

Computer Science — Capstone Project Documentation (2025)

Capstone II team: Luis Delgado (Product Owner and Team Lead), Ana Oliveira (Backend/Frontend developer), Gaeldesh Demosthene (Backend/Frontend developer), Andres Hernandez (Scrum Master and Backend/Frontend developer)

Knight Foundation School of Computing and Information Sciences (KFSCIS) — Florida International University

Instructor: Masoud Sadjadi | Version: 2025 Fall | License: MIT (include in your repo)

Poster: 36"×48" horizontal • Videos: Intro & Demo • Presentation: 10–15 min • Scrum: disciplined, evidence-based.

This template is ACM-aligned and maps to the Capstone grading rubric (Project Documentation = 10%).

ACM Student Outcomes (O1–O6)

Outcome	Description
O1. Problem Analysis & Requirements	Elicit and analyze user/stakeholder needs; define constraints and success criteria.
O2. System Design & Implementation	Design solutions using appropriate abstractions, patterns, and tools; implement maintainable code.
O3. Experimentation, Testing & Evaluation	Devise evaluation methods; collect evidence; interpret results to improve the system.
O4. Communication & Collaboration	Communicate effectively with technical and non-technical stakeholders; collaborate in teams.
O5. Professional, Ethical, Legal & Societal	Recognize responsibilities; address ethics, security, privacy, accessibility, and sustainability.
O6. Algorithmic Thinking & Complexity	Discipline-specific depth outcome; addressed in dedicated sections below.

Outcome & Rubric Crosswalk (Checklist)

Use this to ensure your documentation and artifacts demonstrate each outcome and satisfy the rubric.

Mark each ☐ when complete.

Outcome	Where It Appears in This Document	External Evidence (Links/Appendices)	Status (☐ / ☒)
O1	§2 Problem & Requirements; §3 System Overview	Appendix A (User Research), Backlog, User Stories	☒
O2	§3 System Overview & Architecture; §5 Implementation Notes	Repo structure, Code, CI status badge	☒
O3	§6 Testing & Evaluation	Test reports, coverage, performance dashboards	☒

O4	§1 Executive Summary; §9 Limitations & Future Work	Poster, Slides, Videos	☑
O5	§7 Security/Privacy/Compliance	Threat model / Model card / SBOM	☑
O6	§4 Discipline-Specific Depth	Artifacts noted in §4	☑

1. Executive Summary

Briefly summarize the problem, who benefits, your solution approach, and key results (≤ 300 words).

The problem: many learners struggle to manually summarize lecture slides, notes, research papers, or technical documents, which consumes significant time and often leads to inconsistent study habits. Learners and students need an easy way to condense study materials (lecture notes, practice exams, study guides, etc.) into an easily accessible format without spending the time to create them from scratch. Apart from this, they also need a tool to help recall information from their materials to assist with their learning. BriskRecall solves both of these problems.

Who benefits: BriskRecall benefits university students, exam takers, self-learners, and anyone who relies on memorization and active recall as part of their learning process.

Our solution: we created an AI-powered web application that allows learners to convert their PDF documents like PowerPoints, research papers, and notes into high-quality flashcards. A user can log into their account, upload a file, and give it a title. BriskRecall then extracts the text, parses the content by converting the file to JSON format, and sends clean text to a Gemini AI model to generate cards with question/answer pairs. The user can view generated cards, organize decks, and review them interactively. All flashcards and user data are stored in Supabase for persistent, cross-device access.

Key results: key results include a fully functional system capable of producing meaningful flashcards after upload, stable PDF parsing, seamless login, email verification after sign-up, and reliable deck storage and retrieval. By automating content extraction and flashcard creation, BriskRecall enhances study efficiency and helps learners retain information more effectively.

2. Problem & Requirements (O1)

Context, stakeholders, personas; functional & non-functional requirements; constraints; success criteria; risks.

Include a prioritized backlog (top 10–15 items) and a link to your full backlog board.

Context: BriskRecall operates in the learning-support and study-automation domain, addressing the inefficiency of manually creating study materials from existing ones. Many students lack the time to convert readings into effective memorization and active recall tools.

Stakeholders:

- **Primary Users:** university students, self-learners, professionals preparing for certifications.
- **Project Team**

- **Institutional Stakeholders:** the University and Professor Sadjadi.

