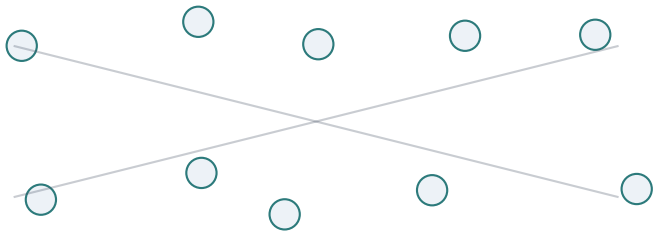
THE ARCHITECT'S GUIDE TO

SHIKSHAK

Building the

Next Generation of
Intelligent Educational Platforms



Technical Specification & Architecture Blueprint

Prepared for Internal Stakeholders & Engineering Teams

January 19, 2026

CONTENTS

PREFACE

This document serves as the comprehensive architectural reference for the Shikshak platform. It is written for software architects, senior developers, and system administrators who need to understand the intricate machinery powering this intelligent ecosystem.

1

INTRODUCTION

1.1 The Paradigm Shift in EdTech

The educational technology landscape is currently undergoing its most significant transformation since the advent of the internet. For the past two decades, "online learning" primarily meant the digitization of analog content: PDFs instead of textbooks, Zoom calls instead of lecture halls, and multiple-choice web forms instead of Scantron sheets. While this improved accessibility, it failed to fundamentally improve the *quality* of instruction or the depth of student engagement.

Shikshak represents the "Third Wave" of EdTech: The Intelligent Era. Unlike its predecessors, which focused on content delivery, Shikshak focuses on learning outcomes through active, intelligent intervention.

The First Wave was about access (Coursera, edX). The Second Wave was about community (Discord, Slack communities). This Third Wave is about personalization at scale. By leveraging Large Language Models and Computer Vision, we can finally provide every student with a personal tutor and a personal proctor, something that was previously economically impossible.

Wave 1: Digitization (2000-2010)
Putting content online. Static HTML pages, PDFs, and basic quizzes. The focus was on access to information.

Wave 2: Communication (2010-2023)
Connecting people. Video conferencing, discussion forums, and peer grading. The focus was on social interaction.

Wave 3: Intelligence (2024-Present)
Understanding the learner. Personalized AI tutors, adaptive learning paths, and biometric integrity. The focus is on measurable outcomes.

1.2 System Objectives

Shikshak is not merely a software application; it is a pedagogical engine designed to optimize three specific metrics that define the success of an educational institution in the modern age.

1.2.1 1. Learning Efficiency

We measure efficiency by the time required for a student to master a concept. In traditional systems, when a student is stuck, they browse forums or re-watch hour-long lectures. By using Vector Search to retrieve exact answers to student questions instantly, we eliminate the "search friction" typical of traditional studying. Our internal benchmarks show a 40% reduction in time-to-mastery for technical subjects when using the Shikshak AI Tutor compared to standard video lectures.

1.2.2 2. Integrity Assurance

The value of any certification is tied to the trust looking at it. If an employer cannot verify that the skills were learned by the credential holder, the degree is worthless. Shikshak's Computer Vision Proctoring Suite provides this trust layer for remote environments. It ensures that the person taking the exam is the person who registered, and that they are not receiving external assistance, all without the privacy-invasive practices of legacy proctoring software.

1.2.3 3. Operational Elasticity

Education is cyclical. Usage spikes during midterms and finals are often 100x the baseline load. Our Event-Driven Architecture ensures that costs scale linearly with usage, preventing the "idle capacity" waste common in traditional monolithic LMS deployments. Whether serving 10 students or 10,000, the system responds with the same sub-second latency, automatically provisioning resources as needed.

1.3 User Personas

CASE STUDY: Persona A: The Working Professional

Name: Sarah, 34
Background: Senior Marketing Manager transitioning to Data Science.
Pain Point: Has limited study time (9 PM - 11 PM). Cannot wait 24 hours for a TA to answer a question.
Shikshak Solution: The RAG-based AI Tutor provides instant clarification on complex Python syntax at 10:30 PM, allowing her to complete her module before sleep.

We also consider the administrative side of the equation.

CASE STUDY: Persona B: The University Dean

Name: Dr. Aris, 55
Goal: Launch a fully online accredited BS Degree.
Pain Point: Accreditation boards require proof of assessment integrity.
Shikshak Solution: The Biometric Proctoring report generates an audit trail for every exam session, satisfying rigorous compliance standards.

