Detailed Design

Or Reshef S 324064849 Gal Azoulay 323859967

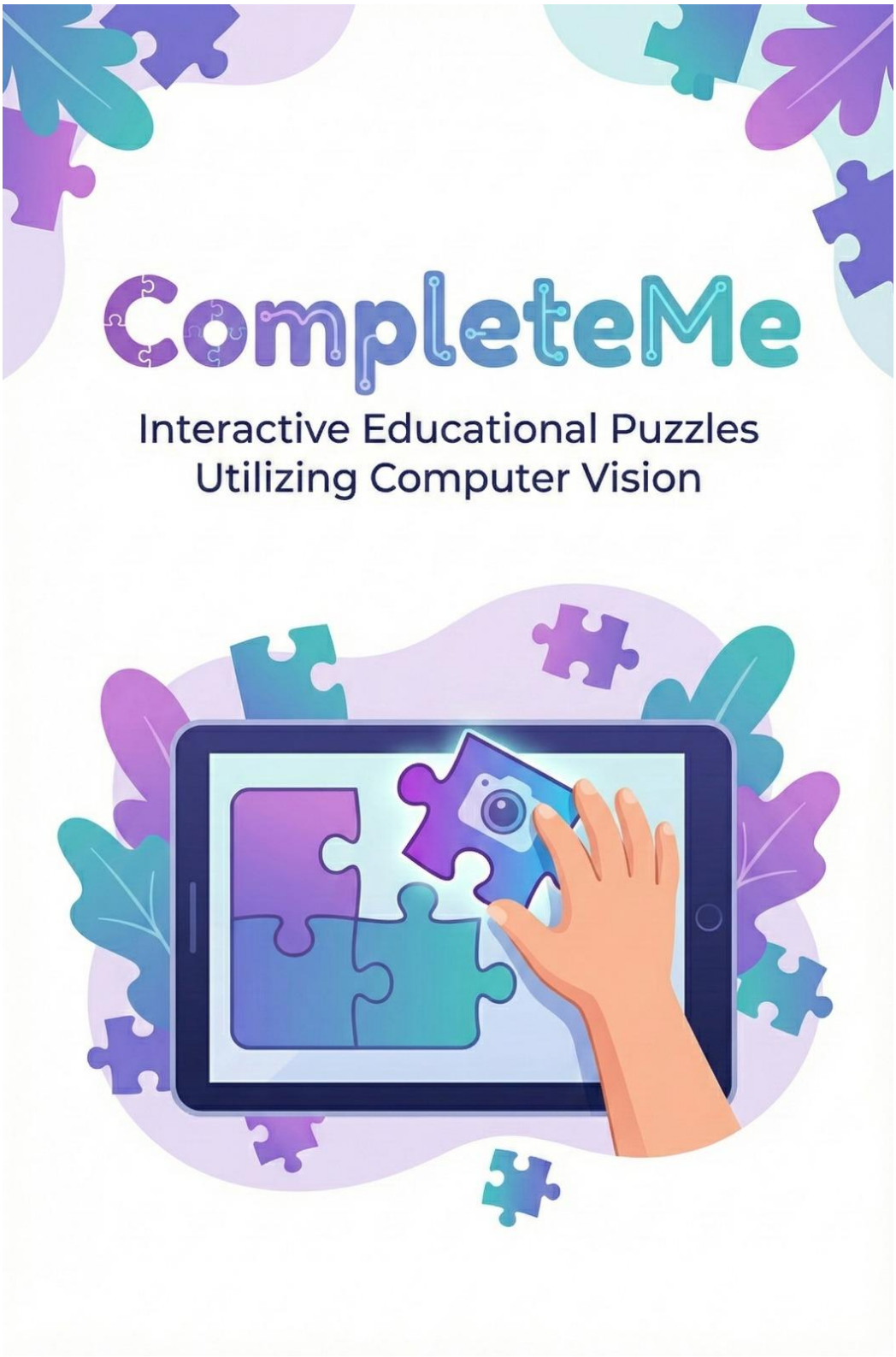


Table of Contents:

1. Database Design

The database serves as the backbone of our puzzle platform, storing user information, game history and system configuration. The following section outlines the database architecture, technology selection rationale and detailed schema design.

