sign-up and brief in-app tutorial, so I can quickly understand the app and log my first achievement right away.”	Deliver sign-up (email/OAuth) and welcome flow to help users take immediate action. Aligns with MVP’s goal of low-friction onboarding and early value realization.
Achievement Logging	“As a user, I want to add a new achievement with structured fields (like title, description, tags), so that I can record an	Covers the core “quick-add” form with prompts and structured fields. Ensures the central functionality of Heonix


	accomplishment in 1–2 minutes and capture key details.”	is fast, simple, and habit-forming.
Categorization & Retrieval	“As a user, I want to tag or categorize my achievements (by skill, project, etc.) and filter them, so that I can easily organize my accomplishments and retrieve them later.”	Adds organizational structure to entries via tagging and filtering. Supports performance review prep and long-term data utility.
Data Export & Portability	“As a user, I want to export my logged achievements to a PDF document, so I can share a professional summary of my accomplishments for performance reviews or job applications.”	Empowers users to make use of their achievements outside the app. Provides downloadable PDFs for external sharing and professional storytelling.
Privacy & Security	“As a security conscious user, I want my achievement data to be private and securely stored, so I feel safe using Heonix as a personal career journal.”	Ensures the application meets data privacy and ownership expectations through secure authentication, encrypted storage, and proper access controls, a critical trust factor for long-term user retention.

Sprint Backlog

The team will commit to the five user stories above, which together constitute the MVP core. These stories and their tasks are the sprint backlog, the plan for achieving the sprint goal and making progress visible.

Each user story is broken into tasks with clear ownership (frontend, backend, design, QA, PM) and estimates in ideal work days to fit within the 2-week sprint. The product is web-only, so implementation will use a React front-end deployed via Vercel and a cloud backend, Supabase for database and authentication, plus any serverless functions to accelerate development.

Task Breakdown by User Story

TASK BREAKDOWN	 Heonix Task Breakdown
----------------	---

Story 1: Onboarding & First Entry – “Frictionless Sign-up & Tutorial”

UX Design: Design the sign-up screen and onboarding tutorial overlay (Designer – 1 day).

Design Review: Review and refine onboarding flow for usability and minimal friction (PM & Designer – 0.5 day).

Front-end Development: Build sign-up form with Supabase Auth and welcome tutorial modal in React (Front-end Dev – 1.5 days).

Back-end Development: Set up Supabase Auth and user account storage (Back-end Dev – 0.5 day).

Front-end Integration: Trigger onboarding flow or dummy entry after first login (Front-end Dev – 1 day).

QA Testing: Verify sign-up, login, and tutorial flow (QA – 0.5 day).

Acceptance Testing: PM tests complete onboarding for smoothness (PM – 0.5 day).

Story 2: Quick Achievement Logging – “Add New Achievement Form”

UI Design: Design quick-add form with fields for title, description, date, and tags (Designer – 0.5 day).

Front-end Development: Build form component with validations in React (Front-end Dev – 1 day).

Back-end Development: Set up achievements schema in Supabase (Back-end Dev – 1 day).

API Integration: Connect form submission to Supabase (Front-end Dev – 0.5 day).

Dashboard View: Display saved entries in list view (Front-end Dev – 1 day).

QA Testing: Test creation, persistence, and edge cases (QA – 0.5 day).

Code Review: Review form logic and backend integration (Leads – 0.5 day).

Story 3: Categorization & Tagging – “Organize and Filter Achievements”

UX Design: Design tagging interface and filtering controls (Designer – 0.5 day).

Schema Update: Add tag support in Supabase (Back-end Dev – 0.5 day).

Tagging Integration: Add tags to the achievement form (Front-end Dev – 0.5 day).

Filter Feature: Implement tag filtering on dashboard (Front-end Dev – 1 day).

Optimization: Apply Supabase query filters for performance (Front-end Dev – 0.5 day).

QA Testing: Verify tagging, filtering, and edge cases (QA – 0.5 day).

Code Review: Review tagging logic and filtering performance (Leads – 0.5 day).

Story 4: Export Achievements to PDF – “Data Portability”

Design: Define layout and visual style of PDF export (Designer – 0.5 day).

PDF Functionality: Create PDF generator using Node or HTML-to-PDF service (Back-end Dev – 1 day).

Front-end Integration: Add export button and hook to PDF function (Front-end Dev – 0.5 day).

QA Testing: Test download, formatting, and multi-entry cases (QA – 0.5 day).

Polish: Refine formatting and resolve visual issues (Developers – 0.5 day).

Story 5: Data Privacy & Security – “Secure & Private Data Storage”

Security Rules: Enable Row-Level Security in Supabase (Back-end Dev – 0.5 day).

Infrastructure Setup: Ensure HTTPS, encryption, and daily backup (Back-end Dev – 1 day).