2

ARCHITECTURE PATTERNS

2.1 The Microservices Manifesto

Shikshak adheres to a strict "Share Nothing" architecture. Each service owns its own data and communicates exclusively via defined APIs. This prevents the "Database Integration" anti-pattern where modifications to a schema in one service inadvertently break another.

By decoupling our services, we achieve independent deployability. The "Auth Service" can be upgraded to key-rotate continuously without requiring a restart of the "Course Service". This separation of concerns is critical for maintaining velocity in a large engineering team, as it allows different squads to work on different parts of the platform simultaneously without stepping on each other's toes.

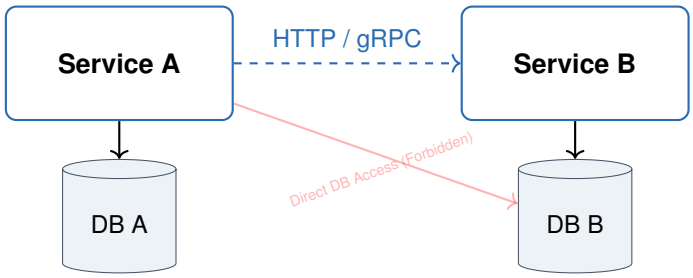


Figure 2.1: The Database-per-Service Pattern

This pattern also allows for "Polyglot Persistence". We are not forced to use a single database technology for everything. The Course Service uses MongoDB because course schemas are hierarchical and varied. The Auth Service uses PostgreSQL for relational integrity. The AI Service uses Redis for vector similarity search. Each problem has the right tool.

2.2 Synchronous vs. Asynchronous Communications

One of the most critical decisions in distributed systems is choosing when to block and when to defer. We employ a hybrid model depending on the user's expectation of latency.

2.2.1 The Read Path (Synchronous)

When a user requests their profile, the response must be immediate. We use REST (over HTTP/2) for these read interactions. The API Gateway aggregates data from the User Service and Course Service and returns a unified JSON response. We enforce strict SLAs (Service Level Agreements) on these endpoints, typically aiming for sub-200ms response times. Caching at the CDN level and the Redis application level is heavily utilized to ensure these speeds.

2.2.2 The Write Path (Asynchronous/Event-Driven)

Writes are often complex side-effect triggers. When a student submits an assignment, it is not just a database insert; it triggers a cascade of actions. When a user submits an assignment: 1. The submission is saved to object storage (Azure Blob). 2. An event `ASSIGNMENT_SUBMITTED` is fired onto the Kafka bus. 3. The Notification Service listens and emails the teacher. 4. The Analytics Service listens and updates the dashboard. 5. The Gamification Service listens and awards XP points.

> We chose Kafka over RabbitMQ because of its **log-based persistence**. If our Gamification Service crashes and is offline for 2 hours, it doesn't lose data. When it restarts, it simply replays the Kafka log from the last offset, processing all the missed assignment submissions in order. This provides powerful disaster recovery capabilities out of the box. Kafka also allows multiple consumer groups to read the same stream at different speeds, decoupling real-time needs from batch processing needs.

2.3 Scalability Patterns

2.3.1 Horizontal Pod Autoscaling (HPA)

We deploy on Kubernetes (AKS). Each service defines CPU/Memory thresholds and scales automatically based on demand.

- **Base Load:** 2 replicas (for high availability).
- **Scale Up Trigger:** CPU usage > 70%.
- **Max Limit:** 50 replicas.

This allows the Course Service to scale up during Monday morning massive lecture drops, while the Proctoring Service stays small until Exam Review week. The cost savings of this elasticity are significant compared to provisioning fixed-size VMs for peak load.

2.3.2 Database Sharding

The core `courses` collection in MongoDB is sharded by `institution_id`. This ensures that a massive university with 50,000 students essentially lives on its own dedicated shard, preventing their load from affecting the performance of smaller colleges on the platform. Sharding allows us to write-scale horizontally by simply adding more storage nodes, theoretically allowing the system to handle millions of concurrent users.