Personas:

- **Ana - Computer Science student studying for a final exam:** takes heavy course loads, wants quick flashcard generation from lecture slides in order to recall content and practice questions for a final exam coming up.
- **Miguel – MCAT Applicant:** handles long PDFs for studying and values consistent formatting and AI accuracy.
- **Sara - Software Engineer:** preps for AWS certification, wants mobile access and easy deck organization.

Functional requirements:

- Users can upload PDFs to generate flashcards.
- Users must create an account with a secure password and verify their email address.
- Flashcards are stored and retrieved via Supabase.
- System generates flashcards using Gemini AI.
- Users can view created decks.

Non-functional requirements:

- **Performance:** 15 flashcards generated in ≤ 15 seconds for PDFs < 20 pages.
- **Reliability:** server uptime $\geq 90\%$ during demos.
- **Usability:** intuitive UI accessible with minimal onboarding.
- **Security:** validation of password criteria when creating an account, JWT-based authentication .
- **Compatibility:** supports most web browsers.

Constraints:

- Limited GPU/compute resources (local development).
- PDF parsing accuracy depends on text extraction quality.

Success criteria:

- Decks persist across sessions and devices.
- System remains functional during the final demo.
- Flashcard deck generation after PDF upload takes under 15 seconds.
- Most users in testing consider the flashcards to be accurate and useful.
- User can upload a PDF and receive a deck of 15 flashcards reliably.

Risks:

- **PDF parsing errors:** Scanned PDFs might extract poorly.
- **API limits:** Gemini API quota can block usage.
- **Authentication failures:** Misconfigured tokens prevent uploads.

Prioritized backlog:

- **High Priority:**
 - Users can upload PDFs to generate flashcards.
 - Users must create an account with a secure password and verify their email address.
 - Flashcards are stored and retrieved via Supabase.
 - System generates flashcards using Gemini AI.
 - Users can view created decks.
- **Medium priority:**
 - Optimize response time to meet $\leq 15s$ flashcard generation requirement
 - Cross-browser testing (Chrome, Firefox, Safari)

3. System Overview & Architecture (O2)

Describe the overall architecture. Paste a current architecture diagram (C4 containers/components or equivalent).

Client: The client(user) mainly has access to functions related to the UI. In the interface the user has the option to access login/logout functions, data uploads, and flashcard deck management.

- Login/Logout; Users have the option to create an account, log in using an existing account, or log out from the application.
- Data Uploads: As of right now, users can upload a pdf file and choose a name for the corresponding deck. Using this will create a deck consisting of 15 flash cards. Currently, there are no restrictions on file size.
- Decks: Users can see a list of all the decks they have created. Once the user has chosen a deck the contents of the deck will appear. They can then cycle between the deck, shuffle the cards, flip the cards, or make guesses to test their knowledge. Part of the guessing feature is earning a streak for every correct guess, and incorrect guesses end the streak.

Server: The server handles user authentication, file verification, file transcription, and interactions With Google Gemini.

- User Authentication: Server references database to check if user information (username, email, password) is correct.
- User Creation: Server can add new entries into the database relating to user information (username, email, password).
- File verification/transcription: server confirms that uploaded file type is valid. Then converts the data into usable text to send to Gemini.
- Gemini integration: Once text is extracted from the file/s, 15 flashcards are created using the gemini-2.5-flash model. The flashcards are then sent to the database where they are stored based on user requests.

Database: Handles storing of user profiles, flashcard data, and deck data

- User profiles: Stores information related login/logout and is used to pair flashcards and decks to specific users.
- Flashcard data: Stores the contents of the flashcards (question, answers) and links them to specific decks.
- Deck data: Stores the contents of a given deck and links them to specific users.

Application: Responsible for handling the logic for some functions (deleting decks, shuffling decks, naming decks, etc.) and organizing data to be displayed on the UI.

List key technologies, frameworks, and services. Include API surface (OpenAPI/GraphQL) and data/storage overview.