1.1 Database Technology Selection

Selecting the appropriate database technology is crucial for ensuring optimal performance, scalability and maintainability. We evaluated two primary options: SQL (relational) databases and NoSQL (document-based) databases.

Comparison: PostgreSQL vs MongoDB

Aspect	PostgreSQL (SQL)	MongoDB (NoSQL)
Data Structure	Structured tables with fixed schema	Flexible JSON-like documents
Query Language	SQL (mature, standardized)	MongoDB Query Language (MQL)
Relationships	Strong foreign keys, joins	Embedded documents, references
ACID Compliance	Full ACID guarantees	ACID within single documents
Scalability	Vertical scaling (scale up)	Horizontal scaling (scale out)
Performance	Excellent for complex queries	Faster for simple lookups
Schema Changes	Requires migrations	Flexible, no migrations needed
Learning Curve	Familiar SQL syntax	New query syntax to learn
Best For	Structured data, complex relationships	Rapidly changing schemas, high write loads

Final Decision: PostgreSQL (Local Installation)

After careful evaluation, we selected **PostgreSQL with local installation** as our primary database solution for the following reasons:

- **Strong ACID compliance** - ensures data integrity for user accounts and game history
- **Structured schema** - provides clear relationships between users, games, and progress tracking
- **Free and open source** - no cloud costs, suitable for academic projects
