Escape Room: EscapeCircuit

“Wire your way out.”

Application Design Document

Academic Advisor: Niv Gilboa

Clients: Gera Weiss and Oded Margalit

Group Members:

Dor Steinlauf

Noam Shlomo Yosef

Mendy Dishon

Yuval Zarmi

version 1.0

Date:June, 2026

EscapeCircuit

Table Of Contents

Chapter 1 – Use Cases 4
1.1 User Profiles - The Actors 4
1. Puzzle Solver (Primary Human Actor) 4
2. Puzzle Creator (Elevated Human Actor) 5
1.2 Use-Cases - Acceptance Tests 6
Use Case 1 - Browse and Search Puzzles 6
Use Case 2 - Solve a Puzzle 8
Use Case 3 - Save and Reuse Circuits 13
Use Case 4 - Create and Publish Puzzle 16
Use Case 5 - Rate a Puzzle 21
Use Case 6 - Earn Rewards and Level Up 24
Use Case 7 - Manage Puzzles (Creators Only) 27
Use Case 8 - Puzzle solution Validation 30
Use Case 9 - Appoint User as Creator 32
Use Case 10 - View Personal Profile & Progress 34
Use Case 11 - Experiment in Sandbox 36
Use Case 12 - Manage Personal Arsenal 38
Use Case 13 - Participate in Discussions 40
1.3 Special Usage Considerations 44
1.3.1. Real-Time Interaction Constraints 44
1.3.2. Deterministic Puzzle Evaluation 44
1.3.3. Creator Privileges & Safety Checks 44
1.3.4. Fair Rating and XP Systems 45
1.3.5. Resource Limits for Gameplay Balance 45
1.3.6. Accessibility and Ease of Use 45
1.3.7. Environment and Operational Dependencies 45
1.3.8. Security and Integrity Considerations 46
Chapter 2 – System Architecture 47
2.1 Architectural Overview 47
2.2 Components and Deployment 47
2.3 Container / Deployment Diagram 49
Chapter 3 – Data Model 50
3.1 Description of Data Objects 50
3.2 Data Objects Relationships 55
3.3 Databases 56
Chapter 4 – Behavioral Analysis 58
4.1 Sequence Diagrams 58
4.1.1 UC1 – Browse and Search Puzzles 58
4.1.2 UC2 – Solve a Puzzle 59

4.1.3 UC3 - Save and Reuse Circuits....60
4.1.4 UC4 – Create and Publish Puzzle
Addition: Before self-solve (step 5), the system validates puzzle metadata: name uniqueness check, field length limits (name ≤ 100, description ≤ 500, instructions ≤ 5,000), board size validation (9–20 rows), and safe upload validation before any files are created....61
4.1.5 UC5 – Rate a Puzzle....62
4.1.6 UC6 – Earn Rewards and Level Up....63
4.1.7 UC7 – Manage Puzzles (Creator/Admin)....64
4.1.8 UC8 – Puzzle solution Validation....65
4.1.9. UC9 - Appoint User as Creator....66
4.1.10. UC10 - View Personal Profile & Progress....67
4.1.11. UC11 - Experiment in Sandbox....68
4.1.12. UC12 - Manage Personal Arsenal....69
4.2 Events....70
4.3 States....72
4.3.1 Puzzle Solving State Machine....72
4.3.2 Puzzle Lifecycle - creation of a puzzle....73
4.3.3 Rating Puzzle....74
Chapter 5 – Object-Oriented Analysis....75
5.1 Class Diagrams....75
Backend (service-oriented) class diagram:....75
5.2 Class Descriptions....76
PuzzleService:....76
SolvingService:....78
XPService:....79
RatingService....80
5.3 Packages....81
Backend (Python):....81
Frontend (React):....82
5.4 Unit Testing....83
Chapter 6 – User Interface Draft....84
6.1 Home / Puzzle Browser....84
6.2 WorkStation (Puzzle Solving Screen)....85
6.3 Profile Page....86
6.4 Creator Dashboard....88
6.5 Admin Panel....89
Chapter 7 – Testing....91
7.1 Testing Approach and Strategy....91
7.2 Unit Testing: Core Logic and Constraints....92
7.2.1 Circuit Cost and Board Constraints....92
7.2.2 Experience (XP) and Rewards Calculation....95

7.2.3 Rating System Weighting..... 99
7.3 Functional and Integration Testing..... 102
7.4 Non-Functional Testing..... 104
7.5.1. Core Principle: Deterministic and Reproducible Evaluation..... 106
7.5.2 What “Correct Solution” Means in This System..... 107
7.5.3 Unit Tests: Proving the Rules in Isolation..... 107
7.5.4 Integration Tests: Ensuring Correctness Across the Full Flow.... 108
7.5.5 UI Tests: Correct Interaction Contract in the WorkStation..... 108
7.5.6 Non-Functional Testing: Performance, Reliability, Security, Coverage..... 109
7.5.7 Continuous Testing and Regression Prevention..... 109

Chapter 1 – Use Cases

1.1 User Profiles - The Actors

With the EscapeCircuit system, several external actors interact by exchanging data with the platform. Each actor has different goals, permissions, and forms of interaction with the system.

1. Puzzle Solver (Primary Human Actor)

Description:

The Puzzle Solver represents the typical user of the system and is usually a student, but could also be a participant in the Faculty Escape Room activity or any random online learner.

Responsibilities and Interactions:

- Accesses the system to browse and search for available puzzles.
- Engages with the puzzle-solving interface by building logical circuits using AND, OR, NOT, and other gates.
- Interacts with the backend via user actions that trigger logic validation, XP updates, and medal assignment.
- Submits ratings for solved puzzles (Difficulty, Fun, and Clearness).
- Saves personal circuits in their private arsenal (capacity scales with user level, from 5 up to 50 saved designs)."
- Receives XP, medals, and profile updates as system outputs.

Exchange Data:

- Inputs: User credentials, puzzle selections, circuit connections, rating values.
- Outputs: Puzzle results, XP increments, medals, saved circuits, updated user level, and per-puzzle leaderboard ranking by time or cost.

2. Puzzle Creator (Elevated Human Actor)

Description:

A Creator is a Solver with additional privileges to design and publish puzzles. Mostly university lecturers, TAs, and experienced students (who have reached a high level).

Responsibilities and Interactions:

- Defines new puzzles by providing all puzzle parameters.
- Uploads and publishes puzzles only after successfully solving them within their own set constraints.
- Assigns initial difficulty ratings and creator comment.
- Receives feedback data such as solver statistics and ratings to improve or balance puzzle design.

Exchange Data:

- Inputs: Puzzle configuration, test cases, gate values, custom circuits, creator comment, board grid size. The Creator may also define an Initial Board for the puzzle: a set of pre-placed, locked components and locked wires that appear on the solver's board from the start of the puzzle. These locked elements cannot be moved, rotated, or deleted by the solver (only by the creator in edit mode), but they count toward the puzzle's budget and are part of the official solution context.
- Outputs: Published puzzles in the shared repository, creator XP from player engagement, and solvers' feedback.

1.2 Use-Cases - Acceptance Tests

Use Case 1 - Browse and Search Puzzles

Actors: Solver player

New components: InfoPopup (reusable click-to-toggle contextual help overlay used across WorkStation, Arsenal, and Create Puzzle pages), PuzzleLeaderboard (per-puzzle podium displaying top 3 solvers by fastest time or lowest cost, with medal icons and full ranking list, togglable between time and cost views).

Trigger: User selects the Browse/Search option after logging in.

Preconditions:

- The user is authenticated.
- At least one puzzle exists in the system.

Parameters:

Search filters: puzzle name, creator, difficulty, solved/unsolved (might be more in the future).

End conditions:

If there are puzzles that match the filters, they will be presented to the user.

Order of Operations:

1. User logs in to the system.
2. User opens home screen.
3. System displays puzzles.
4. User enters search filters.
5. System retrieves matching results.
6. User chooses puzzle or marks for later.

Alternative:

- No results found → system displays appropriate message.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful Search	User 123 is logged in. User 123 searches with the filters: difficulty=Hard, solved=Unsolved	System returns puzzles meeting filters.
Unsuccessful Search: No Matches	User 123 is logged in. User 123 searches with filters, but no matching puzzles	Empty list + message that tells the users no puzzle matches their filters
Unsuccessful unauthorized Access	User 123 is not logged in. User 123 does anonymous request	Appropriate message will be presented

Use Case 2 - Solve a Puzzle

Actors: Solver player

Trigger: User selects a puzzle to solve.

Preconditions:

- The user is authenticated.
- The desired puzzle exists in the system.
- Puzzle loaded successfully with all gates and limits.
- If the puzzle defines an Initial Board, all pre-determined (locked) components and wires are loaded onto the solver's workstation grid before the solver begins building, and remain non-removable throughout the attempt.

Parameters:

Circuit graph (logical connections), time limit, Value usage.

Editor actions available while building the solution include: placing components, wiring, deleting, rotating, selecting a component or a multi-component region, copying the selection to an in-memory clipboard (Ctrl+C / Cmd+C), and pasting it back onto the board at an offset position (Ctrl+V / Cmd+V). The clipboard is client-side only and is not persisted across sessions.

End Conditions:

Puzzle marked as solved or unsolved, XP and medals updated accordingly.

Users' solve time and cost is recorded, and their ranking appears on the puzzle's leaderboard (ranked by time or cost).

Order of Operations:

1. User logs in to the system.
2. User selects a puzzle.
3. System loads the selected puzzle interface.
4. User builds a solution using available gates.

4a. While building, the user may select one or more components and use copy (Ctrl+C / Cmd+C) and paste (Ctrl+V / Cmd+V) to duplicate the selection together with its internal wires. Pasted components are placed at a small grid offset from the originals, clamped to the board bounds, and are subject to the puzzle's budget; pre-determined locked components cannot be copied.

4b. Any pre-determined (locked) components and wires from the puzzle's Initial Board remain on the board for the entire attempt and cannot be moved, rotated, or deleted by the solver.

5. User runs evaluation. System validates the submitted circuit data before processing (safe upload validation).
6. System checks test cases.
7. Displays results and awards XP or medals.
8. System records the user's solve time and cost, and updates the per-puzzle leaderboard with their ranking."

Alternative:

- Solution incorrect → user receives “Not Solved”.
- Time expired → no Timer Medal.
- Exceeded limits → system blocks addition.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful solve within limits	User 123 is logged in. User 123 selects a puzzle. User 123 solves valid solution within limits; time remaining	Puzzle solved, XP & Timer Medal
Unsuccessful wrong solution	User 123 is logged in. User 123 selects a puzzle. User 123 doesn't succeed in solving the puzzle, thus presents an unworking solution.	Not solved, no XP
Unsuccessful exceeded the value limit	User 123 is logged in. User 123 selects a puzzle. User 123, while solving, tries to add a gate to the solution that makes him exceed the allowed amount Addition makes Value > limit	Addition blocked, puzzle unchanged

Unsuccessful time expired	User 123 is logged in. User 123 selects a puzzle. User 123 solves correctly, but the timer expired	Puzzle solved, no Timer Medal
Successful leaderboard ranking	User 123 is logged in. User 123 solves a puzzle with a valid solution. Other users have previously solved the same puzzle.	ranked by fastest time or lowest cost (togglable). Top 3 solvers shown on podium.
Successful copy-paste of a sub-circuit	User 123 selects a group of components and presses Ctrl+C, then Ctrl+V.	The selected components and their internal wires are duplicated on the board at a small offset, clamped to the grid, new unique IDs are assigned, the duplicates are auto-selected, and the puzzle budget is re-validated.
Unsuccessful copy-paste exceeds budget	User 123 is logged in. Puzzle budget remaining is smaller than the cost of the copied selection. User 123 presses Ctrl+V.	Paste is blocked (or the addition is rejected) and an appropriate 'budget exceeded' message is displayed; the board is unchanged.
Locked initial	User 123 is logged in.	The action is rejected

components cannot be modified	User 123 selects a puzzle that has pre-determined (locked) components on the initial board. User 123 attempts to drag, rotate, delete, or copy a locked component.	for locked components; the locked components remain in their original position, orientation, and set.
Initial board restored on Try Again	: User 123 is logged in. User 123 selects a puzzle with an initial board and clicks 'Try Again' after a failed attempt.	All pre-determined locked components and locked wires are re-added to the board exactly as defined by the puzzle, while the solver's own (unlocked) components are cleared.

Use Case 3 - Save and Reuse Circuits

Actors: Solver player

Trigger: User clicks on the button “Save Circuit” after creating a circuit.

Preconditions:

- The user is authenticated.
- The circuit is composed only of default gates or operations, and the user has fewer than the limit of saved circuits in their arsenal.

Parameters:

Circuit name (unique within the user's arsenal), save scope (current puzzle or personal arsenal), circuit composition.

End Conditions:

The circuit is saved to the appropriate scope, or saving is rejected with an explanation.

Order of operations:

1. User logs in to the system.
2. User selects a puzzle.
3. System loads the selected puzzle interface.
4. User enters the sandbox to create a new circuit for the arsenal.
5. User selects the “Save Circuit” option.
6. System prompts the user to enter a name and choose a save scope.
7. System verifies that the circuit uses only default gates and that the selected name is unique.
8. System computes the circuit’s Value, and default set and checks the arsenal capacity.
9. If valid, the system saves the circuit to the chosen scope.

Alternative:

- Circuit includes a component that is not default → Saving is refused.
- Name collision → user must choose a different name.
- Arsenal capacity reached → saving is denied.
- Sent to the current puzzle → circuit sent to the active puzzle and not added to the user's arsenal.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful save to arsenal	User 123 is logged in. User 123 creates a legal circuit. User 123 saves the new circuit to the arsenal	Circuit saved, only used I/O; appears in arsenal
Successful reuse compatibility	User 123 is logged in. User 123 selects a circuit from his arsenal; puzzle includes all required defaults	Circuit appears in the puzzle. The circuit is hidden if defaults are missing, and can't be selected
Unsuccessful name collision	User 123 is logged in. User 123 doesn't succeed in creating a circuit - must have a unique name.	Save rejected, must choose a unique name + appropriate message will be presented
Unsuccessful arsenal full	User 123 is logged in. User 123 doesn't succeed in creating a circuit because arsenal has the maximum amount of circuits	Save denied, capacity reached + appropriate message will be presented

Use Case 4 - Create and Publish Puzzle

Actors: Creator player

Trigger: Creator selects “Create Puzzle” to design a new puzzle.

Preconditions:

- The user is authenticated.
- Creator privileges verified.
- Required fields (name, limits, etc.) are filled.

Parameters:

- Puzzle name
- Number of inputs (0–5)
- Number of outputs (1–20)
- Bit stream lengths (1–20)
- Test cases
- Value limit and greater value limit
- Default dictionary (gates and values)
- Basic circuits for the puzzle
- Description and time limit
- Initial Difficulty rating and optional comment
- Board rows (grid height, default 15)
- Board columns (grid width, default 30)
- Initial Board (optional): a set of pre-placed, locked components (locked_placed) and locked wires (locked_wires) that the creator places on the puzzle grid.
- Each locked placed component has: id, componentId, origin {row, col}, rotation ∈ {0, 90}, and isLocked=true.
- Each locked wire has: id, from/to endpoints (componentId, portId, pinIndex), and isLocked=true.
- The Initial Board is serialized as initial_board_json and stored on the Puzzle; it is loaded onto every solver's workstation at the start of their attempt and cannot be modified by the solver.
- Validation constraints: Puzzle name must be ≤ 100 characters and unique across all puzzles. Description must be ≤ 500 characters. Instructions must be

≤ 5,000 characters. All puzzle data is validated server-side before any file is created on disk (safe upload validation).

End conditions:

- Puzzle is successfully published, with upload datetime set.
- Or, creation is rejected if validation or self-solve fails.

Order of operations:

1. User logs in to the system.
2. Creator opens the new puzzle interface.
3. Inputs all puzzle parameters.

   3a. (Optional) Creator lays out the puzzle's Initial Board on the workstation grid: places components, wires them, and marks them as locked. On submit, all placed components and wires inside the initial board are automatically flagged isLocked=true and serialized into initial_board_json.

4. System validates the data: checks name uniqueness, enforces field length limits (name ≤ 100, description ≤ 500, instructions ≤ 5,000), validates board size, and verifies all data before creating any files on disk.
5. Creator self-solves the puzzle within the greater limits.
6. Upon success with system validation, system records upload datetime and publish the puzzle.

Alternative:

- If self-solve fails, the puzzle is not uploaded.
- If the puzzle's name is not unique, the system asks for a different name.
- If limits are invalid, the system displays an error and blocks upload.
- If name exceeds 100 characters, or description exceeds 500, or instructions exceed 5,000 → system displays a validation error and blocks upload.
- If board rows or columns values are invalid → system displays an error.
- If the Initial Board contains components placed outside the grid, overlapping placements, wires with dangling or invalid endpoints, or references to components not in the puzzle's allowed set → the system displays a validation error and blocks upload.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful valid publish	User 123 is logged in. User 123 creates the puzzle All puzzle data is valid, and the creator solves within greater limits.	Puzzle saved, upload datetime set; initial difficulty uses creator weighting
Successful basic circuits allowed	User 123 is logged in. User 123 adds new Basic Circuits and assigns values	Circuits stored as puzzle-only, solvers cannot save
Unsuccessful self-solve fails	User 123 is logged in. User 123 cannot solve puzzle within greater limits	Upload blocked, prompts revision
Unsuccessful duplicate name	User 123 is logged in. User 123 creates the puzzle All puzzle data is valid, and the creator solves within greater limits. Except that User 123 chooses an existing name.	Name rejected, choose unique one + appropriate message will be presented
Unsuccessful invalid	User 123 is logged in.	Upload prevented + appropriate

limits	User 123 creates the puzzle Greater value ≤ standard limit	message will be presented
Unsuccessful name too long	User 123 is logged in. User 123 creates a puzzle with a name exceeding 100 characters.	Upload blocked, validation error displayed + appropriate message will be presented
Unsuccessful description too long	User 123 is logged in. User 123 creates a puzzle with description exceeding 500 characters.	Upload blocked, validation error displayed + appropriate message will be presented
Successful custom board size	User 123 is logged in. User 123 creates a puzzle and sets board rows to 12 and columns to 25.	Puzzle created with 12×25 grid; solvers see the custom board
Unsuccessful custom board size	User 123 is logged in. User 123 creates a puzzle and sets board rows to -12 and columns to 25.	Puzzle is not created user gets an error message
Successful initial board saved	: User 123 (creator) is logged in. User 123 creates a puzzle, places two AND gates and one wire on the grid, and marks them as the initial board.	Puzzle is saved with initial_board_json containing two locked_placed entries (isLocked=true) and one locked_wires entry (isLocked=true); solvers see these components pre-placed on their board.
Unsuccessful initial board out of bounds	User 123 (creator) is logged in. User 123 adds an initial-board	Upload is blocked, a validation error message is displayed, and no

	component whose origin lies outside the configured boardRows × boardCols grid.	puzzle file is written to disk.
Unsuccessful initial board overlap	User 123 (creator) is logged in. User 123 attempts to publish a puzzle with two initial-board components whose cells overlap.	Upload is blocked, a validation error message is displayed, and the puzzle is not created.

Use Case 5 - Rate a Puzzle

Actors: Solver player

Trigger: User finishes solving a puzzle or has attempted it for at least five minutes and chooses to provide feedback.

Preconditions:

- The user is authenticated.
- User has solved the puzzle or has been engaged for a minimum of five minutes.
- User is allowed to rate (has not rated this puzzle previously unless editing).

Parameters:

- Rating values for Difficulty, Fun, and Clearness on a given scale.

End conditions:

- Ratings are stored and averages recalculated.
- User receives a small XP reward for contributing a rating.

