Cloud-Native Scalable Personal Expense Tracker

Final Project Presentation -Spring 2025

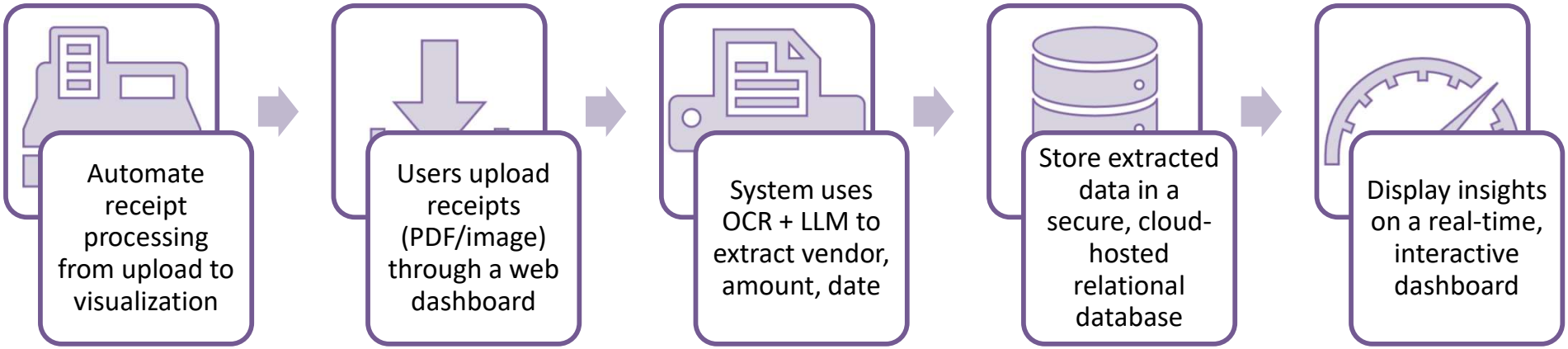
Team Members:

- Abhishek Balasaheb Bhingle (abhingl@iu.edu)
- Shubham Sandip Salunke (ssalunke@iu.edu)
- Aditya Nitin Pise (anpise@iu.edu)

