WP4-A3. Functional Specifications.

Contents

1. INTRODUCTION

This document presents the outcomes of activity WP4.A3: Functional specifications of the RockChain e-Learning Tool. Building on the database and backend architecture delivered in WP4.A1 and on the refined, pilot-ready version of the tool produced in WP4.A2, this activity focuses on describing in a concise and structured way what RockChain is expected to do from a functional point of view. The emphasis is on the behaviour of the tool as experienced by learners, trainers and facilitators during real training sessions, rather than on code-level implementation or low-level technical details.

The main objective of WP4.A3 is to provide a clear description of the functions, modules and flows that make up the RockChain e-Learning Tool after the refinement phase. This includes how users are authenticated and managed, how game sessions and rounds are created and orchestrated, how market and mining mechanics behave, and how scores, rewards and basic indicators are produced for both gameplay and evaluation. The specifications capture how these elements interact in typical VET and adult-learning scenarios, with small groups of participants, often over 45 and with heterogeneous digital skills, guided by a trainer who uses the game as a starting point for discussion on circular economy and waste management.

In practical terms, the functional specifications serve as a bridge between the conceptual and pedagogical design of RockChain and its technical implementation. They translate requirements identified in earlier Work Packages into observable behaviours and rules that can be used as a shared reference for developers, trainers and decision-makers: what must always work, what information needs to be available on screen, and under which conditions the system should progress from one state to another. The document refers to the existing architecture based on a React Native mobile client, Firebase services and a real-time communication layer, but keeps the focus on roles, responsibilities and expected system behaviour. In this way, WP4.A3 supports the preparation of trainer guidelines in WP4.A4, the packaging and deployment work in WP4.A5, and the evaluation activities in WP5, ensuring that the RockChain e-Learning Tool can be maintained, reused and adapted beyond the lifetime of the project in a consistent and reproducible manner.

2. SYSTEM OVERVIEW AND ROLES

RockChain is a multiplayer serious game designed to help adult learners explore how data, markets and validation mechanisms can support circular economy strategies in the ornamental rock and construction sectors. In each session, a small group of learners joins a shared game on their mobile devices and plays through one or more rounds in which they buy and transform products, generate and reduce waste, solve mining challenges and see how their decisions affect both economic and environmental outcomes.

2.1. RockChain concept and main training scenarios

The tool is intended to be used mainly in trainer-led sessions rather than as a standalone app. Typical scenarios include:

- Short interactive blocks within a course or workshop (e.g. a 45–90 minute segment where the class plays one or two games and then discusses the results).
- Demonstration sessions where a trainer uses RockChain to illustrate concepts such as value chains, waste streams, traceability and incentives for circular practices.
- Blended activities where learners first receive theoretical input (e.g. in WP3 materials or other modules) and then use RockChain to experiment with decisions and observe the impact on waste, costs and rewards.

In all cases, RockChain is not meant to replace teaching but to serve as a practical engine for exploration and discussion, providing a shared experience that can be analysed during debriefing.

2.2. Target users and roles

The functional specifications consider three main types of users:

2.2.1. Learners (players)

- Adult participants in VET and adult education, many of them over 45 and with heterogeneous digital skills.
- Use RockChain on their own Android or iOS devices to join a game, make decisions during rounds and see their outcomes.
- Interact only with the mobile client: logging in, joining a game via a code, navigating between market, mining, recycling, stats and profile screens, and receiving feedback and rewards.

2.2.2. Trainers / facilitators

- Teachers, trainers or consultants responsible for delivering the learning activity.
- Use RockChain to set up and manage sessions: creating or hosting games, helping learners join, deciding when to start and end rounds, and steering discussion based on results.
- May also review basic indicators (scores, waste levels, winning industries, selected strategies) to support debriefing and assessment.
- Do not need deep technical skills; their interaction is primarily through the same client interface, possibly with a projected view or shared screen.

2.2.3. Technical administrators

- Staff in partner institutions or technical centres who deploy and maintain the backend (Firebase project, real-time server, builds of the mobile app).
- Ensure that authentication, database, network configuration and updates are correctly set up for the institutions that use RockChain.
- Typically, do not participate in day-to-day training sessions but provide second-line support when new versions are rolled out or when infrastructure changes are needed.

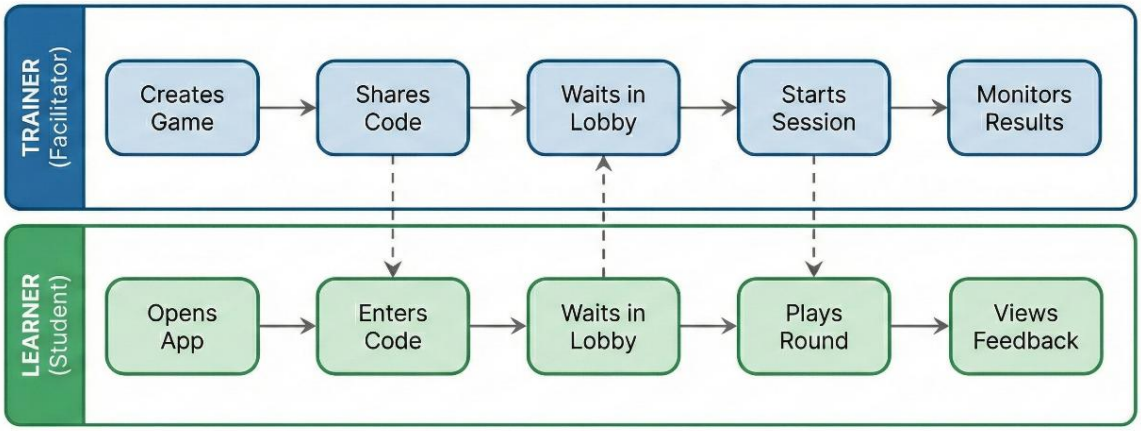

Figure 1: Guided session by trainer.