Order of operations:

1. User logs in to the system.
2. System prompts the user to rate after completion or after sufficient attempt time.
3. User chooses the rate on the scale for each category.
4. System updates the overall and experienced-only averages, adjusting creator weightings when applicable.
5. XP reward is applied to the user.

Alternative:

- User declines to rate → no rating is stored.
- User edits a previous rating → existing averages are updated accordingly.
- User attempts for less than five minutes and didn't solve → rating is denied.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful valid rating (Pre-10)	User 123 is logged in. Puzzle has <10 ratings; user submits a new rating	Difficulty = 80 % creator + 20 % user average, Fun & Clearness = user average
Successful experienced weighting	User 123 is logged in. User 123 submits a new rating Rater level ≥ 5; submits ratings	Rating counted twice: general and experienced-only averages
Successful post-10 shift	User 123 is logged in. User 123 submits a new rating. Puzzle reaches ≥10 total ratings	Creator weighting drops to 40 %, user weighting rises to 60 %
Successful edit rating	User 123 is logged in. User 123 edits their previous rating	Aggregates recalculated; no extra XP granted
Unsuccessful ineligible rating	User 123 is logged in. User 123 attempted <5 min and didn't solve	Rating rejected
Unsuccessful duplicate rating	User 123 is logged in.	Rating rejected; system enforces one rating per

	User 123 has already rated this puzzle. User 123 submits a new rating (not via edit).	user per puzzle. User must use the edit flow instead.

Use Case 6 - Earn Rewards and Level Up

Actors: Solver player

Trigger: User successfully solves a puzzle and completes the solution submission.

Preconditions:

- A valid win is recorded for the user on the puzzle.

Parameters:

- Puzzle difficulty tier (e.g., Easy, Medium, Hard).
- Time status (whether the timer was beaten).
- First-solve flag
- Alternative solution flag.

End conditions:

- XP, medals, and level status are updated to reflect the victory. (Medals may be upgraded based on limits.)

Order of operations:

1. User logs in to the system.
2. User solves a puzzle and completes the solution submission.
3. System calculates base XP according to puzzle difficulty and applies any alternative solution bonus.
4. If it is the user's first win on the puzzle and time remains, the Medal is upgraded.
5. System checks whether the greater value limits were respected; upgrades medal tier accordingly.
6. XP is added to the user's profile, potentially increasing the level.
7. Results and rewards are displayed to the user.

Alternative:

- Subsequent solves on the same puzzle do not grant an alt-solution bonus.
- If limits are not met, the medal tier remains at bronze.

Note: Medal assignment now handles boundary edge cases, cost exactly equal to the greater value limit is treated as within limits.

.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful standard XP	User 123 is logged in. User 123 reaches a new difficulty tier. Puzzle difficulty tier defined	XP awarded per tier
Successful medal upgrade	User 123 is logged in. User 123 is solved within greater limits (value/time)	Medal upgraded (bronze → silver → gold)
Successful level up	User 123 is logged in. user's 123 XP crosses next level threshold	User level increases
Unsuccessful repeat timer	User 123 is logged in. User 123 solved a puzzle within the given time. User 123 does it the sconed time in the same puzzle. Second or subsequent win; time remaining	No additional Timer Medal or alt-solution bonus

Successful medal tie at boundary	User 123 is logged in. User 123 solves a puzzle with cost exactly equal to the greater value limit.	Medal correctly upgraded; boundary value is treated as within limits
Successful leaderboard after reward	User 123 is logged in. User 123 solves a puzzle and earns gold medal.	user appears on puzzle leaderboard ranked by solve time or cost (based on selected view)

Use Case 7 - Manage Puzzles (Creators Only)

Actors: Creator player

Trigger: Creator accesses the “Created Puzzles” section of their profile to manage published puzzles.

Preconditions:

- The creator has at least one published puzzle.

Parameters:

- Puzzle identifier.
- Action: edit comment, delete comment, or delete puzzle.

End conditions:

- Puzzle comment is added, edited, or removed as requested.
- Puzzle is deleted from public listings if removal is confirmed.

Order of operations:

1. User logs in to the system as a creator.
2. Creator navigates to the “Created Puzzles” tab in their profile.
3. System displays the list of puzzles created by the user.
4. Creator selects a specific puzzle and chooses an action (add/edit/delete comment or delete puzzle).
5. System performs the requested action if the user is authorized.

Alternative:

- A non-creator or a different creator attempts to manage a puzzle → the request is denied with a “forbidden” response.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful comment addition	User 123 is logged in. User 123 is on the page of one of his puzzles. User 123 adds a new comment to his puzzle.	Comment stored and visible on puzzle page
Successful edit comment	User 123 is logged in. User 123 is on the page of one of his puzzles. User 123 modifies the existing comment	Comment replaced and timestamp updated
Successful delete comment	User 123 is logged in. User 123 is on the page of one of his puzzles. User 123 chooses to delete comment	Comment removed from puzzle page
Successful delete puzzle	User 123 is logged in. User 123 is on the page of one of his puzzles. User 123 deletes puzzle	Puzzle removed from public list (stats kept)

Unsuccessful unauthorized	User 123 is logged in. User 123 doesn't have permission to the puzzle in edit mode and tries to edit/delete.	Action rejected with error (permission denied)

Use Case 8 - Puzzle solution Validation

Actors: Creator player / Solver player

Trigger: Creator/Solver solves a puzzle and submit the solution for the system to check.

Preconditions:

- The puzzle exists in the system and there is an official solution with the test cases.

Parameters:

- Puzzle identifier.
- Circuit graph (logical connections), time limit, Value usage.
- test cases for the puzzle

End conditions:

- System returns true/false based on solution correctness.

Order of operations:

1. User logs in to the system.
2. User selects a puzzle.
3. System loads the selected puzzle interface.
4. User builds a solution using available gates.
5. User runs evaluation.
6. System checks each test case and returns true if all are correct and false otherwise.

Alternative:

- The solution isn’t suitable for the puzzle’s truth tables.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful correct solution	User 123 is logged in. User 123 selects a puzzle. User 123 solves valid solution within limits; time remaining The system check with the test cases and returns the answer	System returns ture, that signal that the solution is correct
Unsuccessful incorrect solution	User 123 is logged in. User 123 selects a puzzle. User 123 solves invalid solution within limits; time remaining The system check with the test cases and returns the answer	System returns false, that signal that the solution is incorrect
Unsuccessful limits were broken	User 123 is logged in. User 123 selects a puzzle. User 123 solves outsides limits The system check with the test cases and returns the answer	System returns false, that signal that the solution is incorrect

Use Case 9 - Appoint User as Creator

Actors: Admin (Initiator), Puzzle Solver (Target)

Trigger: An Admin selects a user profile or uses a search tool to grant Creator privileges.

Preconditions:

- The Initiator is authenticated and has the "Admin" role.
- The Target user exists and is currently a "Puzzle Solver".

Parameters:

- Target User ID/Username.

End conditions:

- The target user receives the role immediately.

Order of Operations:

1. The admin logs in to the system.
2. The admin navigates to the admin panel.
3. The admin selects the "Appoint as Creator" option.
4. The system verifies that the Initiator has permission to appoint.
5. The system sends an appointment invitation or updates the Target's status.
6. System confirms the action to the Initiator.

Additional Admin Actions:

• Remove Creator role from a user

• Ban/unban users from discussions

• Delete any puzzle

• Unpublish any puzzle

• Edit creator puzzle capacity overrides

• View puzzles flagged for moderation (low ratings, unrated)

• Audit logging of all admin actions

• Manage discussion/reply reports (PENDING, RESOLVED, IGNORED)

Alternative:

- Target user is already a Creator → System displays "User is already a Creator".
- Initiator not authorized → System denies access.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful Appointment	User 123 (admin) is logged in. User 456 is a Solver. User 123 appoints User 456.	System records appointment/sends invite to User 456.
Successful already appointed	User 123 (admin) is logged in. User 456 is a Solver. User 123 appoints User 456.	System notify that this user is already a creator.
Unsuccessful - Unauthorized	User 789 (Solver) is logged in. User 789 tries to access the appointment interface.	System denies access/hides the option.

Use Case 10 - View Personal Profile & Progress

Actors: Puzzle Solver

Trigger: User clicks on their profile icon or name in the header.

Preconditions:

- The user is authenticated.

Parameters:

- User ID.

End conditions:

- The user sees their current level, XP, medals, arsenal and lists of saved/created puzzles(if creator).

Order of Operations:

1. User logs in to the system.
2. User clicks the "Profile" button in the header.
3. System retrieves user data (XP, Level, Medals count).
4. System retrieves "Saved for Later" puzzles and "Personal Arsenal" count.
5. System displays the Profile Page with all metrics.

Alternative:

- Data retrieval error → System displays error message "Could not load profile."

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful Profile Load	User 123 is logged in with 500 XP and 2 Gold Medals. User 123 views profile.	System displays "Level X", "500 XP", and "2 Gold Medals"
Successful Puzzle Access	User 123 is logged in and has saved 3 puzzles for later. User 123 views profile.	System displays a clickable list of the 3 saved puzzles.

Use Case 11 - Experiment in Sandbox

Actors: Puzzle Solver

Trigger: User enters into the "Sandbox" .

Preconditions:

- The user is authenticated.

Parameters:

- Circuit components (Gates, Operations), Circuit Name (for saving).

End conditions:

- A new circuit is created and optionally saved to the user's Personal Arsenal.

Order of Operations:

1. User logs in to the system.
2. User enters Sandbox mode within the WorkStation (integrated, not a separate page).
3. System loads the WorkStation in Sandbox mode without puzzle-specific constraints (Budget/Time). The Sandbox is accessed from within the puzzle WorkStation interface.
4. User drags and drops components to build a circuit.
5. User clicks "Save to Arsenal".
6. User provides a unique name for the circuit.
7. The system checks Arsenal capacity (scales with user level: 5 at level ≤2, 10 at ≤4, 20 at ≤6, 35 at ≤8, 50 at >8) and name uniqueness.
8. The system saves the circuit.

Alternative:

- Arsenal Full (50 circuits) → System blocks save and alert the user to delete old circuits.
- Name Collision → System prompts user to choose a different name.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful Sandbox Save	Successful Sandbox Save User 123 is logged in with arsenal slots available (capacity based on user level). User 123 builds a valid circuit and saves it as "MyAdder".	The circuit is saved to Arsenal and available for future puzzles.
Unsuccessful - Arsenal Full	User 123 has reached their level-based arsenal capacity. User 123 tries to save a new circuit.	The system rejects save and displays a capacity error indicating current capacity and how to increase it (level up)."
Unsuccessful - Name collision	User 123 has circuit named “A” in his Arsenal. User 123 tries to save a new circuit named “A”.	The system rejects save and displays a naming error.

Use Case 12 - Manage Personal Arsenal

Actors: Puzzle Solver

Trigger: User accesses the "Saved Circuits" or "Arsenal" tab within their Profile.

Preconditions:

- The user is authenticated.
- The user has at least one saved circuit in their arsenal.

Parameters:

- Circuit ID, Action (Delete / Rename).

End conditions:

- The selected circuit is removed from the database or renamed.

Order of Operations:

1. User logs in to the system.
2. User navigates to Profile > Arsenal.
3. The system displays a list of saved circuits with their costs and dates.
4. User selects a circuit.
5. User chooses "Delete".
6. The system requests confirmation.
7. User confirms.
8. The system removes the circuit and updates the count (e.g., 38/50).

Alternative:

- Rename Circuit: User chooses → Inputs new name → System verifies uniqueness → Updates name.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful Deletion	User 123 has 50/50 circuits. User 123 deletes "Old_Circuit".	The system deletes the file, count becomes 49/50.
Successful Rename	User 123 renames "Circuit_A" to "Circuit_B".	System updates the name in the list.

Use Case 13 - Participate in Discussions

Actors: Solver player

Trigger: User navigates to the Discussions section.

Preconditions:

• The user is authenticated.

Parameters:

• Discussion title, body, tags.

• Reply content, vote direction (up/down), reaction emoji.

End Conditions:

• Discussion is created, or reply is posted, or vote/reaction is recorded.

Order of Operations:

1. User logs in to the system.

2. User navigates to the Discussions page.

3. System displays list of discussions (filterable, sortable).

4. User creates a new discussion or selects an existing one.

5. User can reply, upvote/downvote, add reactions, or bookmark.

6. Discussion creator can pin replies. Admins can lock discussions.

Alternative:

• User is banned from discussions → action denied with message.

• Discussion is locked → replies are blocked.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful discussion creation	User 123 logged in, navigates to Discussions, creates discussion with title/body/puzzle	Discussion created, appears in list, visible to others
Successful reply with voting	User 123 logged in, opens User 456's discussion, replies and upvotes another reply	Reply posted, vote count updated, one vote per user per reply
Successful discussion locking by admin	Admin locks discussion, User 456 tries to reply	Discussion locked, reply rejected with lock message, existing replies stay visible
Unsuccessful user is discussion-banned	User 123 logged in but banned (is_discussion_banned=true), attempts to create/reply	Action rejected with ban message, nothing created

Use Case 14 - Manage Notifications

Actors: Solver player / Creator player

Trigger: System event generates a notification (solve, rating, role change, etc.).

Preconditions:

• The user is authenticated.

Parameters:

• Notification type (solve, rating, creator_invite, role_change, puzzle_unpublished).

• Filter by type, puzzle, actor, date.

End Conditions:

• User views and/or marks notifications as read.

Order of Operations:

1. User logs in to the system.
2. User navigates to the notifications page.
3. System generates notification on relevant events.
4. User views notification list (unread/all).
5. User can filter notifications by type, puzzle, or date.
6. User marks notifications as read.

Acceptance Tests:

Test Name	Setup & Parameters	Expected Results
Successful notification on puzzle solve	User 123 (creator) is logged in. User 123 has a published puzzle. User 456 solves User 123's puzzle successfully.	A notification of type "solve" is generated for User 123, containing User 456's username and the puzzle name. Notification appears in User 123's unread notifications list.
Successful notification filtering	User 123 is logged in. User 123 has multiple notifications of different types (solve, rating, role_change). User 123 opens the notifications list and filters by type "rating".	Only notifications of type "rating" are displayed. Other notification types are hidden. User can also filter by puzzle, actor, or date.

1.3 Special Usage Considerations

The system, EscapeCircuit, has several usage-specific requirements and environmental considerations that impact how users interact with the platform and the system mu4st operate. These considerations make sure that accessibility, fairness, academic suitability, and system reliability are met.

1.3.1. Real-Time Interaction Constraints

Solving logic-based puzzles in EscapeCircuit is highly interactive and time sensitive, thus,

- The puzzle timer cannot be paused to ensure fairness and consistency across all solvers.
- Circuit evaluation must occur in real time, providing immediate feedback on output behavior and constraint usage.
- Performance must remain stable even as users drag, drop, and connect multiple circuit elements.

1.3.2. Deterministic Puzzle Evaluation

To maintain academic and gameplay integrity,

- All puzzle evaluations must be deterministic, meaning identical circuits always yield identical truth tables and results.
- Circuit cost must be calculated consistently and reproducibly across sessions and users.
- Saved circuits must behave identically regardless of where they are reused.

1.3.3. Creator Privileges & Safety Checks

Puzzle creation introduces elevated responsibilities,

- A creator must successfully solve their puzzle before publishing it, enforcing puzzle validity.
- Constraints (limits, bitstream lengths etc.) must be thoroughly validated before acceptance.
- Editing or deleting puzzles is restricted to the creator, preventing unauthorized modifications.

1.3.4. Fair Rating and XP Systems

Because ratings influence puzzle visibility,

- Users may only rate puzzles after solving or attempting them for at least five minutes.
- Experienced users (Level 5 and above) have double-weighted influence, requiring the system to track and compute dual metrics.
- The XP system must be resistant to abuse (e.g., repeated solves cannot yield repeated bonus XP).

1.3.5. Resource Limits for Gameplay Balance

To maintain balanced progression,

1. User-saved circuits capacity scales with user level (5 slots at level ≤2, 10 at ≤4, 20 at ≤6, 35 at ≤8, 50 at >8).
2. Limits on Value, and greater limits enforce fair puzzle design and prevent degenerate solutions, making the puzzles more challenging .
3. Puzzle metadata is constrained: name ≤ 100 characters (must be unique), description ≤ 500 characters, instructions ≤ 5,000 characters. Board grid rows are limited to 9–20.

1.3.6. Accessibility and Ease of Use

Since users vary widely in background (from beginners to advanced students),

- The UI must remain intuitive, even when handling complex circuit structures.
- All interactions (drag-and-drop, wiring connections) must be visually clear and responsive.
- Puzzles should remain solvable using only standard browser capabilities; no special hardware or plugins are required.

1.3.7. Environment and Operational Dependencies

EscapeCircuit relies on:

- Stable browser performance for rendering the interactive board.
- Backend reliability to compute logic outputs quickly and handle XP and ratings updates.
- Network connectivity for saving progress, retrieving puzzles, and synchronizing user data.

1.3.8. Security and Integrity Considerations

Due to user-generated content:

- All inputs must be sanitized to avoid malicious puzzle definitions or injection attacks. Puzzle upload data is fully validated server-side before any files are created on disk (safe upload validation). Field length limits are enforced both client-side and server-side.
- User authentication must be enforced consistently.
- Google OAuth sign-in is supported as an alternative authentication method alongside username/password login.
- XP, ratings, medals, and level progression must be tamper-proof.

Chapter 2 – System Architecture

2.1 Architectural Overview

EscapeCircuit follows a classic client–server architecture:

- A Next.js 14 application (App Router) in the browser provides the WorkStation UI and all user-facing pages.
- A Python backend exposes REST APIs and contains all business logic, including puzzle management, solving, ratings, XP, and authorization.
- A logic evaluation engine (backend module) evaluates circuits using Boolean logic based on the existing prototype.
- A relational database (SQLite / SQL) persistently stores users, puzzles, circuits, ratings, XP, and related data.

2.2 Components and Deployment

Client (Browser):

- Runs a web interface served by the server.

  Implements:

  - Puzzle Browser – listing and filtering puzzles.
  - WorkStation – interactive BreadBoard with gates, wires, and debugger.
  - Profile & Arsenal – user progress, saved circuits, medals, etc.
  - Creator Dashboard – puzzle creation and management (for creators).
  - Admin Panel – user roles and puzzle moderation (for admins).

Backend Application Server (Python):

1. Hosted on a web server (e.g., free cloud instance of the university, with option to upgrade).

2. Exposes JSON/HTTPS endpoints for:

CRUD:
create / read /
update / delete

a. Authentication & roles (solver/creator/admin).
b. Puzzle CRUD operations and publishing.
c. Circuit saving and retrieval.
d. Solution submission and evaluation.
e. XP & medal updates and profile retrieval.
f. Rating operations.
g. Per-puzzle leaderboard retrieval.
h. Admin operations (assign/remove creator, delete puzzle).
i. Discussion forum operations (create, reply, vote, react, pin, lock, report).
j. Notification retrieval and management.

Files and data:

- Backend code, logic engine, and database files live on the server.

- No persistent application data is stoAddition:
- red on the client beyond session/cookies and in-memory UI state.

Logic Engine:

The Logic Engine is a backend component responsible for evaluating user-built circuits against a puzzle's test cases.

When a user builds a circuit on the BreadBoard in the browser and clicks "Check Solution", the frontend serializes the circuit into a JSON and sends it to the backend via an HTTP request (e.g., POST `/api/puzzles/{puzzleId}/check`).

