**Prep Plan for the next 12 weeks  
  
Overview**

Before commencing full-time development in twelve weeks’ time, we will undertake a sequence of infrastructure and planning tasks. These preparatory steps establish a solid foundation - analogous to laying the groundwork for a building - so that when the main construction begins, we progress swiftly, cohesively and with minimal surprises. Each week’s activities are described in layman’s terms, with accompanying reasons and purposes.

**Week 1:   
  
Establishing Version Control and Drafting the System Blueprint**

**Tasks**

1. **Initialise Version Control Repositories**
   * Create two separate repositories (“repos”) for the project: one for the front-end interface and one for the back-end server code.
   * Set up basic protection rules so that the main branch reflects stable, reviewed work.
   * Add a simple README in each repository, briefly describing its intent.
2. **Define Folder Structure / Project Scaffolding Plan**
   * Decide on a consistent arrangement of folders and files (for instance, where to place user-interface components, service modules, documentation, and configuration).
   * Commit an empty directory scaffold so everyone sees in advance where future work will go.
3. **Draft High-Level System Architecture Diagram**
   * Produce a rough sketch (a diagram or flowchart) illustrating how the various parts of the future system will connect:
     + The user interface (front-end) interacting with the server (back-end).
     + Core matching component (“matching engine”) and data-access parts (“crawler” for external sources).
     + Points where future enhancements (such as AI or quantum-inspired modules) may slot in.
   * Circulate this draft among stakeholders for initial feedback.

**Purpose and Rationale**

* **Version Control Repositories**
  + *Explanation:* Think of version control as a safe, organised filing system that tracks changes over time, notes who made each change, and allows reversal if needed.
  + *Purpose:* Establishing this immediately prevents accidental overwrites, lost work or confusion about which version is current. It ensures that every alteration is recorded from the start.
* **Folder Structure / Project Scaffolding**
  + *Explanation:* Similar to labelling folders in a filing cabinet before adding documents, we decide where specific types of materials (code files, documents, tests) will reside.
  + *Purpose:* A clear, agreed structure avoids chaos later, when multiple contributors add files. Everyone knows where to locate or place a given piece of work.
* **High-Level System Architecture Diagram**
  + *Explanation:* Like sketching the blueprint of a building before construction, this diagram shows at a glance the major components and how they interact.
  + *Purpose:* It gives all stakeholders a shared overview of the intended system, highlights potential areas of concern early, and gathers feedback to avoid costly misunderstandings once development commences.

**Week 2:   
  
Agreeing Coding Standards, Basic Continuous Integration, and Environment Setup**

**Tasks**

1. **Define Coding Standards & Linting/Formatting**
   * Select and configure rules for consistent code style (for example, indentation, naming conventions).
   * Include configuration files that tools can use to verify adherence (even before real code is written).
2. **Set Up Basic Continuous Integration (CI) Configuration (Stub)**
   * Create a placeholder CI workflow: when changes are proposed, automated checks (such as code style verification) run. Initially, these checks will be minimal, serving as a reminder to add actual tests later.
3. **Environment Variable & Secrets Strategy**
   * Draft a template file listing the names of all environment variables that will be needed (for instance, database connection details, secret keys).
   * Document how to store sensitive information securely, without exposing it in public code.
4. **Local Development Setup Guide**
   * Write simple instructions for new developers: how to clone the repositories, install necessary tools or dependencies, and start a basic, placeholder version of the system locally.