The functional behaviour specified in this document is oriented mainly to learners and trainers, while technical administrators act as enablers of the environment in which the tool runs.

2.3. Deployment and runtime environment

Functionally, RockChain assumes a common deployment pattern that reflects realistic VET and adult-learning conditions:

2.3.1. Devices

- Learners use smartphones or tablets running the RockChain mobile app (Android or iOS).
- Trainers may also use a mobile device, a tablet or an emulator on a laptop, and can optionally project their screen to the classroom display to show shared views (e.g. waiting room, end-of-round results).

2.3.2. Connectivity

- Sessions are expected to run with stable internet access: typically, classroom Wi-Fi or institutional networks, but mobile data is also possible.
- The system must tolerate brief interruptions (e.g. Wi-Fi glitches, device sleep) and allow clients to reconnect and rejoin the current game without losing their position.
- Fully offline use is not supported, as RockChain relies on a backend hosted in the cloud.

2.3.3. Session scale and structure

- Games are designed for small groups (for example, 3–10 learners per match), with the possibility of running several games in parallel in a single class if needed.
- Trainers plan sessions so that there is enough time for: login and onboarding, playing one or more games, and a debriefing phase where the in-game outcomes are linked back to the course content.

These assumptions about devices, connectivity and classroom use frame the functional requirements described in the next sections: they determine what the system must do to support smooth, trainer-led sessions in typical VET and adult-learning environments.

3. FUNCTIONAL ARCHITECTURE AND MODULES

The RockChain e-Learning Tool follows a client–server architecture in which a mobile application provides the user interface for learners and trainers, while a set of backend services manage authentication, game logic, persistence and real-time synchronisation. WP4-A1 already described this architecture in technical depth; WP4-A3 reuses the same structure and focuses on the functional responsibilities of each part: what the client must do, what the backend must guarantee, and how both collaborate to deliver a coherent learning experience.

3.1. High-level client–server architecture

At a high level, RockChain consists of three cooperating layers:

- A React Native mobile client (built with Expo) that runs on Android and iOS devices. This client is the only component directly used by learners and trainers. It is responsible for user interaction, local state management during a session and the visual presentation of game states, feedback and results.
- A Firebase backend that provides authentication, persistent storage and controlled business logic:
    - Firebase Authentication identifies users and issues credentials.
    - Cloud Firestore stores games, users, market snapshots, mining events and learning-related data that must survive across sessions.
    - Firebase Functions implement server-side operations that require validation and trusted access (for example, creating games, joining games, updating round status, calculating rewards or updating user progress).
- An authoritative real-time server that coordinates rounds and fast interactions between players using WebSockets. This server maintains an in-memory view of each active game (players, inventories, timers, mining problems) and broadcasts authoritative events (start of round, end of round, market updates, mining problems) to all connected clients.

Functionally, the mobile client never tries to decide alone what the game state should be. It reacts to authoritative information coming from the backend and the real-time server and sends user actions as requests (via HTTPS or WebSockets) that the server validates and applies before any persistent change is made.

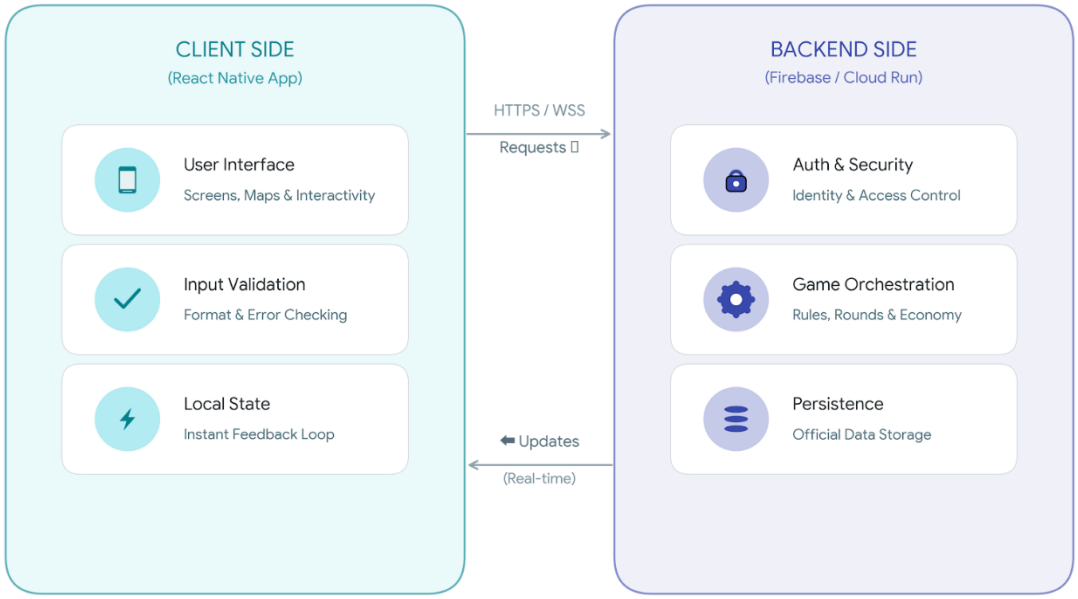

Figure 2: Architecture diagram

3.2. Front-end modules

On the client side, behaviour is organised into a set of modules and screens, all of them supported by shared state contexts (for game data, mining and learning progress):

Authentication and onboarding (Login / Registration): Handles user sign-in and sign-up through Firebase Authentication, basic profile creation and initial checks (for example, whether the user is hosting a game or joining an existing one). From a functional perspective, this module must always lead the user either to create a new game or to enter a valid game code.