Frontend

React

Role: Core UI framework for building the interface

Location: /src/*

Key Areas:

- Pages: /src/pages/
- Components: /src/components/
- Global styles: App.css, index.css
- Entry point: main.jsx

Frontend API Surface

- Uses Supabase Client (supabaseClient.js)
 - REST-style operations (generated by Supabase)
 - Realtime features
 - Auth operations (sign-in, sign-up, user session)
- Uses HTTP requests to the Node server for:
 - file uploads (server.js)

Protocols:

- HTTPS (when deployed)
- Fetch/XHR for upload routes

Vite

Role: Development server, bundler

Location: vite.config.js

Capabilities in this project

- Fast HMR during development
- Plugin support for React

- Builds production assets using Rollup
- Serves static files from /public

API Surface

- No backend API
- Provides dev server features only

CSS

Role: Styling for UI

Files:

- App.css
- index.css
- Component-level CSS inside components

Capabilities

- Provides layout, theming, and animations
- No runtime API or storage

Backend

Node.js + Express server (Custom Upload Server)

Location: /server/server.js

Purpose in this project

- Handles file uploads using Express + Multer
- Stores uploaded content in /server/uploads

API Surface (Express REST Endpoints)

Examples :

Endpoint	Method	Purpose
/upload	POST	Handles file upload (Multer)
/uploads/:filename	GET	Serves uploaded files

Protocol: JSON / Multipart Form Data

Supabase (Primary Backend Service)

Location: /src/supabaseClient.js

Authentication

- Email/password login
- Session management

Database (PostgreSQL)

- Storing:
 - Flashcards
 - Decks
 - AI generated content metadata
- Accessed via Supabase's **auto-generated REST API**

Storage

- For media (text data)

Realtime

- Live updates (e.g., changes to flashcards or deck lists)

Supabase API Surface

1. REST API (PostgREST / OpenAPI)

Features:

- CRUD operations via HTTPS
- Filtering using query parameters
- OpenAPI schema generated automatically

Example:

GET /rest/v1/flashcards

POST /rest/v1/flashcards

2. Supabase Auth API

- /auth/v1/signup
- /auth/v1/token
- /auth/v1/user

Handled automatically by the JS client.

4. Data & Storage Overview

Layer	Technology	Storage Type	Purpose
Frontend	React	Browser/Local	UI state, page navigation
Frontend Dev	Vite	Build Cache	Bundling + dev server
Node Server	Express + Multer	Local filesystem (/server/uploads)	Temporary or permanent file

			storage
Backend	Supabase DB	PostgreSQL	Flashcards, decks, users
Backend	Supabase Auth	Postgres schema	Authentication + sessions
Backend	Supabase Storage	S3-like	Files storage

Architecture Summary

Supabase = database, auth, optional storage

Node server = file upload pipeline

React UI = main user interface

5. Implementation Notes (O2)

Repository structure, coding conventions, notable patterns, and tradeoffs. Reference the README and module layout.

Repository structure

The repository is organized with a React+Vite frontend and Node.js backend.

- **flashcards-project/** - main project directory
 - **public/** - static assets served by Vite
 - **src/** – all frontend logic
 - **pages/** - route-level views such as Home, Upload, SignIn, SignUp, Decks, and DeckView.
 - **components/** - reusable UI components (Nav, Card, UploadForm)
 - **assets/** - images and icons
 - **supabaseClient.js** - initializes the Supabase client using environment variables (Vite's `import.meta.env`)
 - **App.jsx, App.css, main.jsx** - core React entry points
 - **server/** - backend service
 - **server.js** - Express server entry point with upload endpoint, Supabase authentication, PDF parsing, and Gemini flashcard generation logic
 - **uploads/** - temporary PDF upload directory used during development
 - **package.json / package-lock.json** – backend dependencies
 - Project-level configs: README.md, vite.config.js, .gitignore, eslint.config.js, index.html

Coding conventions

- **React conventions:** functional components, hooks, PascalCase filenames, co-located CSS imports, and clean routing with <Routes> and <Route> components.
- **Styling:** custom CSS variables for theme consistency; PicoCSS is imported globally for baseline styling resets.
- Supabase configuration is centralized in supabaseClient.js, which reads VITE_SUPABASE_URL and VITE_SUPABASE_ANON_KEY.
- Backend code uses **CommonJS** (require(...)) consistently.
- All secrets (Supabase service key, Gemini key) are stored in .env files
- Upload handling uses multer with memory storage, PDF text extraction uses pdf2json, and flashcard generation uses the Gemini model.