UI Notice: Display privacy info during onboarding/settings (Front-end Dev – 0.5 day).

QA Testing: Test for unauthorized access and session handling (QA – 0.5 day).

Security Review: Run a checklist audit against PRD security needs (PM & Devs – 0.5 day).

RACI Matrix

HEONIX RACI MATRIC	 Heonix RACI Matrix
--------------------	--

Sprint Execution Timeline

Week 1 – Design & Core Development:

- **Day 1 (Mon):** Sprint planning meeting to confirm scope and tasks. Designer begins creating wireframes for onboarding and the achievement form. Front-end developer sets up the React project and configures the initial project structure on Vercel. Back-end developer sets up the Supabase project, creating the database and enabling authentication providers. The team establishes environment variables/credentials for local and staging use. PM ensures everyone understands the acceptance criteria for each story. (*Outcome:* basic project scaffolding in place, initial designs in progress.)
- **Day 2 (Tue):** Designer finalizes high-fidelity designs for the sign-up flow and achievement form, plus a tags UI concept. Quick review by PM and team to approve the UX. Front-end dev starts implementing the sign-up page and onboarding tutorial in code, following the design. Back-end dev implements the initial database schema including the user table, achievement table with fields for title/desc/date, etc. and configures Supabase Auth (email/password login). QA begins writing test cases for Stories 1 and 2 to be ready for later testing. (*Outcome:* sign-up UI coded, backend auth & DB set up; design assets ready for use.)
- **Day 3 (Wed):** Front-end dev completes the core of the achievement logging form (Story 2) in React and ties it into Supabase, new achievement data can be sent to the database. Back-end dev works on the API integration or direct queries for saving and retrieving achievements using Supabase client libraries. The achievements list view is scaffolded to display saved entries. QA and PM do a brief check-in; PM walks through the app's current state, maybe via Storybook or deployed preview to ensure the form flow aligns with expectations. Daily stand-up indicates no major blockers. (*Outcome:* user can sign up, log in, and add a basic achievement that gets stored; initial listing visible.)
- **Day 4 (Thu):** The team tackles Story 3. Front-end dev implements tag input on the achievement form and basic filtering UI on the list. Back-end dev updates the database to support tags, adding a tags field and writing a query to filter by tag. Meanwhile, the back-end dev also starts the PDF export function (Story 4) logic in parallel, since the tag work may be quick. Designer provides any needed icons or visual tweaks (e.g., a tag icon or export button icon). QA prepares test cases for tagging and exporting features. (*Outcome:* tagging functionality added, pending full testing, and the groundwork for PDF export laid on the backend.)

- **Day 5 (Fri): Mid-sprint review:** The team conducts an internal demo of progress. The front-end and back-end are integrated for core flows: a demo user goes through sign-up, adds a couple of achievements with tags, and sees them listed. The PDF export may be partially working, perhaps generating a rudimentary PDF. The team identifies any UI issues or bugs like validation errors, layout tweaks and addresses quick fixes. In the afternoon, development continues: front-end dev finishes integrating the export button with the backend function, and back-end dev finalizes the PDF generation output ensuring formatting is acceptable. QA begins exploratory testing of the entire flow in the staging environment. (*Outcome:* core features are functionally implemented by the end of week 1; ready to test and polish in week 2.)

Week 2 – Testing, Hardening & Finalization:

- **Day 6 (Mon):** Developers focus on fixing bugs found during the mid-sprint review and QA's initial testing. For example, if the QA found that filtering by tag and categories wasn't updating the UI correctly or form validation allowed an empty title, those are fixed today. The PDF export feature is tested by developers with various data to ensure reliability. The team also implements remaining Story 5 (privacy and security) tasks: the back-end dev applies the security rules (row-level security on Supabase) and double-checks that all API calls require an authenticated user. If any design polish is needed like better layout for the list or adding a logo, the designer and front-end developer can collaborate on these small enhancements. (*Outcome:* All known critical bugs are resolved; the application is feature-complete. Security measures are in place in the backend.)
- **Day 7 (Tue):** Full-cycle QA testing begins. The QA engineer executes test cases for every user story: creating accounts, logging in/out, adding achievements, with and without tags, edge cases, filtering, and exporting a PDF. They also test security by attempting to access data as an unauthorized user ensuring access is denied. Any bugs discovered like incorrect data in PDF, or a tag not saved are logged. Front-end and back-end devs tackle the high-priority bug fixes promptly. The app is deployed to a staging environment on Vercel linked with the test database for end-to-end testing. (*Outcome:* Comprehensive list of minor bugs is identified and being addressed; major flows are confirmed working as expected.)
- **Day 8 (Wed):** Developers continue to fix bugs and polish. For example, if QA found styling issues or a PDF formatting glitch, those are corrected. Performance is checked in a basic way, for example, does the app load quickly, are database queries for filtering efficient, no noticeable lag with test data. The team also verifies that email notifications for sign-up confirmation or similar from Supabase are working, though weekly reminders are out of scope this sprint. PM updates any product documentation like a brief "how to use" guide in case it's needed for the sprint review. By the end of Day 8, most bugs are resolved; the app feels much more refined. (*Outcome:* The product is nearly ready, with all features functioning and tested. The team has a high level of confidence in meeting acceptance criteria.)
- **Day 9 (Thu):** Final validation and freeze. The team conducts a final run-through of all user stories, sometimes called a "pre-demo" check. QA does regression testing on critical flows to