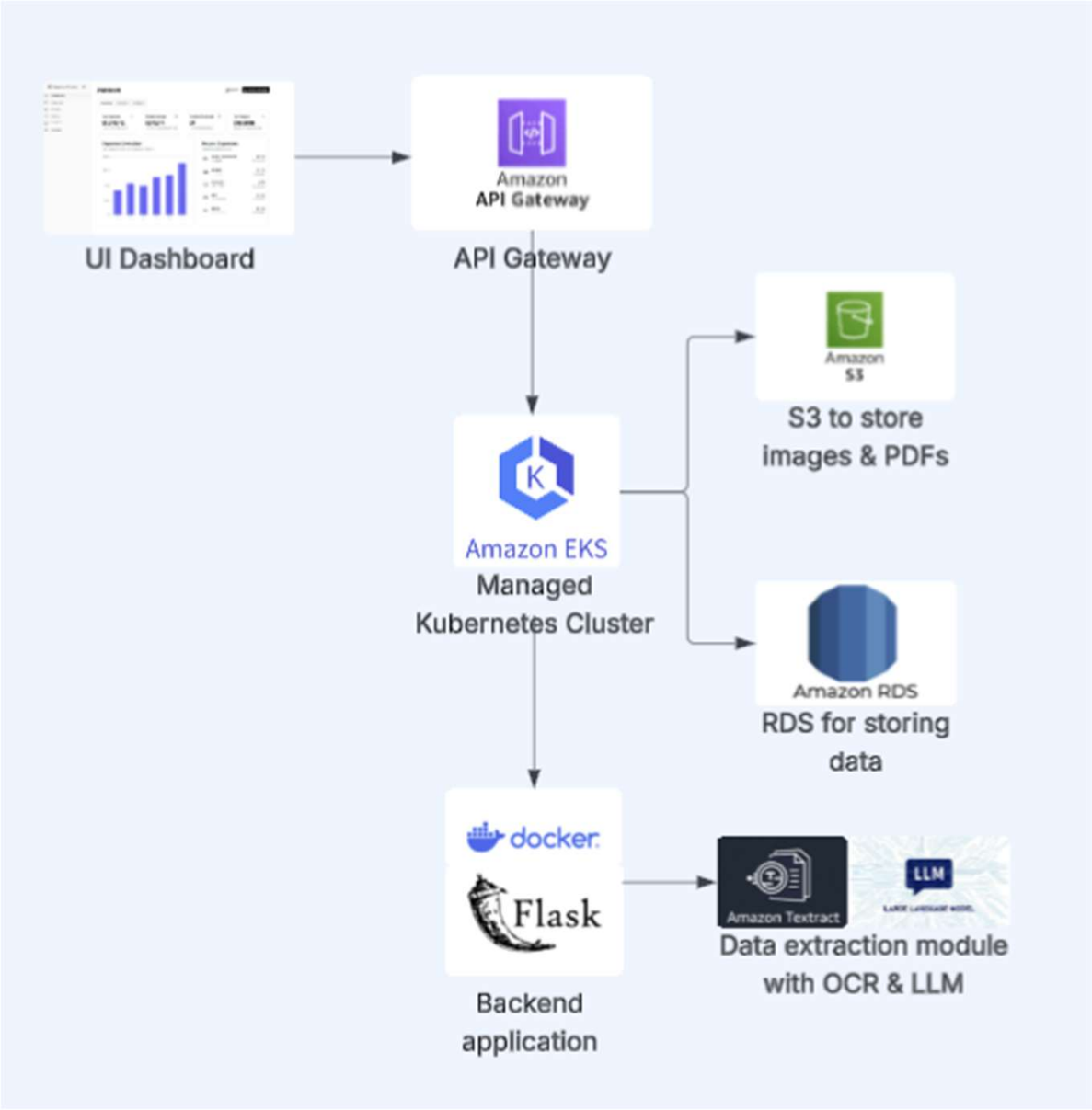
Problem Statement

- Tracking expenses is something everyone knows they should do — but very few enjoy doing it.
- Most people still rely on old-school methods: snapping photos of receipts, manually entering numbers, or letting them pile up.
- This manual process is not only tedious — it often leads to forgotten expenses, misclassifications, and incomplete data.
- More importantly, people don’t get immediate insights from their spending — they’re stuck reviewing their finances reactively, not proactively.
- There’s a clear need for smarter, automated tools that simplify this essential part of everyday life.