- **Simple deployment** - runs locally during development and testing
- **Production flexibility** - can easily migrate to cloud hosting (AWS RDS, Supabase, Railway) if needed in the future

For this academic project, PostgreSQL will be installed and run locally on the development machine, eliminating the need for cloud infrastructure while maintaining all the benefits of a robust relational database system.

1.2 Database Schema Design

The database schema consists of two primary tables: Users and Game_History. This minimalist design ensures efficient data storage while maintaining all necessary information for user authentication, progress tracking and analytics

Table 1: Users

The Users table stores all user account information, authentication credentials and profile settings.

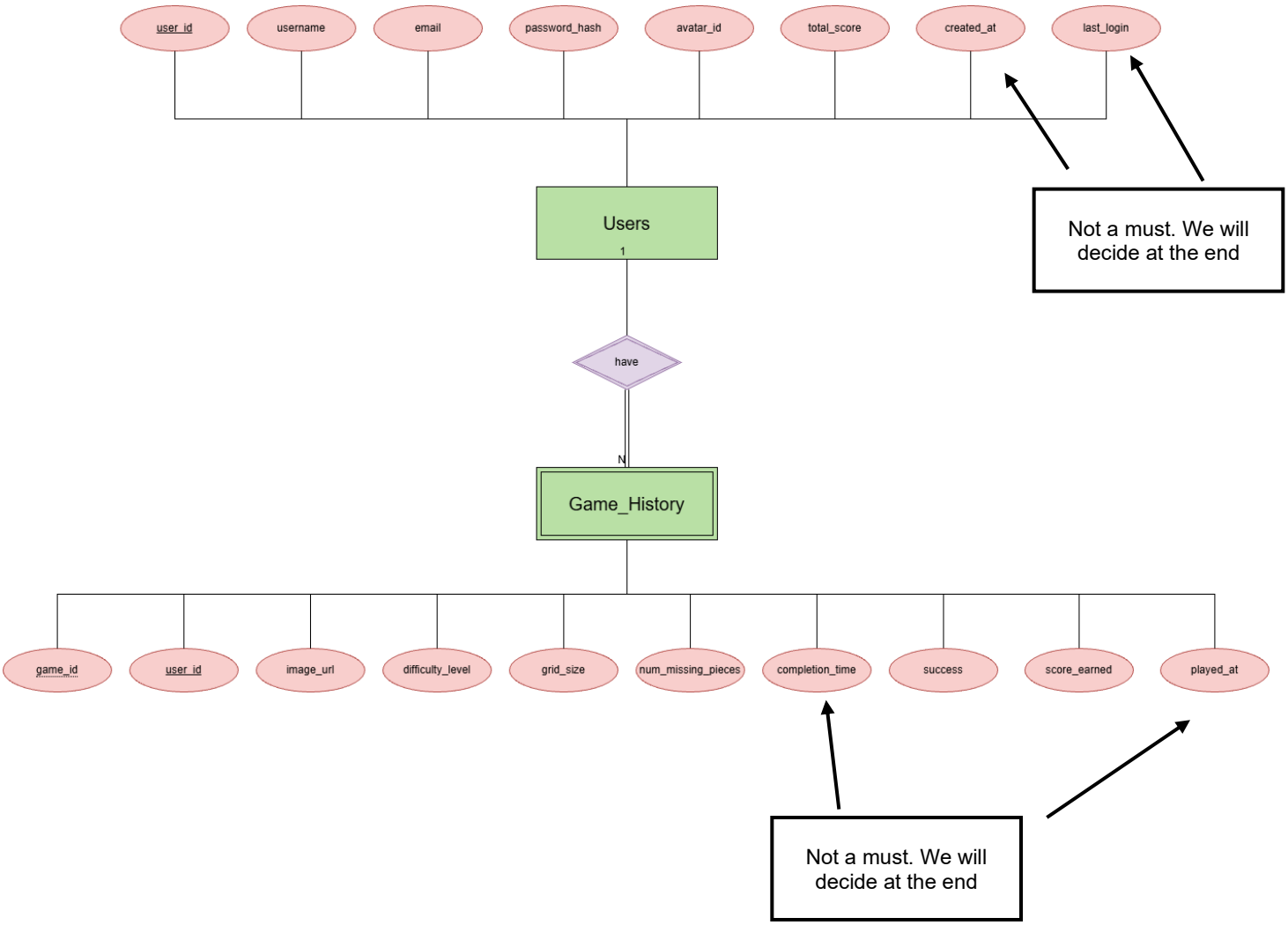
Column Name	Data Type	Description
user_id	SERIAL PRIMARY KEY	Unique identifier for each user
username	VARCHAR(50) UNIQUE NOT NULL	User's display name
email	VARCHAR(100) UNIQUE NOT NULL	User's email address (for login)
password_hash	VARCHAR(255) NOT NULL	Hashed password using bcrypt
avatar_id	INTEGER	Reference to predefined avatar (1-10, we will decide later on in the project)
total_score	INTEGER DEFAULT 0	Cumulative points earned
created_at (not a must)	TIMESTAMP DEFAULT CURRENT_TIMESTAMP	Account creation date
last_login (not a must)	TIMESTAMP	Last login timestamp