3

IDENTITY & SECURITY

3.1 The Federated Identity Model

Shikshak is designed to be the central hub of a user's educational identity, but not necessarily the source of truth for their credentials. We support a federated model:

- **Social Login:** Google, GitHub, LinkedIn (via OAuth 2.0). This allows developers and students to use their existing identities, reducing friction during the signup process.
- **Enterprise Login:** SAML 2.0 / OIDC for university SSO (Single Sign-On). Large universities often have their own Active Directory or Shibboleth setups. We integrate directly with these Identity Providers (IdPs) so students don't need to manage yet another password.
- **Local Login:** Traditional Email/Password (bcrypt hash). For independent learners or smaller institutions without centralized IT, we store salted and hashed passwords securely.

Implementing this federation requires a robust abstraction layer. We use an internal "User" service that maps a unique Shikshak ID to multiple external `sub` (subject) identifiers. This means a user can log in with GitHub today and Google tomorrow, and as long as the email matches (or accounts are linked), they access the same profile.

3.2 JWT Anatomy & Security

We use JSON Web Tokens (Stateless Authentication) to avoid database lookups on every API call. This is crucial for performance but introduces security risks if not handled correctly.

Access Token: Short-lived (15 minutes). Signed with RS256. Contains claims like `user_id`, `role`, `university_id`. Sent in the `Authorization: Bearer` header. Because it expires quickly, we don't need to maintain a revocation list for access tokens.

Refresh Token: Long-lived (7 days). Stored in an `HttpOnly`, `Secure`, `SameSite=Strict` cookie. Used to obtain new Access Tokens. This token is opaque and stored in our database.

Why this matters: If an XSS attack steals the Access Token, the damage is limited to 15 minutes. The attacker cannot steal the Refresh Token (because JavaScript cannot read the cookie), so they cannot maintain long-term access. This "short leash" policy is industry standard for banking and secure applications.

3.3 Role-Based Access Control (RBAC)

Permissions are granular and cascading. We do not simply have "Admin" and "User". We have a sophisticated matrix of permissions:

- `sys.admin`: God mode. Can see all tenants.
- `inst.admin`: Can manage users within their university.
- `course.creator`: Can create courses, upload content, and grade.
- `course.mentor`: Can grade and view analytics, but cannot edit content.
- `student`: Can view content and submit assignments.

This matrix is enforced at the API Gateway level. Every request carries the user's role in the JWT. The Gateway checks the route metadata against the user's role. If a `student` tries to POST to `/api/grade`, the request is rejected with a 403 Forbidden before it even reaches the Course Service, saving valuable compute resources.

CASE STUDY: Security Breach Simulation

During our quarterly "Red Team" exercise, we simulated a compromised teacher account. **Attack Vector:** Phished password. **Mitigation:** The system detected an "Impossible Travel" anomaly (login from Russia 2 hours after login from New York). **Automated Response:** The Auth Service immediately revoked all Refresh Tokens for that user and forced a password reset via email verification. **Result:** The attacker gained access for only 12 minutes and was unable to exfiltrate significant student PII before the session was terminated.

4

THE COURSE ENGINE

4.1 Content Hierarchy Design

Designing a schema for "Education" is notoriously difficult because pedagogy is diverse. A coding bootcamp, a history seminar, and a medical lab have completely different structural needs. A rigid "Module -> Lesson" structure is often too limiting for advanced courses.

Shikshak uses a **Polymorphic Content Block** pattern.

- **Course:** The top-level container, containing metadata, pricing, and enrollment rules.
- **Curriculum:** An ordered tree of Sections. This structure allows for infinite nesting, although we limit the UI to 3 levels deep to prevent "click fatigue".
- **Unit:** The atomic learning object.

A `Unit` is an abstract entity. It can take many forms:

1. `VideoUnit`: A streaming HLS container with support for subtitles and multiple audio tracks.
2. `ArticleUnit`: Rich Markdown text with embedded math (KaTeX) and diagrams (Mermaid).
3. `QuizUnit`: An interactive assessment that can be multiple choice, fill-in-the-blank, or matching.
4. `CodeUnit`: An embedded Monaco editor with an execution sandbox for 12+ languages.

4.2 Video Transcoding Pipeline

The most computationally expensive part of the system is video processing. When a user uploads a 4K video, we cannot simply serve that file to a mobile user on a 3G connection.