**Purpose and Rationale**

* **Coding Standards & Formatting**
  + *Explanation:* Just as a team might agree on a uniform style for document formatting, code style rules ensure a uniform appearance and structure.
  + *Purpose:* Consistent style improves readability, simplifies code reviews and reduces errors caused by misunderstandings over formatting.
* **Basic Continuous Integration (CI) Configuration**
  + *Explanation:* CI is like an automated assistant that checks every proposed update for adherence to agreed rules. A stub, or placeholder, means we set up the mechanism now, even if the real checks are added later.
  + *Purpose:* Establishes the habit and infrastructure of automated checking early, so we do not overlook integrating tests or style checks once actual development begins.
* **Environment Variable & Secrets Strategy**
  + *Explanation:* Many systems require confidential details (e.g., passwords or API keys). Listing these in advance in a template ensures everyone uses the same names without accidentally revealing real secrets.
  + *Purpose:* Prevents accidental leaks of sensitive information and smooths integration when real credentials become available.
* **Local Development Setup Guide**
  + *Explanation:* A “recipe” explaining how to set up the basic project locally, even in placeholder form.
  + *Purpose:* Allows any new team member to get started immediately, avoiding confusion (“How do I run this?”) when the time comes.

**Week 3:   
  
Building the Skeleton of Front-End and Back-End**

**Tasks**

1. **Front-End Skeleton**
   * Scaffold a basic user interface framework with empty pages or placeholders for key screens: e.g., “Search & Retrieval” page, “Match Request” form page, “Data Sources Configuration” page.
   * Ensure these placeholders render without errors, showing labels or dummy text.
2. **Back-End Skeleton**
   * Establish a minimal server framework that listens for requests and returns dummy responses (for example, “sample JSON”).
   * Include basic middleware or setup for parsing requests and handling errors, albeit without real business logic.
3. **Integration Stubs**
   * Wire the front-end placeholders to the back-end stub endpoints, so that clicking a button triggers a request and displays some dummy data in the interface.
4. **Commit & Test**
   * Developers run the skeleton locally, verifying that the front-end and back-end can communicate in principle, with placeholder data flowing.

**Purpose and Rationale**

* **Front-End Skeleton**
  + *Explanation:* This is akin to erecting the bare frame of a building: outlines of rooms (screens) appear, but without furnishings or utilities.
  + *Purpose:* Offers a tangible preview of where future features will reside, enabling stakeholders to visualise the user interface structure early and suggest layout adjustments before real logic is integrated.
* **Back-End Skeleton**
  + *Explanation:* Comparable to installing the piping or wiring conduits without connecting actual appliances: the structure exists, but the functionality is still empty.
  + *Purpose:* Confirms that the server environment is correctly configured to accept requests, and later business logic can be slotted into these stubs without environmental issues.
* **Integration Stubs**
  + *Explanation:* Ensures that the “plumbing” connecting front-end to back-end is in place: a click in the interface triggers a request, and something returns.
  + *Purpose:* Validates the end-to-end communication path, so when real logic is added, integration surprises are minimised.
* **Commit & Test**
  + *Explanation:* Developers verify locally that the skeleton works as intended (no errors, basic connectivity).
  + *Purpose:* Early confidence in the foundational setup means full development can proceed without initial environment or configuration hurdles.

**Week 4:   
  
Designing and Agreeing API Contracts**

**Tasks**

1. **Define API Contracts**
   * Document precisely what data the front-end will send to the back-end (for example: search terms, matching criteria) and what data the back-end will return (for example: lists of results with specific fields).
   * Include details such as which fields are mandatory, optional, and their formats.
2. **Mock Server Generation (Optional)**
   * Based on the agreed contract, create a mock server that simulates the back-end’s responses. The front-end can use this mock to develop alongside while real logic is pending.
3. **Share and Review**
   * Present the documented API agreements to the client and project manager for review: confirm that the fields, formats and workflows reflect business needs.
4. **Finalise API Specification**
   * Incorporate feedback and lock down the data exchange definitions, so that later coding strictly follows these contracts.

