APACHE INCUBATOR

Apache ResilientDB incubating

ResilientDB Application: ResCanvas

Henry Chou

ExpoLab
Creativity Unfolded

UCDAVIS
UNIVERSITY OF CALIFORNIA

What is Blockchain?

A blockchain is a distributed digital ledger that securely records transactions across a network of computers (nodes). Once recorded, data in a block cannot be altered without consensus from the network.

- Each **block** contains a set of transactions and a cryptographic hash of the previous block (forms a chain)
- Within the **distributed network**, every participant (node) stores a copy of the ledger (no single point of control or failure)
- Nodes follow rules to agree on which transactions are valid via **Consensus Mechanisms**

What is ResilientDB?

ResilientDB is a next-generation blockchain platform designed for high performance, fault tolerance, and scalability.

- Developed to address the limitations of traditional blockchain systems such as Bitcoin and Ethereum
- Built for high throughput
- Designed for low latency and fast consensus
- Modular architecture for research and extensibility
- Open-source project available at: https://github.com/apache/incubator-resilientdb

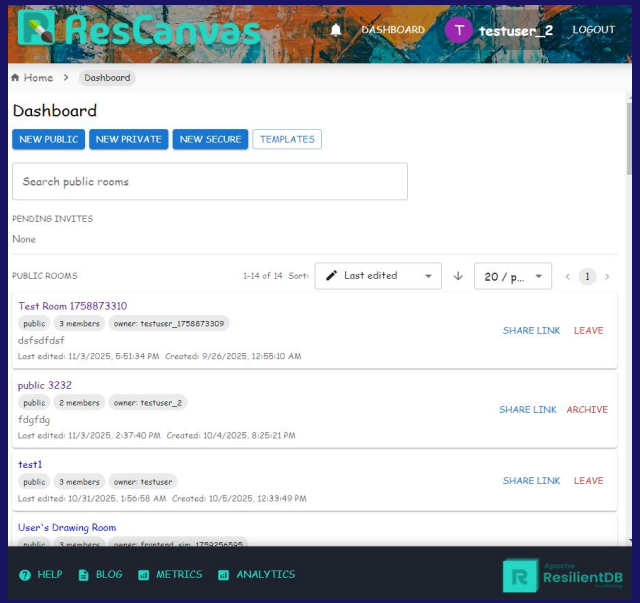
Why ResilientDB?

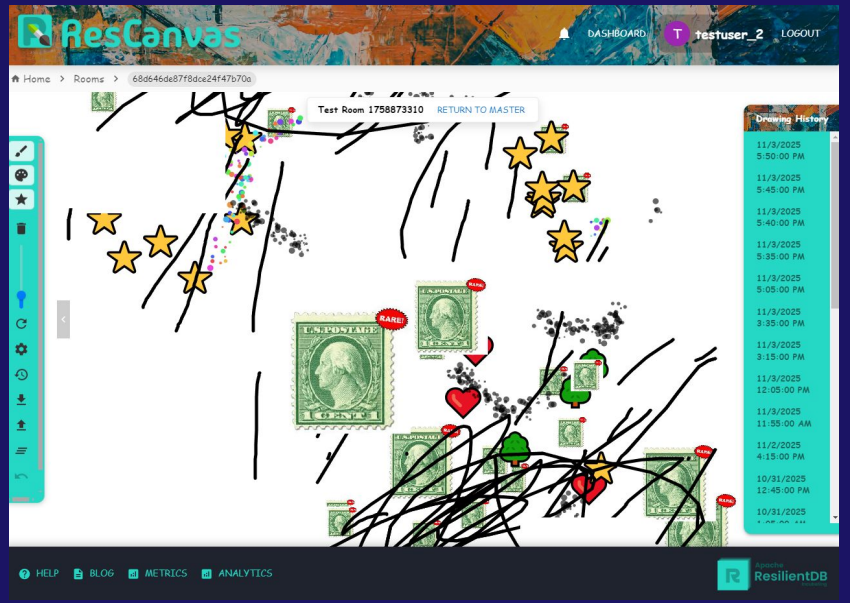
- Traditional blockchains face three key bottlenecks
  - Low throughput — e.g., Bitcoin: ~7 TPS
  - High latency — confirmation may take minutes
  - Energy inefficiency — Proof-of-Work wastes resources

- ResilientDB solves these using:
  - **Practical Byzantine Fault Tolerant** (PBFT) consensus instead of mining
    - PBFT is a method for a network of nodes to agree on a single result even if less than ⅓ nodes fail or act maliciously
  - Parallel execution of transactions
  - Efficient data replication and fault recovery

What is ResCanvas?

ResCanvas, a breakthrough in web-based drawing platforms that utilizes ResilientDB to ensure that user's drawings are securely stored, allowing for multiple users to collaborate and create new works of art and express ideas freely without any limits, tracking, or censorship. The canvas drawing board is the core feature of ResCanvas, designed to allow users to perform drawings.