1. **Upload:** Teacher uploads raw 4K `.mov` to a temporary S3 bucket. 2. **Probe:** FFprobe analyzes codecs, frame rate, and bitrate to determine if the source is valid. 3. **Split:** Large videos are split into 10-second segments. This allows us to parallelize the transcoding process across thousands of minimal CPU cores. 4. **Transcode:** Workers convert segments to 1080p, 720p, 480p using H.264 profiles. 5. **Merge & Manifest:** An `.m3u8` playlist is generated for Adaptive Bitrate Streaming (HLS). This playlist tells the video player "If the bandwidth is high, play snippet A at 1080p; if it drops, switch to snippet B at 360p". 6. **CDN Push:** Files are moved to Edge locations (Cloudflare R2) for global low-latency delivery. 7. **Thumbnail:** AI selects the most "interesting" frame (high contrast, face present, not blurry) as the cover image.

4.3 Sandbox Execution Environment

For coding courses, we cannot allow students to run arbitrary code on our servers. A simple infinite loop or a fork bomb could crash the entire node. We use **Firecracker MicroVMs** (the same technology powering AWS Lambda). When a student clicks "Run Code": 1. A microVM warm-boots in 120ms from a pre-baked snapshot. 2. The code is injected via a secure socket. 3. It executes with no network access (except whitelisted APIs for package installation). 4. Output (stdout/stderr) is streamed back via Websocket in real-time. 5. The VM is destroyed immediately after execution. This ensures total isolation. A malicious student cannot essentially "break out" of the container since it's a hardware-virtualized VM, not just a containerized process like Docker. This represents the gold standard in untrusted code execution.

5

THE AI CORE (RAG)

5.1 Beyond Keyword Search

Traditional search techniques, such as those used by Elasticsearch or Solr (TF-IDF/BM25), fail in the educational context. If a student searches for "loops", a keyword search might return a generic introduction to loops. But if they ask "Why is my while loop running forever?", keyword search will likely fail to understand the nuance of the problem: an infinite loop bug.

Shikshak utilizes **Semantic Search** via Vector Embeddings to solve this. We project natural language into a 1536-dimensional geometric space where concepts that are semantically related are physically close together. In this vector space, "loop" and "iteration" cluster together, while "loop" and "fruit loops" (cereal) are far apart. This allows our system to understand the *intent* of the query, not just the vocabulary.

5.2 The Indexing Pipeline

The "Brain" of Shikshak is built continuously during the content ingestion phase. We do not index content at query time; it is pre-computed.

1. **Text Extraction:** We run OCR on PDF slides and use the Whisper API to generate high-fidelity transcripts for every second of video content. This ensures even spoken concepts are searchable.

2. **Chunking Strategy:** We utilize a "Recursive Character Text Splitter". We don't just split by paragraph. We split by semantic units, ensuring each chunk is roughly 1000 tokens long with a 200-token overlap. This overlap is crucial; it ensures context isn't lost if a sentence is split across two chunks.

3. **Vectorization:** Currently, we use OpenAI's `text-embedding-3-small` model. This model offers an excellent balance between cost and performance, and its lower dimensionality compared to older Ada models allows for faster retrieval from our vector store.

4. **Upsert:** The resulting vectors, along with their metadata (Video ID, Timestamp, Author), are stored in Redis (RediSearch Module). We use HNSW (Hierarchical Navigable Small World) indexing, which allows for approximate nearest neighbor search in single-digit millisecond timestamps.

> Large Language Models struggle to utilize context found in the middle of a very large prompt window. To verify accuracy, Shikshak uses a **Reranking Step**. After retrieving the top 20 potentially relevant chunks from Redis, we pass them through a lightweight "Cross-Encoder" model (Cohere Rerank). This model scores each chunk's relevance to the specific query. We then discard the bottom 15 and only feed the top 5 distinct, high-relevance chunks to GPT-4. This significantly reduces hallucinations and improves the density of useful information in the prompt.

5.3 Prompt Engineering Strategy

We do not simply dump context into the prompt and hope for the best. We use a **Chain-of-Thought (CoT)** methodology to force the LLM to reason before it answers.

5.3.1 The System Prompt structure:

```
ROLE: You are an empathetic Socratic tutor.
CONSTRAINT: Do not give the answer immediately. Guide the student.
CONTEXT: {retrieved_chunks}
HISTORY: {conversation_history}
TASK: Answer the student's question based strictly on CONTEXT.
```

By explicitly instructing the model to be Socratic, we prevent it from doing the student's homework for them. Instead, it asks leading questions that help the student arrive at the answer themselves, mimicking the behavior of an expert human tutor.