ensure that bug fixes didn't introduce new issues. The remaining open bugs, if any are addressed. The code is then merged to the main branch, and a production build is deployed on Vercel. The app should now be in a demo-ready state. The team prepares for the next day's sprint review by compiling demo scenarios like showing adding an achievement, filtering it, exporting. In the afternoon, buffers are in place in case any last-minute critical bugs appear. (*Outcome:* Code is finalized and deployed; the team is confident all acceptance criteria are met going into the review.)

- **Day 10 (Fri):** Sprint Review and Retrospective. In the morning, the team presents the completed features to stakeholders. The demo shows a user signing up and using Heonix's core functions: logging an achievement and exporting it, highlighting the habit-forming design. All user stories are confirmed as done/accepted by the Product Manager, each story meets its acceptance criteria. In the afternoon, the team holds a retrospective to discuss what went well like smooth use of Supabase accelerated development and what can be improved, maybe test automation could be better, or design was a bottleneck early on. They also identify any carry-over tasks or nice-to-have features such as the Delighter features like reminders or badges not done in this sprint to consider in future sprints. The sprint is then formally closed. (*Outcome:* Stakeholders see working MVP features aligning with the PRD vision; team learns and readies for next sprint.)

Throughout the sprint, **daily stand-ups** are held each morning to synchronize the team. Any impediments are raised immediately, for example, if the front-end hits a roadblock integrating the PDF library, the team lead or back-end dev would assist the same day rather than waiting. This continuous communication ensures that minor blockers like environment setup issues or design clarifications are resolved quickly, keeping the sprint on schedule. Thanks to using realistic, high-level components (React, Supabase, Vercel), the team can focus on product behaviour rather than infrastructure, which helps meet the tight two-week timeframe.

Dependencies and Potential Blockers

- **Design Asset Readiness:** Timely delivery of UI/UX designs is critical. The designer must provide the sign-up and form designs by early Week 1 so that development isn't held up. *Mitigation:* We scheduled design completion by Day 2 and involved the devs in design reviews to eliminate ambiguity. If designs were delayed, developers would use wireframes or placeholders to continue progress, then adjust once final assets arrive.
- **Supabase Setup & Reliability:** The project relies on Supabase for auth and data. Any delay in setting up the Supabase project like waiting for API keys or encountering permission issues could block development of core features. Also, any downtime in the service could impact testing. *Mitigation:* PM/Back-end dev set up Supabase on Day 1 and tested the connection. We have a contingency to switch to a local mock or Firebase if a severe Supabase issue arises, unlikely within a sprint.

- **PDF Generation Library:** Using a third-party library or service for PDF export means we depend on that library's capabilities. If the chosen solution had bugs or formatting limitations, it could slow down Story 4. *Mitigation:* We chose a well-known PDF generation approach like generating HTML and using a headless browser or a library like pdfkit. We planned buffer time (Day 8) to tweak formatting issues. In the worst case, if PDF export proved too problematic, we would communicate early and possibly deliver a simpler data export (CSV) as a fallback, but this was not needed.
- **Scope Creep:** There is a risk of trying to include “delighter” extras like email reminders or badge graphics within this sprint. *Mitigation:* The team strictly followed the MVP PRD scope. Any ideas for additional features were noted for future sprints. We kept this sprint focused on the must-have functionality to avoid diluting effort.
- **Team Availability:** All roles (FE, BE, Designer, QA, PM) are needed. If any team member were to be unavailable (sick day, etc.), it could affect the timeline. *Mitigation:* The team lead (PM or a senior dev) can cover basic tasks in another area if needed for example PM can assist with testing, front-end dev can adjust minor design gap to keep work moving.
- **Integration Issues:** As we integrate front-end with back-end services, unforeseen issues like auth tokens, CORS errors, etc.) might arise. *Mitigation:* We tackled integration early (by mid-Week 1) to surface such issues. Daily stand-ups ensured any integration blocker was resolved same day with collaborative debugging.

Overall, these dependencies were managed proactively, and no major blockers derailed the sprint. The two-week timeline was **tight but achievable**, given the use of proven tech stack components and a clear, focused scope.