Why ResCanvas? Comparison with Existing Tools

Feature / Aspect	Figma / Miro / Google Jamboard	**ResCanvas**
Architecture	Centralized since it is hosted by company servers (e.g., AWS, Firebase)	**Decentralized as transactions are recorded on ResilientDB blockchain**
Collaboration Model	Real-time editing via a single backend server	**Peer-synchronized using Redis, MongoDB, and blockchain replication via ResilientDB**
Data Ownership	Controlled by platform provider	**User controlled and cryptographically signed**
Fault Tolerance	Single point of failure as an outage or compromise of the main server can disrupt operations	**Practical Byzantine Fault Tolerant (PBFT) consensus ensures continued operation**
Transparency and Auditability	Limited since internal logs are inaccessible to users	**Fully auditable edits and other operations logged on the chain**

ResCanvas: Feature Overview

- Multiple user concurrent drawing and viewable editing history on a per user, per room basis with custom room and user access controls
- Drawing data and edit history is synchronized efficiently and consistently across all users within the canvas drawing room
  - Fast, efficient loading of data from backend via Redis cache
- Persistent, secure storage of drawing data in ResilientDB allowing for censorship free expression
  - No sharing of data to third parties, advertisers, government entities, .etc with decentralized storage
- JWT-based authentication for API and Socket.IO access
- Real time collaboration using Socket.IO for low latency stroke broadcasting, user notifications, and user activity communication

ResCanvas: Collaborative Features

- Room-Based Collaboration
  - Users can create or join drawing rooms, each backed by its own Redis and MongoDB workspace.
  - Supports isolation of session data between users and scalable multi-room collaboration.
- Account Management
  - JWT-secured login and registration system implemented in Flask with hashed credentials.
  - User authentication integrated with Redis session caching for quick validation.
- Dashboard & Sharing
  - Central dashboard listing all canvas rooms and shared public, private and secure type canvases.
  - Users can generate and share room links to control access for collaboration.
  - Displays real-time status notification of other users in same canvas (e.g., joining/leaving canvas)
- Undo/Redo & Recovery
  - Undo/redo actions are committed to ResilientDB just like strokes to ensure complete replayability.
  - On restart or Redis flush, the app rebuilds state directly from blockchain commits.
- Real-Time Collaboration
  - Socket.IO channels broadcast drawing events, ensuring millisecond synchronization.
  - Handles disconnect/reconnect gracefully and pulls missing strokes from MongoDB.

ResCanvas: Key Components

- **Frontend (React)**: Handles drawing input and display, UI state (tools, color, thickness), optimistic local rendering, and Socket.IO for real-time updates. Stores auth tokens in localStorage and the frontend API wrappers (frontend/src/api/) automatically attach JWT access tokens to all protected requests.

- **Backend (Flask + Flask-SocketIO)**: The authoritative security boundary which validates all JWT tokens server-side using middleware (backend/middleware/auth.py), enforces room membership and permissions, verifies client-side signatures for secure rooms, encrypts/decrypts strokes for private/secure rooms, and commits transactions to ResilientDB via GraphQL. All protected API routes and Socket.IO connections require valid JWT access tokens. The backend performs all security checks so clients cannot bypass authentication or authorization.

- **ResilientDB**: The persistent, decentralized, immutable transaction log where strokes are ultimately stored. Strokes are written as transactions so the full history is auditable and censorship-resistant.

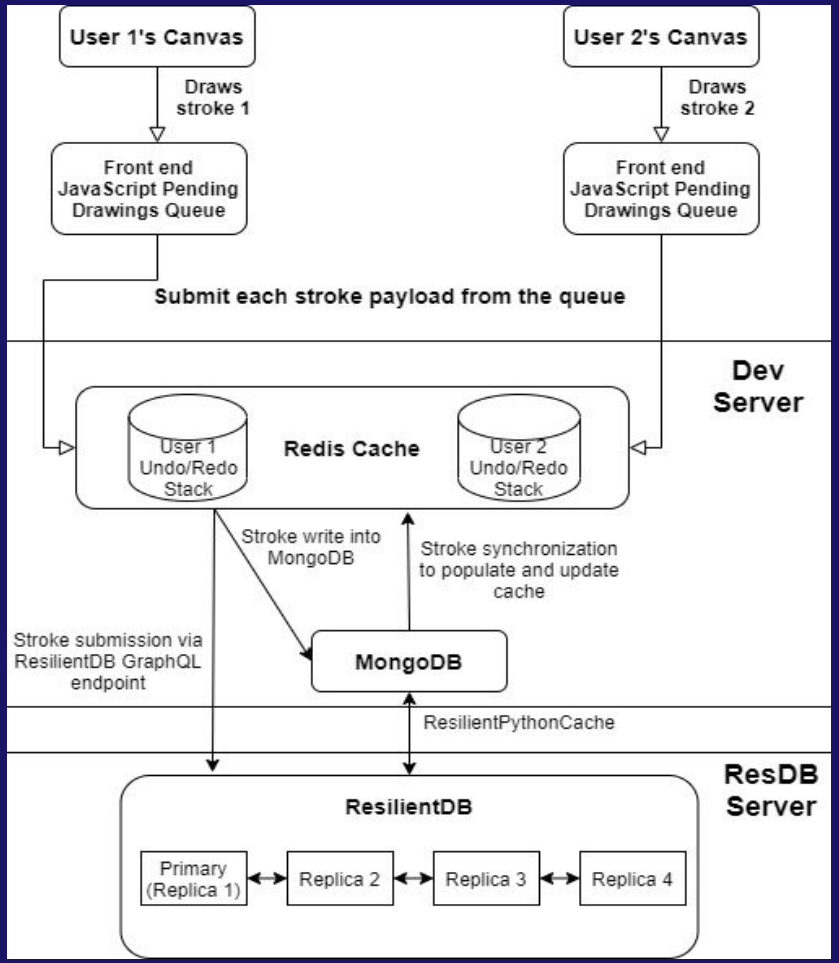
- **MongoDB**: A warm persistent cache and queryable replica of strokes so the backend can serve reads without contacting ResilientDB directly for every request. A sync bridge mirrors ResilientDB into MongoDB.