Table 2: Game_History

The Game_History table records every completed puzzle attempt, enabling progress tracking and analytics.

Column Name	Data Type	Description
game_id	SERIAL PRIMARY KEY	Unique identifier for each game session
user_id	INTEGER FOREIGN KEY	References Users(user_id)
image_url	TEXT NOT NULL	URL of Unsplash image used
difficulty_level	VARCHAR(20) NOT NULL	Beginner, Intermediate, Advanced, Expert
grid_size	INTEGER	Grid dimensions (2, 4, 8, 16, 32)
num_missing_pieces	INTEGER	Number of pieces to find (1-4)
completion_time (not a must)	INTEGER	Time taken in seconds
success	BOOLEAN NOT NULL	True if puzzle completed correctly
score_earned	INTEGER	Points awarded for completion
played_at (not a must)	TIMESTAMP DEFAULT CURRENT_TIMESTAMP	When the game was played

Database Schema Diagram:



2. System Architecture

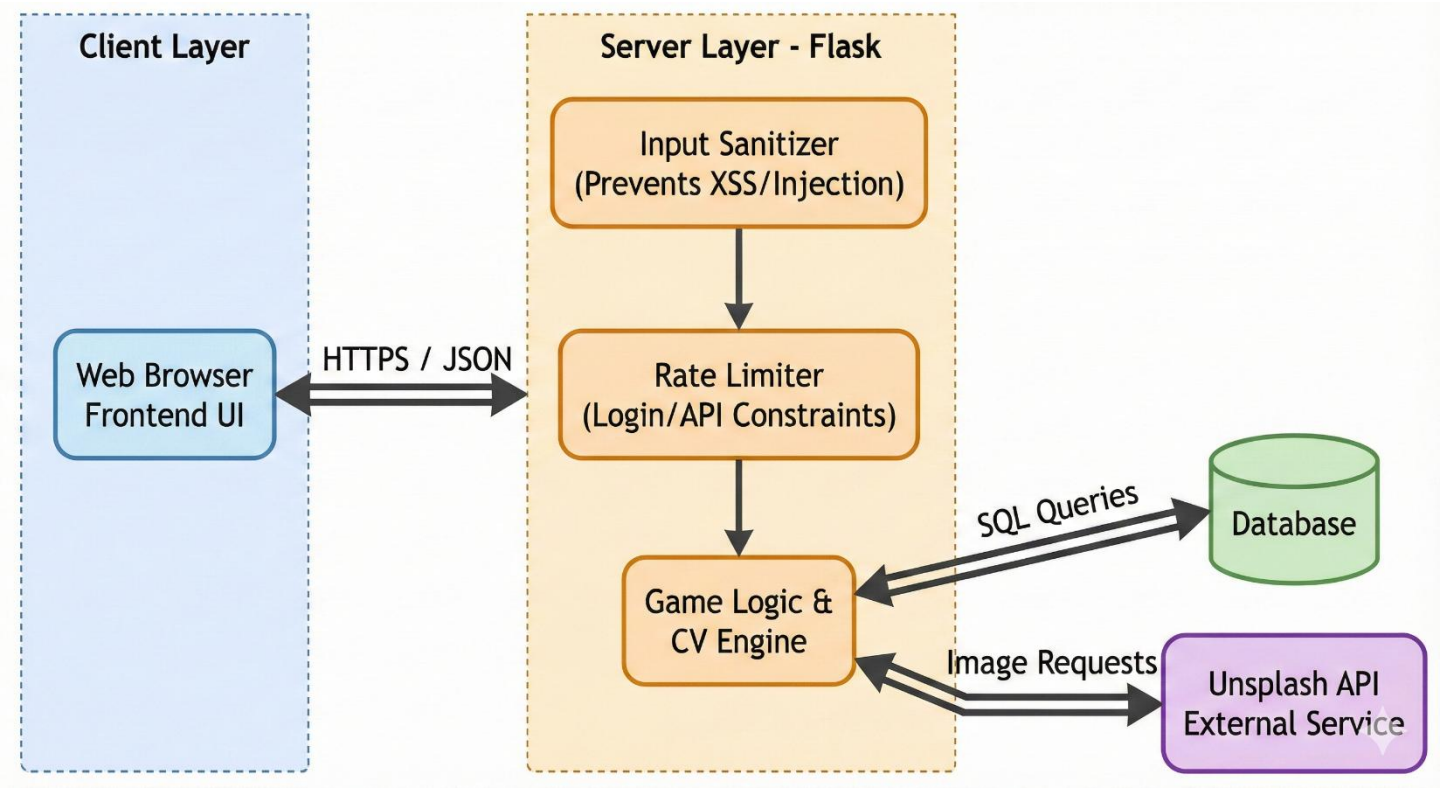
The system follows a modern Client-Server Architecture utilizing a RESTful API for communication between components. This architecture ensures clear separation of concerns, scalability and maintainability.

2.1 System Architecture Diagram

The following diagram illustrates the complete system architecture, highlighting the data flow between the client, the server, the database, and the external APIs.

Key Security Components:

- **Rate Limiter:** Monitors request frequency to enforce limits (e.g., login attempts) and prevent abuse.
- **Input Sanitizer:** Validates and cleanses all incoming user data (search queries, registration details) to prevent SQL Injection and XSS.
- **Authentication Handler:** Manages user identity using JWT.



2.2 Component Description

The platform consists of three primary layers:

2.2.1 Client Layer (Frontend)

The client layer is a responsive web application that provides the user interface for all interactions.

User Interface: Built with HTML5, CSS3, and JavaScript to ensure compatibility across devices.

HTTP Client: Uses the Fetch API to communicate asynchronously with the backend.

Key Responsibilities:

- Display puzzle interface with incomplete images
- Handle user input (piece selection, drag-and-drop)
- Send validation requests to backend
- Render visual feedback (animations, scores)
- Manage session state and authentication tokens

2.2.2 Server Layer (Backend)

The server layer, built with Flask (Python), handles all business logic, authentication and external API integration.

Input Sanitization: Automatically strips malicious characters from user inputs before processing to prevent code injection attacks.

Game Logic Module: Handles puzzle generation, difficulty scaling, and scoring.

Computer Vision Engine: Integrates OpenCV and PyTorch to validate user moves.