On the server side, the SolvingService first validates the circuit (allowed gates, cost within budget, all inputs/outputs connected). If the circuit is structurally valid, the service invokes the Logic Engine:

evaluate(circuitJson, testCases) → { pass/fail, messageOutput }

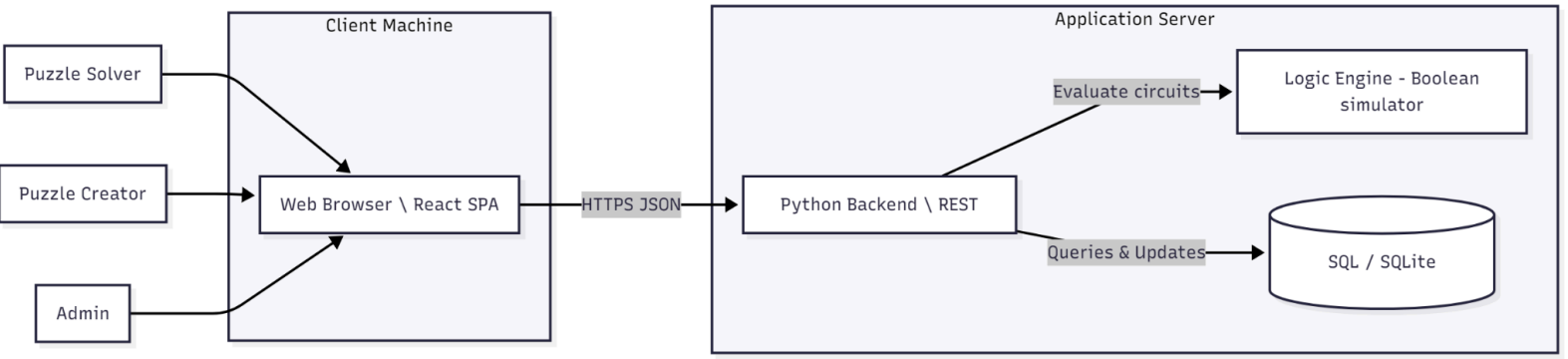
The Logic Engine runs the Boolean simulation for all of the puzzle's test cases and returns whether the circuit passes all of them (pass/fail), and optionally the detailed outputs.

Importantly, the Logic Engine is entirely server-side:
it doesn't run in the browser and is accessed only through the backend services.

Database

- Relational DB (SQLite or similar) for persistent storage of all domain entities: users, puzzles, test cases, circuits, ratings, XP events, solve attempts, and medals..

2.3 Container / Deployment Diagram



Chapter 3 – Data Model

This chapter describes the core information domain: the objects the system manages and their attributes.

3.1 Description of Data Objects

Puzzle:

PK = Primary Key
FK = Foreign Key

- puzzle Id (PK)
- creatorId (FK → User)
- name (unique per creator)
- description
- input Count (0–5)
- inputStreamLength (1–20)
- outputsCount (1–20)
- outputStreamLength (1–40)
- budget (1–999)
- tightBudget (≥ creator solution cost, < budget)
- time Limit Seconds (time for get medal)
- defaultGateSet (allowed basic gates/operations)
- specialCircuits (set of SpecialCircuit IDs)
- boardRows (default 15, grid height for the puzzle workspace)
- boardCols (default 30, grid width for the puzzle workspace)
- initialBoardJson (optional): JSON defining the puzzle's pre-placed locked components and wires that load onto every solver's board.
- difficultyCreatorRating
- rating aggregates: difficultyAvg, funAvg, clearnessAvg, plus experienced-only variants
- create Comment
- upload Datetime
- isPublished
- is Popular

Puzzle TestCase:

- testCaseId (PK)
- puzzle Id (FK → Puzzle)
- inputVector (bitstream or encoded vector)
- expectedOutputVector
- type ∈ {BlackBox, WhiteBox}

User:

- user Id (PK)
- username
- email
- password Hash
- role ∈ {Solver, Creator, Admin}
- xpTotal – accumulated XP
- level – derived from XP via thresholds
- createdAt
- Unique constraint: (puzzleId, userId) — each user may rate a puzzle only once.
- totalSavedCircuits
- is_discussion_banned
- whether user is banned from discussions max_published_puzzles
- admin override for puzzle capacity max_unpublished_puzzles
- admin override for puzzle capacity
- maxAmountPublishedPuzzle

Circuit (user-saved):

- circuitId (PK)
- userId (FK → User)
- name (unique per user)
- structureJson (truth table, gates that are used)
- cost (derived from basic gate counts and nested circuits)
- gatesCountsMap (per-gate type counts)

Rating:

- ratingId (PK)

- puzzleId (FK → Puzzle)
- userId (FK → User)
- difficulty
- fun
- clearness
- createdAt

Medal:

- userId (FK → User)
- puzzleId (FK → Puzzle)
- current Tier

SpecialCircuit:

- specialId (PK)
- puzzleId (FK → Puzzle)
- name
- inputsCount
- outputsCount
- truthTableJson
- cost

Discussion:

- discussionId (PK)
- authorId (FK → User)
- puzzleId (FK → Puzzle, optional)
- title
- body
- tags
- isPinned
- isLocked
- createdAt
- updatedAt

Reply:

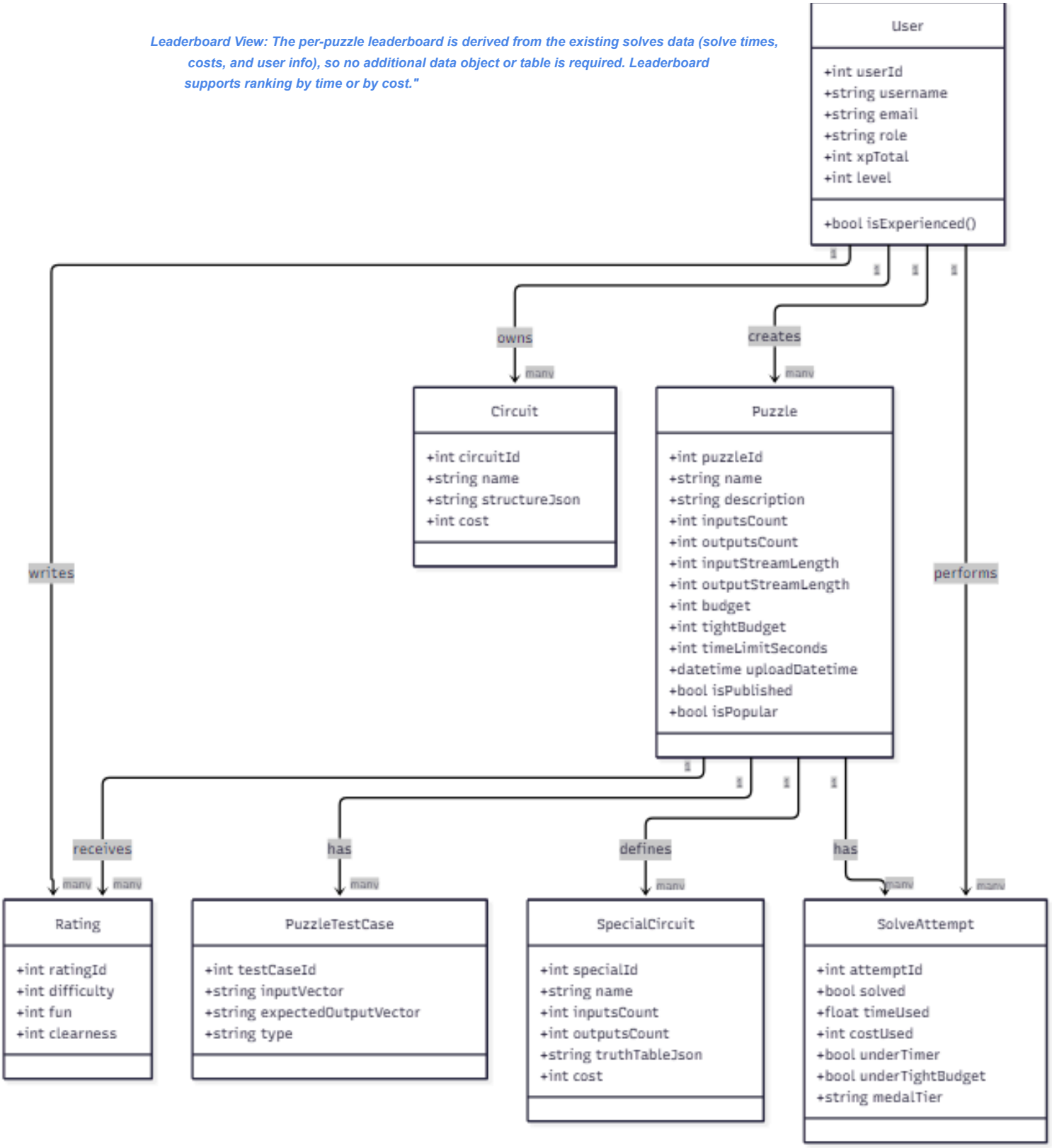
- replyId (PK)
- discussionId (FK → Discussion)
- authorId (FK → User)
- body
- votes (upvote/downvote count)
- isAccepted
- createdAt

Notification:

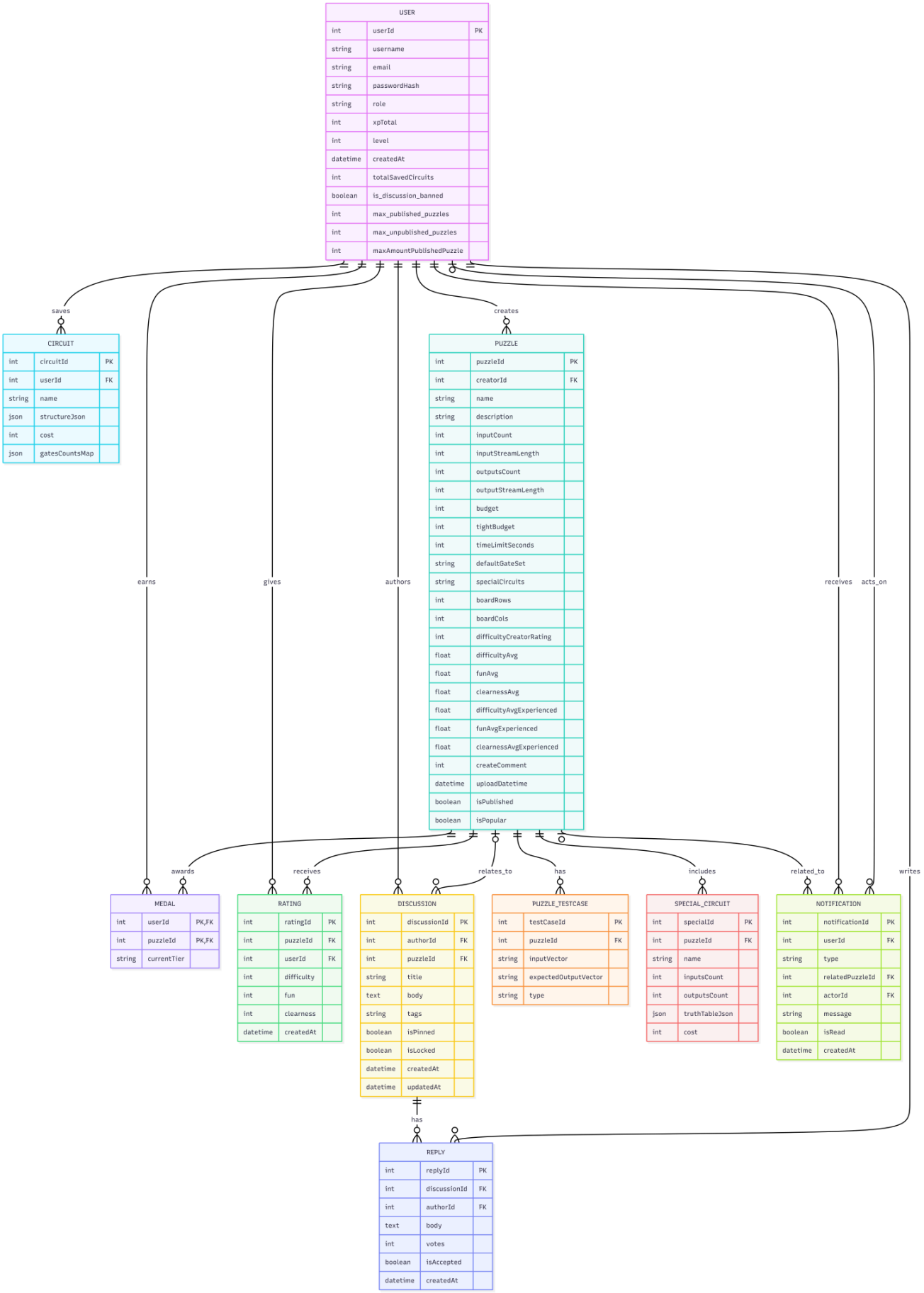
- notificationId (PK)
- userId (FK → User)
- type ∈ {solve, rating, creator_invite, role_change, puzzle_unpublished}
- relatedPuzzleId (FK → Puzzle, optional)
- actorId (FK → User, optional)
- message
- isRead
- createdAt

3.2 Data Objects Relationships

Leaderboard View: The per-puzzle leaderboard is derived from the existing solves data (solve times, costs, and user info), so no additional data object or table is required. Leaderboard supports ranking by time or by cost."



3.3 Databases



Typical DB transactions:

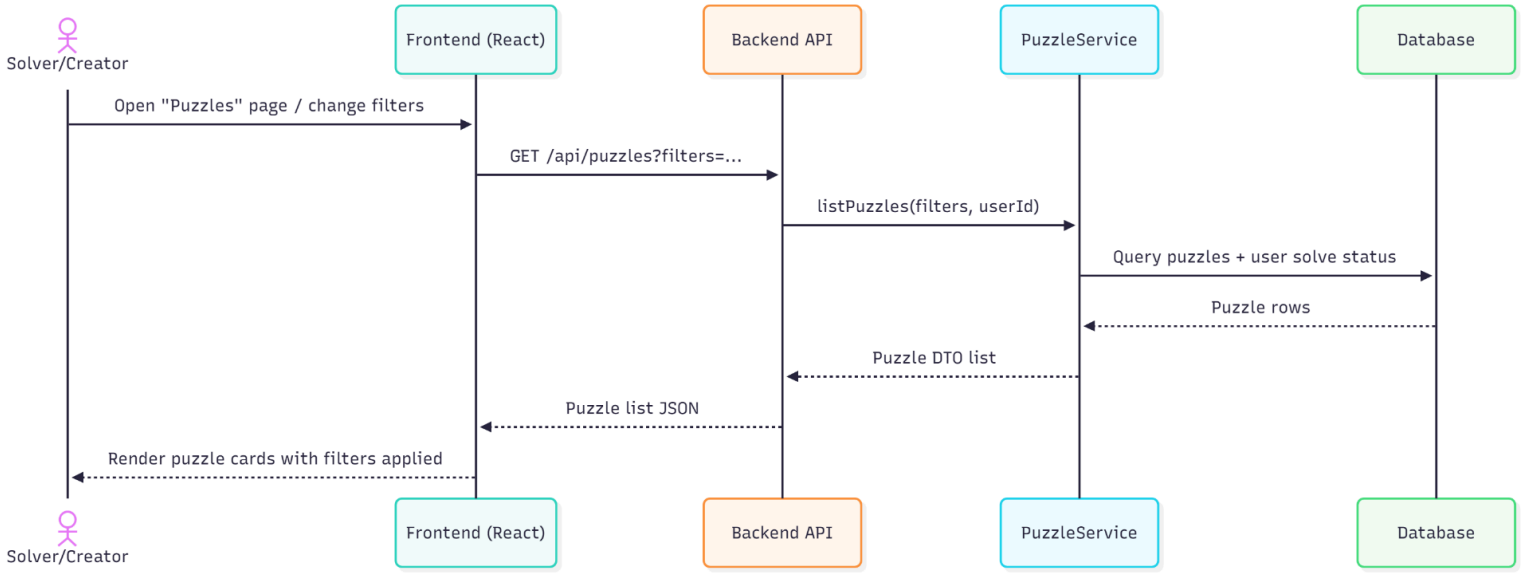
- Insert/update/delete for puzzles and their test cases (creation and publishing).
- Inserting circuits into the arsenal with unique-name and capacity constraints.
- Logging solves attempts and XP events when solutions are submitted.
- Inserting/updating ratings and recalculating aggregates in the puzzle row.
- Querying per-puzzle leaderboard rankings (derived from solve-time and cost records, no dedicated table).
- Insert/update/delete for discussions and replies (forum operations).
- Recording votes and reactions on discussion replies.
- Inserting notifications on system events and marking as read.
- Managing discussion reports and moderation status.

Chapter 4 – Behavioral Analysis

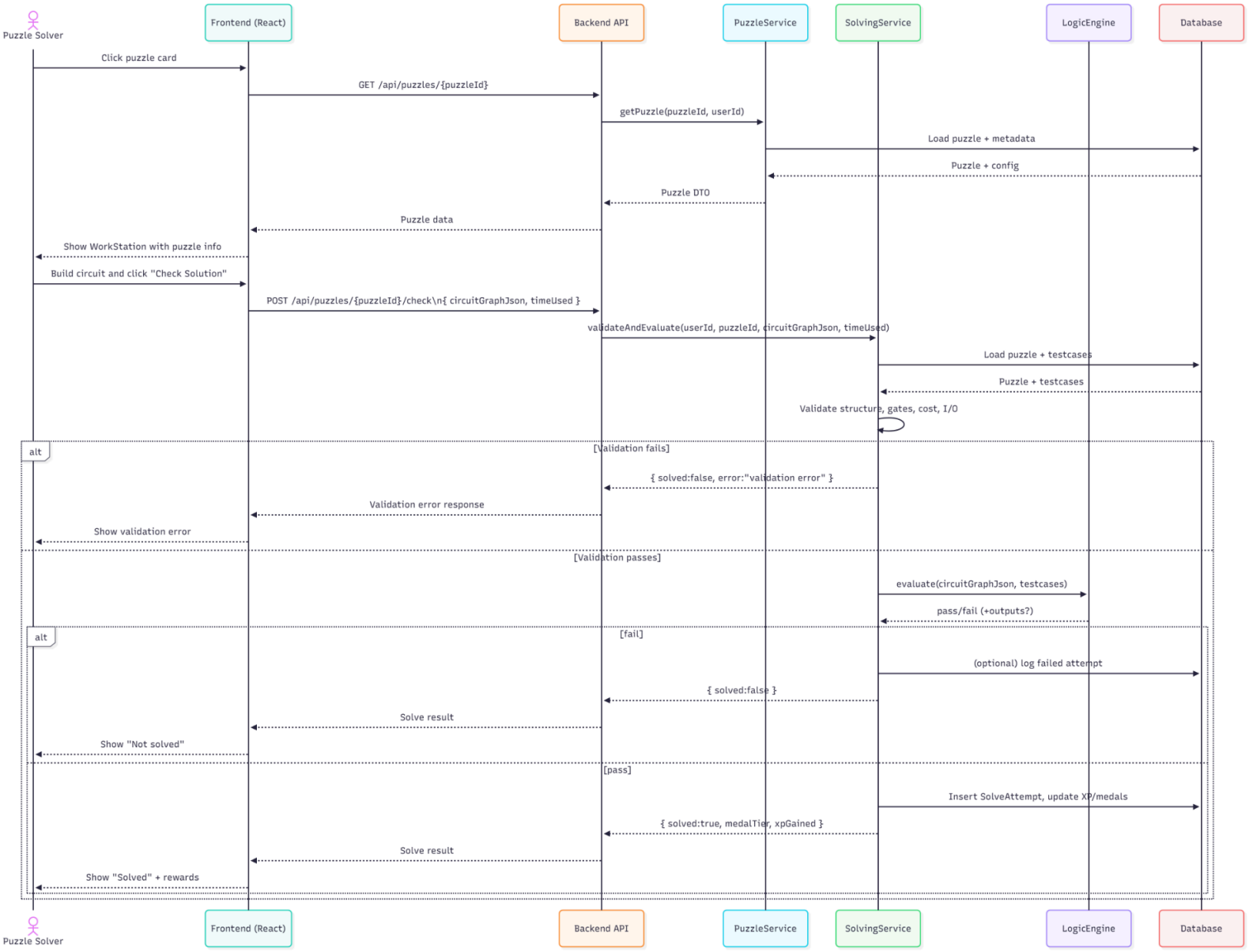
4.1 Sequence Diagrams

We provide sequence diagrams for three core flows: solving a puzzle, creating and publishing a puzzle, and saving a circuit.