- **Redis**: Short-lived, in-memory store keyed by room for fast read/write and undo/redo operations. Redis is intentionally ephemeral: it allows quick synchronization of live sessions while ResilientDB acts as the long-term durable store.

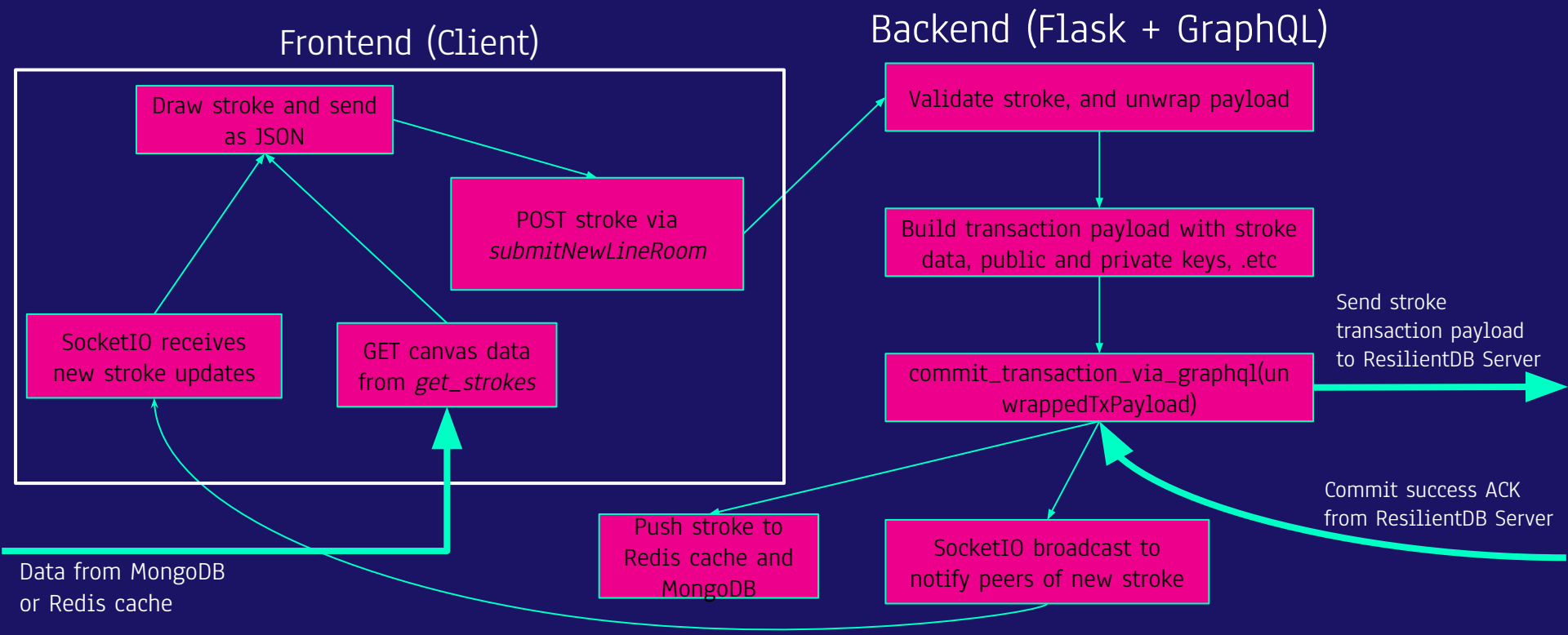
ResCanvas: Architecture Overview

- Front end interfaces with application layer backend services via REST API, contains components for drawing, toolbar, ResVault integration

- Backend handles user authentication via JSON Web Token (JWT) middleware, routes for various operations (get stroke data, submit stroke, clear canvas, undo, redo, .etc), and services such as graphql_service, SocketIO notification and stroke broadcasting
    - Secure submission and receiving of stroke data between Redis cache and MongoDB vía ResilientPythonCache (watches ResilientDB WebSocket for new blocks and writes transactions to MongoDB)