**Purpose and Rationale**

* **Define API Contracts**
  + *Explanation:* Similar to specifying the ingredients and format of a dish in a restaurant: the kitchen (back-end) and waiter (front-end) must agree exactly on what is ordered and what is served.
  + *Purpose:* Prevents mismatches later, such as the front-end expecting certain data that the back-end does not provide or vice versa, which otherwise leads to wasted effort and debugging.
* **Mock Server Generation**
  + *Explanation:* A pretend back-end that follows the agreed contract, even though the real logic is not yet in place.
  + *Purpose:* Allows front-end development to proceed in parallel, reducing idle time while back-end logic is under construction.
* **Share and Review**
  + *Explanation:* Stakeholders confirm that the data being requested and sent matches their understanding of business requirements.
  + *Purpose:* Early alignment prevents discovering mid-project that the interface expects different data, necessitating rework.
* **Finalise API Specification**
  + *Explanation:* Lock in the definitions so that both front-end and back-end teams reference the same “contract.”
  + *Purpose:* Ensures consistency and clarity for subsequent development phases.

**Week 5:   
  
Designing the Permissions/Consent Flow**

**Tasks**

1. **Functional Requirements for Permissions**
   * Identify the types of external data sources users might wish to connect (for instance, their in-house database, partner directories, public registries).
   * Define the steps the user must take to grant permission: e.g., selecting source type, entering credentials, confirming access.
2. **UI Wireframes / Mockups**
   * Sketch simple screens showing where and how a user connects or disconnects a data source, how consent wording is displayed, and how the user sees a list of active connections.
3. **Technical Design Document**
   * In plain terms, describe how the system will store and use the permissions: for example, “When the user agrees, we store an encrypted token; whenever we fetch data, we use that token to access their system.” Also cover how to handle expired permissions or revocation.
4. **Review & Feedback**
   * Present the wireframes and design notes to the client and project manager, ensuring the approach aligns with legal, compliance or business policies.

**Purpose and Rationale**

* **Functional Requirements for Permissions**
  + *Explanation:* Clarify exactly how users give the system permission to access their own or partner data.
  + *Purpose:* Without a clear plan, attempts to link data sources may fail or be insecure. Detailed requirements ensure we build a robust, user-friendly permission process.
* **UI Wireframes / Mockups**
  + *Explanation:* Simple drawings of the screens where users will interact to manage connections.
  + *Purpose:* Non-technical stakeholders can visualise and confirm the user journey, and legal teams can assess the consent language.
* **Technical Design Document**
  + *Explanation:* A plain-language description of how tokens or credentials are handled, how the system uses them when fetching data, and how expiry or revocation is managed.
  + *Purpose:* Demonstrates that security and reliability have been considered; prevents later surprises if tokens expire or connections fail unexpectedly.
* **Review & Feedback**
  + *Explanation:* Stakeholder review ensures that the proposed design meets compliance requirements and matches expectations.
  + *Purpose:* Early sign-off reduces the risk of having to redesign permission flows mid-development due to overlooked policies.

**Week 6:   
  
Building Mock External Connectors and a Crawler Prototype**

**Tasks**

1. **Set Up Mock External APIs**
   * Create simple, stand-in services that mimic external data sources with differing data formats. These may be local or hosted mocks returning sample data.
2. **Crawler Service Prototype**
   * Develop a basic module that “reaches out” to each mock source, retrieves their data, and converts it into a single, uniform format that the system can understand.
3. **Unit Tests for Normalisation**
   * Create automated checks that feed sample mock data into the normalisation code and verify that the output matches the agreed unified format.
4. **Integration with Stubbed Matching**
   * Connect the crawler prototype to the existing stubbed matching endpoint, so that when the front-end requests a match, it displays realistic-looking combined data from multiple mock sources.
5. **Documentation**
   * Write a simple guide explaining how to add new connectors in the future, specifying that each connector must fetch data and normalise it according to our shared schema.

