# B2C Education Platform: Phase-wise Sitemap & Architecture

## Overview

This B2C education platform is structured around a **three-phase learning journey**: **Learn**, **Churn**, and **Earn**. In the **Learn** phase, students enroll in online courses with video lessons, quizzes, assignments, and live mentor sessions, earning certificates upon completion. The **Churn** phase simulates a startup/incubator environment where students form teams to work on real-world projects under mentor guidance, completing milestones in a mock company setting. Finally, in the **Earn** phase, students monetize their project output – either by showcasing and launching their product or by taking on freelance tasks through an internal marketplace. The platform is built on a **React** front-end with a **Node.js** back-end, following a microservices-friendly architecture for scalability and modularity[[1]](https://github.com/harshagrawal2503/Learnhub#:~:text=%E2%9A%99%EF%B8%8F%20Back,course%20content%20and%20user%20data)[[2]](https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture#:~:text=Technically%20speaking%2C%20the%20key%20to,accessible%20and%20open%20as%20feasible). There are three primary user roles with distinct interfaces and permissions: **Student**, **Mentor**, and **Admin**. Each user type has its own navigation flow and capabilities, described in detail per phase below. Short, clear navigation menus and dashboards are provided for each role to ensure easy access to relevant features.

## Phase 1: Learn (Course Phase)

In the **Learn** phase, the platform functions as a comprehensive Learning Management System (LMS). Students can browse an extensive **Course Catalog**, enroll in courses (with payment integration if courses are paid), and then progress through structured learning content. Course content is typically organized into modules with **video lectures**, reading materials, and interactive **quizzes** or **assignments** at the end of each module. Students can track their progress, earn points or badges, and ultimately receive a **completion certificate** for each course. Live components are included: students can attend **live sessions** (webinars or Q&A sessions) scheduled by course instructors or mentors. Throughout, students have access to **mentor support** – they can ask questions in discussion forums or via chat, and submit assignments for feedback. Mentors, in this phase, act as instructors or teaching assistants: they monitor student progress, answer questions, grade assignments, and host live sessions. Admins manage the course content and user enrollment: they create or approve courses, upload videos/quizzes, assign mentors to courses, and oversee overall student performance metrics.

### Learn Phase – Sitemap & Feature Mapping

The table below outlines the key pages/sections in the Learn phase, along with the features available to each user type:

| **Page / Section** | **Student View & Actions** | **Mentor View & Actions** | **Admin View & Actions** |
| --- | --- | --- | --- |
| **Dashboard** | Landing page after login. Shows enrolled courses with progress bars, upcoming live sessions, and recent notifications. Allows quick resumption of next lesson. | Mentor dashboard showing an overview of their assigned students and courses. Highlights mentee progress, pending questions or submissions to review, and upcoming mentor sessions. | Admin dashboard with overall platform stats: number of users, course enrollments, completion rates, etc. Shows alerts (e.g., flagged content) and quick links to admin functions. |
| **Course Catalog** | Browse all available courses (search, filter by category/level). View course details (syllabus, instructor info, reviews). Enroll in a course (triggering payment if required).[[3]](https://github.com/harshagrawal2503/Learnhub#:~:text=,Some%20of%20these%20pages%20are) | View list of courses, particularly those they mentor or have created (if mentors also serve as instructors). Can access course details to see content structure (for mentoring purposes). | Manage course offerings: create new courses (title, description, category), upload course content (videos, PDFs), set quizzes and assignments. Publish or unpublish courses. Also able to **assign mentors** to courses and set course pricing or prerequisites. |
| **Course Player / Content** | The main learning interface. Watch lecture videos (with playback controls), read lesson text, and download resources. Mark lessons as complete. Navigate through modules. | Access course content to monitor and guide students. Can see which students have completed each lesson. May have a **Mentor Notes** panel to view or add teaching notes. Possibly join live sessions as host/co-host. | *Admin typically does not take courses*, but can **preview content** for quality assurance. (Admins can impersonate a student view to ensure content plays correctly.) |
| **Quizzes & Assignments** | Take quizzes at end of modules (auto-graded with immediate feedback). Submit assignments or projects (file uploads or text) for mentor review. View quiz scores and assignment feedback. | Review and grade student assignment submissions (provide scored feedback and comments). For quizzes, view analytics of common mistakes or manually grade subjective answers if needed. Can also reset attempts for a student if necessary. | Design quiz questions (multiple-choice, etc.) and assignment prompts during course creation. Set passing criteria. Oversee grading standards. Can override grades or grant extra attempts if required (for support cases). |
| **Live Sessions** | Join scheduled live classes or Q&A sessions via integrated video conferencing. Can view upcoming session schedule and register or join with one click. Interact via chat or audio during session. | Schedule and **host live sessions** (e.g., weekly webinar or doubt-clearing session). Manage live class (present slides, moderate student questions). After session, access attendance and recording. | Configure the live session tool integration (e.g., Zoom API or similar). View all scheduled sessions across platform, drop in to observe if needed. Manage recordings and ensure they are accessible to students after the session. |
| **Mentor Interaction** | Contact mentor for help: options include a **Q&A forum** on each course page and one-on-one **chat or messaging** with assigned mentor. View mentor profile and available office hours. | Respond to student questions in forums or chat. May have a queue of help requests. Schedule one-on-one mentoring sessions with students as needed (outside of live class schedule). Track which students have asked for help and ensure follow-up. | Monitor mentor–student communications for quality and safety (possibly via an admin-visible log or by intervening in forums if inappropriate content is flagged). Assign or reassign mentors to students if needed. |
| **Progress & Certificates** | View course progress overview (percentage complete, quiz scores). Upon 100% completion and meeting all requirements, unlock and download **certificate** for the course. Certificates can be accessed from a **Certificates** page in profile. | Track mentee progress from the mentor dashboard or a course roster – see which students are lagging or struggling (low quiz scores). Verify that a student has met criteria for completion. Mentor may need to mark certain tasks as complete for a student to get certified (e.g. project work). | Design certificate templates and criteria (e.g., auto-generate with student name and course). The admin can revoke or reissue certificates if needed. Also access overall course completion reports. |
| **Profile & Settings** | Manage personal profile (name, contact, bio). List earned certificates and skills on profile. Account settings for password, notification preferences. | Manage mentor profile (bio, expertise, availability hours). Possibly toggle willingness to take new mentees. Account settings similar to student. | Access any user’s profile (for moderation). Manage platform settings (site settings, content moderation tools, etc.). Admin may have a separate **Admin Panel** section for user management, not visible to others. |