Proposed Solution



Architecture



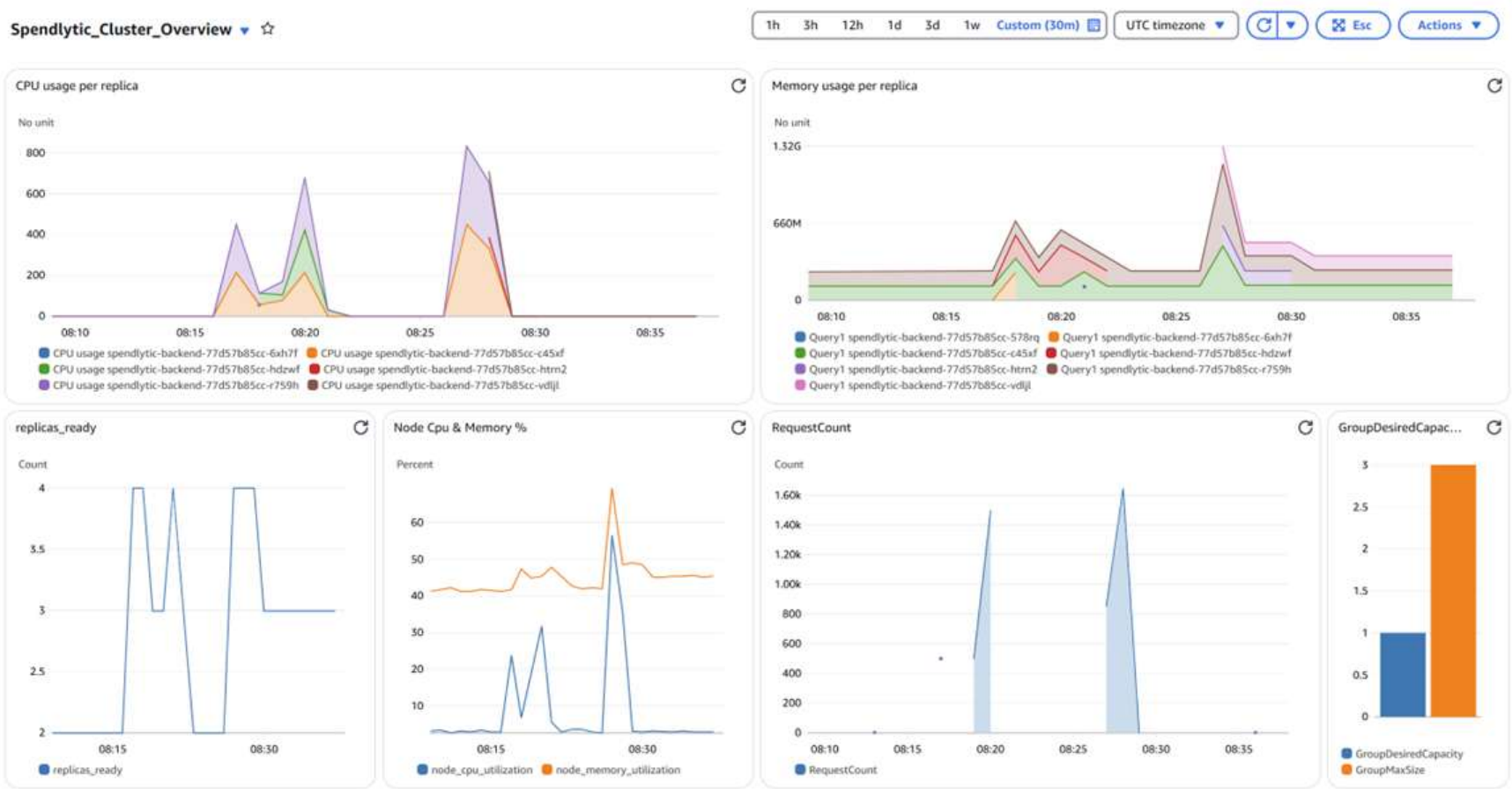
Implementation

- **Modular, Scalable, Cloud-Native:** Built end-to-end for rapid expansion and easy maintenance
- **User Interface:** React dashboard for receipt uploads and instant, interactive expense visualizations
- **Backend Logic:** Flask API processes files and integrates services, each component Docker-containerized
- **Cloud Services:** AWS API Gateway, S3 for receipts, RDS for expense data, and EKS for auto-scaling workloads
- **Security & Access:** IAM policies, HTTPS encryption, and CORS controls ensure proper permissions and data safety
- **Operational Outcome:** Highly responsive, enterprise-grade secure, and ready to scale on demand

Kubernetes Cluster Deployment

- **Cluster Platform**: AWS Elastic Kubernetes Service (EKS)
- **Node Instance Type**: t3.medium
- **Node Group Size**:
  - Minimum: 1 node
  - Maximum: 3 nodes
- **Deployment**: Backend application via Kubernetes Deployment manifest
- **Pod Autoscaling (HPA)**:
  a) Enabled with Horizontal Pod Autoscaling
  b) Metrics: CPU & memory utilization
  c) Replica range: 2–6 pods
  d) Scaling intervals:
    a) Scale up every 20 seconds
    b) Scale down every 60 seconds
- **Configuration Management**: All resources defined using declarative YAML scripts