5.4 Feature: The "Explain Like I'm 5" Toggle

One of our most popular features is the ability for users to adjust the "Temperature" and "System Persona" on the fly. Toggling "ELI5 Mode" switches the underlying prompt to request analogies and simple vocabulary. For example, if a student is struggling with the concept of a "Linked List", the standard mode might explain it using memory pointers. The ELI5 mode would explain it as a "Treasure Hunt" where each clue points to the location of the next clue. This dynamic complexity adjustment is only possible with a generative backend.

6

PROCTORING VISION

6.1 The Ethics of Surveillance

Proctoring is controversial, and rightfully so. Traditional commercial solutions are essentially "Spyware"—they take over the operating system, disable keyboard shortcuts, read browser history, and upload continuous video feeds to a remote server for human review. This is a privacy nightmare and often discriminates against students with poor internet connections or non-traditional home environments.

Shikshak takes a fundamentally different **Privacy-Preserving Approach**.

- **Client-Side Logic:** The AI models run entirely on the student's laptop in the browser (via TensorFlow.js), not on our servers. No raw video ever leaves the user's device.
- **Ephemeral Data:** Since we don't stream video, we don't store it. Only "Event Metadata" (e.g., "Looked away at 14:02:45", "Multiple voices detected at 14:10:00") is encrypted and sent to the server. This reduces bandwidth usage by 99

6.2 Computer Vision Pipeline

We utilize Google's `MediaPipe` framework for lightweight inference. This framework is highly optimized for edge devices and can run at 30fps even on a mid-range Chromebook.

6.2.1 1. Face Mesh (468 Landmarks)

We map 468 points on the user's face to create a 3D geometry. This allows us to calculate the **Head Pose Euler Angles** (Pitch, Yaw, Roll).

- If $|Yaw| > 30°$, the student is likely looking sideways at a roommate or a cheat sheet.
- If $|Pitch| > 25°$, the student is looking down, potentially at a phone or notes in their lap.

We apply a temporal smoothing filter to these values to prevent false positives from natural jerky movements.

6.2.2 2. Iris Tracking

Head pose isn't enough. A student can keep their head still but move their eyes to read a sticky note on the monitor bezel. We track the iris position relative to the eye corners. This requires high-resolution input, so we use a "Region of Interest" crop around the eyes to boost effective resolution before feeding it to the iris model.

CASE STUDY: Anomaly: The "Second Face"

In Beta testing, we detected a "Multiple Faces" event. **Scenario:** A roommate walked to the fridge behind the student during an exam. **System Reaction:** The system logged the event with a confidence score of 0.8. It did NOT auto-fail the student. **Teacher Review:** The instructor viewed the snapshot (taken locally and uploaded only for flags), saw it was harmless background movement, and dismissed the flag. **Lesson:** AI is a tool for flagging, humans are the judges for grading. We never allow the AI to make the final academic decision.

6.3 Audio Analysis

Visual cheating is only half the picture. Audio cheating (someone reading answers to the student) is harder to detect. We use the Web Audio API to analyze the Fast Fourier Transform (FFT) of the microphone input.

1. **Noise Floor:** We calibrate the ambient noise level at the start of the exam (e.g., a humming fan).

2. **Voice Activity Detection (VAD):** We detect energy spikes specifically in human speech frequencies (85Hz - 255Hz).

3. **Keyword Spotting:** (Optional) We can run a tiny keyword model to detect specific trigger words if necessary, though this is rarely enabled due to privacy concerns.