**Purpose and Rationale**

* **Set Up Mock External APIs**
  + *Explanation:* Since we do not yet have real external systems to connect, we simulate them with dummy services that return example data in varied formats.
  + *Purpose:* Allows us to test the normalisation logic and demonstrate how multiple sources’ data will be combined even before actual connections exist.
* **Crawler Service Prototype**
  + *Explanation:* This module represents the “translator” that gathers data from different sources and unifies it.
  + *Purpose:* Confirms our approach to handling diverse formats, uncovers inconsistencies (e.g. different field names or units) early, and ensures the system can merge data effectively.
* **Unit Tests for Normalisation**
  + *Explanation:* Automated checks verify that for any given sample input, the normalisation code produces the correct unified output.
  + *Purpose:* Guarantees reliability of the normalisation logic and prevents silent errors when real data arrives.
* **Integration with Stubbed Matching**
  + *Explanation:* Combining the crawler output with placeholder matching produces realistic mock-ups of match lists in the user interface.
  + *Purpose:* Gives stakeholders a tangible preview of how combined data appears, facilitating early feedback on presentation and expectations.
* **Documentation**
  + *Explanation:* A straightforward guide on how to create additional connectors in future phases.
  + *Purpose:* Ensures future developers or partners understand how to plug in new data sources without confusion, supporting system extendibility.

**Week 7:   
  
Designing and Prototyping the Rule-Based Matching Engine**

**Tasks**

1. **Gather Initial Matching Rules from Client**
   * Request from the client clear guidance on how to prioritise matches: for example, “price closeness is most important,” “in the same region preferred,” “licensed suppliers receive a small advantage,” etc.
2. **Prototype Scoring Function**
   * Implement a simple scoring module that takes a buyer’s request and a list of candidate offers (from the crawler) and calculates a numeric score for each, ranking them accordingly.
3. **Unit Tests for Scoring Logic**
   * Automate checks that confirm the scoring behaves as intended in sample scenarios (for instance, exact price match yields the highest score).
4. **Integration Demo**
   * Connect the scoring prototype to the front-end, so that when a user submits a mock request, the interface shows a sorted list of matches with their scores.
5. **Document Matching Engine Module**
   * Provide a plain-language description of how the scoring is calculated, how weights may be adjusted, and where in the code these rules reside.

**Purpose and Rationale**

* **Gather Initial Matching Rules from Client**
  + *Explanation:* Define the business priorities clearly, ensuring the matching reflects real preferences.
  + *Purpose:* Prevents guessing or mismatches between system behaviour and client expectations; aligns technical logic with business needs.
* **Prototype Scoring Function**
  + *Explanation:* Translates the agreed rules into working code that assigns a score to each potential match.
  + *Purpose:* Demonstrates that the system can apply those rules in practice and provides a basis for evaluation and refinement.
* **Unit Tests for Scoring Logic**
  + *Explanation:* Automated tests confirm the code implements the rules exactly as specified.
  + *Purpose:* Catches errors early, ensuring reliability before deeper integration or real data arrives.
* **Integration Demo**
  + *Explanation:* Shows stakeholders how the ranked matches appear in the interface, enabling them to confirm or adjust the rules.
  + *Purpose:* Early demonstration fosters confidence, uncovers misunderstandings, and allows tweaks before full-scale development.
* **Document Matching Engine Module**
  + *Explanation:* A non-technical overview explaining how the scoring is done and how to adjust it later.
  + *Purpose:* Serves as a reference for all parties, ensuring transparency and facilitating future adjustments.

**Week 8:   
  
Setting Up Interaction Logging (“Human Memory”) for Future Learning**

**Tasks**

1. **Define Logging Schema for Interactions**
   * Decide precisely what information to record whenever a user interacts with matches: for example, the request details, the list of matches shown, which match the user selected, and whether a trade was completed.
2. **Implement Logging Infrastructure (Stub)**
   * Build a simple mechanism that saves these interaction records in a basic store (e.g. a database collection), even if the data is minimal.
3. **Privacy & Retention Policy Draft**
   * Draft a plain-language note explaining that these logs will be used solely to improve the matching system, kept securely, anonymised as needed, and retained for a defined period.
4. **Test Logging Flow**
   * Simulate a mock user action (submit request, select a match), and verify that the system properly records all intended details.