4.1.1 UC1 – Browse and Search Puzzles

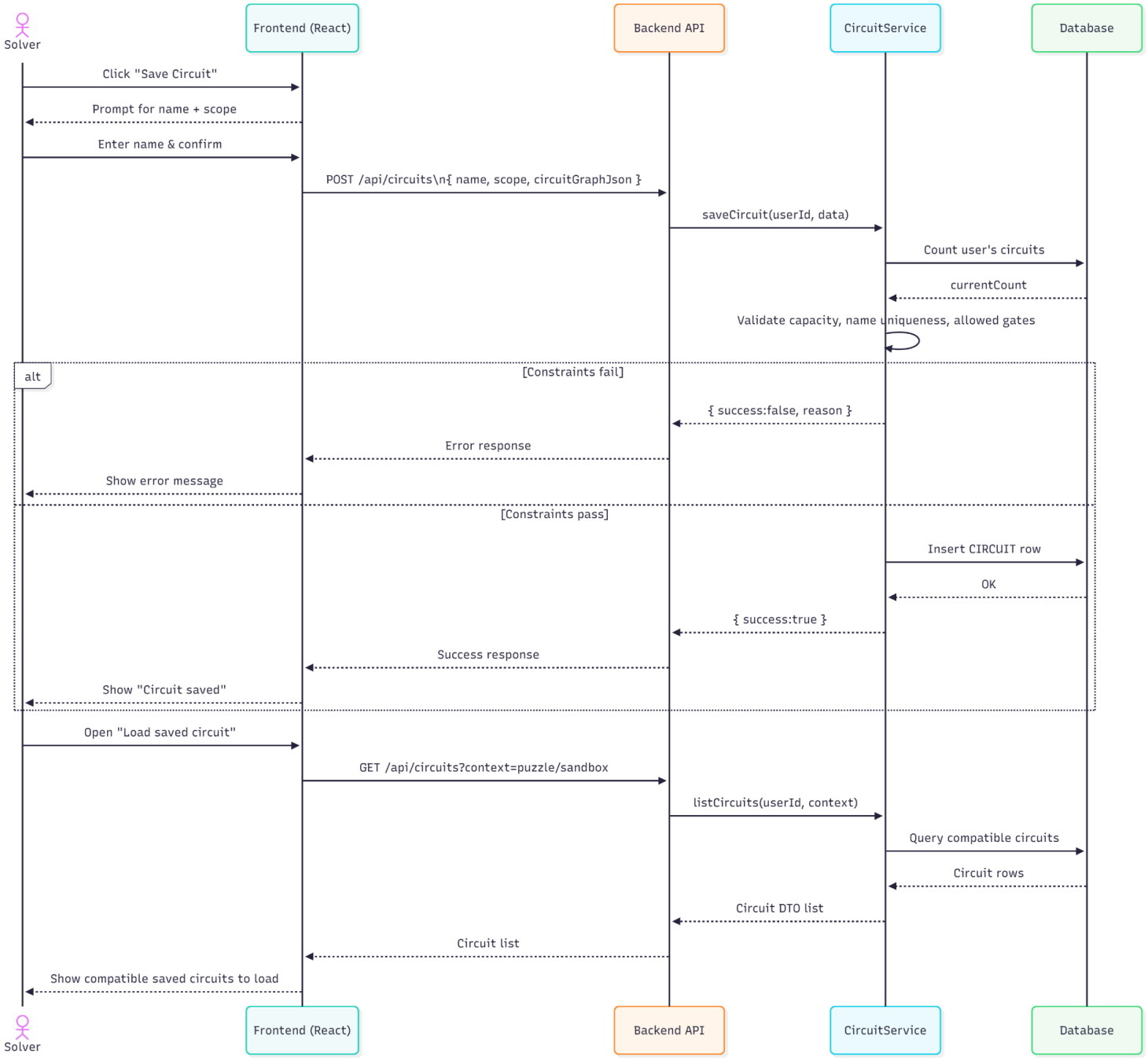


4.1.2 UC2 – Solve a Puzzle



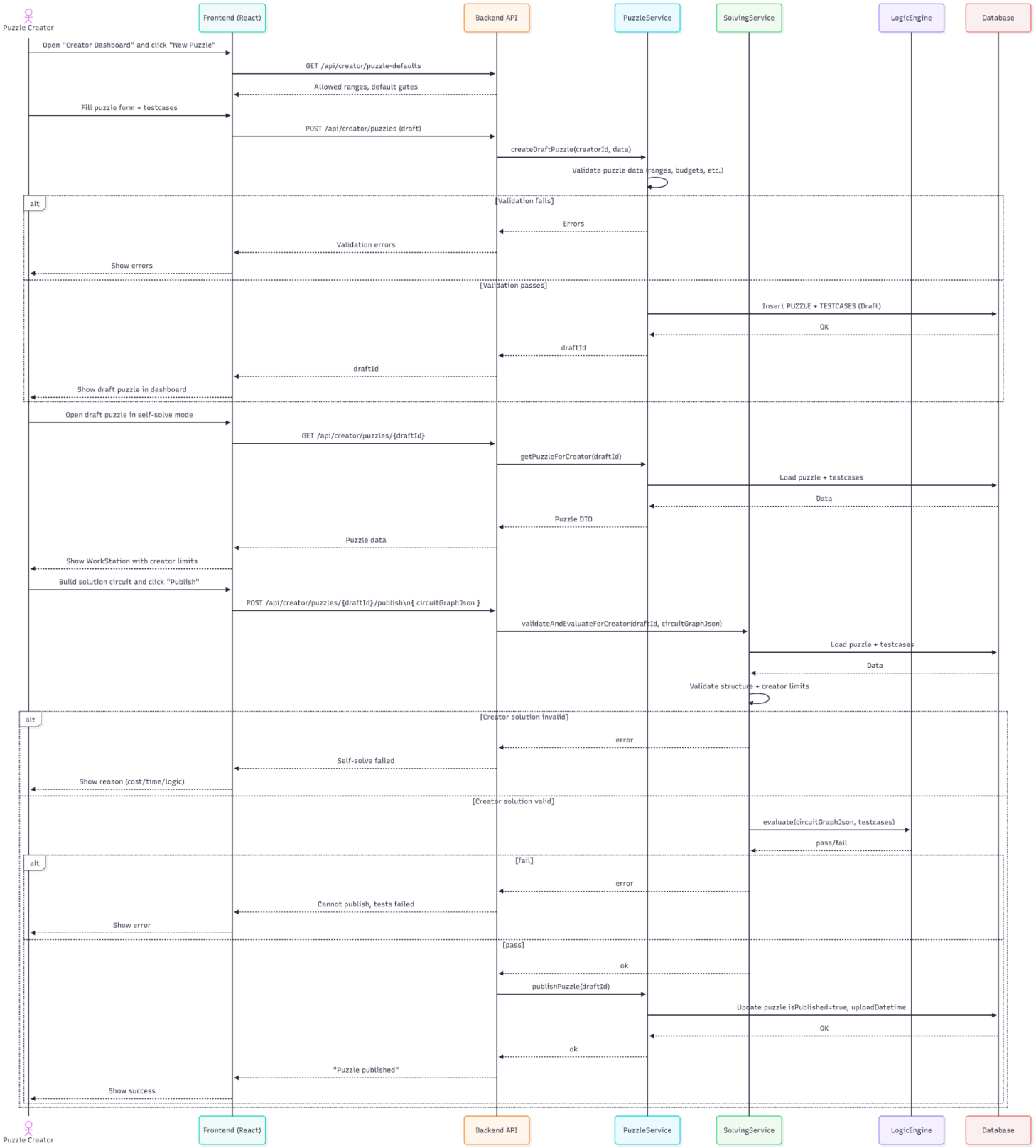
After step 7 (results displayed), the system records the solve time and cost, and updates the per-puzzle leaderboard. The frontend fetches GET time or cost to display the top solvers podium (togglable between time and cost ranking).

4.1.3 UC3 - Save and Reuse Circuits

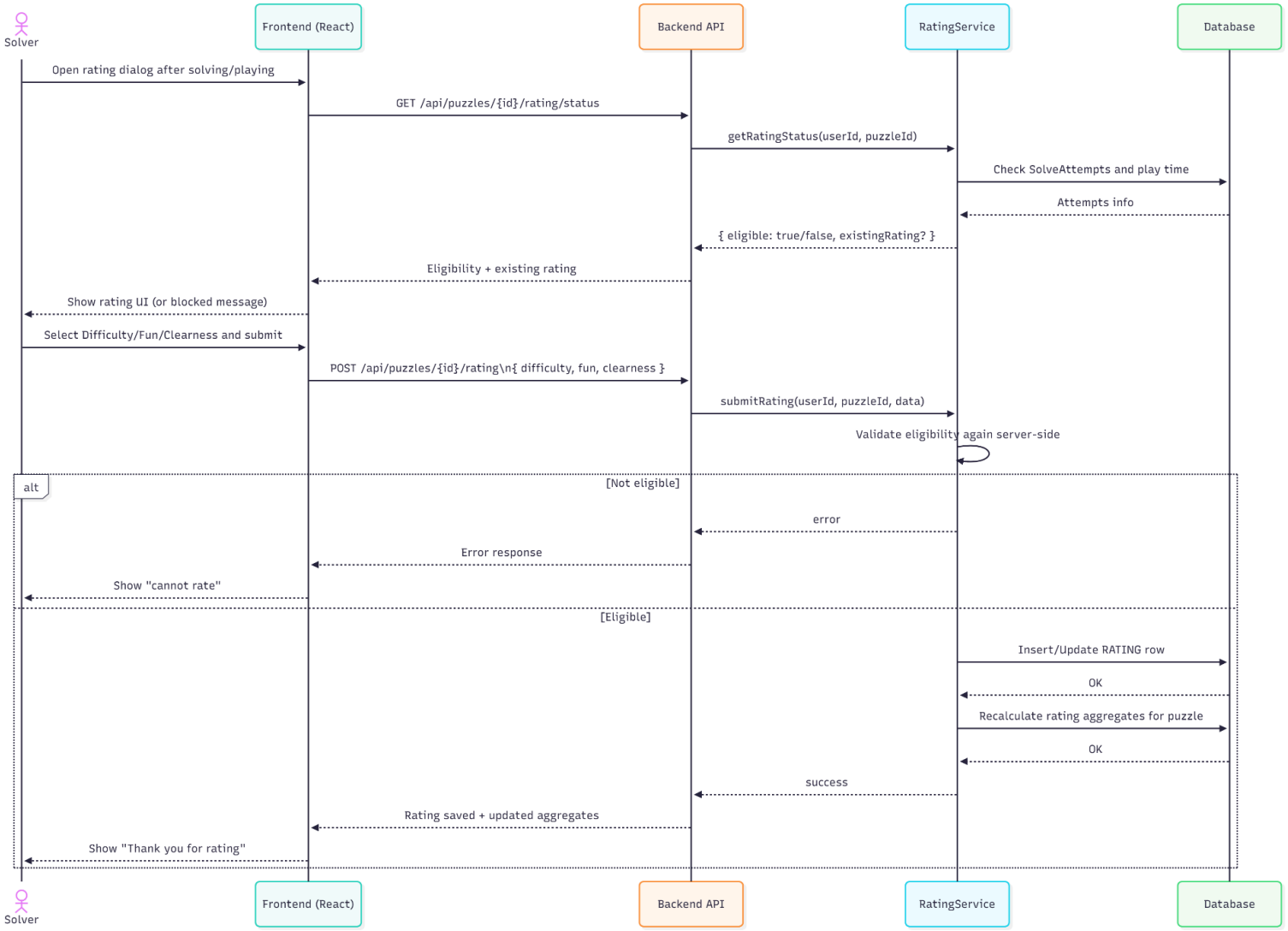


4.1.4 UC4 – Create and Publish Puzzle

Addition: Before self-solve (step 5), the system validates puzzle metadata: name uniqueness check, field length limits (name ≤ 100, description ≤ 500, instructions ≤ 5,000), board size validation (9–20 rows), and safe upload validation before any files are created.

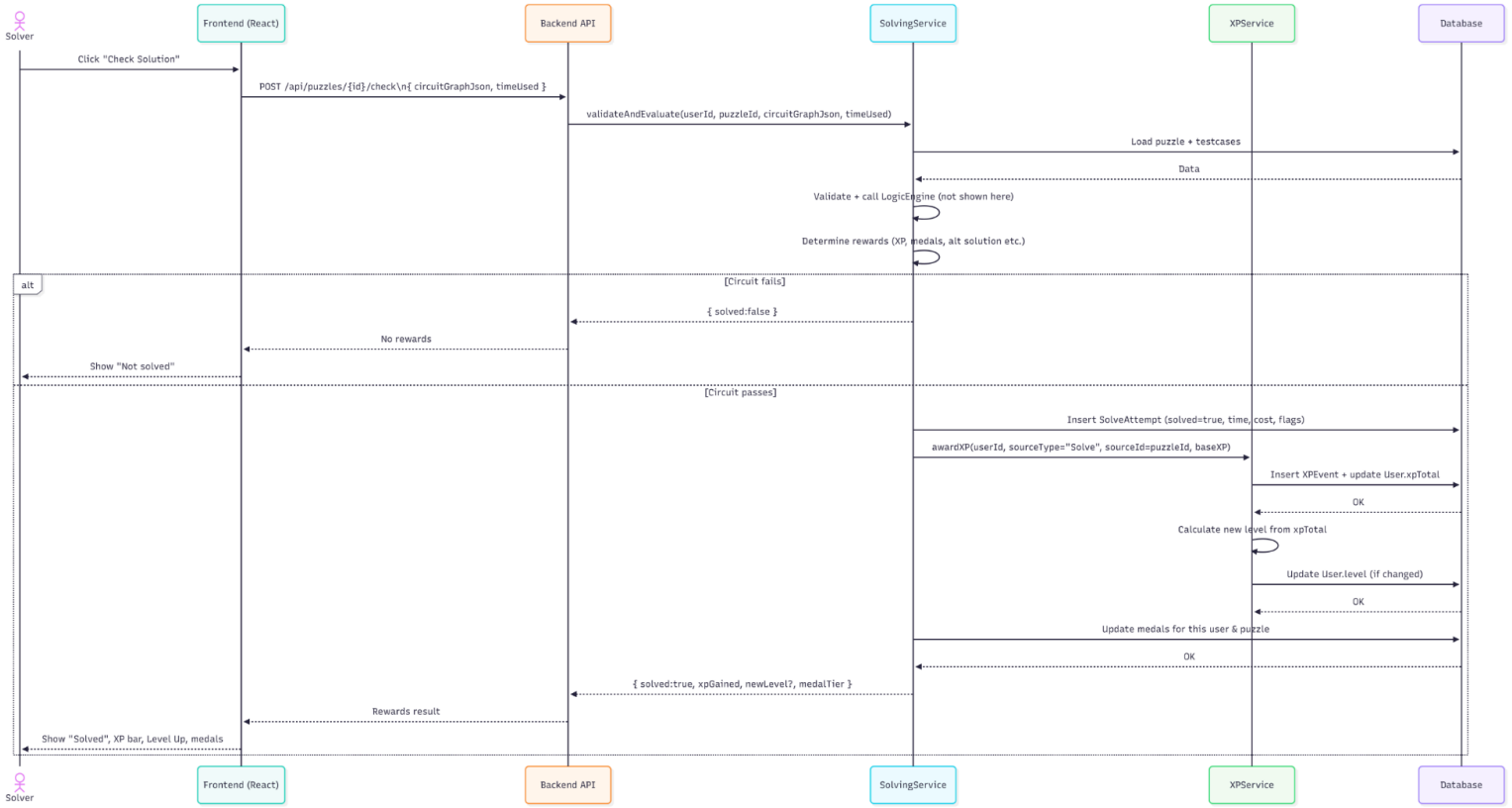


4.1.5 UC5 – Rate a Puzzle



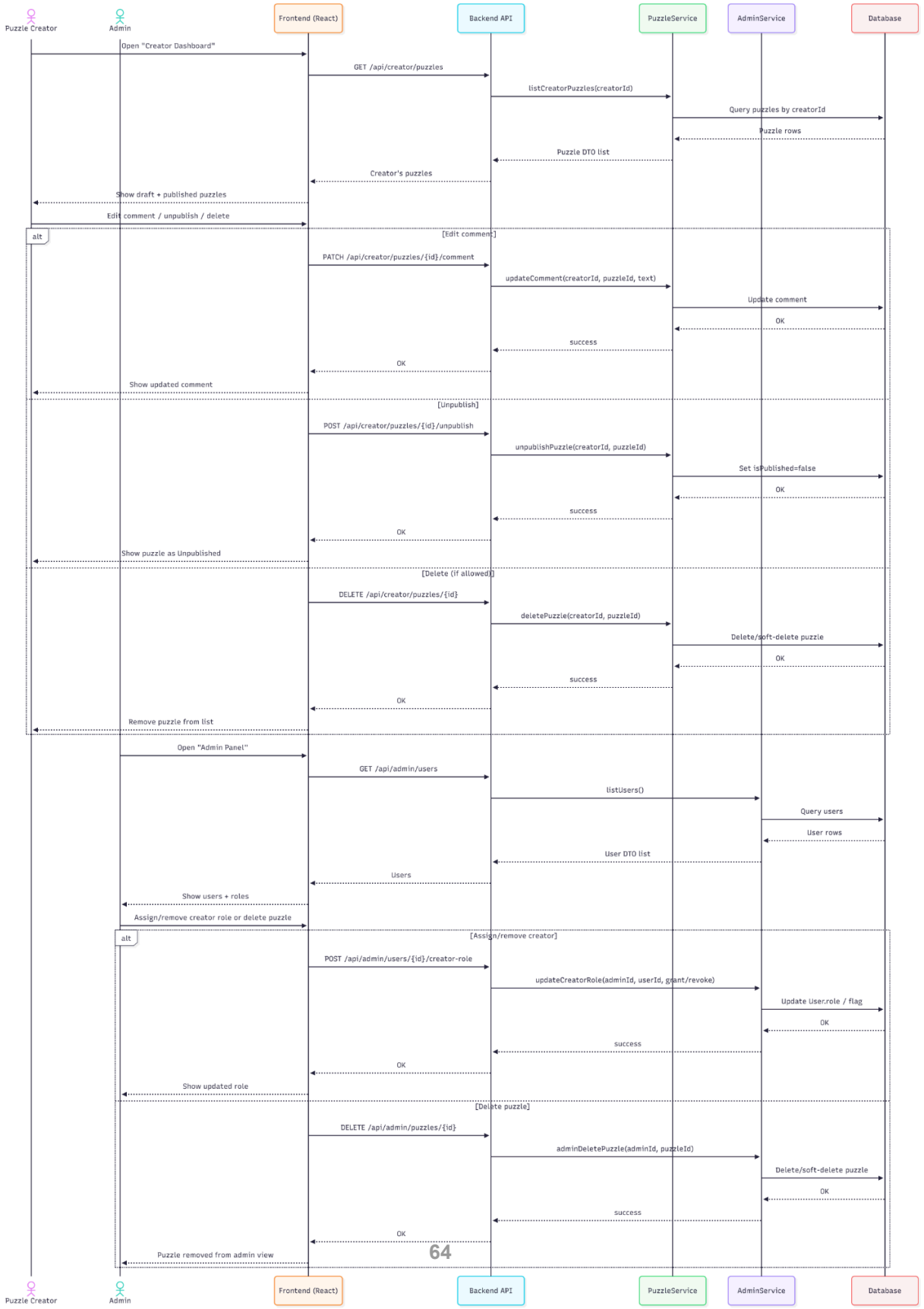
Addition: Before storing a new rating, the system checks for an existing rating by the same user on the same puzzle. If found, the user is directed to use the edit flow instead (meaning it is replacing the existing rating).