Acceptance Criteria

Each user story is considered done when all its acceptance criteria are met. Below are the criteria used to validate the completion of each story by the end of the sprint:

- **Story 1: Onboarding & First Entry**
 - Users can create a new account via email/password or OAuth and access the app immediately after sign-up, without any errors or undue complexity.
 - After account creation, the user is presented with a clear prompt or tutorial guiding them to add their first achievement. The onboarding experience should feel “*as simple as a spreadsheet*” in terms of ease.
 - The user is able to complete their first achievement entry during onboarding or skip and do it later and sees a confirmation of their entry in the list.

- No obvious drop-offs: test participants all managed to sign up and add an entry without assistance, indicating the flow is truly frictionless.

- **Story 2: Quick Achievement Logging**

- The “Add Achievement” form includes structured fields (e.g. Title, Description, Date, Tags) as per design, and it is quick to fill out on the order of 1-2 minutes per entry in testing.
- Submitting the form successfully creates a new achievement entry associated with the logged-in user. The new entry immediately appears in the user’s list of achievements without page refresh or requiring re-login.
- The entry data is stored persistently in the database, verified by refreshing the app or checking the database directly. All key fields save correctly.
- Form validations work: required fields like Title cannot be empty, the user is notified if they try to save without a title. Error messages (e.g., for missing data) are minimal and user-friendly.
- The user interface remains responsive during entry addition (e.g., a loading indicator or feedback is given on save, and the app does not freeze).

- **Story 3: Categorization & Tagging**

- Users can add one or multiple tags to an achievement when creating or editing it. Tags appear on the achievement entry in the list (e.g., as small labelled chips).
- A filtering mechanism is available (e.g., a dropdown of tags or a search field). When a user selects a tag to filter, the achievement list updates to show only entries that have that tag.
- Filtering by a tag yields correct results (tested by adding known tags to entries and ensuring only those are shown). If no achievements have the selected tag, the list shows a “no entries” message.
- The user can clear the filter to see all achievements again. Filtering does not require a page reload and is fast, the app quickly responds to filter inputs, indicating efficient query or client-side filter.
- Tags and filtering operate reliably across sessions for example if the user logs out and back in, their entries still have tags, and filtering still works.

- **Story 4: Export Achievements to PDF**

- There is an easily accessible “Export to PDF” option on the user’s dashboard or settings.
- Clicking “Export to PDF” generates a PDF file that includes the user’s achievements with their key details. The PDF is either automatically downloaded or the user is prompted to download it.
- The content of the PDF is verified to match the data in the app: for example, for each achievement, the title, date, and main description in the PDF correspond exactly to what the user entered. Tags might be included in a simple way or listed per entry.
- The PDF format is clean and readable: entries are separated clearly for new lines or pages per entry if needed, and the document could reasonably be shared or printed for a performance review.
- Exporting does not expose any other user’s data, the PDF only contains the current user’s information. (QA tested that one user cannot somehow get another’s data via the export function.)
- The export operation completes within a few seconds for a typical number of entries (say 10-20 achievements), indicating performance is acceptable for MVP.

- **Story 5: Data Privacy & Security**

- **Authentication Enforcement:** All main interactions like adding/viewing achievements, exporting require the user to be authenticated. If a user is not logged in, they cannot access the dashboard or data (e.g., navigating to the app’s internal routes redirects to login).
- **Access Control:** Users can only access their own data. In testing, User A was unable to retrieve or modify User B’s achievements by any means, attempts to guess item IDs or use network calls with another user’s token were denied by the backend. This confirms row-level security is effective.
- **Data Encryption:** It’s verified that the app communicates over HTTPS, so data in transit is encrypted. (Our Vercel deployment and Supabase endpoints use HTTPS by default – QA confirmed the URLs were HTTPS and no mixed content warnings.)
- **Data Retention/Backup:** The team has a backup procedure in place (e.g. the database can be exported from Supabase). While not directly user-facing, for acceptance, we ensured that if the database had an issue, we could restore user data from a backup.

- **User Control:** By default, all achievements are private. The app does not share any user content publicly without user action (e.g., the “Share” feature is separate and user-initiated). This meets the PRD’s requirement that a user’s data “remains accessible to them throughout their career... all entries are private unless the user chooses to share them”.
- **Stability:** Basic security testing like vulnerability scan or using common security testing tools on the web app reveals no critical vulnerabilities in how the MVP handles user input and authentication. (This is a qualitative acceptance criterion – essentially, no glaring security holes exist in session management, data storage, etc., for MVP level.)

Each of the above criteria was checked off during the sprint’s final days. Meeting these ensured that the implemented features not only function but also align with Heonix’s stated goals and user needs from the PRD, from ease-of-use and habit formation to trust and data ownership. The Product Manager confirmed that all stories met the definition of done, and the team moved forward knowing the MVP foundation was successfully built in this sprint.