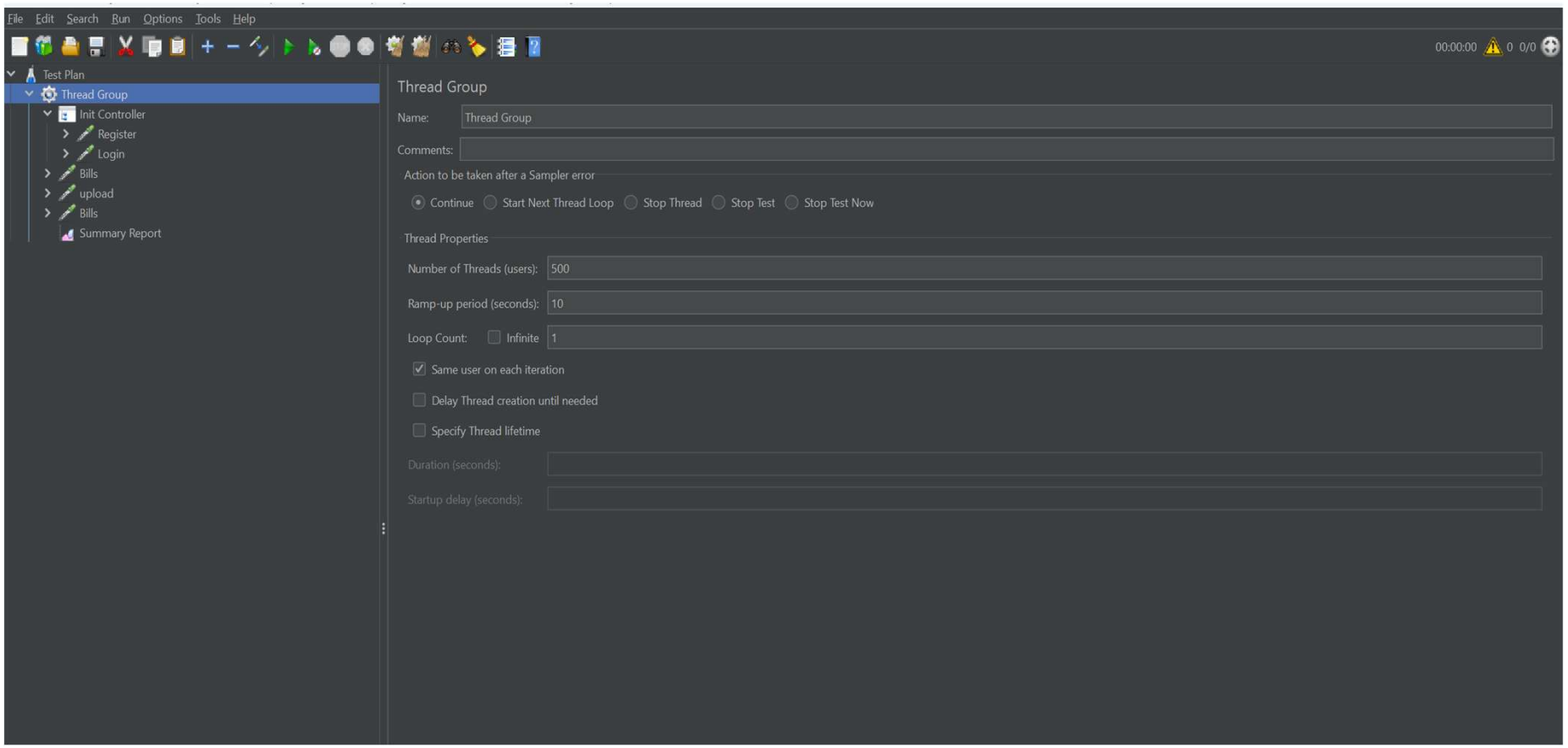
Cluster Overview Dashboard



Cluster Overview Dashboard

- Key Insights:
- Tested with **500 users** (~1.6K requests)
- CPU peaked at **~800 mCPU**, memory at **~1.3 GB** per pod
- HPA scaled pods **2 → 4** (bounds: 2–6)
- Node CPU ~**60%**, memory ~**50%**