Waiting room (Lobby): After creating or joining a game, all players are taken to the waiting room.

- shows who has joined and their status (waiting/ready)
- indicates who is acting as host or trainer
- allows the host to start the game when the group is complete
- displays any countdown or message related to the start of the first round. Its main function is to keep the group aligned before gameplay begins

GameTabs container and shared header: Once the game has started, the user enters the main in-game area, structured as a set of tabs (Market, Mining, Recycle, Stats, Profile, Chat) wrapped by a shared header. The header shows core indicators (round number, time remaining, key resources) in a consistent way across all tabs. Functional

specifications require that this header always reflects the authoritative state coming from the backend so that learners can rely on it for their decisions.

MarketScreen: Displays the products available in the current round, their prices, waste implications and other attributes. It lets players buy or sell items, subject to the rules enforced by the backend. Functionally, this screen must:

- always reflect the current round's market configuration
- prevent invalid operations (e.g. spending more RockCoins than available)
- update the local view promptly when transactions succeed

MiningScreen: Presents mining problems (simple arithmetic or logic tasks) linked to specific game events and allows players to submit answers under time pressure. The screen must:

- show active problems and their time limits
- disable submission when a problem is no longer active
- clearly indicate when a winner has been determined for a block

RecycleScreen: Enables players to transform parts of their inventory or waste into more valuable states or to reduce accumulated waste according to game rules. Its functional role is to make circular strategies visible and actionable, and to feed updated waste and resource values back into the overall game state.

StatsScreen: Provides an overview of performance: scores, waste levels, rankings, and other indicators per round or per game. It is a key module for debriefing within the session, helping both learners and trainers to interpret what happened and which strategies were more effective.

ProfileScreen and settings: Lets users view and adjust their profile (name, avatar, language preferences) and see a summary of their past activity or achievements in the current game. It also acts as an entry point for basic accessibility and localisation options.

ChatScreen and auxiliary components: Supplies a simple chat channel to help players coordinate during the game. Auxiliary components (such as a network status banner or tutorial overlays) inform users about connection issues, provide contextual help and support smoother onboarding.

Together, these modules define what learners and trainers can see and do at each moment. The functional specifications ensure that each screen reacts in a predictable way to changes in the backend state and to user actions.

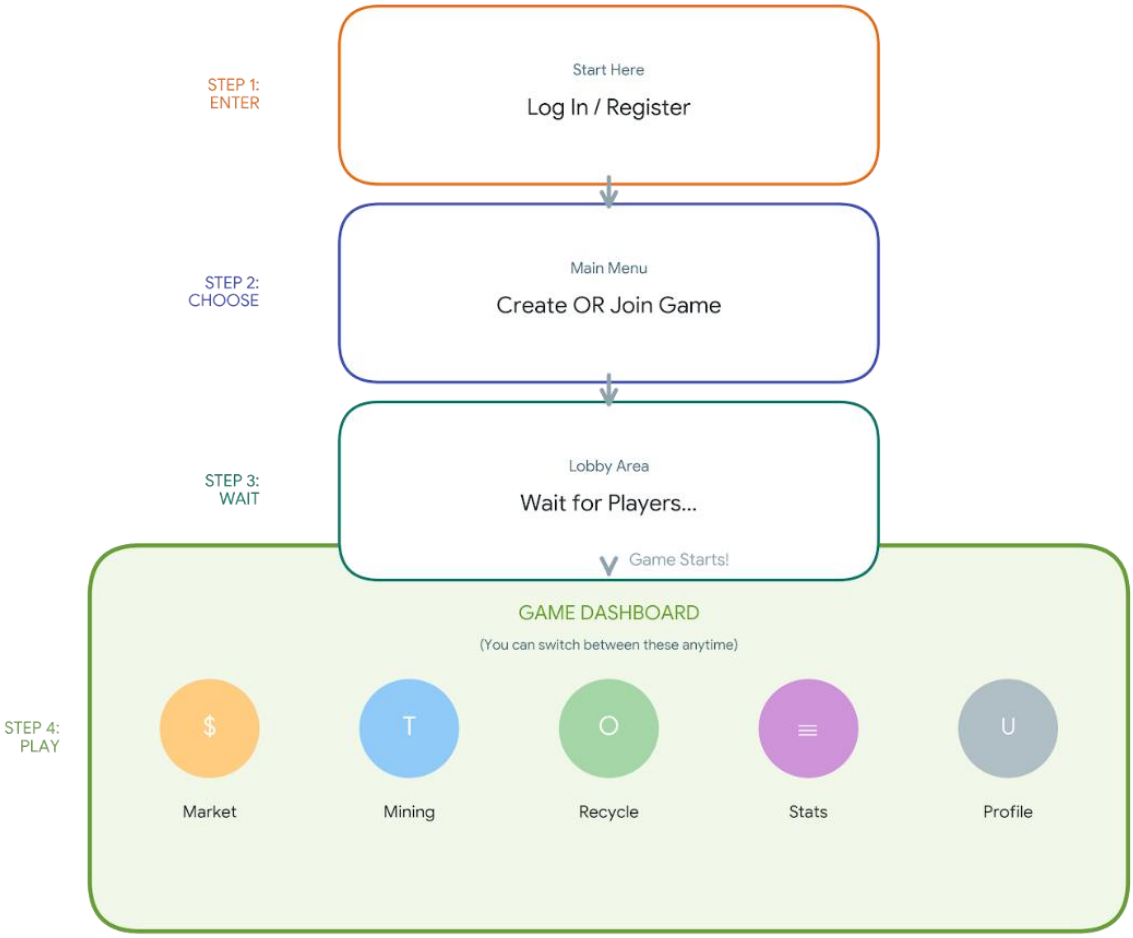

Figure 3: Frontend modules diagram

3.3. Backend modules

On the backend side, functionality is grouped into several logical modules, implemented across Firebase Functions, Firestore rules and the real-time server:

- User and profile management: Handles registration, authentication and storage of basic user information in Firestore. Functionally, this module must ensure that:
    - Each authenticated user has a unique and stable identifier.
    - Profile updates are validated and consistent.
    - And only authorised users can access or modify their own data.
- Game and round orchestration: Covers the lifecycle of games and rounds: creating and deleting games, joining and leaving, starting and ending rounds, and

determining when the game finishes. This module defines the valid state transitions and guarantees that:

  - Game codes are unique and resolvable.
  - No more than the allowed number of players join a game.
  - Changes in status (waiting, in progress, round end, finished) are applied consistently for all players.

- Market and resource logic: Manages the generation of market configurations per round, price and waste rules, and the validation of transactions. From a functional perspective, this module ensures that:
  - Each round's market obeys predefined rules and randomness constraints.
  - Player inventories and balances are updated atomically and consistently.
  - The resulting data can be traced back for analysis (e.g. Through snapshots or logs).
- Mining and validation logic: Generates mining problems, records responses and determines winners according to the rules of the game. This module must:
  - Ensure that problems are associated with a specific game context.
  - Record attempts in a way that can be audited.
  - Apply fair criteria (e.g. First correct answer) when assigning rewards.
- Real-time coordination layer: Implements the WebSocket-based server that keeps active game state in memory and broadcasts authoritative events. Its functional responsibilities include:
  - Keeping all connected players in sync regarding round status, timers and key updates.
  - Handling disconnections and reconnections so that players can rejoin ongoing rounds when possible.
  - Interacting with firestore and functions at well-defined points (for example, when persisting end-of-round results).
- Learning progress and logging: Records relevant events and summaries needed for educational monitoring and later evaluation in WP5. This module defines which actions and outcomes (e.g. blocks mined, waste reduced, choices of industry) are stored as learning indicators and how they can be retrieved and aggregated at user, game or cohort level.

By clearly assigning these responsibilities to front-end and backend modules, the functional architecture ensures that RockChain behaves in a predictable, testable and extensible way. The next section builds on this structure to define the core functional requirements in more detail.

4. DETAILED FUNCTIONAL REQUIREMENTS

The following requirements describe what the RockChain e-Learning Tool must do from the point of view of learners, trainers and the training context. They build on the architecture outlined above but are expressed in terms of observable behaviour and system rules rather than implementation details.

4.1. User management and authentication

The system must identify users consistently across sessions and keep their basic information and game participation in a stable form.

Functionally, this means that the system shall:

- Require users to authenticate before accessing any game-related functionality.
- Create and maintain a unique, stable user identifier and a basic profile (e.g. Display name, language preference) for each authenticated user.
- Allow users to update selected profile fields while preserving links to past games and learning data.
- Restrict access so that each user can only read or modify their own profile and in-game state, except for information explicitly shared within a game (e.g. Public rankings).

4.2. Game creation, joining and lobby management

The system must allow trainers or designated hosts to create games and other users to join them easily, ensuring that all participants are correctly registered before play starts.

The system shall:

- Allow a user to create a new game and automatically assign a unique game code and identifier.
- Ensure that a newly created game is initialised with default parameters (round count, timers, maximum number of players, initial resources).
- Allow users to join an existing game by entering a valid game code, with clear feedback if the code is incorrect or the game is not joinable.
- Enforce limits on the number of players per game and prevent late joins once the game has progressed beyond an agreed point.

- Provide a waiting room (lobby) view where all joined players and their readiness status are visible, and where the host can start the game when conditions are met.

4.3. Game and round lifecycle

Once a game is created, the system must manage its lifecycle in a controlled way, moving between well-defined states that are reflected consistently on all clients.

The system shall:

- Maintain an explicit game status (e.g. Waiting, in progress, finished) and round status (e.g. Not started, active, resolving, completed) for each game.
- Allow only valid transitions between these states, triggered by authorised actions (e.g. Host starting the game, round timer expiring, explicit end-of-game).
- Start each round by broadcasting the necessary information (round number, time remaining, initial market and mining configuration) to all players.
- End each round based on authoritative conditions (timer, completion of required steps) and ensure that all clients receive a clear indication that play has stopped.
- Mark the game as finished when the configured number of rounds has been completed or when the host explicitly ends the session and prevent further actions in that game afterwards.

4.4. Market, resources and circular-economy logic

RockChain's market and resource mechanics must reflect the intended learning focus on waste, value and circular strategies.

The system shall:

- Generate, for each round, a set of available products with associated attributes (type, material, quality, price, waste implications) according to predefined rules.
- Present to each player a market view that reflects the authoritative state for that round and update it when products are bought or sold.
- Validate all transactions against the player's current resources (e.g. Rockcoins, inventory) and prevent actions that would violate constraints.
- Update player inventories, balances and waste levels consistently after each valid transaction.

- Ensure that the market configuration and resulting states can be reconstructed afterwards for analysis (e.g. Through stored snapshots or derived data).