Key Responsibilities:

- User authentication and authorization (JWT)
- Fetching images from the Unsplash API with child-safe content filters
- Image processing and puzzle generation
- Computer Vision validation algorithms
- Database operations and query management

2.2.3 Data Layer (Database)

PostgreSQL database provides persistent storage for all application data.

Password Protection: Stores user passwords strictly as hashed values using bcrypt (never in plain text).

Key Responsibilities:

- Store user accounts and authentication credentials
- Maintain game history and progress records
- Ensure data integrity through ACID transactions

2.3 System Data Flow Example

The following sequence describes how data moves through the system during a standard game session:

1. **Initialization:** User selects a difficulty level and image preference (random or specific) → Client sends GET /api/random-image.
2. **External Fetch:** Server fetches image from Unsplash API.
3. **Processing:** Server replaces selected regions with black, caches pieces, generates decoy pieces (from another API request to Unsplash), and returns the puzzle data to the client.
4. **Interaction:** Client displays the puzzle. User selects piece → Client sends POST /api/validate.
5. **Validation:** Server runs Computer Vision algorithms for validation.
6. **Completion:** If correct, the Server saves to the database. The Client displays the victory animation.

2.4 Technology Stack

The following table summarizes the complete technology stack used in our platform:

Component	Technology Used
Frontend	HTML5, CSS3, JavaScript
Backend Framework	Flask (Python 3.x)
Database	PostgreSQL (SQL)
Authentication	JWT, bcrypt
Computer Vision	OpenCV
Deep Learning (not a must)	PyTorch – if we decide to add DL to our project
External API	Unsplash API (RESTful)
HTTP Client	requests library (backend), Fetch API (frontend)
Version Control	Git, GitHub

3. Component Design

The application follows a Component-Based architecture pattern, with clear separation between authentication, game logic, image processing and data management. Each component has well-defined responsibilities and interfaces.

3.1 Authentication Component

Purpose: Handles secure user registration, login and session management.

Key Functions:

- User Registration: Creates new accounts with hashed passwords (bcrypt)
- User Login: Validates credentials and issues JWT tokens
- Session Management: Verifies tokens for protected endpoints
- Guest Mode: Allows unauthenticated temporary access

3.2 Game Management Component

Purpose: Handles game sessions, difficulty settings and progress tracking.

Key Functions:

- Game Initialization: Sets up puzzle based on user-selected difficulty
- Score Calculation: Computes points based on number of puzzles solved
- History Management: Stores last 10+ completed puzzles

3.3 Image Processing Component

Purpose: Fetches images from Unsplash API, processes them and generates puzzle pieces.

Key Functions:

- Image Retrieval: Fetches child-safe images via Unsplash API
- Grid Division: Segments image into 2-32 pieces based on difficulty
- Pixel Masking: Replaces selected regions with black squares
- Piece Extraction: Generates correct piece and decoy options
- Temporary Storage: Caches masked image data in session

3.4 Computer Vision Validation Component

Purpose: Validates user-selected pieces by performing visual analysis to ensure they fit correctly into the puzzle.

Key Functions:

- Edge Detection: Analyzes boundary continuity using standard edge detection algorithms such as Canny or Sobel to ensure smooth transitions between the piece and the surrounding area.
- Color Compatibility: Compares the colors of the piece with the surrounding area to ensure they match, using methods like HSV histogram analysis.
- Texture Analysis: Verifies that the texture consistency on the piece matches the rest of the image using texture analysis techniques such as LBP or Gabor filters.

4. Design Considerations

4.1 Image Retrieval Strategy

The platform provides users with three distinct methods for selecting puzzle images, each designed to balance personalization with child safety.

4.1.1 Image Selection Methods:

1. Keyword Search (User-Driven)

Users can type a specific search term (e.g., "cat", "airplane", "monkey") to retrieve images matching their interests. The system sanitizes user input and queries the Unsplash API with the content_filter=high parameter to ensure child-appropriate results.

2. Random Image (Surprise Me Button)

Users can request a completely random image by clicking the "Surprise Me" button. This triggers an Unsplash API call without any category filter, relying solely on Unsplash's child-safe content filtering (content_filter=high) to return appropriate images. This provides maximum variety and discovery.

