Case Study Report

1. Mobile App (React Native)

Problem/Observation: Existing React Native app experiences delay in major user flows like course list loading and chat updates. Navigation between screens is inconsistent, and onboarding is unclear, which causes increased user drop-offs. Large image assets and unnecessary libraries also contribute to increasing bundle size, affecting download/install times. Offline mode is poor, and users end up reconnecting more often.

Proposed Improvement:

- Improve FlatList performance with `getItemLayout`, `React.memo`, and windowed rendering when dealing with long course lists.
- Add skeleton loaders and caching methods (AsyncStorage, react-query) for smoother perceived performance.
- Make navigation smoother with deep linking support and more intuitive onboarding (guided tooltips, progress indicators).
- Minimize app size by compressing images (WebP), enabling Hermes JS engine, and stripping unused dependencies.
- Use offline-first design with caching of key data and retry policies for network requests.

Why it matters (Impact): These optimizations decrease latency below 100ms for typical UI operations, enhancing retention and user confidence. A smoother onboarding process decreases early churn, while offline features guarantee reliability in low-connectivity environments (crucial for student users). A lighter app enhances adoption, especially in bandwidth-restricted environments.

2. Backend (Node.js, Express, MongoDB, Kafka)

Problem/Observation: Backend is monolithic, and scaling is not easy when usage increases. Many database queries are not indexed correctly, so reads/writes for huge data sets are slow. Kafka events have no schema validation yet, which can introduce data inconsistency between microservices. Event workflows are tightly coupled, making them less resilient.

Proposed Improvement:

- Split backend into domain-driven microservices (User, Courses, Chat, Notifications), each with its own database.
- Implement MongoDB indexing techniques (compound indexes, TTL indexes for logs, text indexes for search). Employ schema validation with Mongoose to enforce structure.
- Use Confluent Schema Registry with Avro/Protobuf for Kafka events to provide strict schema validation and safe evolution.
- Implement event-driven orchestration patterns using Kafka Streams or ksqlDB for workflows such as course enrollment or real-time chat.

Why it matters (Impact): Microservices enhance scalability and reliability, while query optimization slashes response times by half. Schema validation eliminates the expensive debugging of corrupted or malformed events. Event-driven workflows render the system reactive, fault-tolerant, and scalable to future increases.

3. Infrastructure (AWS + Kubernetes)

Problem/Observation: Today's AWS configuration is not well utilized with some services provisioned in excess and others without autoscaling. Spikes in traffic are not managed well, and there is a risk of downtime. Media and static assets are directly served from app servers rather than a CDN. Disaster recovery (DR) is minimal with no automated backups and multi-region plan.

Proposed Improvement:

- Run services on EKS (AWS on Kubernetes) with Horizontal Pod Autoscaler (HPA) for scalability.
- Utilize AWS ALB/NLB for smart load balancing and CloudFront CDN for hosting static assets worldwide.
- Implement AWS ElastiCache (Redis) for caching high-hit data.
- Automate backup and restore policies with AWS Backup and multi-region replication for mission-critical databases.
- Implement Infrastructure-as-Code (IaC) through Terraform or AWS CDK for repeatable and predictable deployments.

Why it matters (Impact): Autoscaling is cost-effective by reducing in idle hours and automatically coping with bursts. CDN enhances latency for worldwide users, and Redis caching decreases database loading. A solid DR plan ensures business continuity, which is critical when scaling to thousand-strong universities.

4. Security

Problem/Observation: APIs at present utilize basic authentication, which is at risk of replay attacks. There is no rate limiting, and endpoints are exposed to brute-force or DoS attacks. AWS IAM roles are too general and are too permissive. Secrets are stored in plain text in CI/CD pipelines, and dependency scanning is manual.

Proposed Improvement:

- Implement JWT-based authentication with role-based access control (RBAC) for APIs.
- Implement rate limiting and API gateway throttling for abuse protection.
- Implement least privilege IAM policies and switch on AWS GuardDuty, WAF, Shield for protection of infrastructure.
- Harden CI/CD by keeping secrets in GitHub Actions Secrets / AWS Secrets Manager, incorporating image scanning (Trivy, Snyk) and dependency audits.
- Support HTTPS everywhere with TLS certificates controlled by AWS ACM.