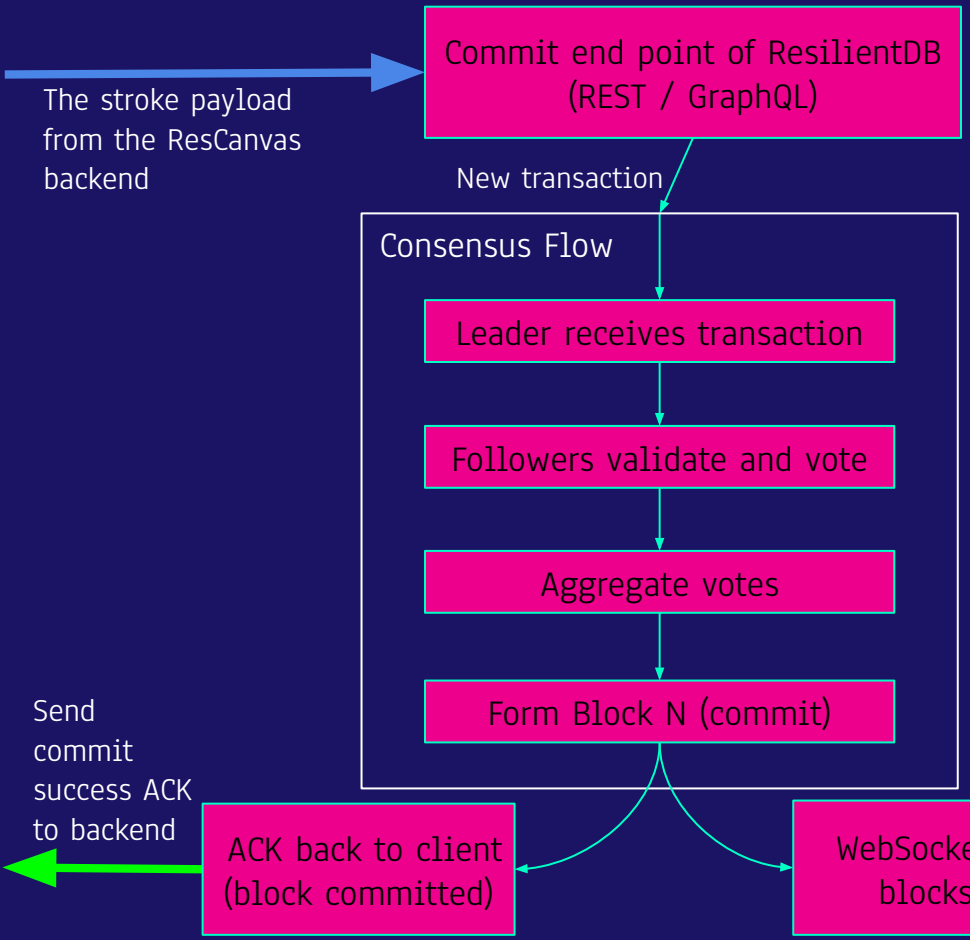


Frontend to Backend Stroke Data Flow



From the **Frontend** (Client Browser) to the **Backend** (Flask + GraphQL) to the **ResilientDB Server** (PBFT Transaction Flow and Consensus Protocol). This data path design serves as the key architecture of ResCanvas.