4.1.6 UC6 – Earn Rewards and Level Up

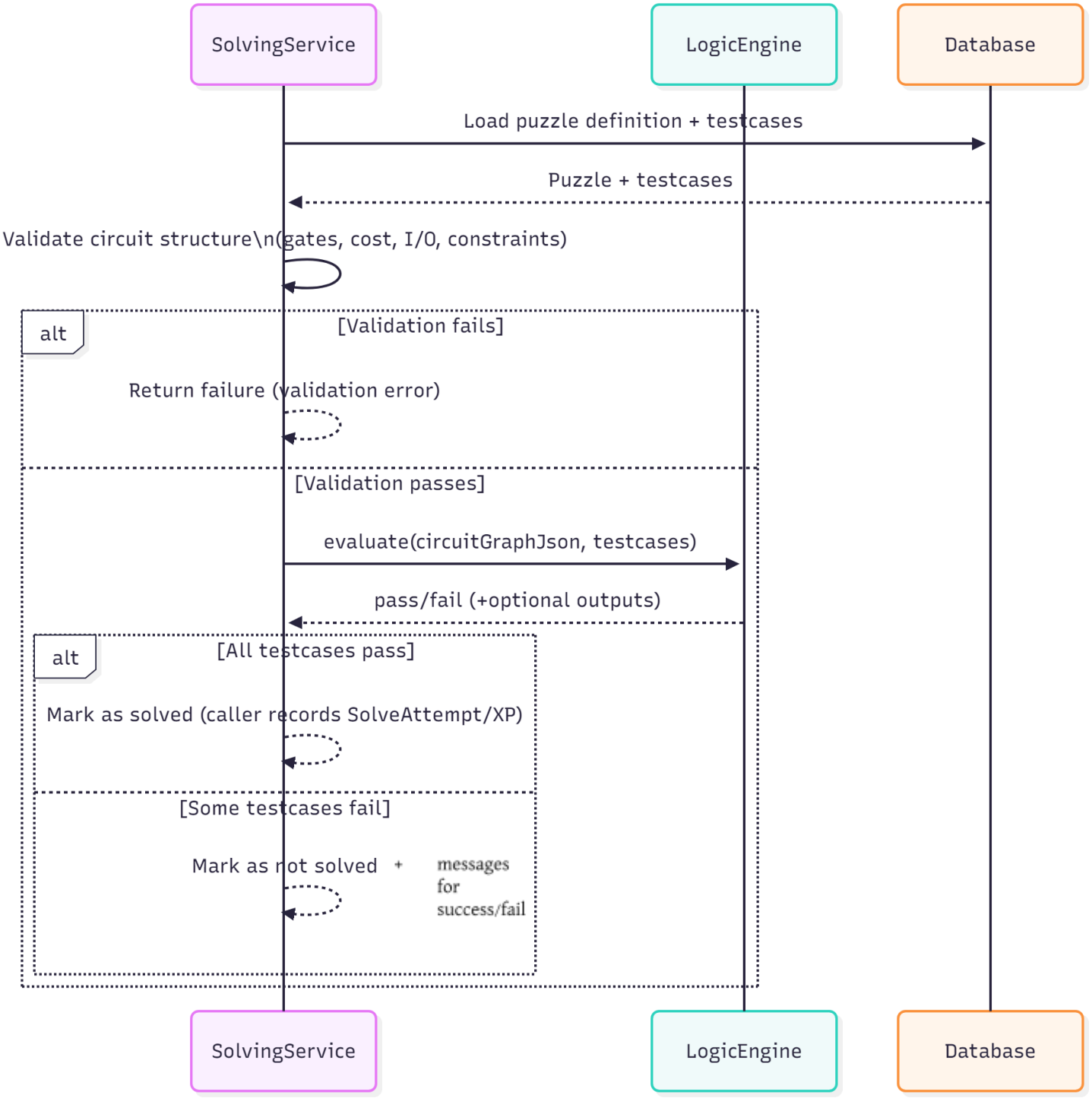


Medal assignment now handles boundary edge cases: cost exactly equal to the greater value limit is treated as within limits. After rewards are assigned,the user appears on the per-puzzle leaderboard ranked by solve time or cost (based on the selected view).

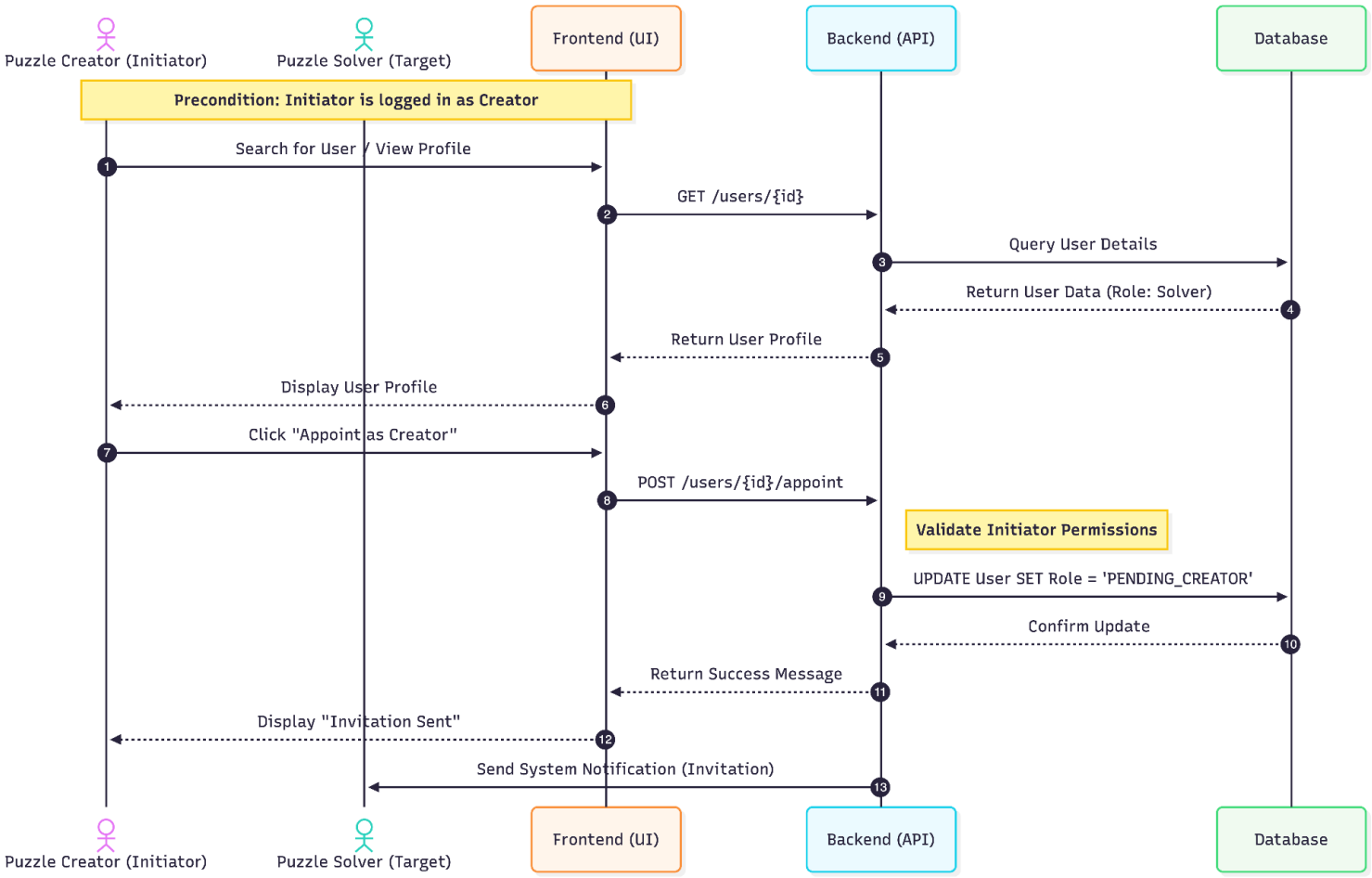
4.1.7 UC7 – Manage Puzzles (Creator/Admin)



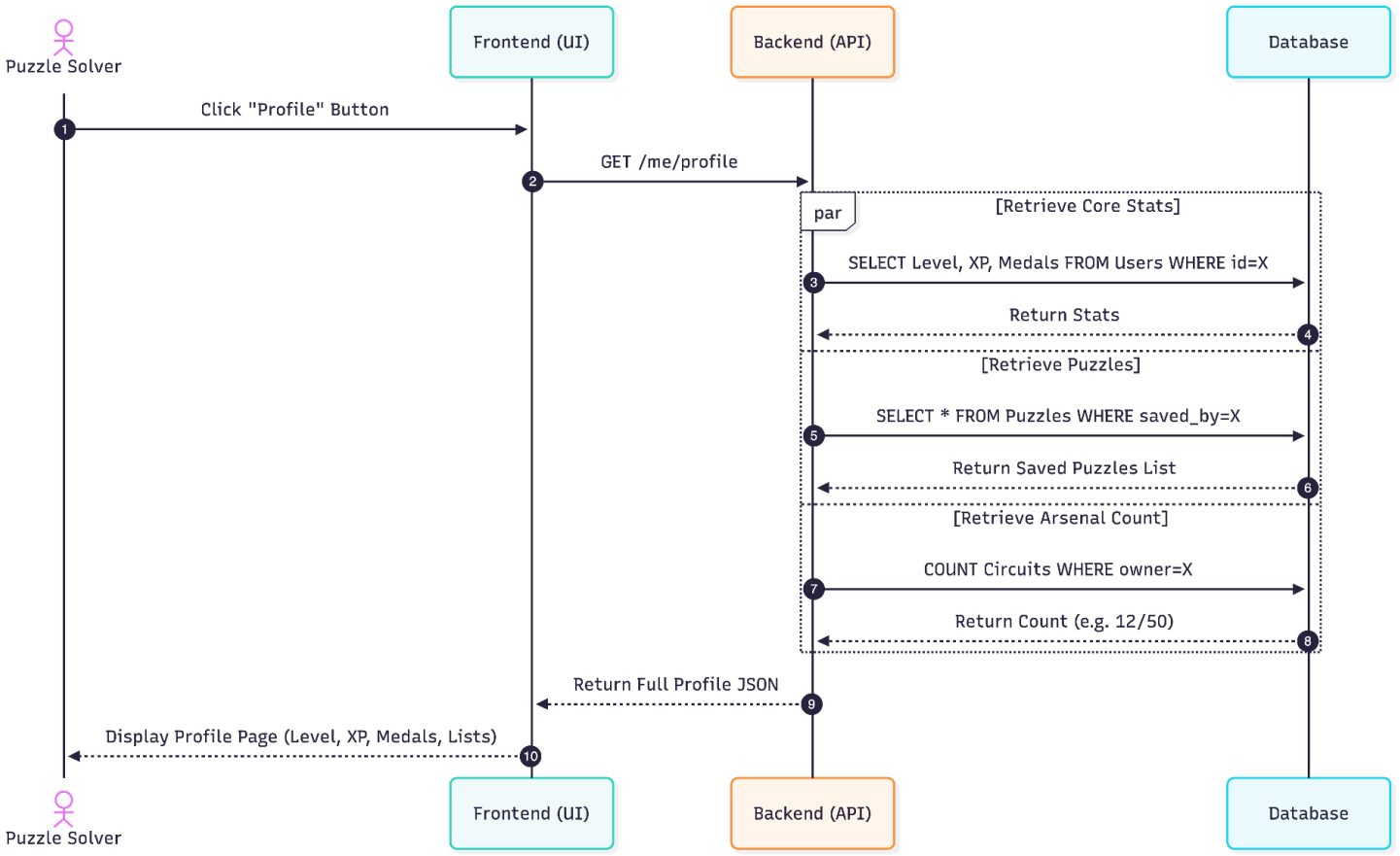
4.1.8 UC8 – Puzzle solution Validation



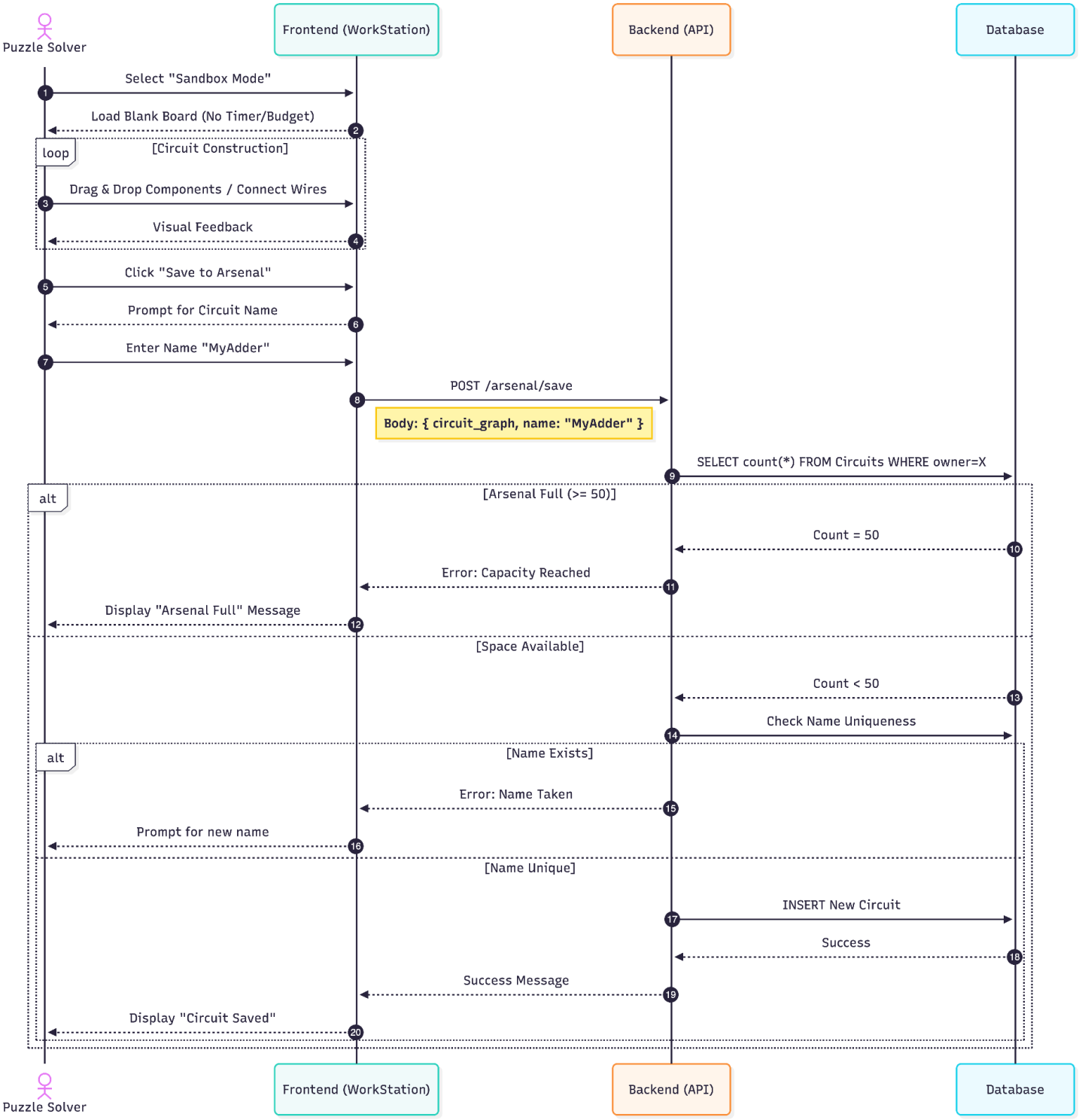
4.1.9. UC9 - Appoint User as Creator



4.1.10. UC10 - View Personal Profile & Progress



4.1.11. UC11 - Experiment in Sandbox



4.1.12. UC12 - Manage Personal Arsenal

4.2 Events

EscapeCircuit is naturally event-driven from the user's point of view. Key events:

UserLogin(username, password):

Triggers authentication, session creation, and loading of profile data.

PuzzleSelected(puzzleId):

Triggers loading puzzle definition, allowed gates, budgets, and test cases into the WorkStation.

CheckSolutionClicked:

Triggers validation and circuit evaluation flow described in UC2.

TimerTick:

Every second, the timer is updated in the UI; color changes depending on remaining time (green → yellow → red, then stopwatch).

TimerExpired:

When the timer reaches zero, the WorkStation switches to stopwatch mode with a "+" prefix, and the puzzle is no longer eligible for a Timer Medal.

SaveCircuitClicked:

Triggers arsenal saving logic.

PublishPuzzleClicked:

Triggers server-side metadata validation (name uniqueness, field length limits, board size), then creator self-solve and publishing checks.

RatePuzzleSubmitted:

Triggers duplicate-rating check (one rating per user per puzzle), rating validation (must have solved or played for ≥5 minutes), and recalculation of aggregates.

DiscussionCreated(discussionId, authorId):

Triggers creation of a new discussion thread and optional puzzle association.

ReplyPosted(replyId, discussionId, authorId):

Triggers notification to discussion participants.

NotificationGenerated(userId, type, relatedData):

Triggers insertion of a new notification record for the target user.

SoundEffectTriggered(effectType):

Triggers playback of the appropriate audio (drop, wire, error, success) based on user volume/mute settings.