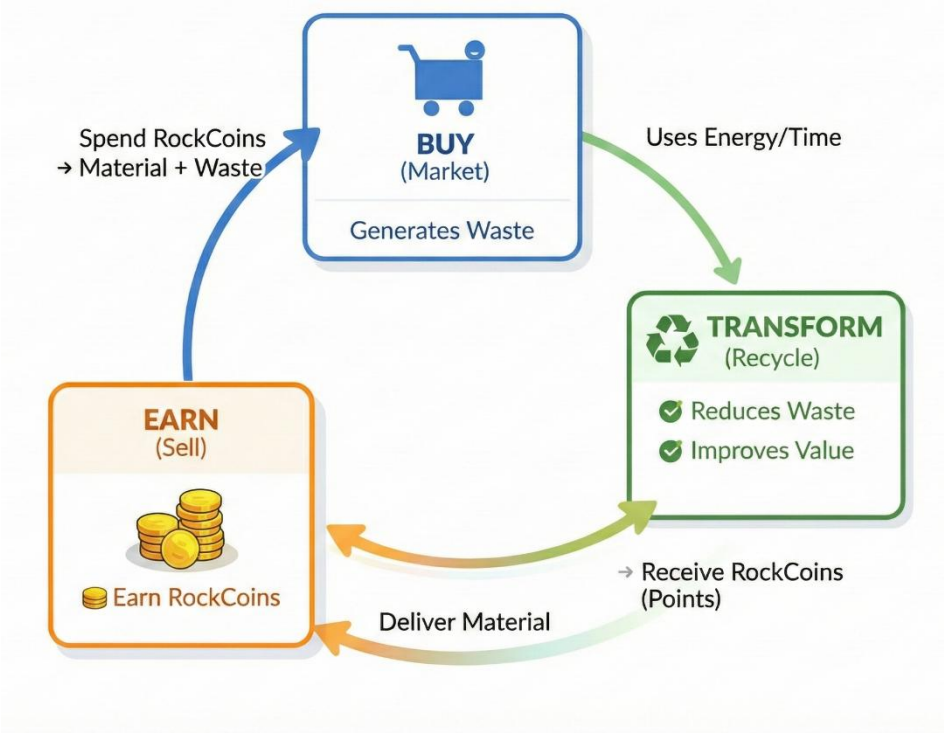

Figure 4: Circular economy in Rockchain.

4.5. Mining challenges and validation of solutions

Mining events provide time-limited challenges that connect decisions and validation mechanisms to the game's blockchain-inspired narrative.

The system shall:

- Generate mining problems that are clearly associated with a game and round context and make them available only while they are active.
- Notify all relevant players when a new mining problem becomes available and indicate how long it will remain open.
- Accept and record player responses while the problem is active, rejecting late submissions once it has been closed or solved.
- Determine winners according to clearly defined rules (e.g. First correct response) and ensure that only one winner is recorded per problem.
- Update the game state (e.g. Mining rewards, history of solved problems) and inform all players of the outcome in a transparent way.

4.6. Scoring, rewards and industry selection

Scoring and rewards must help learners understand the consequences of their decisions on both economic results and waste/circularity indicators.

The system shall:

- Maintain, for each player, a current score and key indicators such as accumulated waste and resources.
- Allow players to select or confirm an industry or strategy where relevant and use this information in the calculation of rewards at the end of each round.
- Compute per-round rewards based on transparent rules that combine market behaviour, mining results and circular-economy choices.
- Apply these rewards atomically to player states (updating scores, waste levels and inventories) and store a summary of the round results per player.
- Present end-of-round and end-of-game summaries that clearly show how scores and indicators were obtained, supporting debriefing and discussion.

4.7. Tracking, logging and data needed for WP5 evaluation

Finally, the system must record enough information to support educational evaluation in WP5 and future analysis, while respecting data protection principles.

The system shall:

- Record key events and states relevant for learning analysis, such as products bought and sold, waste generated and reduced, mining attempts and successes, and choices of industry or strategy.
- Associate these records with pseudonymous user identifiers and game identifiers so that patterns can be analysed at player, game and cohort level.
- Provide a way to extract or aggregate this information (e.g. Through dedicated collections or snapshots) for partners responsible for evaluation and reporting.
- Log technical events (e.g. Disconnections, reconnections, errors) in a way that helps interpret anomalies in gameplay and supports continuous improvement.
- Respect data minimisation and retention policies agreed in the project, ensuring that logs and learning data are stored only as long as needed for evaluation and reporting.

These core functional requirements define what RockChain must reliably do in every deployment, serving as a shared reference for the more detailed flows and non-functional constraints described in the following sections.

5. GAME FLOWS AND STATE TRANSITIONS

The functional behaviour of RockChain is organised around a small number of recurrent flows that repeat in every session. These flows determine how a game starts, how rounds progress, what happens when time runs out and how the system reacts if players disconnect and reconnect. They are the backbone that trainers and learners experience during use, and they must remain predictable across deployments.

5.1. Overall game lifecycle

From a functional point of view, each game follows a common lifecycle:

Game setup

- A new game is created when logging in, receiving a game code.
- The game is initialised with default parameters (maximum players, number of rounds, round duration, starting resources) and marked as waiting.

Joining and lobby phase

- Learners authenticate and enter the game code to join the same session.
- The system registers each player in the game, shows them in the waiting room and tracks their readiness status.
- The host monitors who have joined and decide when the group is complete.

Game in progress

- When the host starts the game, the status changes to in progress and the first round is initialised.
- Players are taken into the in-game area (tabs for market, mining, recycle, stats, etc.) while a shared header shows the current round and remaining time.
- The game proceeds through a fixed number of rounds, each following the round lifecycle described below.

End of game and debriefing

- After the configured number of rounds, or if the host decides to end the game early, the system marks the game as finished.
- A final summary view presents scores, waste indicators and key outcomes for all players.

