Design Thinking Stages for AppLens - Group 8

Github link - https://github.com/TanmayRanaware/AppLens

1. Empathize

Goal: Understand the pain points of developers and DevOps engineers who debug microservice systems.

Deliverables:

- User Research Plan:
    1. Target users - Backend developers, DevOps engineers, site reliability engineers (SREs).
    2. Methods- Interviews, surveys, observing debugging sessions, analyzing GitHub issue threads

- Observation Notes:

  Quotes like “It takes me hours to trace an error across services.” and “A small API change can break multiple test suites downstream.”

- Empathy Map:

Persona 1:Sam, DevOps Engineer

Says:

“Every time there’s an alert, I have to jump between five tools.”

“By the time I find the root cause, the system is already unstable.”

Thinks:

“If only I could see how one service failure affects others instantly.”

Does:

Digs through logs, traces, and dashboards manually

Pings teammates to confirm which service might be impacted

Feels:

Frustrated with fragmented data

Relieved when issues are finally resolved, but exhausted

Persona 2: Priya, Backend Developer

Says:

"I just want to know which service broke without hunting for hours."

"I hate writing postmortems after every incident."

Thinks:

"If the system could show me dependency links, I'd fix things faster."

"Debugging shouldn't be this stressful."

Does:

Reviews error logs and traces to find the faulty API

Collaborates with DevOps to isolate the issue

Feels:

Curious but often confused during root cause analysis

Overwhelmed when logs contradict each other

- Journey Map

Phase	Action	Emotion	Pain Point	Opportunity
Initial Diagnosis	Team opens traces, logs, dashboards	Frustrated	Information is scattered across tools	Centralize data and let users see everything in one place
Deep Dive	Correlate service interactions manually	Confused	Hard to connect cause and effect across services	Automatically map and visualize dependencies
Change Assessment	Evaluate what-if or rollback scenarios	Risky	No clear visibility on what changes might break	Simulate impact of code or config changes before deploy

- Key Insights:
    1. Developers struggle with fragmented data and lack of visibility .
    2. They want faster, visual, and intelligent root cause analysis tools

2. <u>Define</u>
Goal: Turn empathy insights into a clear, actionable problem statement for AppLens.
Deliverables:

- Point of View (POV) Statement:
  Developers who manage microservice systems need a smarter way to understand service dependencies and trace errors quickly, because current tools are too slow and fragmented.
- How Might We (HMW) Questions:
  1. How might we help developers instantly identify the root cause of errors?
  2. How might we visualize microservice connections?
  3. How might we let teams query their architecture in plain English?
- Problem Framing Document / Storyboard:
  Storyboard showing:
  1. A production error is detected and triggers an alert.
  2. The on-call engineer initiates the investigation workflow.
  3. AppLens automatically visualizes relevant service dependencies and highlights the most probable source of failure.
  4. The engineer reviews supporting evidence, including recent pull requests, deployment logs, and configuration changes.
- User Personas:
  1. DevOps Engineer “Sam” – manages large-scale microservice deployments
  2. Backend Developer “Priya” – wants to debug API failures faster

---

3. <u>Ideate</u>
Goal: Brainstorm and design creative solutions to the defined problems.
Deliverables:

- Brainstorming Outputs / Idea Sketches:
  Ideas like “AI-based error log analyzer,” “interactive dependency graph,” “natural language query system.”
- Concept Clusters or Prioritization Matrix:
  1. Group ideas into categories (Debugging, Visualization, Q&A, Impact Analysis)
  2. Prioritize based on feasibility (AI + existing data) and impact (developer productivity)
- Storyboard or Concept Cards for Top Ideas:
  1. Concept 1: “AI Error Analyzer” : paste a log → see root cause & impact
  2. Concept 2: “Service Graph Visualizer” : auto-generate architecture map
  3. Concept 3: “What-If Simulator” : test code changes before deployment