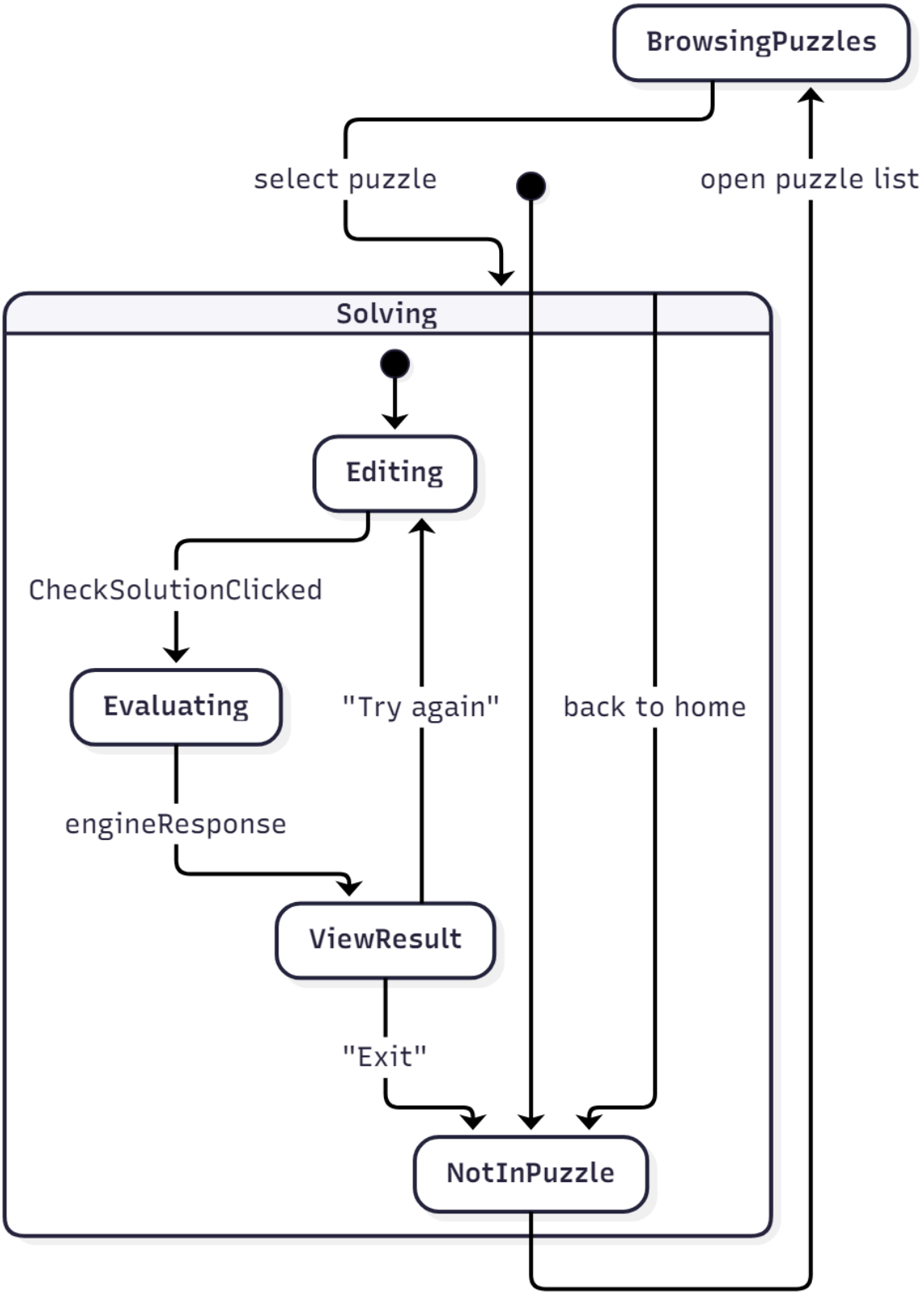
AdminAssignCreator(userId):

Triggers role update and access to creator-only APIs

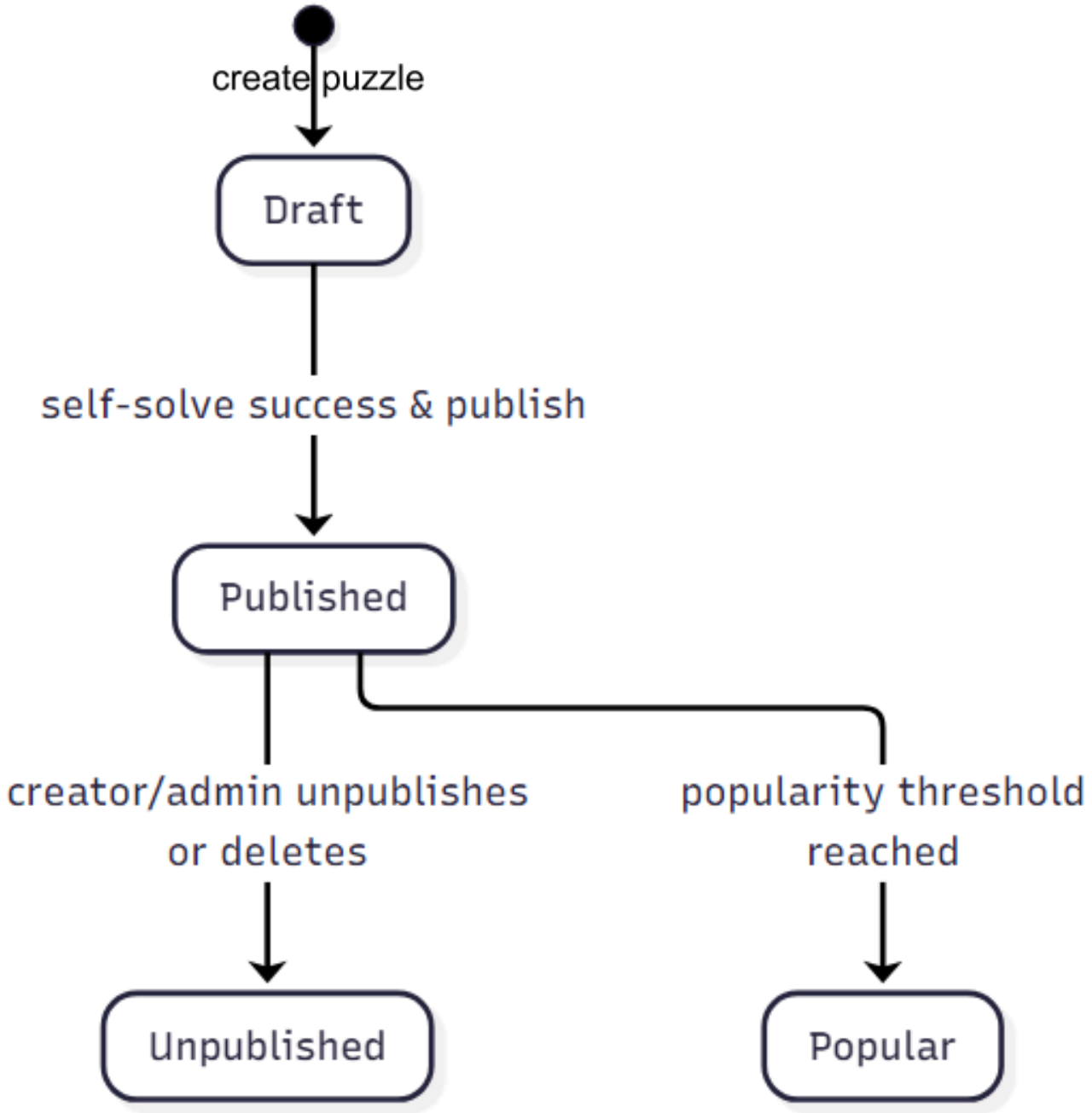
4.3 States

Let’s describe several subsystems as state machines.

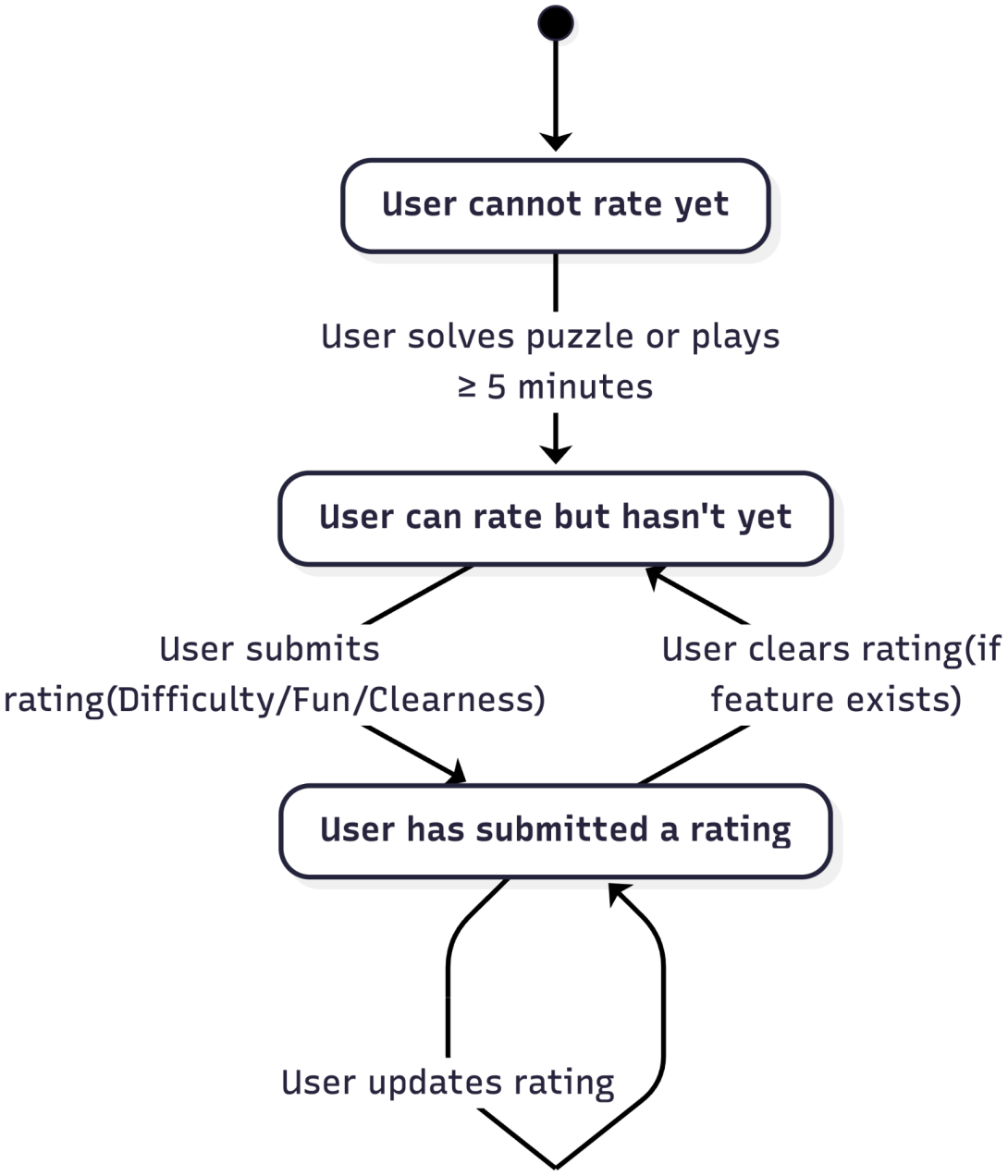
4.3.1 Puzzle Solving State Machine



4.3.2 Puzzle Lifecycle - creation of a puzzle



4.3.3 Rating Puzzle



Chapter 5 – Object-Oriented Analysis

We combine the data model and behaviors into classes, services, and packages.

5.1 Class Diagrams

Backend (service-oriented) class diagram:

too big to fit. its in this link:

https://drive.google.com/file/d/12NdltDhg5mvZ9TGaSEl434eERipyhsk9/view?usp=sharing

5.2 Class Descriptions

PuzzleService:

- Responsibilities
  - Manage the lifecycle of puzzles: creation, saving, publishing, retrieval, and unpublishing.
  - Enforce puzzle constraints (input/output limits, budgets, time limit, number of special circuits, etc.).

- Key Methods (examples)
  - `createDraftPuzzle(creatorId, data)`
    - *Preconditions: user is authenticated; puzzle exists in Draft; metadata passes validation (name ≤ 100 chars, description ≤ 500, instructions ≤ 5,000, boardRows ∈ [9, 20]); name is unique; creator has successfully self-solved it; tightBudget and budget are valid.*
    - *Postconditions:* a new puzzle in Draft state is inserted with the provided content; associated test cases are stored.
  - `publishPuzzle(puzzleId)`
    - *Preconditions:* user is authenticated; puzzle exists in Draft; creator has successfully self-solved it; name is unique; tightBudget and budget are valid.
      ARD
    - *Postconditions:* puzzle is marked `isPublished = true`; uploadDatetime is set; puzzle enters public listings.
- Invariants
  - `inputsCount ∈ [0, 5]`, `outputsCount ∈ [1, 20]`.
  - `tightBudget >= creatorSolutionCost` and `tightBudget < budget`.

- name.length ≤ 100, description.length ≤ 500, instructions.length ≤ 5,000.
- boardRows defaults to 15; boardCols defaults to 30.

SolvingService:

- Responsibilities
  - Validate circuits against puzzle constraints (cost, allowed components, connection of all inputs/outputs).
  - Invoke LogicEngine and interpret the results.
  - Log SolveAttempts and trigger XP/medal logic.

- Key Methods
  - `getLeaderboard(puzzleId, type="time")`
    - Preconditions: puzzle exists and is published.
    - Postconditions: returns a ranked list of solvers ordered by solve time or cost (ascending, based on the type parameter: "time" or "cost"), derived from existing SolveAttempt records.

  - `validateAndEvaluate(userId, puzzleId, circuitGraph, timeUsed)`

    - *Preconditions:* user is authenticated; puzzle exists; circuit graph is structurally valid (no cycles if forbidden, no disconnected outputs, only allowed gates).

    - *Postconditions:* returns an evaluation result (solved flag, medal tier change, XP gained); if solved and constraints are satisfied, a SolveAttempt row is inserted and XP is awarded.

  * Invariants

  - No SolveAttempt is persisted where `costUsed > budget`.
  - Timer medal is only awarded if `timeUsed ≤ timeLimitSeconds`.

XPService:

- Responsibilities
  - Centralize XP and level progression logic.
  - Ensure levels are always derived from XP thresholds.

- Key Methods
  - `awardXP(userId, sourceType, sourceId, baseXP)`
    - *Preconditions:* user is authenticated;
    - *Postconditions:* new XPEvent inserted; user's xpTotal increased; user's level recalculated and updated if threshold crossed.
  - `calculateLevel(xpTotal)`
    - *Preconditions:* user is authenticated;
    - *Postconditions:* deterministic mapping from XP to level; monotone non-decreasing.

RatingService

- Responsibilities
  - Enforce rating eligibility rules (user must solve or attempt 5 minutes).
  - Record ratings and recalculate aggregates, including experienced-only and creator-weighted difficulty.

- Key Methods
  - `submitRating(userId, puzzleId, ratingData)`
    - *Preconditions: user is authenticated; puzzle exists; no existing rating by this user for this puzzle (duplicates are rejected — user must use the edit flow instead); user solved the puzzle or played at least 5 minutes; rating values within valid range.*
    - *Postconditions:* rating inserted or updated; puzzle's aggregated rating fields recomputed.

- Invariants
  - Creator difficulty weighting ≥50% before 10 ratings, reduced later (as specified in ARD).

5.3 Packages

Backend (Python):

- src/Backend/
  - APILayer/ # HTTP controllers
    - PuzzleController.py
    - CircuitController.py
    - UserController.py
    - AdminController.py
    - DiscussionController.py
    - RatingController.py
    - ArsenalController.py
    - auth_utils.py
  - ServiceLayer/ # Business logic
    - PuzzleService.py
    - SolvingService.py
    - XPService.py
    - RatingService.py
    - UserService.py
    - AdminService.py
    - DiscussionService.py
    - ReplyService.py
    - NotificationService.py
    - logicEngineService.py
  - DomainLayer/ # Core domain models
    - Puzzle.py
    - User.py
    - Circuit.py
    - Rating.py
    - SolveAttempt.py
    - Discussion.py
    - Reply.py
    - Enums.py
    - Utils.py
  - PersistantLayer/ # ORM / repositories
    - PuzzleRepo.py
    - UserRepo.py
    - CircuitRepo.py

        - SolveRepo.py
        - RatingRepo.py
        - DiscussionRepo.py
        - ReplyRepo.py
        - NotificationRepo.py
        - _db.py
        - main.py
        - settings.py"
    - engine/ # LogicEngine wrapper
        - evaluator.py
    - config/
        - settings.py

Frontend (React):

- apps/nextjs-app/
    - src/
        - app/ : Next.js App Router pages (puzzles/, profile/, discussions/, arsenal/, auth/, create-puzzle/)
        - features/ : Feature modules (puzzles/, discussions/, ratings/, auth/)
        - components/ : Reusable UI components (InfoPopup, etc.)
        - hooks/ : Custom React hooks (useAudio, etc.)
        - lib/ : Utilities (api-client, auth, react-query config)
        - types/ : TypeScript type definitions
        - context/ : Settings context (sound/volume), auth context
        - utils/ : Helper functions

5.4 Unit Testing

Unit tests will focus on invariants, preconditions, and postconditions rather than specific internal implementation.

Examples:

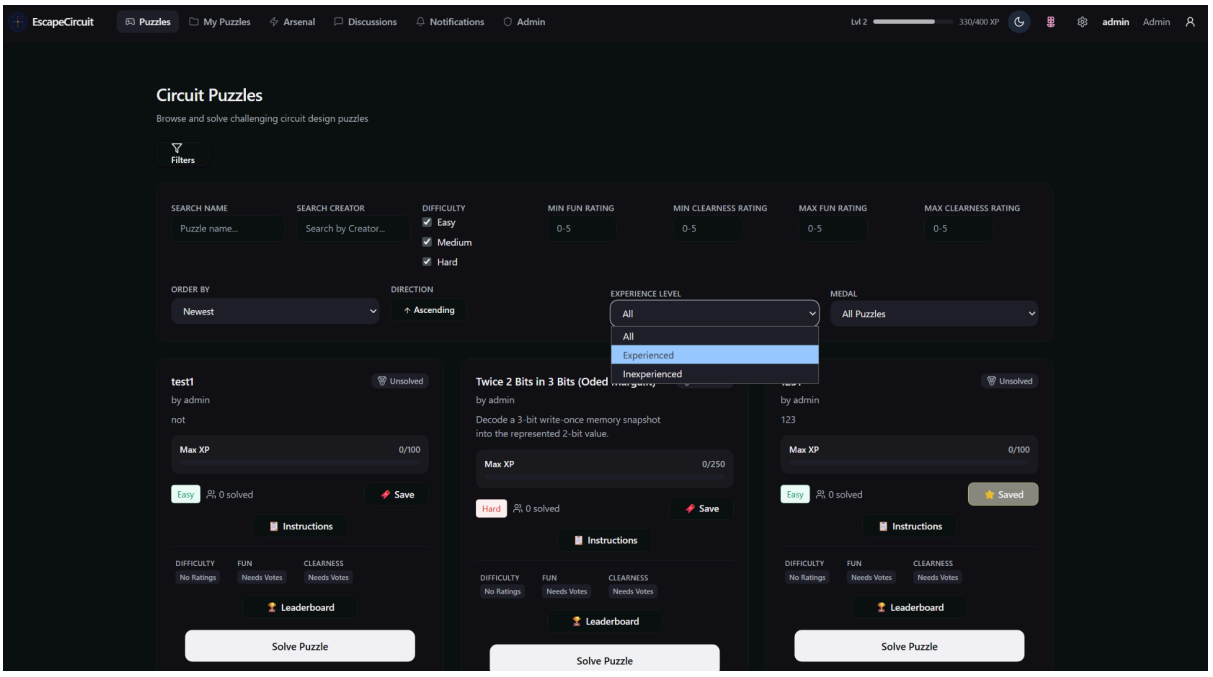
- PuzzleService tests
  - Creating puzzles outside allowed input/output ranges should fail.
  - Setting `tightBudget` ≥ `budget` or `< creatorSolutionCost` should be rejected.
  - Name exceeding 100 characters, description exceeding 500, or instructions exceeding 5,000 should be rejected.
  - Board size outside 9–20 rows should be rejected.
- SolvingService tests
  - Circuits that don't connect all required inputs/outputs must be rejected.
  - Circuits exceeding budget must fail validation.
  - Known correct circuits for a given puzzle must always return `solved=true`.
- XPService tests
  - Level increases when XP crosses exact thresholds and never decreases.
- RatingService tests
  - Rating without solve or 5-minute attempt ⇒ rejected.
  - Duplicate rating submission by the same user on the same puzzle ⇒ rejected.
  - Weighted difficulty calculation respects creator and experienced-user rules.
- Medal & Leaderboard tests
  - Cost exactly equal to the greater-value limit is treated as within limits (boundary medal awarded).
  - Leaderboard returns solvers ranked by ascending solve time or ascending cost, based on the requested ranking type.

All core logic (engine integration, cost calculation, XP, rating) contributes toward the ≥80% test coverage requirement.