- Trainers can use this summary, along with stored data, as the basis for discussion and reflection.

Once a game is finished, no further actions are allowed in that session; learners can leave or join a new game if the trainer decides to run another one.

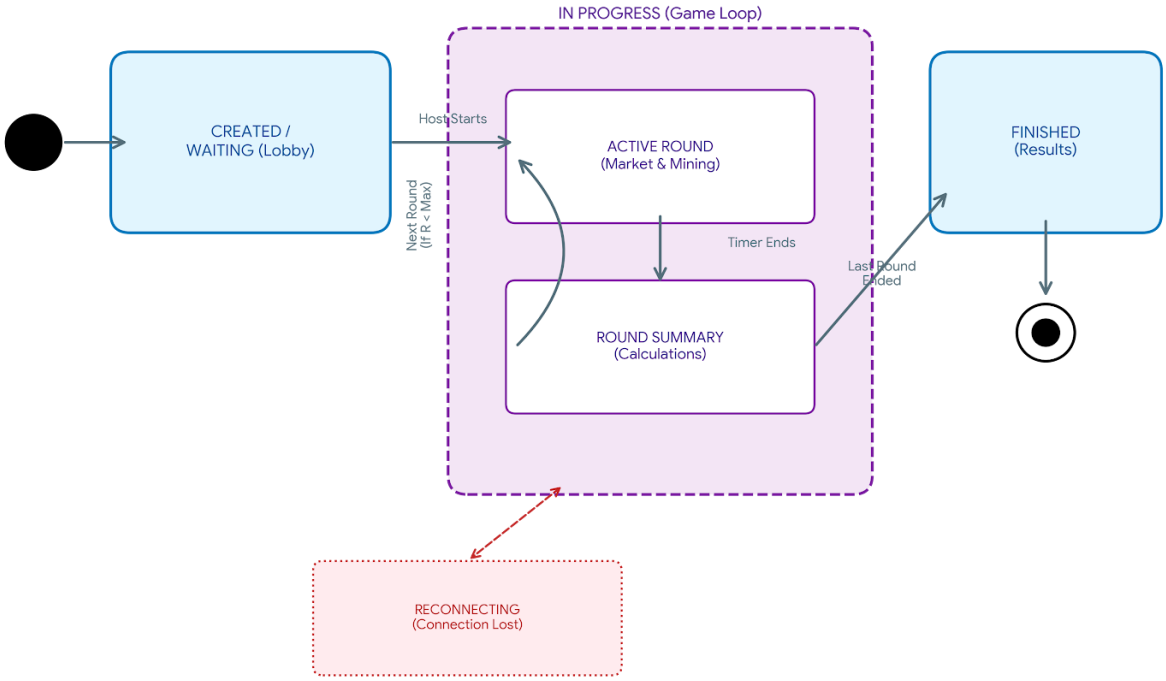

Figure 5: State machine diagram

5.2. Round lifecycle

Within the overall game, each round has its own internal lifecycle. The system manages this lifecycle so that all players experience the same phases at the same time:

Round initialisation

- At the start of a round, the backend defines the authoritative state: round number, end time, market configuration and any initial mining context.
- This information is broadcast to all connected clients, which update their views accordingly. The header starts the countdown.

Active play phase

- During active play, players can:
    - Interact with the market (buying/selling products).
    - Solve mining problems when available.

- Use recycling options.
- Consult statistics or chat.

- All actions are sent as requests to the backend; only validated operations are applied and reflected to other players.

Resolution phase

- When the authoritative timer expires, or when the system decides that all necessary conditions have been met, the round enters a resolving state.
- New actions are blocked or limited. The backend calculates:
  - Mining winners and rewards.
  - Industry-related bonuses.
  - Changes in scores, inventories and waste levels.
- A consistent set of results is written to persistent storage and broadcast to all clients.

End-of-round feedback

- Players see an end-of-round summary: their updated resources, waste levels, rewards obtained and any winning industries.
- Trainers can use this view to comment on strategies and link them to circular-economy concepts before starting the next round.

Transition to next round

- If more rounds remain, the host or the system triggers the next round.
- The cycle of initialisation, active play, resolution and feedback repeats until the game finishes.

This round lifecycle ensures that, in every group, all learners experience a clear rhythm: *prepare → act → see consequences → reflect*.

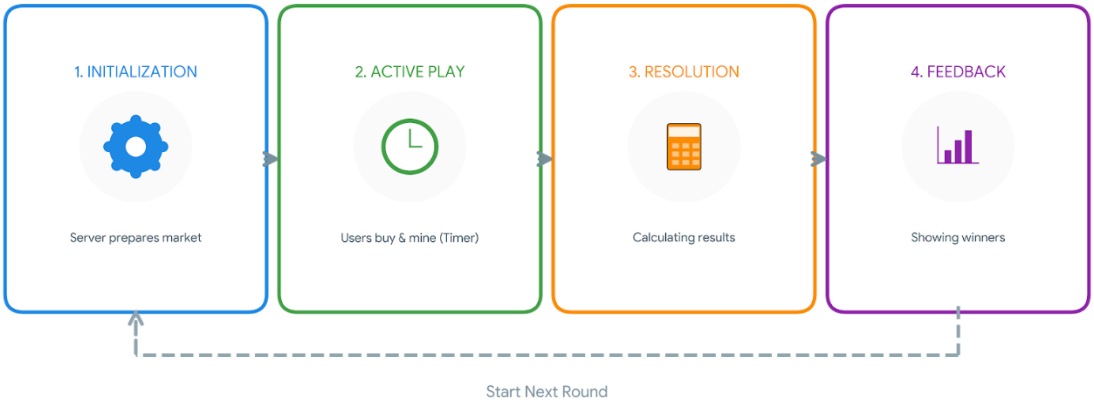

Figure 6: Round lifecycle.