**Purpose and Rationale**

* **Define Logging Schema for Interactions**
  + *Explanation:* Determine exactly which pieces of information are valuable for later “learning” (e.g., price points, chosen suppliers, outcomes).
  + *Purpose:* Without these records, we cannot train or refine an automated matching model. Early planning ensures no critical data is omitted.
* **Implement Logging Infrastructure (Stub)**
  + *Explanation:* A simple set-up that captures events, storing them safely without exposing sensitive details.
  + *Purpose:* Collects the “memory” of user behaviour from the outset. If we wait until later, early interactions are lost and cannot inform future improvements.
* **Privacy & Retention Policy Draft**
  + *Explanation:* A clear statement on how the collected data will be used, how long it will be kept, and how privacy is protected.
  + *Purpose:* Reassures stakeholders and users, ensures compliance with data-protection expectations, and prevents legal or reputational risk.
* **Test Logging Flow**
  + *Explanation:* Verify that the logging code functions as intended, capturing complete and accurate records of user actions.
  + *Purpose:* Confirms the system is correctly recording data from Day 1, so that when we reach the learning phase, we have reliable datasets.

**Week 9:   
  
Designing the Future Machine-Learning (ML) Pipeline and Feature Engineering**

**Tasks**

1. **Define ML Objectives & Data Requirements**
   * Clarify the goals of introducing automated learning: for instance, improving match accuracy by learning from past user choices, personalising suggestions, or identifying patterns that humans might overlook.
2. **Design Data Flow for ML**
   * Sketch how interaction logs will be processed into structured inputs for model training: “raw logs” → “features” → “training data” → “model.”
3. **Select ML Framework & Tools (Propose Options)**
   * Outline possible approaches (e.g., simpler decision-tree methods versus more complex algorithms) and the tools or environments likely needed, noting that actual implementation will occur later when sufficient data exists.
4. **Draft Feature Engineering Specifications**
   * Specify which data attributes will transform into features (e.g., price difference ratios, region match flags, past success rates between a buyer and seller). For each feature, explain how it is calculated and why it matters.
5. **Define Evaluation Metrics & A/B Testing Approach**
   * Determine how to measure the success of a learning model: for example, does the suggested top match lead to a higher acceptance rate? Propose a plan to compare new learning-based suggestions against the original rule-based approach in a controlled manner.
6. **Deliver Design Document**
   * Compile a plain-language document summarising objectives, data flow, feature definitions, tool options, evaluation plan and prerequisites for moving to the ML phase.

**Purpose and Rationale**

* **Define ML Objectives & Data Requirements**
  + *Explanation:* Decide clearly what benefits we hope to gain by adding automated learning, and what data we must collect and store.
  + *Purpose:* Focuses the future effort, preventing aimless experimentation. Stakeholders understand why ML is proposed and what outcomes to expect.
* **Design Data Flow for ML**
  + *Explanation:* Visualise the path from raw logs to trained models.
  + *Purpose:* Ensures that when the time comes, we can rapidly build the necessary pipelines and avoid guesswork about the steps needed.
* **Select ML Framework & Tools (Propose Options)**
  + *Explanation:* Identify candidate methods and environments so stakeholders appreciate the resource implications (expertise, computational resources) of ML.
  + *Purpose:* Prepares for budgeting and recruiting the right skills when we reach the learning phase.
* **Draft Feature Engineering Specifications**
  + *Explanation:* Determine how each piece of logged data becomes a useful input (feature) for a model.
  + *Purpose:* Feature engineering is often the most time-consuming ML task; planning now avoids delays when sufficient data accrues.
* **Define Evaluation Metrics & A/B Testing Approach**
  + *Explanation:* Decide how to prove that the new model is genuinely better than the original rule-based logic, using controlled comparisons.
  + *Purpose:* Provides confidence and evidence before rolling out a learning-based system widely; prevents blindly deploying a model that does not improve outcomes.
* **Deliver Design Document**
  + *Explanation:* A non-technical summary that describes how ML will work in the future, why it matters, and what is needed.
  + *Purpose:* Gives stakeholders clarity and a roadmap for future investment, ensuring all parties share the same understanding.