Chapter 6 – User Interface Draft

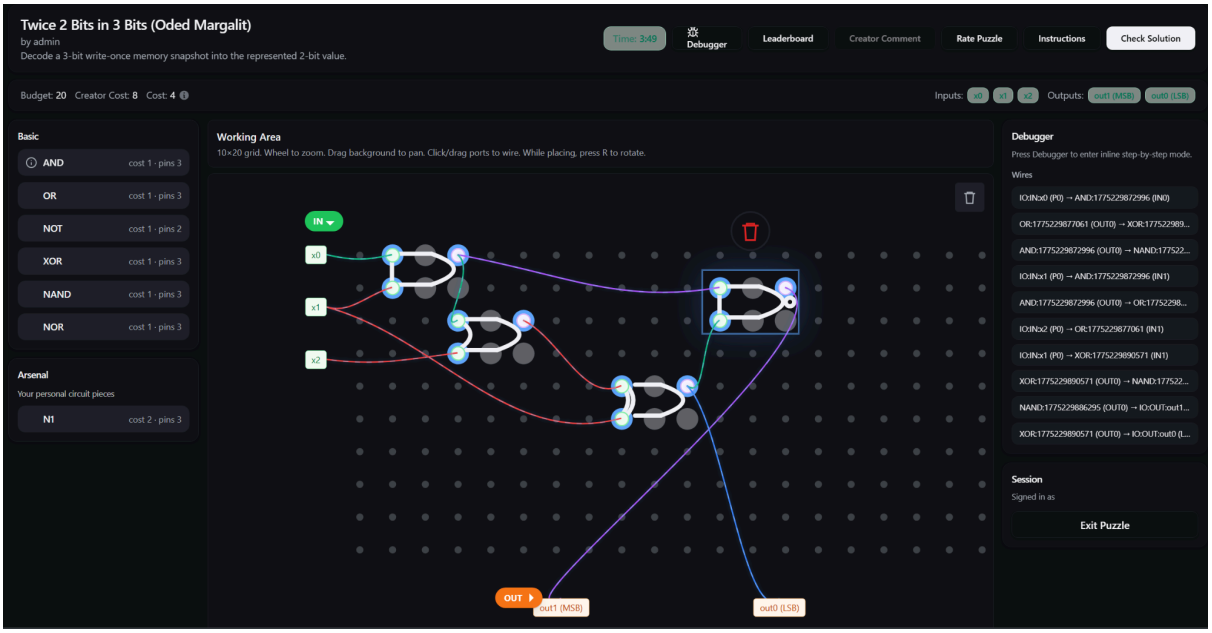
6.1 Home / Puzzle Browser

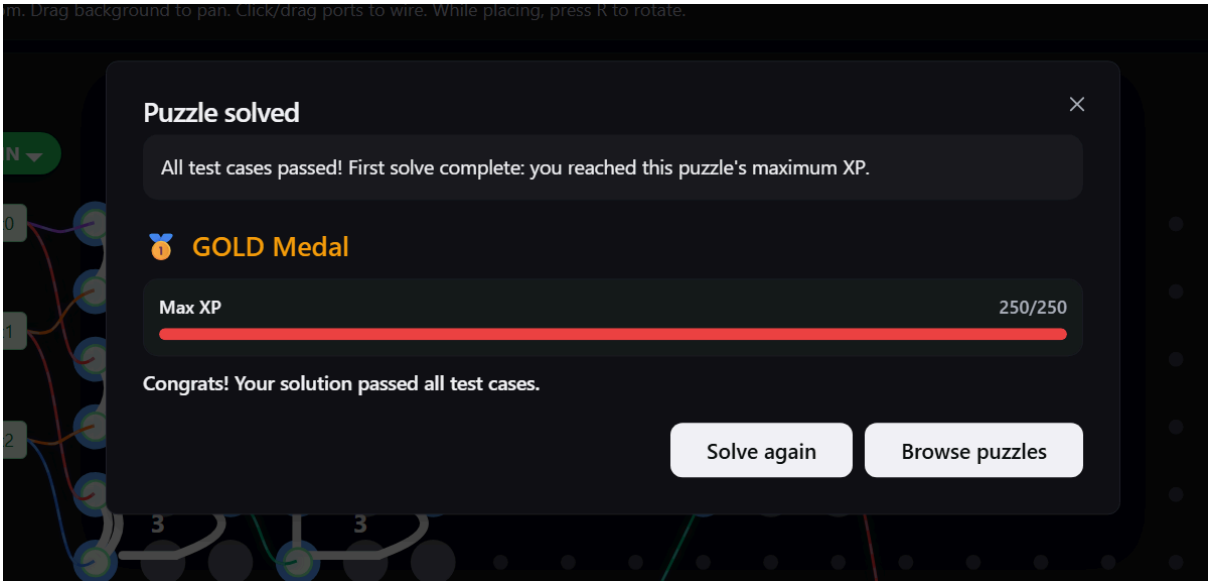
- Inputs
  - Search text
  - Filters (difficulty, time limit, solved/unsolved, rating, popular)
- Outputs
  - List of puzzle cards with:
    - Puzzle name, creator name
    - Difficulty rating (with experienced overlay)
    - Time limit and medals icons
    - Whether the puzzle is solved or not for this user
    - Leaderboard button (🏆) on each puzzle card to open leaderboard modal
    - Leaderboard modal with toggle between Fastest Time and Lowest Cost views



6.2 WorkStation (Puzzle Solving Screen)

- Inputs
    - Gate selection from:
        - Default gates/operations
        - Special circuits for this puzzle
        - Saved circuits (from the user's arsenal, if allowed)
    - Mouse/touch actions: drag gate, rotate gate, place gate, draw wire between pins, delete components.
    - "Check Solution" button
- Outputs
    - Visual BreadBoard with placed gates and wires
    - Timer display with color changes and stopwatch behavior
    - Current cost, budget, tight budget, and usage indicators (always visible). each accompanied by an InfoPopup (ⓘ) icon explaining the mechanic on hover.
    - Result dialog: success/failure, medals, XP gained.
    - Per-puzzle leaderboard podium displaying the top solvers ranked by solve time or lowest cost (toggleable between Fastest Time and Lowest Cost views)."
    - Optional debugger visualizing signal propagation.
    - Sound effects system: audio feedback for component drop, wire connection, errors, and success, with toggleable mute button and volume slider in settings.

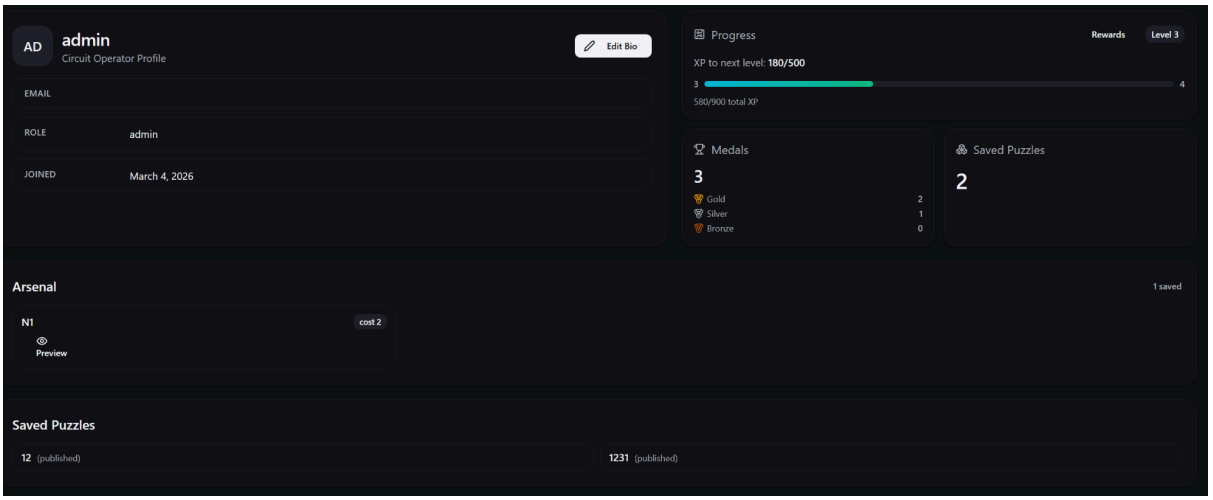




6.3 Profile Page

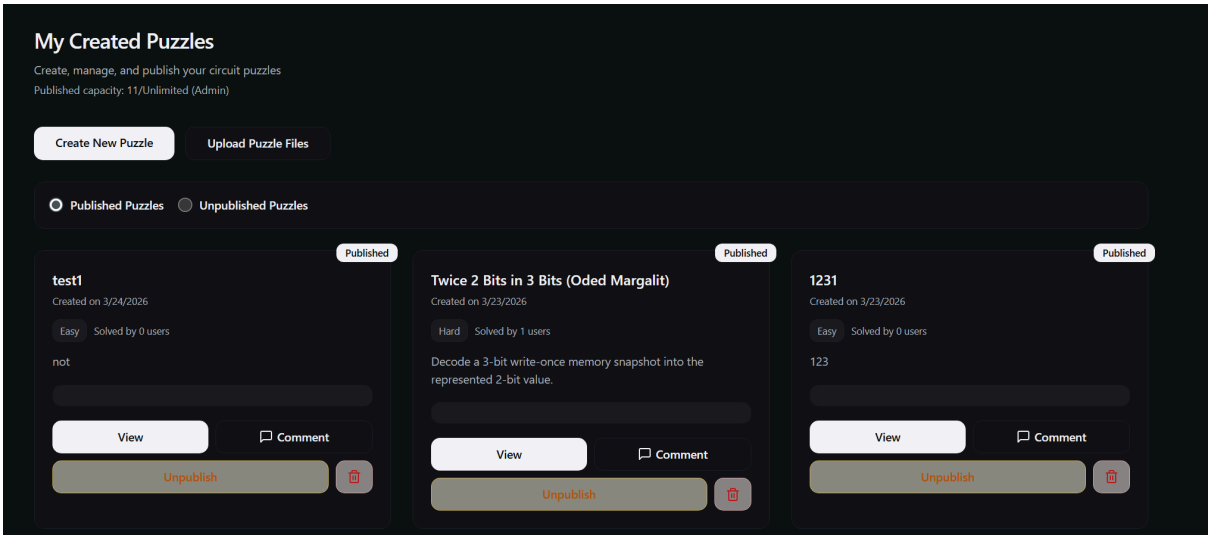
- Inputs
    - Navigation from header to “Profile”.
    - Option buttons to view saved puzzles, saved circuits, and created puzzles (for creators).
- Outputs
    - User name, XP bar, current level, XP required for next level.
    - Counts of solved puzzles and medals by tier.
    - Lists of:
        - Saved puzzles

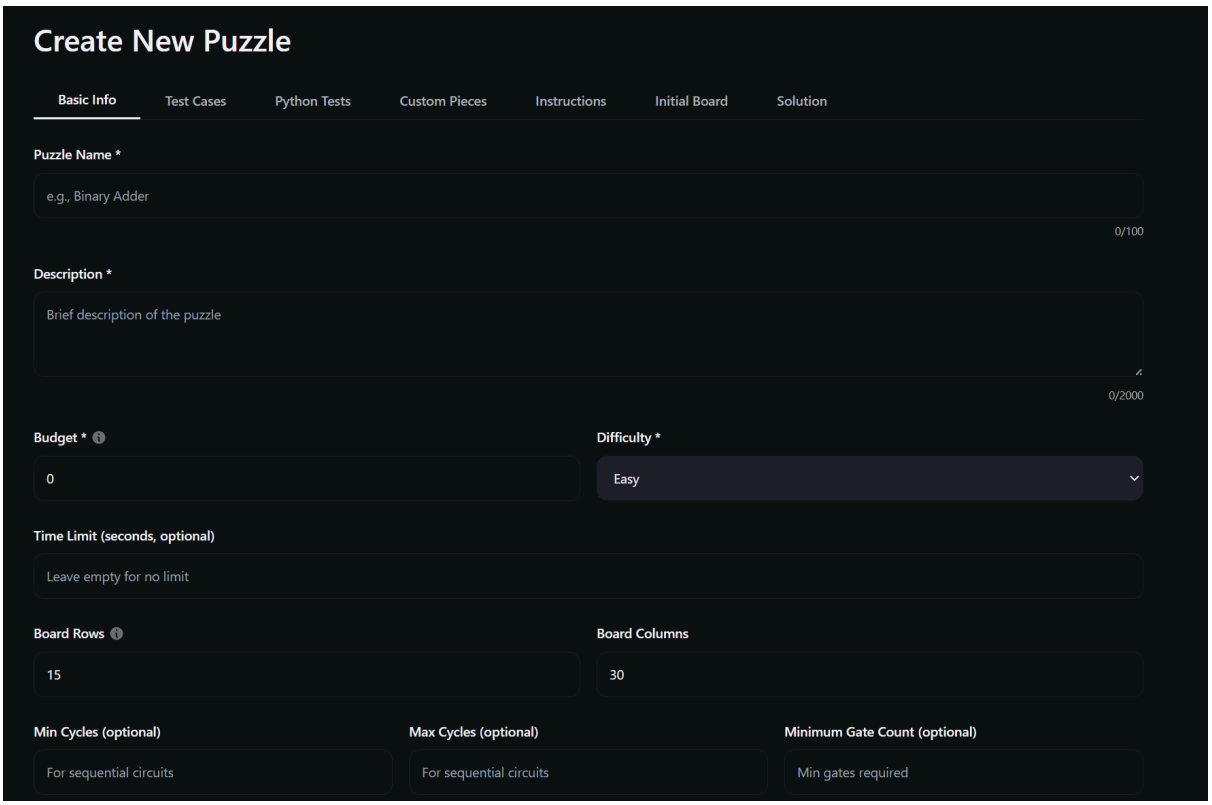
- Saved circuits (Arsenal) with InfoPopup (ⓘ) explaining how user level determines circuit storage capacity
- Created puzzles (for creators)



6.4 Creator Dashboard

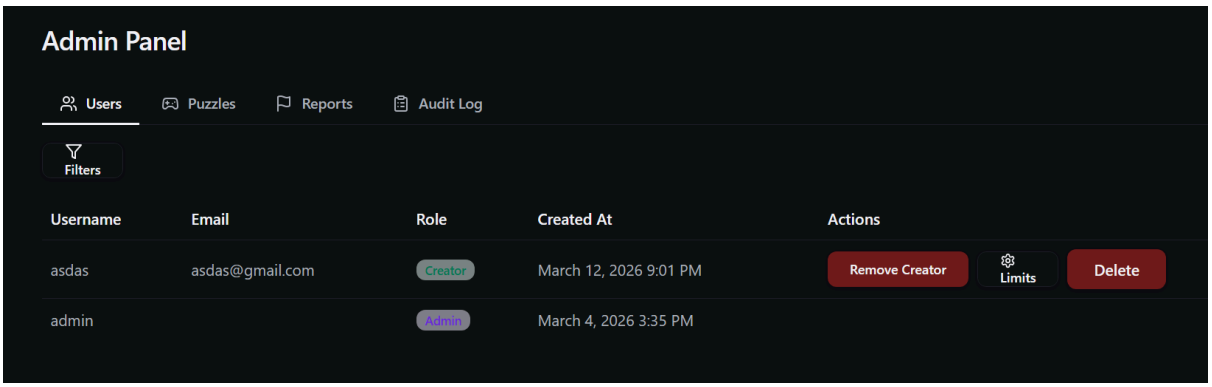
- Inputs
  - “New Puzzle” button to open puzzle creation form.
  - Puzzle creation/edit form includes a board size selector (boardRows: default 15, boardCols: default 30).
  - Action buttons to edit, publish, unpublish, or delete existing puzzles.
- Outputs
  - InfoPopup (ⓘ) explaining capacity tiers (how the creator's level determines the maximum number of publishable puzzles).
  - List of draft and published puzzles with:
    - Name, upload date
    - Current rating summaries
    - Popular flag, if applicable

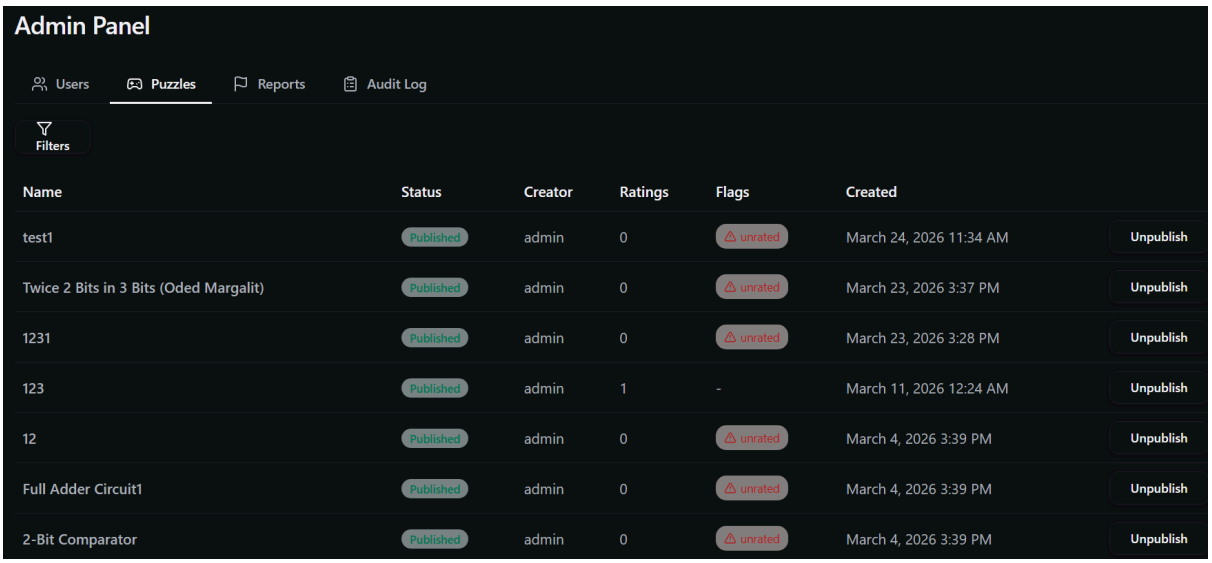




6.5 Admin Panel

- Inputs
  - User search/filter.
  - Buttons to assign/remove Creator role.
  - Buttons to delete or unpublish problematic puzzles.
- Outputs
  - List of users with roles.
  - List of puzzles with flags for moderation (e.g., low rating, low popularity, etc.).




Admin Panel
Users
Puzzles
Reports
Audit Log
Filters
Name
Status
Creator
Ratings
Flags
Created
test1
Published
admin
0
unrated
March 24, 2026 11:34 AM
Unpublish
Twice 2 Bits in 3 Bits (Oded Margalit)
Published
admin
0
unrated
March 23, 2026 3:37 PM
Unpublish
1231
Published
admin
0
unrated
March 23, 2026 3:28 PM
Unpublish
123
Published
admin
1
-
March 11, 2026 12:24 AM
Unpublish
12
Published
admin
0
unrated
March 4, 2026 3:39 PM
Unpublish
Full Adder Circuit1
Published
admin
0
unrated
March 4, 2026 3:39 PM
Unpublish
2-Bit Comparator
Published
admin
0
unrated
March 4, 2026 3:39 PM
Unpublish

Chapter 7 – Testing

Unit Tests for core logic, integration tests for complex user flows, and specific non-functional tests designed to verify performance, reliability, and constraints compliance.

7.1 Testing Approach and Strategy

1. Unit Testing: Core logical components (XP calculation, rating aggregation, circuit cost calculation, Boolean evaluation engine), to make sure the core of the system functions properly.

2. Continuous Integration: Tests will be rerun constantly during development every commit to a branch, allowing for immediate response to committed errors and reducing the chances of bugs in the main branch.