Why it matters (Impact): Enhanced security minimizes breach risks, safeguards user information, and guarantees compliance with privacy laws. Secure CI/CD pipeline stops supply chain attacks. Least privilege policies reduce blast radius on compromise, providing assurance to universities relying on the platform.

5. System Design & Scaling

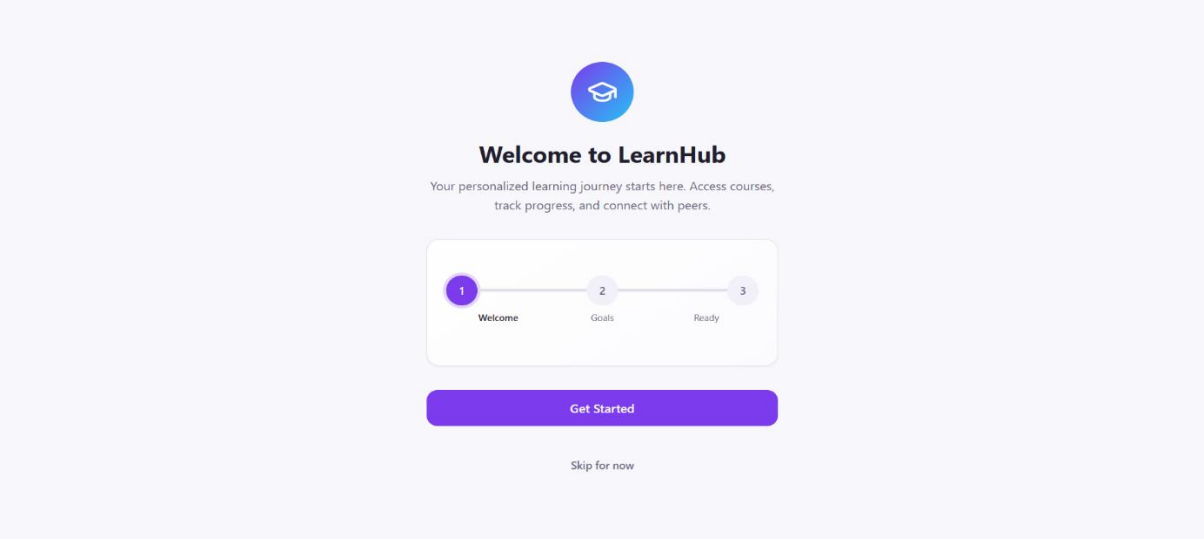
Problem/Observation: Since the platform is growing to span multiple universities, user count could grow 10x. Existing caching is minimal, monitoring is rudimentary (logs only), and chat/notifications are inextricably bound with the core backend. Scaling real-time features such as chat would be problematic under load without architectural changes.

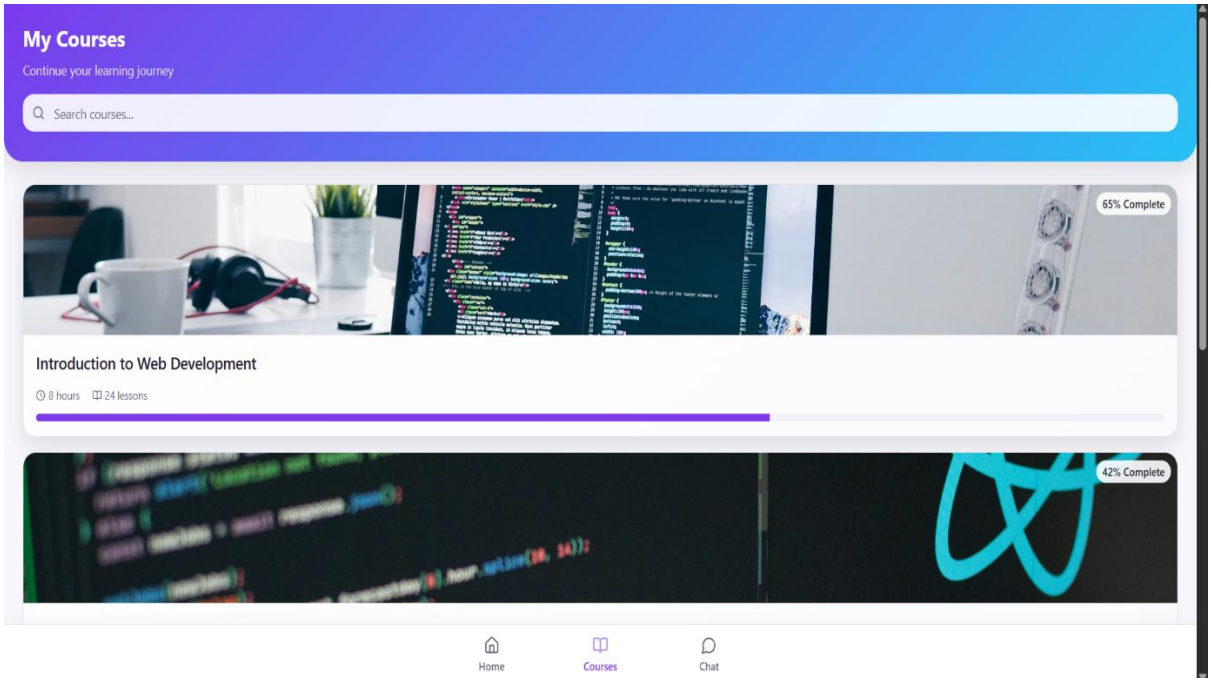
Proposed Improvement:

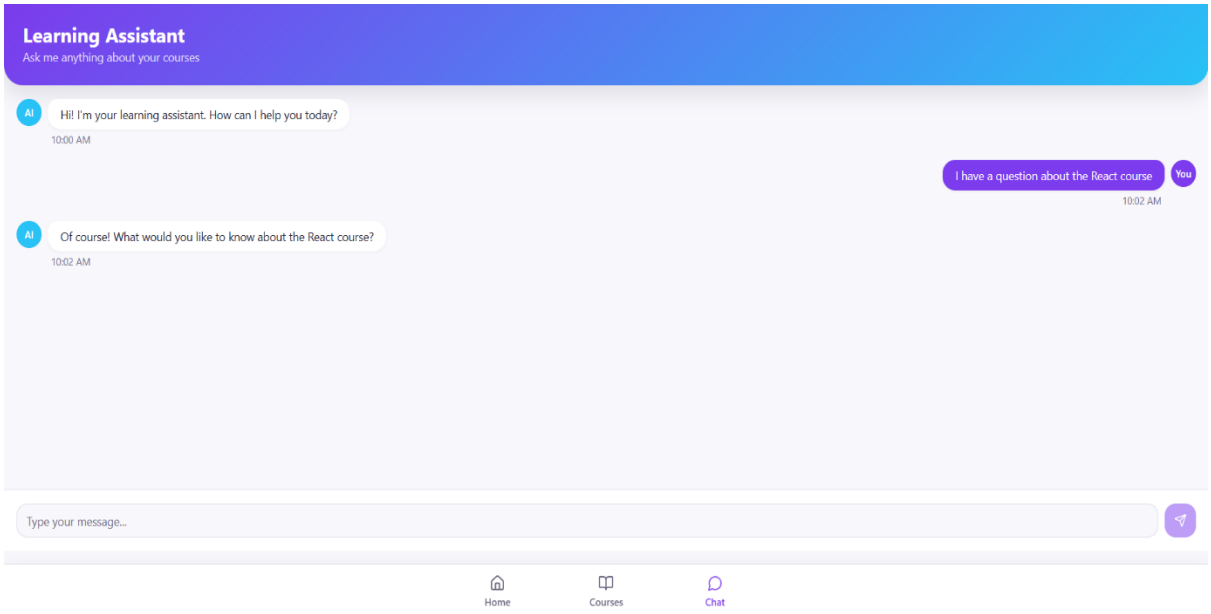
- Add Redis caching for session, query results, and pub/sub notifications.
- Apply CDN strategies for static content, with cache invalidation policies.
- Establish observability stack: Prometheus + Grafana for metrics, ELK/EFK for logs, and OpenTelemetry for tracing.
- Segregate chat/notifications into a real-time service via WebSockets or MQTT, with Kafka for message distribution at scale.
- Follow multi-tenant design so new universities can be onboarded without affecting others.

Why it matters (Impact): Caching and CDN minimize database load and latency, allowing scale-up to 10x traffic with smoothness. Observability gives pre-emptive warnings and quicker problem-solving. A specific real-time service guarantees low-latency chat and notification even under heavy usage. Multi-tenancy design guarantees that the platform can grow quickly without expensive rewrites.

Screens Example:







Prepared by: Paluri Mythri Prasanna

Internship Assignment – LogikSutraAI Connections Pvt Ltd