**Week 10:   
  
Security Review, Performance Baseline and Monitoring Plan**

**Tasks**

1. **Security Review & Hardening Plan**
   * Identify potential vulnerabilities or areas requiring protection (for example, safe handling of credentials, encrypted connections). Draft a checklist of best practices.
2. **Performance Baseline Tests (with Mock Data)**
   * Using the existing prototypes, measure how long key operations take when provided with varying amounts of mock data. Document the timings and note potential bottlenecks.
3. **Monitoring & Logging Strategy**
   * Plan how to observe the system in production: which metrics to track (e.g. number of requests, error rates, speed of responses), how to record them, and how alerts will be raised if issues arise.
4. **Disaster Recovery & Backups (Outline)**
   * Outline how essential data (especially user interaction logs) will be backed up, how often, and how to restore if something goes wrong.

**Purpose and Rationale**

* **Security Review & Hardening Plan**
  + *Explanation:* Even if full security features are not yet implemented, we identify best-practice measures: encrypted communications, secure storage of secrets, input validation to prevent misuse.
  + *Purpose:* Lays the groundwork to avoid vulnerabilities later, ensuring that when full development begins, security has been considered from the outset.
* **Performance Baseline Tests (with Mock Data)**
  + *Explanation:* Measure how quickly prototype matching or data-normalisation steps operate when handling small, medium or larger amounts of sample data.
  + *Purpose:* Reveals whether the chosen approach will scale acceptably, allowing us to plan optimisations before real data arrives.
* **Monitoring & Logging Strategy**
  + *Explanation:* Decide which indicators to watch (e.g., slow responses, rising errors) and how to be alerted (e.g., email or dashboard warnings).
  + *Purpose:* Ensures that once the system is live, problems can be detected and addressed swiftly, maintaining reliability.
* **Disaster Recovery & Backups (Outline)**
  + *Explanation:* Plan how to back up critical data (particularly the logs needed for future learning) so that if a failure occurs, we can recover.
  + *Purpose:* Protects against data loss that would hinder further development or degrade system performance; essential for business continuity.

**Week 11:   
  
Deployment and DevOps Planning**

**Tasks**

1. **Deployment Architecture Draft**
   * Sketch how the system will be deployed: where the server-side code runs, how the user interface is served, how data is stored, and how these parts connect over secure channels.
2. **CI/CD Pipeline Design**
   * Outline how code changes progress automatically from development to testing to production, including automated checks and deployment steps.
3. **Infrastructure as Code (IaC) Considerations**
   * Although actual scripts may be prepared later, plan how we would define servers, databases, and network configurations in a repeatable, scripted way.
4. **Rollback & Versioning Strategy**
   * Decide on version numbering for releases and a procedure for reverting to a prior version if a new release behaves unexpectedly.
5. **Cost Estimation Framework**
   * Provide guidelines on how to estimate hosting and operational costs once a specific hosting provider is selected, advising to begin modestly and scale as usage grows.

**Purpose and Rationale**

* **Deployment Architecture Draft**
  + *Explanation:* Like planning how and where to build the house once the blueprint is finalised, this shows the arrangement of hosting environments, databases and network flows.
  + *Purpose:* Ensures we know in advance how the system will be launched, highlighting any missing pieces (e.g., certificate management) before the development phase.
* **CI/CD Pipeline Design**
  + *Explanation:* Continuous Integration/Continuous Deployment automates building, testing, and releasing updates.
  + *Purpose:* Reduces manual steps, lowers risk of errors during deployment, and facilitates rapid updates once development is active.
* **Infrastructure as Code (IaC) Considerations**
  + *Explanation:* Describing how we would script the creation of servers and services ensures environments can be reproduced reliably.
  + *Purpose:* Promotes consistency across development, testing and production, and supports disaster recovery or scaling.
* **Rollback & Versioning Strategy**
  + *Explanation:* Agreeing on version numbers (e.g. v1.0, v1.1) and how to revert if a release misbehaves.
  + *Purpose:* Provides a safety mechanism so that, should an update introduce issues, we can quickly return to a stable state without extensive downtime.
* **Cost Estimation Framework**
  + *Explanation:* Outlining the factors affecting ongoing expenses (servers, data storage, monitoring) helps in budgeting once the hosting environment is chosen.
  + *Purpose:* Prevents surprises when monthly bills arrive, and guides choice of scalable services to match usage growth.