3. Validation and Integration Testing: End-to-end flows (e.g., UI Backend evaluation XP update) will be tested to confirm seamless integration between the Frontend (React), Backend (Python logic engine), and Persistence Layers.

4. Acceptance Testing: Use Cases and non-functional constraints will be tested manually and automatically to confirm that the system meets customer requirements.

7.2 Unit Testing: Core Logic and Constraints

This section outlines the unit testing strategy for the application's core deterministic logic. Tests are structured using the **AAA (Arrange, Act, Assert)** pattern to validate isolated functions, algorithms, and state transitions without relying on external dependencies (databases or UI).

7.2.1 Circuit Cost and Board Constraints

Target Module: *CircuitValidator / BoardEngine* This test suite ensures that puzzle constraints are mathematically and logically enforced at the domain level.

Test Case	Target Function/ Class	Arrange	Act	Assert (Expected)
Circuit cost calculation with basic gates	Circuit.py	Create Circuit with structure_json containing 3 AND gates, cost=1	Call calculate_cost({})	Returns 3 (sum of basic gate costs)
Circuit cost includes custom arsenal pieces	Circuit.py	Create Circuit with gates ["AND", "CustomPiece_1"], cost=1. Pass special_gates={"CustomPiece_1": arsenal_circuit} where arsenal_circuit cost is 5	Call calculate_cost(special_gates)	Returns 6 (1 for AND + 5 from arsenal piece)

Budget limit exceeded raises error	Puzzle.py	Create Puzzle with budget=10	Call enforce_budget(15)	Raises ValidationError("Circuit cost 15 exceeds puzzle budget 10")
Budget validation passes within limit	Puzzle.py	Create Puzzle with budget=20	Call enforce_budget(15)	No exception raised
Enforce Budget Limit	Puzzle.py	Set puzzleBudget = 10. Initialize a circuit with currentCost = 9. Create a new component (special component from the arsenal) with cost = 2.	Call validateComponentAddition(newComponent, currentCost, puzzleBudget)	Addition of the component is blocked, and an appropriate message is displayed Throws BudgetExceededException.
Filter arsenal by allowed basic gates	ArsenalService.py	Create arsenal pieces: P1 (uses AND, OR), P2 (uses XOR). Allowed gates = {AND, OR}	Call get_available_pieces_for_puzzle(token, allowed_gates)	Returns only P1; P2 excluded because XOR not allowed
Puzzle name max length enforced	Puzzle.py	Create Puzzle with name="" (empty string)	Instantiate	Raises ValidationError("Puzzle.name is required")

Puzzle name length validation	settings.py	Reference PUZZLE_NAME_MAX_LENGTH	Check value	Equals 100 characters
Puzzle description max length	settings.py	Reference PUZZLE_DESCRIPTION_MAX_LENGTH	Check value	Equals 2000 characters
Board Bounds Validation	puzzle.py	Set boundary config to MAX_ROWS = 20. Attempt to initialize grid with boardRows = 25.	Call GridFactory.create(boardRows, boardCols)	Creation is rejected with a validation error indicating invalid board size. Throws OutOfBoundsException.

7.2.2 Experience (XP) and Rewards Calculation

Test Case	Target Function/Class	Arrange	Act	Assert (Expected)
Base XP for EASY difficulty	XPService.py	Reference settings constant	Access BASE_XP[PuzzleDifficulty.EASY]	Returns 50 XP
Base XP for MEDIUM difficulty	XPService.py	Reference settings constant	Access BASE_XP[PuzzleDifficulty.MEDIUM]	Returns 100 XP
Base XP for HARD difficulty	XPService.py	Reference settings constant	Access BASE_XP[PuzzleDifficulty.HARD]	Returns 200 XP
Medal bonus for SILVER medal	XPService.py	Reference settings constant	Access MEDAL_BONUS[Medal.SILVER]	Returns 25 XP bonus
Medal bonus for GOLD medal	XPService.py	Reference settings constant	Access MEDAL_BONUS[Medal.GOLD]	Returns 50 XP bonus

Solve XP with delta (first solve)	XPService.py	Create XPService. Solve HARD puzzle with GOLD medal, previous_best_xp=0	Call calculate_solve_xp(PuzzleDifficulty.HARD, Medal.GOLD, 0)	Returns 250 (200 base + 50 bonus - 0 previous)
Solve XP delta prevents double rewards	XPService.py	Solver previously earned 250 XP on same puzzle	Call calculate_solve_xp(PuzzleDifficulty.HARD, Medal.BRONZE, 250)	Returns 0 (Current score 200 - 250 previous = -50. max(0, -50) = 0)
Creator reward per solve	XPService.py	Create two users: creator (id=2) and solver (id=3)	Call award_creator_solve_xp(2, 3)	Awards creator 10 XP (from SOLVE_REWARD_CREATOR setting)
Creator gets no XP for self-solve	XPService.py	Create user with id=5 (creator and solver are same)	Call award_creator_solve_xp(5, 5)	Returns 0 (no self-reward)
Leaderboard sorted by fastest time	SolveRepo.py	Create 3 passed solve attempts with times: 120s, 90s, 150s	Call get_leaderboard(puzzle_id, 50) *Note:* 50 represents the best 50 results on the puzzle.	Returns entries ordered by time ASC: 90s (rank 1), 120s (rank 2), 150s (rank 3)

Leaderboard sorted by lowest cost	SolveRepo.py	Create 3 passed solve attempts with costs: 20, 10, 15	Call get_leaderboard_by_cost(puzzle_id, 50)	Returns entries ordered by cost ASC: 10 (rank 1), 15 (rank 2), 20 (rank 3)
Leaderboard respects passed=1 filter	SolveRepo.py	Create 2 attempts: passed=1 (120s), passed=0 (90s)	Call get_leaderboard(puzzle_id, 50)	Returns only passed attempt; failed attempt excluded
Level calculation from XP	XPService.py	Total XP = 400 (with LEVEL_XP_DIVISOR=100)	Call calculate_level(400)	Returns level 3 (floor(sqrt(400/100)) + 1 = 2 + 1)
User becomes experienced at level 5	XPService.py	User with EXPERIENCED_LEVEL_MIN=5, level=4 (XP=1600) and level=5 (XP=2500)	Check is_experienced(2500) vs is_experienced(1600)	Level 4 returns False; Level 5 returns True
Medal boundary evaluation (Inclusive)	Puzzle.py / XPService.py	Puzzle has a Silver medal limit of cost=15. Solver submits a valid circuit with	Call medal evaluation logic	Returns Medal.SILVER (evaluates as <=, not <).

		exactly `cost=15`.		
Leaderboard immediate update	`SolveRepo.py`	User successfully passes puzzle. Record is saved to DB.	Call `get_leaderboard(puzzle_id, 50)` immediately after save	The newly inserted solve attempt is present in the returned list and ranked correctly.

7.2.3 Rating System Weighting

Test Case	Target Function/Class	Arrange	Act	Assert (Expected)
Rating values clamped to 1-5	Rating.py	Create Rating with difficulty=6, fun=0, clearness=3	Instantiate	Raises ValidationError or clamps via clamp_int()
Weighted average calculation	RatingService.py	Ratings: [2, 3, 5], weights (experienced): [1, 2, 1]	Call _weighted_avg([2, 3, 5], [1, 2, 1])	Returns (2×1 + 3×2 + 5×1) / 4 = 3.25
Creator dual-weight for experienced raters	RatingService.py	Create Rating with is_experienced_at_rating=True/False	Call _exp_weight(rating)	Returns 2 if experienced, 1 if not
Difficulty blending with few ratings	RatingService.py	Puzzle with creator_difficulty=HARD (maps to 5), 3 user ratings [3, 4, 3]. Count < threshold (10)	Call get_puzzle_metrics(puzzle_id)	weighted_difficulty = 0.8×5 + 0.2×3.33 = 4.67
Difficulty blending with many ratings	RatingService.py	Same puzzle with 12 user ratings (avg=3.5). Count >= threshold (10)	Call get_puzzle_metrics(puzzle_id)	weighted_difficulty = 0.4×5 + 0.6×3.5 = 4.1

Threshold constant for rating count	settings.py	Reference RATING_USER_COUNT_THRESHOLD	Check value	Equals 10
Few ratings weight tuple	settings.py	Reference RATING_DIFF_WEIGHT_FEW_RATINGS	Check value	Equals (0.8, 0.2) (creator weight, user weight)
Many ratings weight tuple	settings.py	Reference RATING_DIFF_WEIGHT_MANY_RATINGS	Check value	Equals (0.4, 0.6) (creator weight, user weight)
Duplicate rating validation on create	RatingService.py	User (id=5) already rated puzzle (id=1). Store allow_existing=False	Call create_rating(token, 1, payload)	Raises ValidationError("You already rated this puzzle. Use the edit flow instead.")
Duplicate rating allowed on update	RatingService.py	User (id=5) already rated puzzle (id=1). Store allow_existing=True	Call edit_rating(token, 1, 5, 4, 3)	Updates existing rating; no error
Rating requires minimum attempt time	RatingService.py	User hasn't solved but attempted for 250s (threshold=300s)	Check _can_rate(user_id, puzzle_id, client_elapsed=250)	Returns False

Rating allowed after solve	RatingService.py	User passed solve_attempts with passed=1	Check _can_rate(user_id, puzzle_id)	Returns True regardless of time
Experienced snapshot at rating time	RatingService.py	User at level 4 (not experienced) submits rating	Database stores is_experienced_at_rating=False	Value persists; never updates even if user later levels up
Rating distribution counting	RatingService.py	5 ratings: difficulty scores [2, 3, 3, 4, 5]	Access difficulty_dist in return	difficulty_dist[2]=1, [3]=2, [4]=1, [5]=1
Difficulty tier mapping from setting	settings.py	Reference RATING_DIFFICULTY_MAP	Check keys/values	{"EASY": 1, "MEDIUM": 3, "HARD": 5}
Rating XP awarded (first time only)	XPService.py	first_time_rating=True, rater (not the puzzle's creator)	Call award_rating_xp(rater_id=5, creator_id=10, first_time_rating=True)	Rater gets 5 XP, creator gets 1 XP (from settings)
Rating XP skipped on duplicate	XPService.py	first_time_rating=False	Call award_rating_xp(5, 10, False)	Returns 0 (no XP awarded)

7.3 Functional and Integration Testing

Integration tests verify the complex flows between the UI and the server logic, focusing on acceptance criteria derived from the Use Cases.

Test Goal	Test Scenario / Setup	Expected Result
End-to-End Solving Flow	User logs in, selects a puzzle, builds circuit, submits solution.	Backend verifies correctness, UI displays success/failure status and updates user XP/Level.
Solution Validation Constraint	User builds a circuit that solves the puzzle but omits one of the defined puzzle inputs.	System prevents checking the solution, stating that the circuit does not meet essential conditions (must use all defined inputs/outputs).
Medal Upgrade (Tight Budget)	User solves a puzzle successfully, and the Cost is less than or equal to the Tight Budget limit.	Medal is upgraded (e.g., Bronze -> Silver). User is granted bonus XP.
Medal Upgrade (Timer)	User solves a puzzle successfully for the first time before the timer expires.	A Timer Medal is granted, and the user's medal tier may be upgraded (e.g., Bronze Silver).
Puzzle Creation Self-Solve	Creator defines a new puzzle with constraints and attempts to publish.	Creator must successfully self-solve the puzzle within the tight budget and fill any necessary details before the puzzle is published.

Authorization Check (Creator)	A Solver (non-Creator) attempts to access the puzzle creation tool or delete a published puzzle.	Action is rejected with a "forbidden" or "permission denied" error.
Leaderboard After Solve	User logs in, solves a puzzle successfully.	After result dialog, the per-puzzle leaderboard is fetched and displayed showing the user's ranking among top solvers.
Metadata Validation on Publish	Creator attempts to publish a puzzle with name > 100 chars and boardRows = 25.	Backend rejects the publish request with specific validation errors before self-solve is attempted.
Duplicate Rating Integration	User who already rated a puzzle submits a new rating via the API.	Server returns an error indicating a rating already exists; no duplicate row is created.
Discussion Integration	User creates discussion, another user replies and votes	Discussion and reply are persisted, vote counts updated correctly.
Notification Integration	User solves a puzzle	Notification is generated for the puzzle creator with correct type and metadata.

7.4 Non-Functional Testing

Non-functional tests verify the system's compliance with qualitative requirements related to performance, reliability, and security.

Constraint	Test Description	Expected Result
Performance: Validation Speed	Submit a solution for a complex, maximum-size puzzle (5 inputs, 20 outputs, high gate count).	Validation of the circuit against the puzzle’s test cases shall be completed in a timely manner.
Performance: WorkStation Latency	Perform rapid drag-and-drop actions, component connections, and component removals in the WorkStation.	The system should respond to user actions in a seamless way, with a response time of less than 300 ms.
Usability: Timer Visual Cues	Set a short time limit (e.g., 20 seconds) and monitor the on-screen timer.	Timer color must reliably transition from green yellow at the final 10% (2 seconds left) and to red at zero, then display the stopwatch prefix "+", and start counting up.
Reliability: Fault Tolerance	Intentionally trigger a network loss (client side) immediately after submitting a solution but before receiving the final confirmation.	Solution validation shall be atomic, preventing partial saves or inconsistent XP reward states upon retry.
Security: Data Integrity	Attempt to manipulate or tamper with the request packet containing XP rewards, medals, or ratings	XP rewards, medals, and ratings must be validated server-side to prevent tampering, and the

	sent from the client to the server.	manipulated request must be denied.
Usability: Error Feedback	Attempt to add a gate that exceeds the Budget or attempt to publish a puzzle with missing required content.	The system must display a clear on-screen message explaining the reason for the blocked action.

7.5 Explanation about Testing Design and Execution

EscapeCircuit is a puzzle platform where “correctness” is not only whether a circuit produces the right outputs, but also whether it does so under explicit constraints (Budget/Value limits, time constraints, allowed components, role permissions, etc.). Therefore, the testing strategy must validate three things together:

1. **Logic correctness** (the circuit evaluates to the expected outputs for the puzzle’s test cases),
2. **Constraint correctness** (the user did not violate puzzle limits and required structure), and
3. **System integrity** (rewards, ratings, and persistence cannot be corrupted or exploited).

This section describes how these tests are designed and executed across unit, integration, UI, and non-functional layers.

7.5.1. Core Principle: Deterministic and Reproducible Evaluation

At the heart of EscapeCircuit is the rule that identical circuits must always produce identical truth tables and results. This is not just a quality goal; it is necessary for fairness, academic suitability, and grading consistency.

Testing therefore treats the backend evaluation engine as the single source of truth (“oracle”) and verifies that:

- the evaluation outcome is deterministic given the same circuit graph JSON and puzzle definition,
- cost/value is computed consistently and reproducibly, and
- saved circuits behave identically when reused in other contexts (as long as they are permitted by the puzzle’s allowed components).

Practically, this means unit tests focus heavily on pure logic and pure computation: the evaluation function, cost calculation, constraint checks, XP/rating calculations. Any randomness is avoided; any time-based behavior (timer medals, stopwatch behavior) is validated through explicit inputs and controlled test conditions rather than “real time” waiting.

7.5.2 What “Correct Solution” Means in This System

Solution validation is defined by a set of **selected test cases** associated with each puzzle. During validation, the system checks the circuit against **only** those selected cases. This design keeps validation efficient and lets the puzzle creator define what it means to be correct. Validation is treated as a strict pipeline:

1. **Pre-validation checks (eligibility + constraints):**
   Before computing outputs, the system verifies that the submission is eligible to be tested: required inputs/outputs exist, connections are valid, illegal structures are rejected, and puzzle constraints (such as budget/value or allowed components) are satisfied. If these conditions are not met, the system rejects the attempt immediately.
2. **Cycle-based evaluation (stream/cycle semantics):**
   The system evaluates the circuit across cycles, feeding the selected input patterns into the circuit and computing outputs according to the circuit graph semantics. Tests verify that this “step-by-step” behavior is consistent and stable across repeated runs.
3. **Decision rule:**
   A solution is accepted only if it satisfies the expected outputs for **all** selected test cases. Any mismatch produces a deterministic failure outcome.

This pipeline is important for security and correctness: constraints must be enforced **before** evaluation and before any persistence/reward logic is applied.

7.5.3 Unit Tests: Proving the Rules in Isolation

Unit tests focus on the smallest rule-heavy components, ensuring correctness independently of networking, UI rendering, and database state. This layer aims to prove the “math and rules” of the system:

- Circuit evaluation behavior under cycle-based semantics.
- Cost/value calculations and budget enforcement logic.
- Constraint validation logic (legal wiring, allowed components, required inputs/outputs).
- Reward and progression logic (XP, medals, levels), including first-time and eligibility rules.

- Rating calculations and weighting logic (thresholds and recalculation rules).

Unit tests are deterministic and fast, making them suitable for continuous execution during development and in CI. Negative paths are first-class: rejecting invalid actions is as important as accepting valid ones.

7.5.4 Integration Tests: Ensuring Correctness Across the Full Flow

Even if each unit is correct, failures often happen where components meet: serialization, API boundaries, persistence, and authorization. Integration tests validate complete workflows across:

UI → API request → backend validation → evaluation → persistence → returned response.

Key properties integration tests enforce:

- **Backend authority:** the server validates constraints and eligibility regardless of what the client claims.
- **Correct ordering:** validation happens before evaluation, and evaluation happens before rewards/persistence.
- **Atomicity:** when validation succeeds, updates (solve status, rewards, ratings) are committed together; failures or network interruptions never produce partial updates.
- **Idempotency where required:** repeated submissions or retries do not corrupt state or duplicate rewards.

Integration tests are especially important for protecting system integrity because they verify the real behavior of the system under realistic interactions.

7.5.5 UI Tests: Correct Interaction Contract in the WorkStation

The circuit-building interface is a core part of the experience, and it must behave consistently under complex user actions. UI tests focus on the interaction contract rather than internal implementation details:

- Drag/drop and wiring actions remain stable and predictable.
- Connection feedback (highlighting, invalid connection states) is consistent.

- Copy-paste interactions (Ctrl+C / Cmd+C and Ctrl+V / Cmd+V) produce the expected duplicated components and internal wires, apply the correct grid offset, generate fresh unique IDs, auto-select the pasted region, respect the puzzle budget, and never copy locked initial-board elements.
- Pre-determined (locked) initial-board components and wires are rendered as non-interactable in solver mode: drag, rotate, delete, and copy operations on them are blocked and provide clear UI feedback, while remaining fully editable in creator edit mode.
- Constraint-related UI feedback is clear (e.g., budget reached, illegal action blocked).
- Timer behavior is correct and fair (state transitions and visual feedback).

UI tests are designed to prevent regressions that would break usability or allow confusing/inconsistent behavior. They complement backend tests rather than replace them.

7.5.6 Non-Functional Testing: Performance, Reliability, Security, Coverage

Non-functional requirements directly affect fairness and trust, so they are tested explicitly:

- **Performance:** validation completes within the expected time limits even for complex circuits and higher puzzle sizes; WorkStation interactions remain responsive.
- **Reliability:** network interruptions or failures do not cause inconsistent saves, partial reward grants, or corrupted rating states.
- **Security/integrity:** client-side tampering cannot grant XP/medals/ratings; all sensitive actions are verified server-side.
- **Coverage:** automated tests cover a high percentage of logic-heavy code paths, prioritizing validation, evaluation, and reward/rating logic.

7.5.7 Continuous Testing and Regression Prevention

Tests are not only for final verification—they actively guide development. The suite is designed for quick feedback and regression prevention:

- Unit tests protect rule correctness.

- Integration tests protect end-to-end correctness and persistence integrity.
- UI tests protect the WorkStation experience.
- Non-functional tests protect fairness, speed, and tamper-resistance.

Together, these layers ensure the system remains correct, consistent, and secure as features evolve.