3. Category-Based Random Selection

Users can select from a predefined list of categories (e.g., Animals, Space, Toys). The system then fetches a random image from the chosen category via the Unsplash API. This approach combines the excitement of randomness with the curated topics.

4.1.2 Predefined Categories:

The following categories will be hardcoded in the backend configuration file (config.py).

PUZZLE_CATEGORIES = ["animals", "nature", "space", "cars", "candy", "toys", "food", "flowers"]

The decision to hardcode categories rather than store them in the database offers several benefits:

- **Simplicity:** No database queries required for category retrieval
- **Easy Updates:** Categories can be modified in the configuration file without database migrations
- **Performance:** Eliminates database lookups for static category data

4.1.3 API Integration:

All three selection methods interface with the Unsplash API as follows:

Selection Method	API Endpoint	Query Parameters
Keyword Search	/photos/random	query={user_input}, content_filter=high
Surprise Me	/photos/random	content_filter=high
Category Selection	/photos/random	query={selected_category}, content_filter=high

4.2 Deep Learning Library Comparison (optional)

We will add this validation layer to our project if time permits. It's not mandatory - Sharon said it's not a must at all. Our project is based on computer vision validation, and this would be a deep learning validation type.

Selecting the appropriate deep learning framework is critical for implementing the advanced (optional) Computer Vision validation layers, specifically for semantic analysis and feature extraction. We evaluated the two industry standards: TensorFlow with Keras and PyTorch.

Aspect	TensorFlow + Keras	PyTorch
Ease of Use	High-level Keras API is very beginner-friendly	More explicit, "Pythonic" syntax that mimics NumPy
CV Ecosystem	Good (tf.keras.applications)	Excellent (torchvision offers extensive pre-trained model support)
Community	Large, Google-backed	Larger for CV research
Debugging	Static graph origins can make debugging complex errors harder	Dynamic computational graphs allow easy line-by-line debugging
Flexibility	Structured and somewhat rigid	Highly flexible, ideal for custom validation logic
Deployment	Industry standard for mobile/web (TFLite)	Growing rapidly (TorchServe, ONNX), but slightly less mature for mobile
Learning Curve	Moderate	Steeper initially, clearer long-term
Performance	Optimized for large-scale production	Excellent for rapid iteration and research tasks
Pre-trained Models	TensorFlow Hub	torchvision.models (ResNet, VGG)

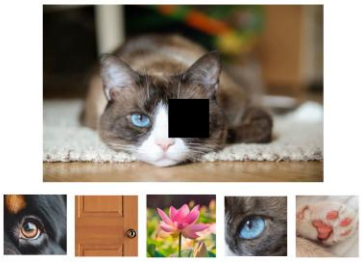
Final Decision: PyTorch

After thorough evaluation, we have selected PyTorch as our framework for any deep learning implementation in this project. This decision is driven by the following reasons:

- Superior Computer Vision Support: torchvision provides pre-trained models (ResNet50, VGG16) optimized for image analysis
- Dynamic Computational Graphs: Allows flexible algorithm implementation and easier debugging
- Stronger CV Community: More active research community producing cutting-edge CV algorithms
- Better Integration with OpenCV: Seamless data flow between PyTorch tensors and OpenCV operations
- Explicit Operations: Clearer understanding of each processing step, essential for custom validation logic
- Production-Ready: Mature ecosystem with excellent deployment options

5. Algorithms

The core functionality of our platform relies on four primary algorithms: Image Processing & Masking, Decoy Piece Generation, Computer Vision Validation and Hint System. Each algorithm is designed to work independently while contributing to the overall puzzle experience.



5.1 Image Processing & Masking Algorithm

Purpose: Transforms a complete image into a puzzle by selectively masking regions with black squares.

Algorithm Steps:

1. Image Acquisition: Fetch image from Unsplash API
2. Grid Division: Divide image into N×M grid based on difficulty level (1×2 for Beginner, up to 8×4 for Expert)
3. Region Selection: Randomly select 1-4 regions to mask based on difficulty setting
4. Piece Extraction: Extract the pixel data from these selected regions to serve as the "Correct Pieces"
5. Pixel Masking: Replace selected regions with black pixels
6. Session Storage (this is not a must, we will decide later): Cache original pieces in Flask session or Redis
7. Response Generation: Return masked image and piece dimensions to frontend

5.2 Decoy Piece Generation Algorithm

Purpose: Creates convincing incorrect puzzle pieces to challenge users.