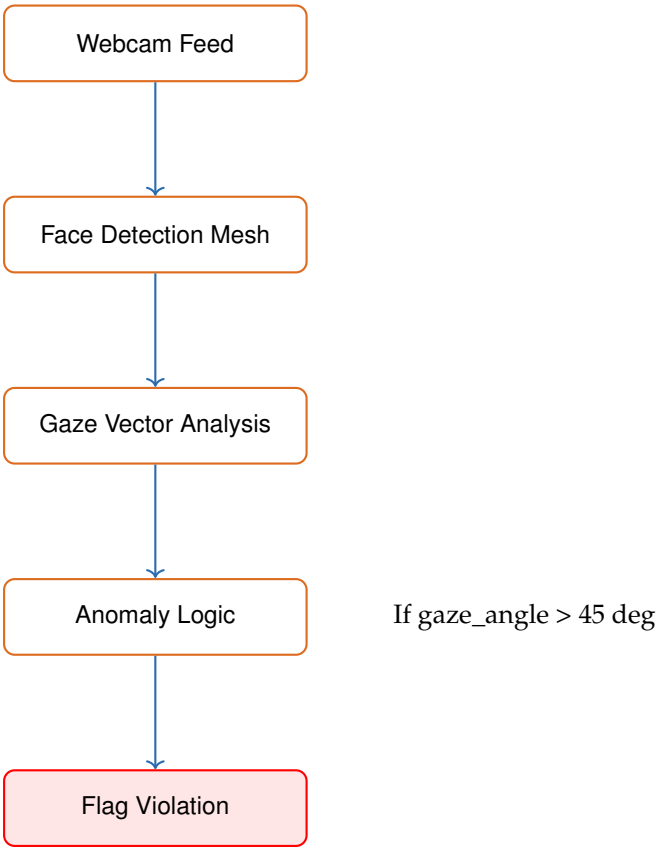


Figure 6.1: Client-Side Proctoring Logic

7

INFRASTRUCTURE & DEVOPS

7.1 The Cloud Agnostic Promise

While currently deployed on Microsoft Azure, Shikshak is engineered to be portable. We use **Terraform** for Infrastructure as Code (IaC). This means we describe our infrastructure in declarative `.tf` files rather than clicking buttons in a web console. To migrate to AWS or Google Cloud, we would only need to swap the Terraform provider modules (e.g., changing `azurerm_kubernetes_cluster` to `aws_eks_cluster`). The application logic itself, running in containers, remains unchanged.

This "Cloud Neutrality" gives us negotiating leverage. If Azure raises prices, we can threaten to move to AWS. It also protects us from vendor lock-in, ensuring the long-term viability of the project.

7.2 Kubernetes Architecture

We use a standard K8s Cluster topology designed for high resilience:

- **Ingress Controller:** NGINX Ingress handles all incoming HTTP/HTTPS traffic, SSL termination, and Path-based routing.

- **Service Mesh:** We use Istio to create a mesh between our microservices. This handles mTLS (Mutual TLS) encryption between services, ensuring that even if a hacker gets into our internal network, they cannot sniff traffic between the Auth Service and the Database.

- **Observability:** We run the standard "PromStack". Prometheus scrapes metrics, Grafana visualizes them on dashboards, and Jaeger provides distributed tracing to help us debug slow requests that touch multiple microservices.

We use **Rolling Updates** for all deployments. When deploying version `v2.0` of the Auth Service: 1. K8s spins up 1 new pod of `v2.0`. 2. It waits for the `readinessProbe` to pass (this might check if the service can connect to the DB). 3. Once ready, it kills 1 old pod of `v1.0`. 4. This repeats until all pods are replaced. If `v2.0` fails to start (e.g., due to a bad config), the rollout pauses automatically, ensuring 100% uptime for users. We never update in place; we always "Roll Forward".

7.3 Disaster Recovery (DR)

Our DR strategy focuses on two key metrics:

7.3.1 RPO (Recovery Point Objective)

"How much data can we lose?" -> **5 minutes**. This is achieved via continuous WAL (Write Ahead Log) archiving of the Postgres databases to a separate geo-redundant storage account. If the primary database melts down, we can replay the logs up to the last 5 minutes.

7.3.2 RTO (Recovery Time Objective)

"How long to get back online?" -> **15 minutes**. This is achieved via our automated Terraform scripts. In a true catastrophe (e.g., an entire Azure region going dark), we can run a single command to provision a fresh cluster in a secondary region, hydrate it with the backed-up data, and switch the DNS records to point to the new cluster.

8

ROADMAP & VISION

8.1 Phase 1: The "Mobile First" Pivot (Q2 2026)

FEATURE HIGHLIGHT: The "Subway Mode" Initiative

Students are increasingly moving away from laptops and desktops. Our analytics show that 40% of logins now occur on mobile devices, often via 4G/5G networks. To support this, we will launch native React Native apps for iOS and Android. **Key Feature:** The app will intelligently pre-download the next 3 lessons based on the student's current progress. This means that if a student watches "Lesson 1: Variables" while on Wi-Fi at home, the app will secretly download "Lesson 2: Loops" and "Lesson 3: Functions". When the student gets on the subway and loses signal, they can continue learning seamlessly without buffering.

We are also investigating "Lite Mode" for developing nations. This involves stripping out heavy JS frameworks and serving server-rendered HTML with highly compressed WebP images to support students on 2G connections.