Notable patterns

- Clean client-server separation
- Prompt pipeline pattern in the backend: receive PDF (multer), extract text (pdf2json), send prompt to Gemini, clean Gemini output, return JSON flashcards
- Component-based frontend: Pages (e.g., Decks.jsx, DeckView.jsx) load data and pass props down to reusable components like Card.jsx
- Environment-based configuration
- Local state managed by components and Supabase as data layer

Tradeoffs

- Single-file backend design: all backend logic is implemented by server.js, which makes the backend simple and fast to build but harder to scale.
- In-memory file uploads: multer.memoryStorage() stores uploaded PDFs in RAM. It is very fast, good for small files, with zero disk writes, but cannot handle large PDFs and has high RAM usage for big uploads.
- Synchronous AI Generation: /api/upload route waits for Gemini to finish before responding. There may be AI latency.
- Frontend handles database inserts: this reduces backend complexity, but more work is done on the client-side and errors must be handled in the browser.
- No caching or chunking of PDFs

6. Testing & Evaluation (O3)

Strategy (unit/integration/e2e), coverage, performance and reliability testing; KPIs/metrics; defect summary.

Unit Testing:

Individual components and functions are tested in isolation, including Card flip logic, deck navigation functions (handleNextCard/handlePreviousCard), and array manipulation utilities (shuffleArray).

Integration Testing:

Tests verify proper interaction between frontend and backend endpoints, including /api/upload and Supabase database queries. All major data flows—PDF upload → AI flashcard generation → database persistence—are validated.

End-to-End (E2E) Testing & Performance:

Simulated user flows—including signup, PDF upload, flashcard review, and deck management—are validated.

Known defects observed:

- Occasional parsing errors on malformed PDFs.
- Flashcard shuffle inconsistencies where order repeats unexpectedly.
- Minor UI misalignment on small screen resolutions.
- Rare delays in AI flashcard generation under heavy PDF loads.

7. Security, Privacy, Accessibility & Compliance (O5)

Summarize how the solution addresses ethical, legal, and societal impacts; include security, privacy, and accessibility considerations.

BriskRecall is designed to support students by automating flashcard generation from their study materials. Because it processes user-uploaded PDFs and interacts with third-party AI services, the solution incorporates several ethical, legal, and societal safeguards.

Ethically, the system protects user autonomy by requiring explicit user authentication before any document is uploaded. Users maintain full control over their content; the system only generates flashcards and does not reuse or share uploaded materials. The application avoids academic dishonesty concerns by focusing on comprehension and active recall rather than generating assignments or solutions.

Legally, BriskRecall handles documents that may contain copyrighted or proprietary content. To address this, the system does not store PDF files on the backend and only processes text transiently through memory-based parsing. The use of Supabase’s managed authentication and database ensures compliance with modern data protection practices. Environment-based configuration (.env files) prevents hard-coding credentials, reducing legal exposure related to data breaches.

The tool aims for a positive **societal impact** by equitable access to study support by reducing the burden of creating learning materials. It benefits students with limited time, those balancing work and school, and non-native English speakers who may struggle with complex texts. The solution enhances learning efficiency without replacing learning practices that promote critical thinking.

Security and privacy are addressed through JWT-based authorization (only authenticated users can access AI generation), secure Supabase session handling, and encrypted communication channels when deployed. The backend never stores PDFs, and sensitive keys (Supabase service key, Gemini API key) remain protected via environment variables.

Accessibility considerations include a clean UI, large text options, simple navigation, responsive layouts, and dark/light themes. The interface avoids cognitive overload.

In summary, BriskRecall’s architecture and features minimize ethical risks, respect user privacy, and promote fair, secure, and accessible learning support.

8. Deployment & Operations

Briefly describe how the system is deployed and operated; link to detailed guides in the appendices.

BriskRecall is deployed as a lightweight full-stack application with a separate frontend and backend. The **frontend** runs on Vite’s development server and serves the React interface. The **backend** operates as a Node.js/Express server that exposes the `/api/upload` endpoint for PDF processing.

The system relies on **Supabase** for authentication and database storage, requiring configuration of the project URL, anon key (frontend), and service role key (backend).