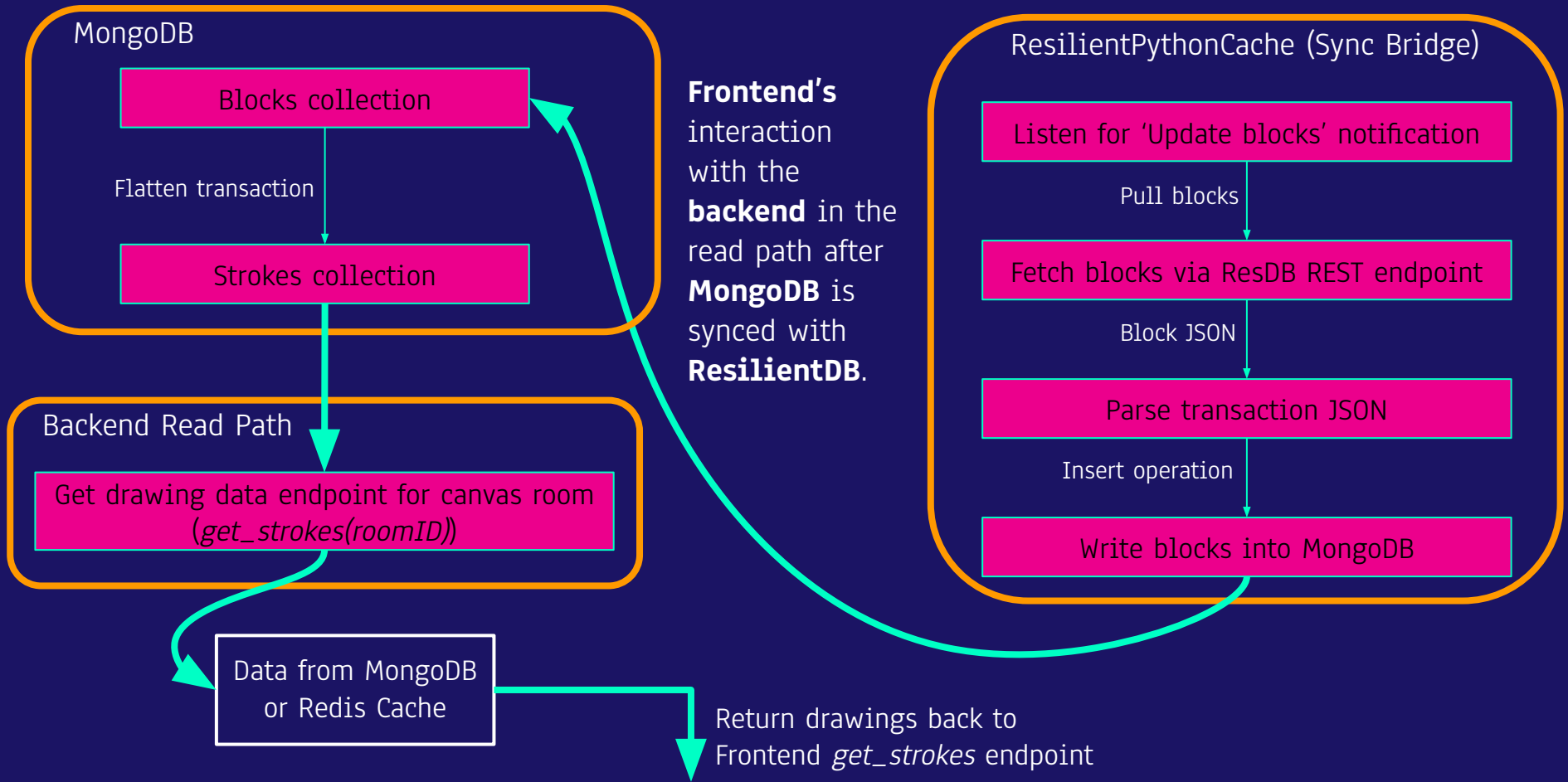
Backend to ResilientDB Data Flow



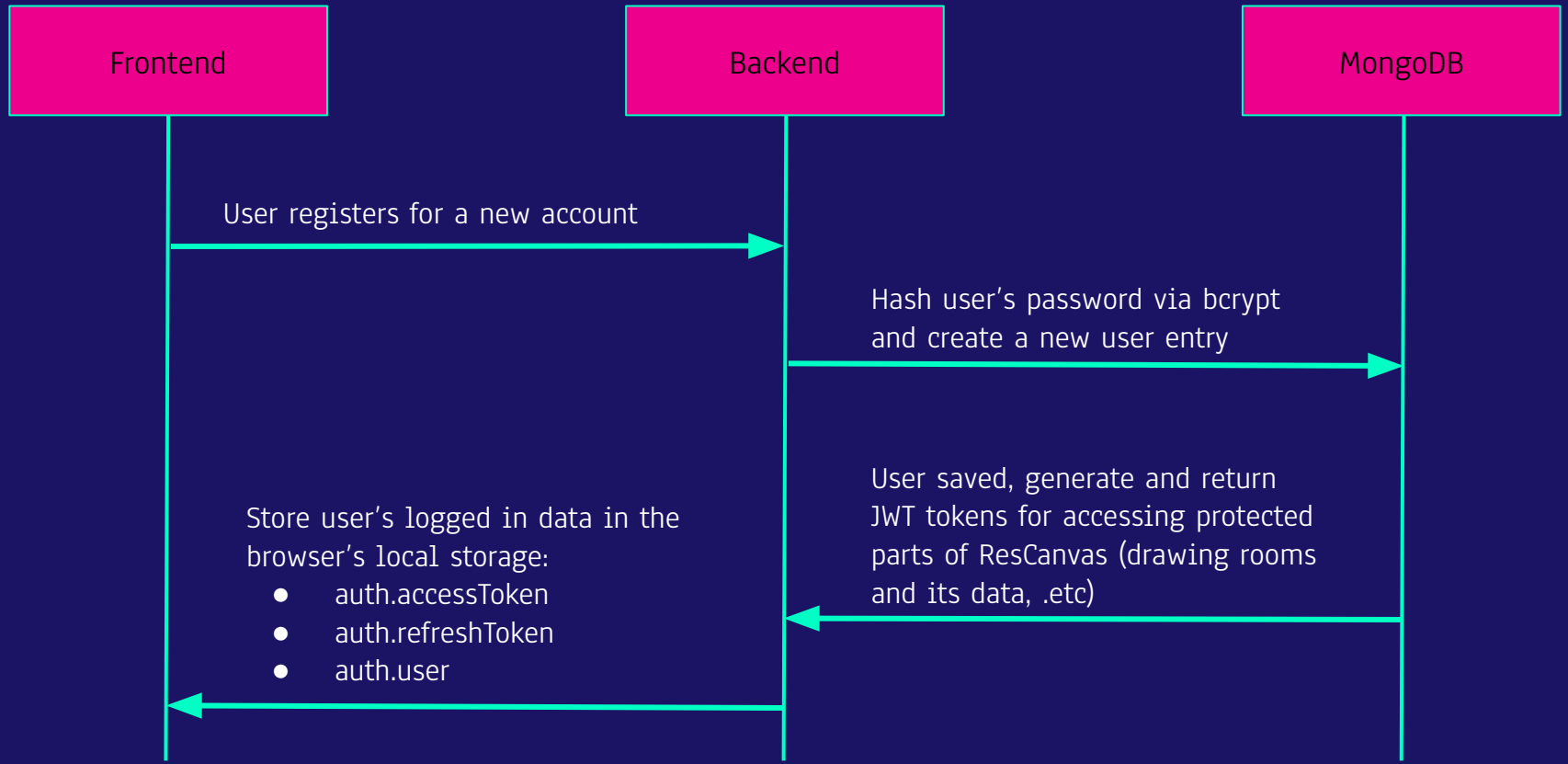
From the **ResilientDB Server** to the sync bridge library **ResilientPythonCache** for synchronizing with **MongoDB** of which the **Backend Read Path** reads from in order to obtain canvas data back to the **Frontend** client's browser.