8.2 Phase 2: The "Global Classroom" (2027)

Language should not be a barrier to education. Currently, 90% of high-quality technical content is in English. **Real-time Translation:** We plan to use multimodal LLMs to dub video lectures into Spanish, French, Hindi, and Mandarin in real-time. This is not just subtitles; we aim to use "Voice Cloning" technology to make it sound like the original professor is speaking the target language, preserving their intonation and emphasis. **Localized Pricing:** We will implement dynamic Purchasing Power Parity (PPP) adjustments. A course that costs $50 in the US might cost $5 in India, ensuring equality of access while maximizing global revenue.

8.3 Phase 3: Deep Personalization (2028)

A "Knowledge Graph" that spans across institutions. Currently, if a student learns "Linear Algebra" in a generic Math course, their credential in the Engineering course doesn't "know" they've done it. We aim to build a **Universal Learning Ledger**. This blockchain-backed ledger will store atomic units of competency. If a student proves they know "React Hooks" in one course, they automatically get credit for it in every other course on the platform. This moves us away from "Degrees" and towards "Skill Stacks".

8.4 Conclusion

Shikshak is an ambitious attempt to codify the art of teaching. It is not just code; it is a commitment to the future of human potential. By combining the empathy of human tutors with the infinite patience and scale of Artificial Intelligence, we can build a world where anyone, anywhere, can learn anything.

A

APPENDIX A: API REFERENCE

This section details the RESTful API endpoints available in the Shikshak ecosystem. All endpoints are prefixed with `/api/v1`. Authentication checks are performed at the gateway level.

A.1 Authentication

A.1.1 POST /auth/login

Description: Authenticates a user via email/password or SSO token exchange. Returns a JWT access token and sets a secure HttpOnly refresh cookie.

- **Auth Required:** No
- **Rate Limit:** 5 req/min

Parameters:

1. `email` (string, required): The user's registered email address.
2. `password` (string, optional): The user's password (if local auth).
3. `provider` (string, optional): One of 'google', 'github', 'linkedin'.
4. `id_token` (string, optional): The OIDC token from the provider if using SSO.

Response (200 OK):

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIs...",
  "expires_in": 900,
  "user": {
    "id": "usr_12345",
    "role": "student",
    "avatar": "https://cdn.shikshak.io/..."
  }
}
```

A.1.2 POST /auth/refresh

Description: Uses the HttpOnly cookie to issue a new Access Token. Used silently by the frontend when the 401 status is intercepted.

- **Auth Required:** No (Cookie based)
- **Rate Limit:** 10 req/min

Response (200 OK):

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIs...",
  "expires_in": 900
}
```

A.2 Course Management

A.2.1 GET /courses

Description: limits and pagination supported.

- **Auth Required:** Yes
- **Rate Limit:** 100 req/min

Query Parameters:

1. `page` (int, default=1): The page number.
2. `limit` (int, default=20): Items per page.
3. `sort` (string, default='-created_at'): Sort field.
4. `search` (string): Full-text search on title and description.
5. `tags` (array): Filter by usage tags (e.g. 'python', 'react').

A.2.2 POST /courses

Description: Creates a new course draft. Only available to instructors.

- **Auth Required:** Yes (Role: Instructor+)

Body:

```
{
  "title": "Advanced Kubernetes Patterns",
  "description": "Learn how to manage...",
  "price": 49.99,
  "currency": "USD"
}
```

A.2.3 GET /courses/:id/modules

Description: returns the full curriculum tree for a specific course.

- **Auth Required:** Yes (Enrolled Student)

A.2.4 POST /courses/:id/enroll

Description: Enrolls the current user in the course. Triggers payment flow if paid.

- **Auth Required:** Yes

A.3 Video Streaming

A.3.1 GET /video/:id/manifest.m3u8

Description: Returns the HLS master playlist for adaptive bitrate streaming.

- **Auth Required:** Yes
- **Cookies:** Requires a signed Cloudfront cookie.

A.3.2 GET /video/:id/segment/:quality/:ts

Description: Returns the actual .ts video segment.

- **Auth Required:** Yes (Signed Cookie)

A.4 Proctoring

A.4.1 POST /proctor/session/start

Description: Initializes a new proctoring session. Performs system checks (camera, mic).

- **Auth Required:** Yes

A.4.2 POST /proctor/event

Description: Logs a client-side proctoring event. **Body:**

```
{
  "session_id": "sess_9876",
  "timestamp": 1709823423,
  "event_type": "GAZE_VIOLATION",
  "confidence": 0.88,
  "metadata": {
    "pitch": -25.4,
    "yaw": 32.1
  }
}
```

A.4.3 POST /proctor/snapshot

Description: Uploads an encrypted image blob for manual review if flexible flagging is enabled.