Environment variables are managed through two `.env` files—one for the client and one for the server—to protect API keys and connection details. Google Gemini is accessed via the `@google/generative-ai` SDK, which also requires API key configuration.

Once both servers are running locally, users authenticate through Supabase, upload a PDF through the UI, and the backend generates flashcards which the frontend saves into Supabase.

9. Limitations & Future Work

Known limitations, shortcuts, technical debt, and prioritized roadmap items for future iterations.

Limitations:

- PDF parsing can fail or produce incomplete flashcards for unusually formatted or very large PDFs.
- The current flashcard review interface supports only sequential navigation.

Future Improvements:

- Improve PDF text extraction robustness and error handling.
- Improve guess answer text input for it to only match certain parts of the flashcard answer.
- Integrate user analytics and progress tracking dashboards.
- Enhance UI/UX with responsive design improvements, accessibility, and mobile optimization.

- Implement real-time collaboration for shared decks between users.

10. References

Use a consistent citation style (e.g., IEEE/ACM).

Node.js, “*Node.js: JavaScript runtime*,” Node.js Foundation, [Online]. Available: <https://nodejs.org/>. [Accessed: Dec. 3, 2025].

npm, “*npm: Node Package Manager*,” npm, [Online]. Available: <https://www.npmjs.com/>. [Accessed: Dec. 3, 2025].

React, “*React — A JavaScript library for building user interfaces*,” Meta, [Online]. Available: <https://reactjs.org/>. [Accessed: Dec. 3, 2025].

React Router, “*React Router v6*,” Meta, [Online]. Available: <https://reactrouter.com/>. [Accessed: Dec. 3, 2025].

Supabase, “*Supabase: Open Source Firebase alternative*,” [Online]. Available: <https://supabase.com/>. [Accessed: Dec. 3, 2025].

Multer, “*Multer: Node.js middleware for handling multipart/form-data*,” [Online]. Available: <https://www.npmjs.com/package/multer>. [Accessed: Dec. 3, 2025].

pdf2json, “*pdf2json — npm Package*,” [Online]. Available: <https://www.npmjs.com/package/pdf2json>. [Accessed: Dec. 3, 2025].

dotenv, “*Dotenv — Loads environment variables from .env file*,” [Online]. Available: <https://www.npmjs.com/package/dotenv>. [Accessed: Dec. 3, 2025].

Google Generative AI API, “*Google Generative AI documentation*,” [Online]. Available: <https://developers.googleai.google/>. [Accessed: Dec. 3, 2025].

Pico.css, “*Pico: Minimal CSS framework*,” [Online]. Available: <https://picocss.com/>. [Accessed: Dec. 3, 2025].

Appendices (Evidence & How-To)

A. Installation Guide (step-by-step with screenshots)

Check the BriskRecall Master Testing/Setup Guide: [Click Here](#)

B. User Manual (task-oriented walkthroughs)

Check the BriskRecall Master Testing/Setup Guide: [Click Here](#)

C. Admin/Operations Guide (runbook, on-call, backup/restore)

Check the BriskRecall Master Testing/Setup Guide: [Click Here](#)

D. API Reference (OpenAPI/GraphQL schema)

User Authentication

- POST /auth/signup — Register a new user
- POST /auth/signin — Sign in an existing user

Decks

- GET /decks?user_id=<user_uuid> — Fetch all decks for the current user
- POST /api/upload — Upload PDF and generate a deck using AI
- DELETE /decks/:id — Delete a deck and cascade delete flashcards