5.3. Player flows

In real training sessions, participants may arrive late, temporarily lose connection or accidentally close the app. RockChain defines functional rules to handle these situations smoothly:

Joining before start

- Players can join the game freely while it is in the **waiting** state.
- The system shows late arrivals in the lobby and updates the list for everyone.
- Once the host starts the game, new joins are either prevented or restricted according to the agreed rules (for example, no joins after the first round has begun).

Re-entry during an ongoing game

- If a player temporarily loses connection or closes the app:
    - On reconnecting and logging in, the system uses their identity to reassociate them with the current game where possible.
    - They are taken back to the appropriate screen and round, with the current authoritative state applied.
- If reconnection is no longer possible (e.g. the game has ended), the system informs the user and offers to return to the main menu.

Behaviour during disconnections

- While offline, the user cannot perform actions that modify the game state.

- The interface may indicate limited or "waiting for reconnection" status; once connectivity is restored, the latest state is fetched and the view is updated.

Voluntary exit

- A player who leaves the game deliberately (for example, via a quit option) is marked as absent.
- Their previous contributions remain in the game history, but they no longer participate in subsequent actions or rewards unless they rejoin under rules defined by the trainer.

These flows are designed to reduce disruption in classroom situations where brief connectivity issues or device problems are common, while keeping the game state consistent and fair for all participants.

5.4. Synchronisation rules between clients

Because RockChain is a real-time multiplayer tool, a central aspect of its functional behaviour is keeping all clients aligned around a single, authoritative view of the game. The main synchronisation rules are:

Authoritative backend state

- The backend, not individual devices, decides:
    - Which game and round are active.
    - When rounds start and end.
    - Which market and mining configurations apply.
    - What the current scores and indicators are.
- Clients listen to this state and adjust their local views accordingly.

Shared timing reference

- Round countdowns and time-limited challenges use a shared reference time provided by the backend.
- Clients may display the remaining time with minor local adjustments, but the decision to end a round or close a mining problem always follows the authoritative timer, not the device clock.

Consistent updates

- When a valid action is performed (purchase, sale, mining success, recycling operation), the backend:

- Updates the game state.
- Persists relevant changes.
- Broadcasts the updated state or event to all players.

- Clients update their interfaces only after receiving confirmation, ensuring that everyone sees the same outcomes.

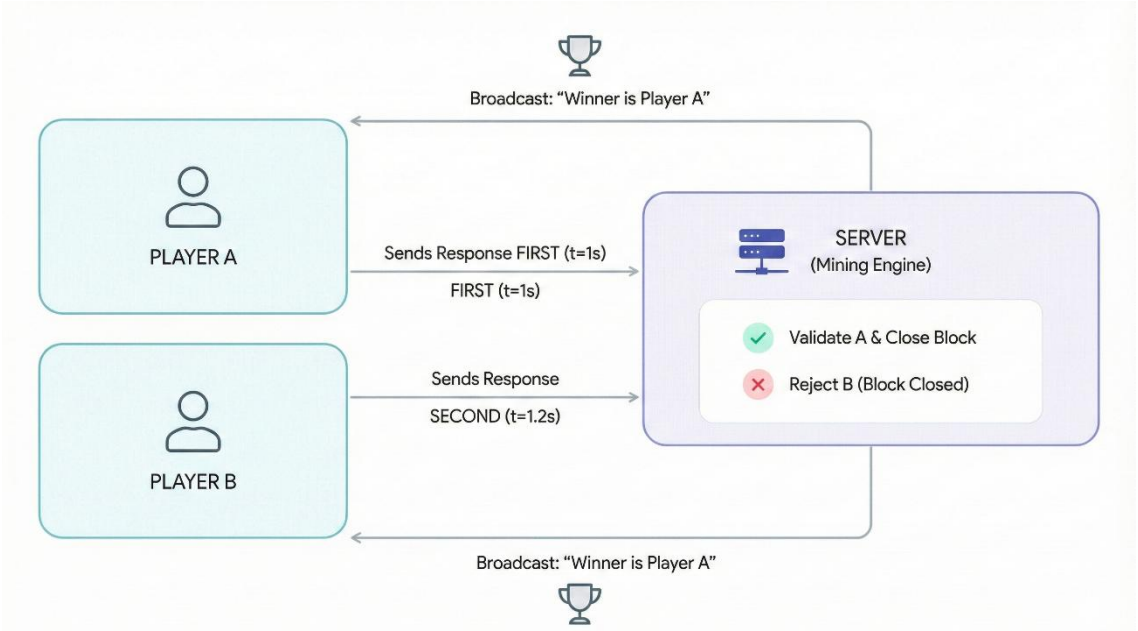

Figure 7: Mining response and connection with server.

Graceful handling of late or conflicting actions

- Actions arriving after a round has ended or a problem has been solved are rejected, with clear feedback to the user.
- If two actions would conflict (e.g. two players racing for the same mining reward), the backend applies deterministic rules (such as "first correct response") and notifies all clients of the final result.

These synchronisation rules ensure that, even in the presence of network variability, RockChain behaves as a single shared game rather than a collection of independent local simulations. For trainers and learners, this translates into a coherent experience where rounds start and end together, results are agreed by all devices, and the discussion after each round can rely on a stable and trustworthy representation of what happened.

6. NON-FUNCTIONAL REQUIREMENTS

Beyond what RockChain must do functionally, the tool must also satisfy a set of non-functional requirements so that it can be used reliably in real VET and adult-learning environments. These requirements concern performance, reliability, security and data protection, and the usability of the tool for adult learners with heterogeneous digital skills.