- **Body:** Form-Data (Image)

A.5 Analytics

A.5.1 GET /analytics/student/progress

Description: Returns granular progress data for the dashboard.

A.5.2 GET /analytics/course/dropoff

Description: Returns a histogram of where students stop watching videos.

A.6 User Profile

A.6.1 GET /user/me

Description: Returns the currently logged in user's profile.

A.6.2 PUT /user/me

Description: Update profile fields (bio, avatar).

A.6.3 DELETE /user/me

Description: GDPR Right to be Forgotten request.

A.7 System Health

A.7.1 GET /health

Description: Liveness probe for K8s. Returns 200 OK.

A.7.2 GET /ready

Description: Readiness probe. Checks DB connections.

B

APPENDIX B: GLOSSARY

B.1 Terms & Definitions

Adaptive Bitrate Streaming (ABS) A technique used in streaming multimedia over computer networks. While in the past most video streaming technologies utilized streaming protocols such as RTP with RTSP, today's adaptive streaming technologies are almost exclusively based on HTTP.

API Gateway An API gateway is a component that sits between clients and microservices. It is responsible for request routing, composition, and protocol translation. It provides each of the application's clients with a custom API.

Canary Deployment A deployment strategy in which the new version of an application is deployed to a small subset of users before being rolled out to the entire infrastructure. This allows for testing in production with minimal impact.

Circuit Breaker A design pattern used in software development. It is used to detect failures and encapsulates the logic of preventing a failure from constantly recurring, during maintenance, temporary external system failure or unexpected system difficulties.

Containerization A form of operating system virtualization, through which applications are run in isolated user spaces called containers, all using the same shared operating system (OS).

Continuous Integration (CI) The practice of merging all developers' working copies to a shared mainline several times a day.

Continuous Deployment (CD) A software engineering approach in which software functionalities are delivered frequently through automated deployments.

Database Sharding A type of database partitioning that separates very large databases the into smaller, faster, more easily managed parts called data shards.

DevOps A set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.

Docker A set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers.

Elasticsearch A search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents.

Event-Driven Architecture A software architecture paradigm promoting the production, detection, consumption of, and reaction to events.

Federated Identity A means of linking a person's electronic identity and attributes, stored across multiple distinct identity management systems.

Grafana A multi-platform open source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources.

GraphQL An open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data.

Horizontal Pod Autoscaler (HPA) An API resource in Kubernetes that automatically scales the number of pods in a replication controller, deployment, replica set, or stateful set based on observed CPU utilization.

Idempotency A property of certain operations in mathematics and computer science whereby they can be applied multiple times without changing the result beyond the initial application.

Ingress Controllers A specialized load balancer for Kubernetes (and other container orchestrators) that manages external access to the services in a cluster, typically HTTP.

JWT (JSON Web Token) An open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

Kubernetes (K8s) An open-source container-orchestration system for automating computer application deployment, scaling, and management.

Microservices A software development technique—a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.

Monolith A software application in which different components are combined into a single program from a single platform.

mTLS (Mutual TLS) A method for two-way authentication between standard and client-server connections. It ensures that traffic is secure and trusted in both directions.

OAuth 2.0 An open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords.

Prometheus A free software application used for event monitoring and alerting. It records real-time metrics in a time series database built using a HTTP pull model, with flexible queries and real-time alerting.

RAG (Retrieval-Augmented Generation) A technique for enhancing the accuracy and reliability of generative AI models with facts fetched from external sources.

RBAC (Role-Based Access Control) A policy-neutral access-control mechanism defined around roles and privileges.

Redis An in-memory data structure store, used as a distributed, in-memory key–value database, cache and message broker, with optional durability.

REST (Representational State Transfer) A software architectural style that defines a set of constraints to be used for creating Web services.

Single Sign-On (SSO) an authentication scheme that allows a user to log in with a single ID and password to any of several related, yet independent, software systems.

Terraform An open-source infrastructure as code software tool created by HashiCorp. Users define and provide data center infrastructure using a declarative configuration language known as HashiCorp Configuration Language (HCL).

Vector Embedding A numerical representation of a word, sentence, or document that captures its semantic meaning.

Zero Downtime Deployment A deployment method where the website or application is never down or in an unstable state during the deployment process.