Algorithm Steps:

1. Determine Count: Calculate number of decoy pieces (3-5) based on difficulty
2. Fetch Random Images: Request random images from Unsplash API
3. Extract Random Regions: Select random regions from fetched images
4. Resize to Match: Scale decoy pieces to match missing piece dimensions
5. Shuffle Options: Randomize order of correct piece and decoys

5.3 Computer Vision Validation Algorithm

Purpose: Validates whether a user-selected piece correctly completes the puzzle using multi-layered computer vision analysis.

Algorithm Layers:

Layer 1: Edge Continuity Analysis

- Apply edge detection algorithms (e.g., Canny, Sobel) using OpenCV to both the placed piece and adjacent regions
- Extract edge pixels along all four boundaries
- Calculate edge alignment score using gradient direction similarity

Layer 2: Color Compatibility Analysis

- Convert images to HSV color space (separates color from brightness)

Compute color histograms for piece and boundary regions

- Calculate histogram similarity using distance metrics (e.g., chi-square, Bhattacharyya distance, histogram intersection) to ensure color harmony

Layer 3: Texture Consistency Analysis

- Apply texture analysis methods to capture structural patterns (e.g., Local Binary Patterns, Gabor filters)
- Extract texture descriptors from both piece and boundary regions
- Compare texture features to evaluate consistency across the boundary

Layer 4: Deep Feature Extraction (This is a deep learning algorithm. This is not a must in our project, but if we will have time we will add this)

- Use a pre-trained CNN model (such as ResNet50 or VGG16) for deep feature extraction
- Calculate similarity scores between the piece's features and the context features to detect semantic mismatches

Layer 5: Semantic Validation (This is a deep learning algorithm. This is not a must in our project, but if we will have time we will add this)

- Ensure logical spatial coherence
- Detect implausible compositions

5.4 Hint System Algorithm

Purpose: Provides intelligent hints without revealing the answer.

Algorithm Steps:

1. Analyze Boundary: Examine the pixel data surrounding the empty slot on the canvas
2. Extract Dominant Colors: Compute the average color or dominant color cluster of these surrounding pixels
3. Color Naming: Map the resulting RGB values to human-readable names
4. Generate Hint: Provide color-based guidance

6. APIs & Interfaces

The platform interfaces with external services through the Unsplash API and exposes internal functionality through a RESTful API.

6.1 External API: Unsplash

- **Endpoint:** GET https://api.unsplash.com/photos/random
- **Parameters:** client_id, content_filter=high, query (optional)
- **Response:** JSON object containing id and urls, example for a JSON response: { "id": "abc123", "url": { "regular": "https://images.unsplash.com/..." } }

6.2 Internal API Endpoints

The Flask backend exposes the following RESTful API endpoints:

Method	Endpoint	Description
POST	/api/register	Create new user account
POST	/api/login	Authenticate user, return JWT token
POST	/api/logout	Invalidate JWT token
GET	/api/random-image	Fetch random image (Surprise Me)
GET	/api/category-image/<category>	Fetch image from specific category
POST	/api/generate-puzzle	Process image, create puzzle with masked regions
POST	/api/validate	Validate user-selected piece using CV algorithms
GET	/api/hint	Generate color-based hint for current puzzle
GET	/api/user/profile	Retrieve user profile and statistics
GET	/api/user/history	Fetch last 10 completed puzzles
PUT	/api/user/avatar	Update user avatar selection

7. Security & Cyber Defence

Security is critical since children are the primary users. This section outlines comprehensive security measures across authentication, data protection, and application security.

7.1 Authentication Security (JWT)

Token Strategy: The system uses a dual-token architecture to balance security and usability.

- Access Token: Short lifespan to minimize risk if a token is stolen.
- Refresh Token: Long lifespan to allow persistent sessions without frequent re-login.

Storage Mechanism: Tokens are stored strictly in HTTP-only, Secure Cookies to prevent theft via Cross-Site Scripting (XSS) attacks. They are never stored in localStorage or sessionStorage.