Furthermore, backend's SocketIO notifies the client that there are new stroke updates available once it receives the commit success ACK from ResilientDB server.

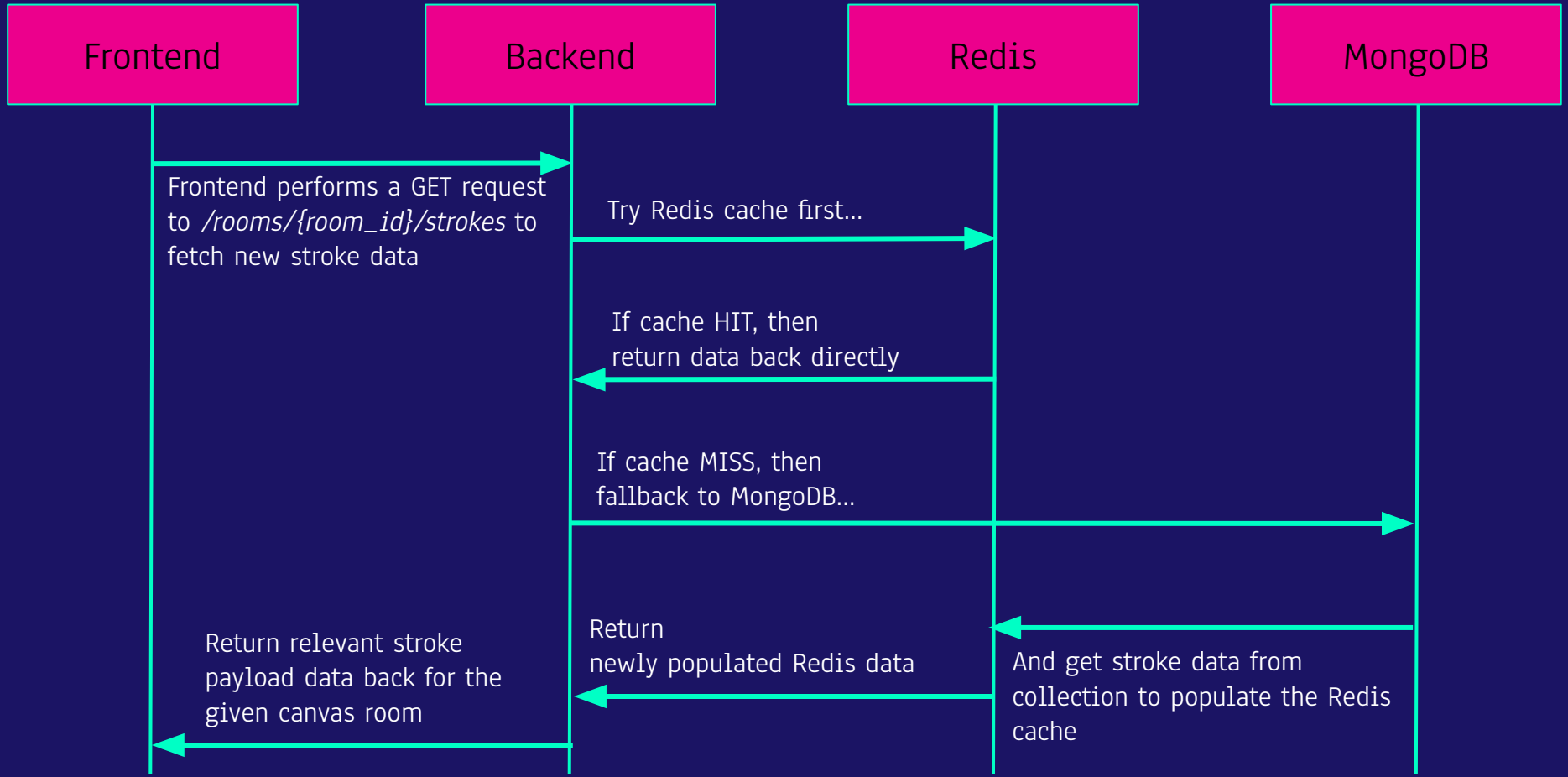
ResilientDB to MongoDB to Frontend Data Flow