**Week 12:   
  
Consolidating Documentation and Preparing for Full-Time Kickoff**

**Tasks**

1. **Aggregate All Documentation**
   * Compile every diagram, specification document, wireframe, design note, policy draft and test report into a single, organised repository or document collection with a clear table of contents.
2. **Prepare Onboarding and Handoff Presentation**
   * Create a concise slide deck or summary document that walks through the preparatory work: what has been completed, why each piece matters, and how everything fits together.
3. **Finalise Milestones & Next-Phase Plan**
   * Based on the groundwork, confirm the detailed plan for the full-time development phase: exact dates for starting the MVP build, schedules for integrating real external data sources, planned timing for moving to ML once sufficient logs exist, etc. Identify any pending prerequisites (for example, obtaining actual API credentials or legal approvals).
4. **Set Up Communication & Collaboration Channels**
   * Ensure all stakeholders have access to the repositories and documentation, and agree on communication channels (e.g. which messaging platform, meeting cadence) and responsibilities (who to contact for approvals, queries, or issues).

**Purpose and Rationale**

* **Aggregate All Documentation**
  + *Explanation:* Collecting all materials into a central “manual” is like assembling all building permits, blueprints, inspection reports and guidelines in one folder before construction begins.
  + *Purpose:* Provides a single source of truth, so that when full-time development begins, everyone can quickly locate the relevant information and avoid duplicating effort or overlooking requirements.
* **Prepare Onboarding and Handoff Presentation**
  + *Explanation:* A clear walkthrough for stakeholders, summarising what we have done and why, so they understand the foundation.
  + *Purpose:* Aligns expectations, clarifies the current state, and ensures that, once full-time work starts, all parties share a common understanding of the starting point and next steps.
* **Finalise Milestones & Next-Phase Plan**
  + *Explanation:* Translate the preparatory work into a concrete schedule for the main development: specific tasks, dependencies, deadlines and prerequisites.
  + *Purpose:* Ensures a smooth transition into the development phase, with realistic timelines and awareness of any outstanding items that must be in place before proceeding.
* **Set Up Communication & Collaboration Channels**
  + *Explanation:* Confirm which platforms and protocols (e.g. regular update meetings, document review flows) will be used, and ensure permissions are granted.
  + *Purpose:* Prevents delays or confusion once development starts, so that queries, feedback and approvals flow smoothly.

**Summary and Analogy**

Imagine you plan to construct a custom house. You would not immediately begin building walls and installing fixtures. First, you would:

* Arrange secure storage for blueprints and contracts (version control).
* Agree on style guidelines (coding standards).
* Sketch an initial floorplan (architecture diagram).
* Ensure the plumbing and electrical plans are drawn (API design, permissions flow).
* Test how different parts connect in a mock environment (mock connectors, skeleton code).
* Decide how rooms will be furnished and how occupants will use them (matching rules, logging for future improvements).
* Prepare for safety inspections and energy efficiency (security review, performance baseline, monitoring).
* Plan how to actually erect the house and maintain it (deployment and DevOps planning).
* Compile all permits, plans and instructions in one folder and hold a walkthrough with the builder and owner (aggregate documentation and handoff presentation).

Only once these preparatory steps are complete do you commence full-time construction, confident that the foundation is solid, the plan is clear, and the necessary approvals and tools are in place.

Similarly, over the next twelve weeks we will not build the full features, but will lay the necessary groundwork so that when the main development begins:

* The team can start coding actual functionality immediately, without environment or specification delays.
* Stakeholders share a unified understanding, avoiding mismatches and rework.
* Security, performance and data-collection considerations have been addressed in principle, reducing later risk.
* A clear roadmap to future enhancements (machine learning, advanced matching methods) is defined, providing confidence in the longer-term vision.