7.2 Password Security (bcrypt)

- Passwords never stored in plain text
- bcrypt automatically generates unique salt

7.3 Input Sanitization

- SQL Injection Prevention
- XSS Prevention
- Command Injection: Never execute user input

7.4 Rate Limiting

- Login Attempts: Max 5 failed attempts per 15 minutes.
- API Requests: Max 50 requests per hour (aligning with Unsplash API constraints).

7.5 Additional Security

- CORS Policy restrictions
- Safe Content: All external API calls (Unsplash) strictly use the content_filter=high flag to ensure no inappropriate imagery reaches the child.

8. User Interface Design

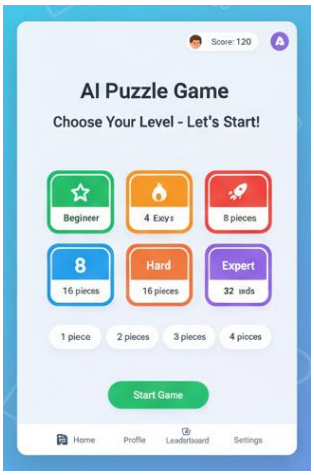
The UI is designed for children, emphasizing large buttons, colorful visuals, clear feedback, and minimal text.

8.1 Design Principles

- High Contrast Colors: Bright, distinct colors
- Minimal Text: Icon-driven interface
- Immediate Feedback: Animations for every action
- Progress Visualization: Uses visuals (e.g., stars, badges, bars) to show progress

8.2 Screen Walkthrough

8.2.1 Home Screen

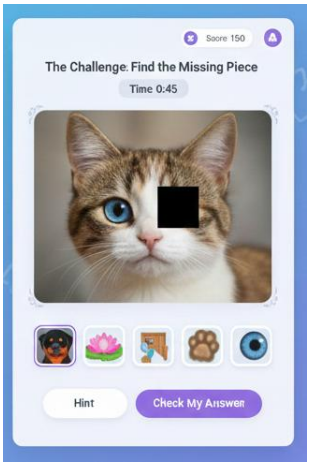


- 4 difficulty buttons: Beginner (2), Easy (4), Hard (16), Expert (32)
- Missing pieces selector: 1-4 pieces

Topic Selection:

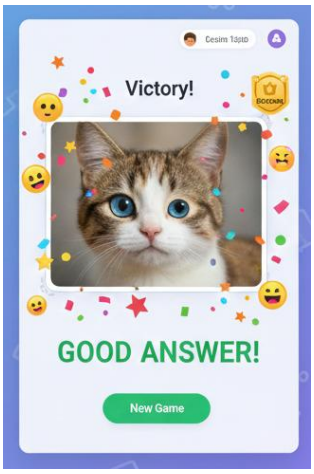
- Search Bar: Allows users to type specific interests (e.g., “Space”, “Cats”)
- “Surprise Me” Button: Fetches a random image
- Category-Based Selection: Allows users to choose a category from a predefined list of categories (e.g., “Animals”, “Cars”)
- "Start Game" button

8.2.2 Game Screen



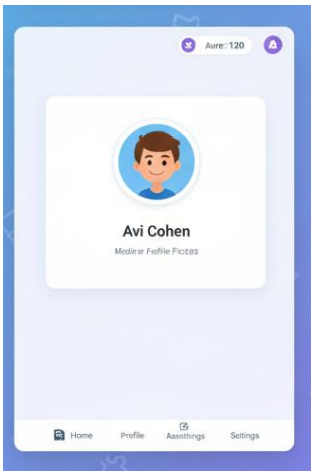
- Main canvas with black square(s)
- Piece tray holding the correct piece(s) and generated decoys
- Hint and Check Answer buttons

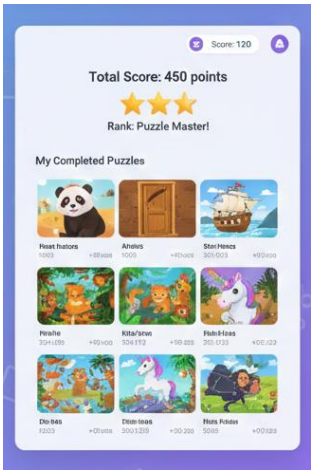
8.2.3 Victory Screen



- Completed image revealed
- Confetti animation
- Success message
- New game button

8.2.4 Profile & Progress





- Editable user avatar icon with a username display
- Total score display
- Completed puzzles gallery showing the last 10 completed puzzle images