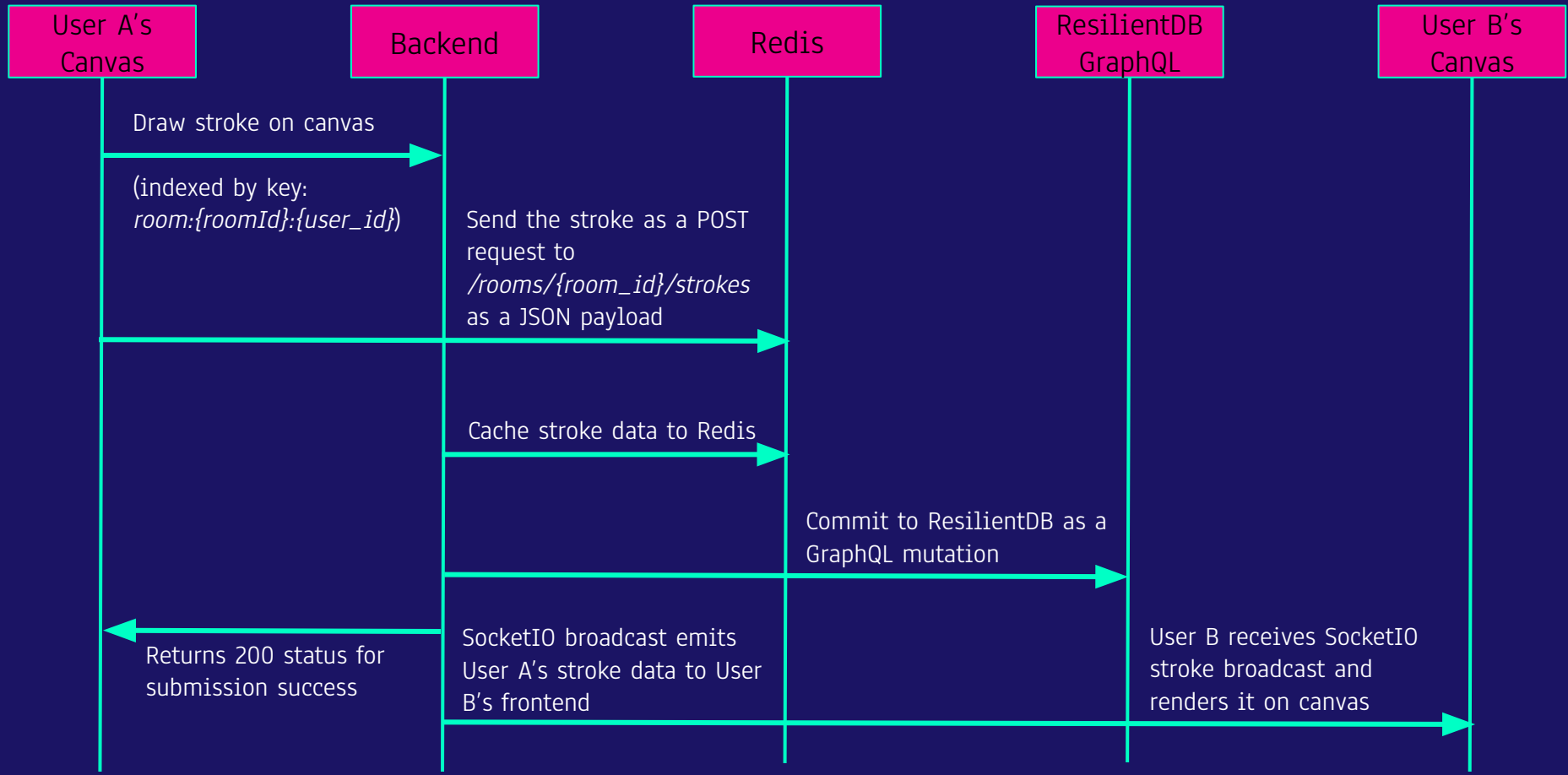
ResCanvas: User Authentication Flow



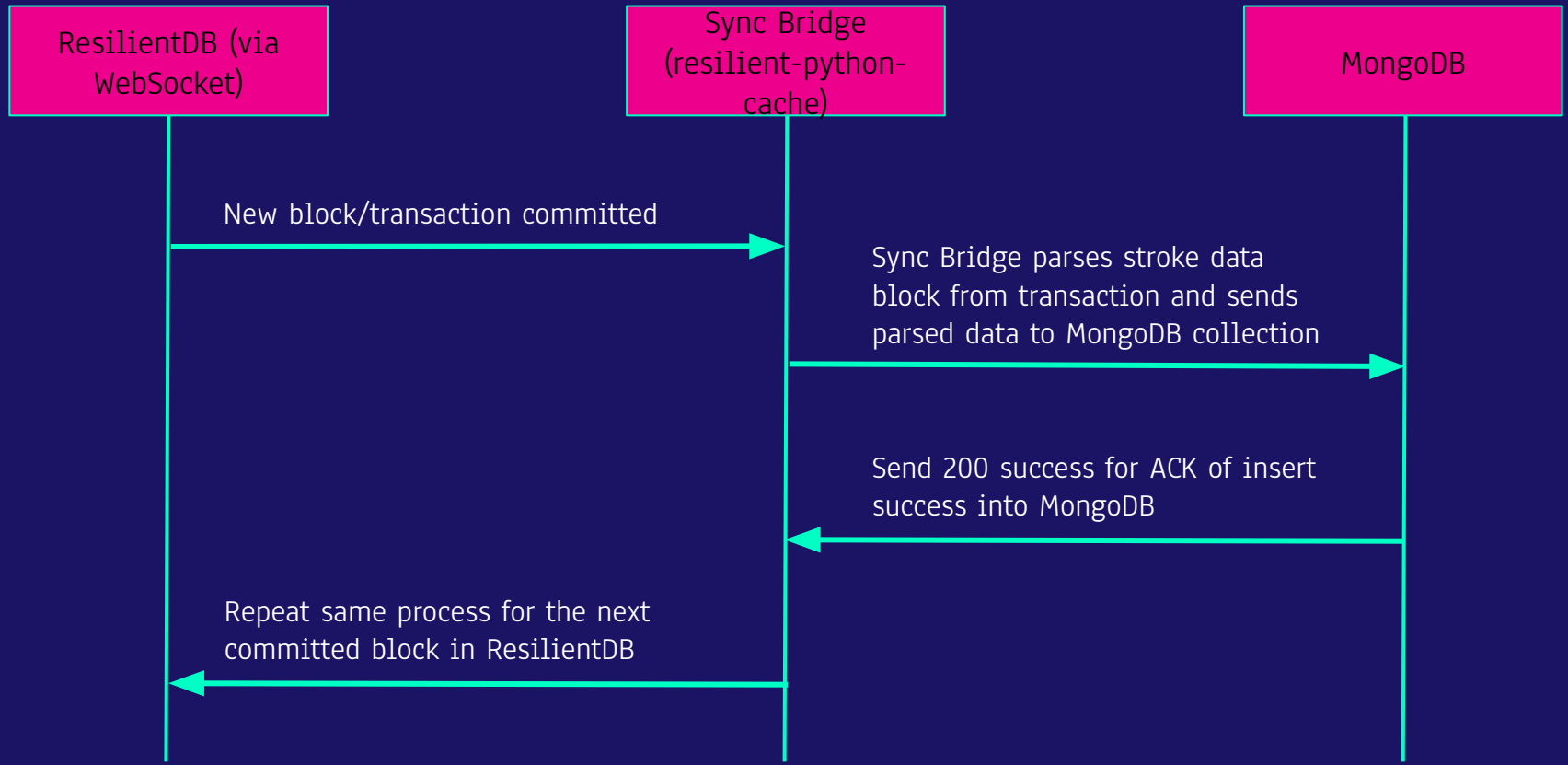
ResCanvas: Stroke Read Path



ResCanvas: Stroke Write Path



ResCanvas: Background Sync Process



ResCanvas: My Core Contributions

- **System Architecture**
  - Designed the end-to-end architecture integrating React, Flask, GraphQL, Redis, MongoDB, and ResilientDB.
  - Implemented secure JWT authentication for login, registration, and session handling.
  - Created GraphQL commit and recovery flows that ensure consistency between Redis and the ResilientDB blockchain database.
- **Backend Development**
  - Wrote all Flask route handlers for submitting new strokes (including new lines and shapes, cut and paste), undo and redo operations, canvas history recall, and room based collaboration.
  - Designed the transaction schema (stroke JSON, user and room related metadata, timestamps).
  - Integrated ResilientPythonCache to automatically fetch committed blocks and sync Mongo collections.
- **Frontend Engineering**
  - Built the modular canvas interface supporting stroke drawing, cut/paste, undo/redo stacks, shape tools, and dynamic selection with real-time synchronization across connected clients using Socket.IO.
  - Developed dashboard and room-selection UI, enabling users to join, create, or share collaborative rooms.
- **Blockchain & Research Layer**
  - Adapted ResilientDB's PBFT based consensus for real-time application workloads.
  - Demonstrated GraphQL based transaction commits, fault recovery, and full stroke auditability.
  - Optimized for low-latency transaction finality while preserving full blockchain integrity.

Demo

Project Repository:
https://github.com/ResilientApp/ResCanvas