Flashcards

- GET /flashcards?deck_id=<deck_uuid> — Fetch all flashcards in a deck

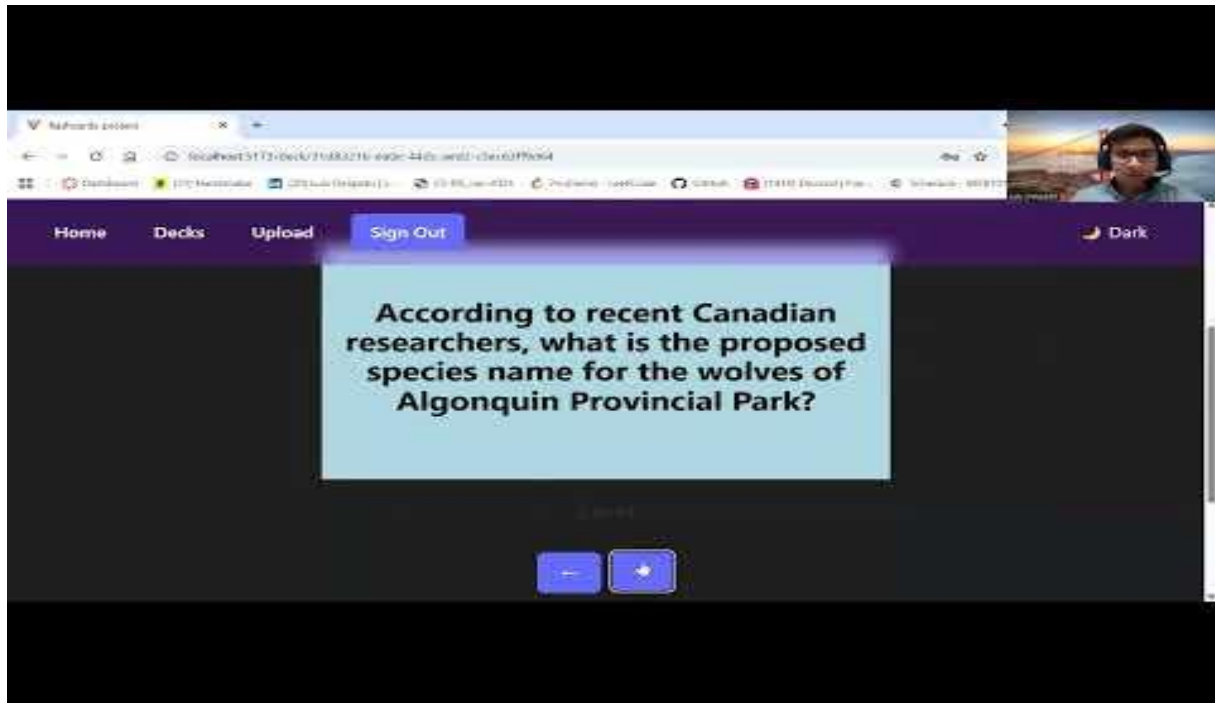
E. Poster (36"×48" horizontal), Slides, and Video Links (Intro, Demo)

Intro Video:

<https://youtu.be/or0020-0lF4?si=wpxvzdlGfo0d-AZi>

Demo Video:

<https://youtu.be/S1LjWKJfRUw?si=jtzoQatV1RiDEyD>



Poster Link -> [BriskRecallPoster](#)

Slides -> [BriskRecallSlides](#)

F. Scrum Evidence Index (links to Sprint Planning, Dailies, Reviews, Retros)

Folder with all links to Sprint Planning, Dailies, Reviews, Retros Link: [Scrum Cycles](#)

4. Discipline-Specific Depth (O6)

- Algorithms & Data Structures: key choices, complexity analysis, and performance implications.

Algorithms & Data Structures

BriskRecall uses a simple relational schema in Supabase (users → decks → flashcards) for efficient lookups. On the frontend, flashcards are stored in arrays; navigation is $O(1)$, while shuffling uses `sort()` for $O(n \log n)$. The upload pipeline is effectively $O(P)$ in PDF size, but real performance is dominated by I/O (PDF parsing, Supabase, Gemini). Overall, costs are mainly linear over small arrays and external latency, not heavy computation.

Architecture & Patterns

- **Frontend (React + Vite):** Handles UI, routing, and Supabase CRUD. Pages are split into reusable components.

- **Backend (Node + Express):** Exposes /api/upload for PDF parsing and Gemini flashcard generation.
- **Cloud (Supabase + Gemini):** Supabase manages auth and data; Gemini generates Q&A.
- **Concurrency:** Managed via async I/O (Node event loop + async/await), allowing multiple requests without threading.

BriskRecall separates **UI state** and **persistent state**. React handles transient state (auth session, form inputs, loading flags, flashcard position, feedback) using useState and useEffect. Persistent data (users, decks, flashcards) lives in Supabase and is fetched as needed. Authentication is managed by Supabase's auth listener (onAuthStateChanged) and controls protected routes. The backend is stateless, validating JWTs per request and returning results without storing sessions.