6.1. Performance and scalability in typical VET/ADU settings

RockChain is not intended for massive online play, but for small to medium-sized groups in classroom or workshop settings. In this context, the system shall:

- Support at least 2–5 concurrent players per game, with the possibility of running several games in parallel within a single institution, without noticeable degradation of response times.
- Ensure that core real-time interactions (round start, market updates, mining events) are reflected on client devices within a short and predictable delay (on the order of seconds, not tens of seconds), under normal classroom connectivity.
- Maintain acceptable performance when multiple rounds are played in sequence, without requiring restarts or manual interventions between games.
- Avoid excessive consumption of device resources (CPU, memory, battery) so that the app can run smoothly on typical mid-range smartphones and tablets used by adult learners.

These performance expectations reflect realistic use in VET/ADU courses and ensure that technical constraints do not overshadow the learning experience.

6.2. Reliability and fault tolerance in classroom conditions

In real sessions, temporary network glitches, device sleep, or brief disconnections are common. RockChain must be resilient enough to handle these situations without derailing the activity. The system shall:

- Tolerate short connectivity interruptions by allowing clients to reconnect and resynchronise with the current game and round, whenever technically feasible.
- Prevent data corruption or inconsistent states when disconnections occur during critical operations (e.g. end-of-round calculations, reward assignment).

- Avoid crashes or unrecoverable errors in the mobile client under normal use, providing clear messages if a critical error does occur.
- Ensure that trainers can complete a planned session (one or more games) without having to restart the infrastructure, even if individual players experience transient technical issues.

The aim is that reliability concerns do not become a barrier for trainers who may have limited time and technical support during their classes.

6.3. Security, privacy and data protection

Security, privacy and data protection requirements complement the technical safeguards already defined for the backend and database in WP4.A1. From a functional point of view, the system shall:

- Ensure that only authenticated users can access game sessions and that game codes are not guessable in practice.
- Restrict access to personal and game-related data so that each user can only see:
    - Their own profile and in-game state.
    - Information that is explicitly shared within a game (e.g. scores and rankings visible to all players in that match).
- Store and process only the minimum personal data necessary to run sessions and evaluate learning outcomes, using pseudonymous identifiers wherever possible.
- Support the project's obligations under data protection regulations (e.g. GDPR), including:
    - The possibility to honour requests for access, correction or deletion of personal data.
    - The application of agreed retention periods for logs and learning data.
    - Secure handling of credentials and configuration secrets.
- protect communication between client and backend (for example, by using encrypted channels) and apply appropriate safeguards to prevent unauthorised access or manipulation of game states.

These requirements ensure that RockChain can be deployed in educational institutions without compromising learners' privacy or institutional policies.

6.4. Usability and accessibility for adult learners with heterogeneous digital skills

Finally, the tool must be usable by adult learners, many of them over 45, who may not be familiar with mobile games or complex interfaces. The system shall:

- Present a simple and consistent navigation structure, with clear labels and minimal steps between logging in, joining a game and starting to play.
- Use language and visual elements that are easy to understand, avoiding unnecessary technical jargon and providing brief explanations where needed (e.g. For timers, rewards, or industry choices).
- Keep key information (time remaining, round number, core resources) visible in a stable location on screen, so that learners do not need to search for it while making decisions.
- Support multi-language use by allowing trainers and learners to select their preferred language from those provided in the project, and by ensuring that essential messages and feedback are available in those languages.
- Behave predictably when learners make mistakes or take longer to act, providing clear feedback and avoiding sudden changes that could cause confusion or anxiety.

By meeting these usability and accessibility requirements, RockChain can be integrated into courses with diverse adult audiences, reducing the need for constant technical assistance and allowing trainers to focus on facilitation and discussion rather than troubleshooting the tool.

7. CONCLUSIONS AND NEXT STEPS

WP4.A3 has consolidated the RockChain e-Learning Tool into a clearly specified functional system, building directly on the data layer and backend architecture delivered in WP4.A1 and the refinements carried out in WP4.A2. While earlier activities focused on making RockChain technically robust and pedagogically usable, these deliverable captures what the tool is expected to do in practice: how users are authenticated and organised into games, how rounds unfold, how markets and mining challenges behave, how scores and indicators are produced, and which events must be recorded to support evaluation. In doing so, it translates the project's learning goals into a set of observable behaviours and rules that can be consistently implemented, tested and reused.

These functional specifications now serve as a shared reference point for both technical and pedagogical partners. For WP4-A4, they provide the backbone for trainer guidelines and facilitation materials: step-by-step descriptions of the game flows, states and screens that trainers will see when preparing and running sessions. For WP4-A5, they reduce uncertainty in the final packaging and deployment work, since the expected behaviour of the tool is stable and documented, allowing partners to focus on completing assets, translations and build configurations without questioning the core mechanics. The same specifications also clarify which data and indicators are available for analysis, directly informing how WP5 can plan its evaluation instruments and data collection strategies.

Looking ahead, the behaviour defined in WP4.A3 is meant to be robust enough for pilot use and flexible enough for future adaptation. The pilots in WP5 will test these specifications under real conditions across different countries, institutions and learner profiles, providing evidence on how well the functional design supports learning about circular economy and waste in the ornamental rock sector. Feedback from those pilots may lead to incremental adjustments, but the core flows and requirements described here are expected to remain the reference framework. In this way, WP4.A3 marks a transition from designing and refining RockChain to deploying and using it systematically, ensuring that the tool can be maintained, replicated and, if needed, extended beyond the lifetime of the project.