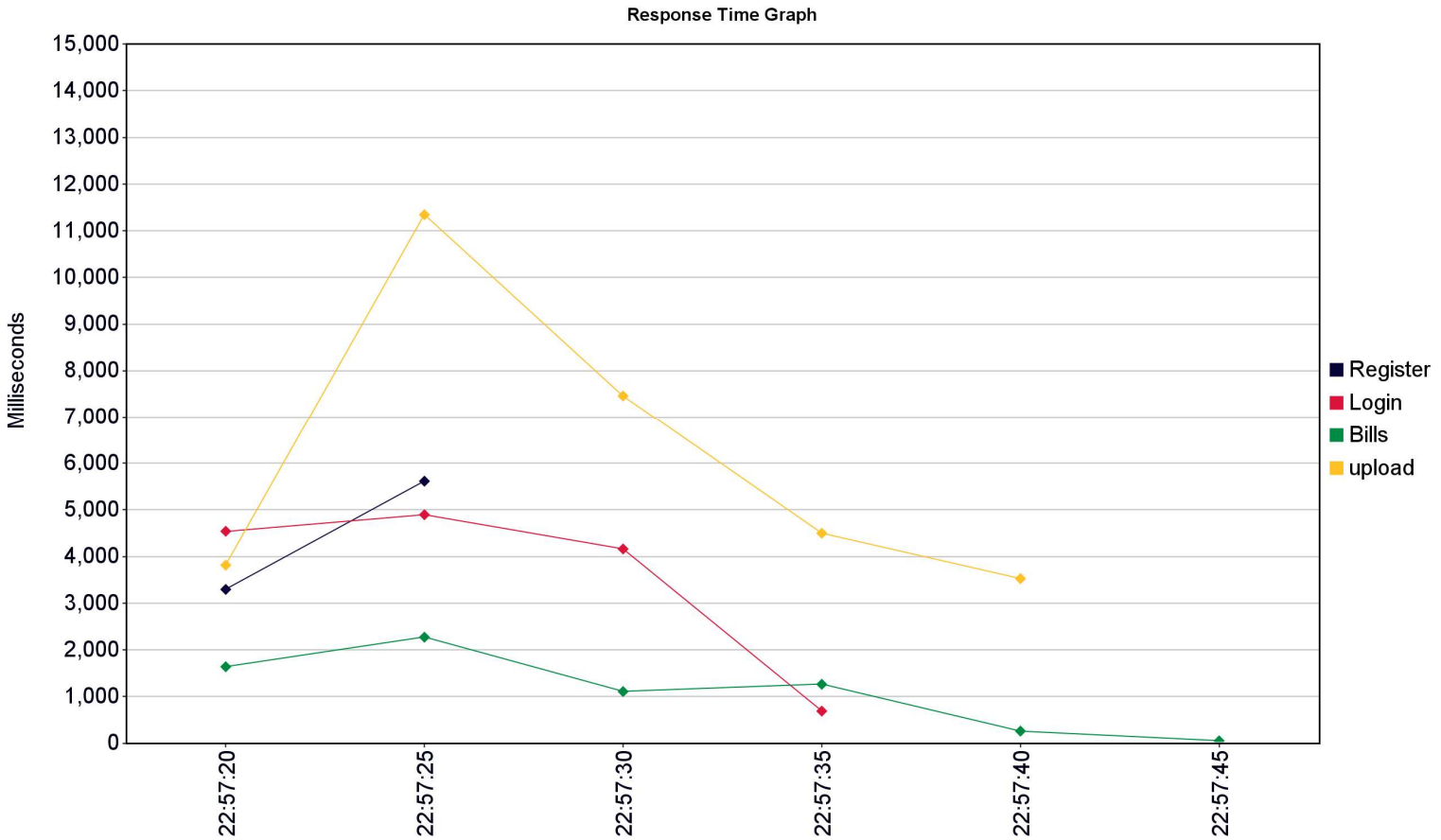
Load Testing – Jmeter



Load Testing – Jmeter

- Simulated **500 users** arriving over a **10-second** span
- Each user went through the full flow **once**
- Steps performed by each user:
  - Register
  - Log in
  - View bills
  - Upload a file
  - View bills again
- Collected a **summary report** of overall system behavior under load

Latency Comparison



Latency Comparison

- **Upload incurs the highest overhead**
  - Peaks at ~11 s on the second sample before tapering to ~3–4 s
  - Average latency ~5.7 s, 16 % error rate—reflecting OCR & LLM extraction work
- **Register shows moderate, consistent performance**
  - Averages ~4.8 s (max ~10.4 s) with 0 % errors
  - Throughput ~3.8 req/sec
- **Login warms up quickly**
  - Initial latency ~4.2 s drops below 0.7 s on subsequent calls
  - Average ~4.2 s overall, 0 % errors, ~3.3 req/sec
- **Bills is the fastest and most predictable**
  - Averages under 0.9 s (max ~5.1 s) with 0 % errors
  - Highest throughput at ~4.2 req/sec
- **Clear downward trend across all endpoints**
  - Latencies decrease over the 25 s window, **showing system warm-up and effective autoscaling**
  - Total average latency ~3.3 s, combined throughput ~9 req/sec

Demo

Here's how it works from a user perspective:

- Step 1: Open the dashboard and upload a receipt
- Step 2: The backend receives the file, saves it to S3, and launches our OCR + LLM pipeline
- Step 3: Extracted data — like vendor name, date, and total — appears on the dashboard
- Step 4: Users can search, filter, and visualize their expenses in real-time

Conclusion

- Solved a real problem made personal expense tracking effortless
- Cloud-native architecture ensures scalability, security, and easy expansion
- Automated the most painful steps, delivering instant, actionable insights
- Proven reliability under load, with dynamic autoscaling (2 → 4 pods)
- **Next steps:**
  - Smarter receipt extraction with next-gen LLMs
  - Custom categories, budgets & real-time alerts
  - Full multi-currency & multilingual receipt support

Contribution of the Project

Aditya

- Introduced an intelligent, cloud-native approach to automating receipt-based expense tracking.
- Designed and deployed a fully scalable backend using Docker, Flask, AWS, and Kubernetes.

Shubham

- Enabled real-time financial visibility through a modern React dashboard with dynamic insights.
- Demonstrated practical use of OCR + LLMs to transform unstructured receipt data into structured records.

Abhishek

- Emphasized data privacy and security through token-based authentication and HTTPS flows.
- Validated the system's stability by testing different payload sizes and ensuring consistent performance under file-heavy requests.

Q&A

Thank you!

We welcome your questions and feedback!