*Notes:* In this phase, **Students** primarily see course-related sections and their own progress, **Mentors** see tools to assist and evaluate students, and **Admins** have content management and oversight capabilities. The **navigation menu** for a Student might include links like *Dashboard*, *My Courses*, *Catalog*, *Certificates*, *Messages*, and *Profile*. For a Mentor, it might include *Dashboard* (mentees & courses), *Q&A*, *Schedule*, and *Profile*. Admins will have an additional admin navigation (possibly a separate admin portal or menu) for managing courses, users, and platform settings.

## Phase 2: Churn (Simulation Company Phase)

The **Churn** phase is a project-based learning experience that simulates working in a startup or company environment. After or alongside completing courses, students join teams (or “simulation companies”) to build a product or complete a capstone project. Each team functions like a small company or startup within the platform. They are provided with a project scope or allowed to pitch their own, then guided through an **incubator-style process** with defined **milestones** (e.g., idea validation, prototyping, testing, final product). This phase is highly collaborative: students must **work together** (using tools like task boards and team chat) and apply skills from the Learn phase. A dedicated **mentor (or project coach)** is assigned to each team to act as a project supervisor – offering guidance, code reviews, design feedback, and ensuring the team stays on track. The Churn phase includes features for project management: task assignments, deadlines, progress tracking, and possibly integration with version control (for code) or at least file sharing for deliverables. Mentors periodically review the team’s work at each milestone, providing feedback or approval to proceed. At the end of the Churn phase, students produce a working prototype or project deliverables, and receive an evaluation (which could be pass/fail or a performance score). Admins oversee this phase by creating the project templates or challenges, grouping students into teams, assigning mentors, and monitoring all projects’ health/status.

### Churn Phase – Sitemap & Feature Mapping

Key sections/pages in the Churn (simulation project) phase are listed below, with role-specific functionalities:

| **Page / Tool** | **Student (Team Member)** | **Mentor (Project Coach)** | **Admin** |
| --- | --- | --- | --- |
| **Project Dashboard** | Central hub for the simulation project. Displays project name, description, and current milestone. Lists team members and their roles. Shows overall progress bar and next deadline. Students can see project updates and mentor notes here. | Overview of the project(s) the mentor is supervising. Can edit the project description or objectives if needed. Sees team roster and can view each member’s profile/role. Updates project status or milestone completion from here. Posts high-level feedback or announcements to the team. | Create and initialize projects (define project outline or select template). Add/remove students to project teams. Assign a mentor to each project. View any project’s dashboard to monitor progress. Can edit project settings (timeline, milestones) or intervene in team composition. |
| **Task Board / Kanban** | Interactive task management board (e.g., Kanban style). Students create and assign tasks to themselves or peers, move tasks across columns like *To Do*, *In Progress*, *Done*. They can set due dates and add checklists or attachments to tasks. This helps break the project into actionable items. | Monitor the task board: mentors can also create tasks (e.g., “Prepare presentation by Friday”) or adjust existing ones. They track if tasks are being completed on time. May re-prioritize tasks or add comments on tasks with guidance. | Oversee all project boards (possibly via admin panel, to ensure teams are active). Not typically editing tasks, but can step in to reassign or prompt teams that are stalled. Admin might have a **global view** of all tasks across projects for analytics, but mainly for oversight rather than daily management. |
| **Team Communication** | Built-in communication tools for the team. This could include a **Team Chat** (real-time messaging), and/or a **Discussion Forum** specific to the project. Students use this to collaborate daily – share updates, ask teammates questions, upload files or links. It’s essentially the virtual “office” for the simulated company. | Joins the team’s communication channels to guide the discussion. The mentor can post updates (e.g., “Don’t forget the deadline tomorrow”) and answer team questions. They might hold stand-up meetings via chat or video. Mentors ensure the conversation stays productive and all students are participating. | Monitor communications for appropriateness and engagement. Admins might not actively participate in chats, but they have access to logs or can join if needed. They ensure the tool is used in compliance with code of conduct (possibly with automated moderation flags). |
| **Milestones & Deliverables** | A page listing all **milestones** in the project (with due dates). For each milestone, students can upload **deliverables** (documents, presentations, code links, etc.) for mentor review. They mark milestones as submitted, and see status (e.g., “Under Review” or “Approved”). Also see mentor feedback for each milestone. | Define expectations for each milestone and review submissions. When students submit deliverables, the mentor reviews the files, then provides feedback or marks the milestone as approved/complete. If not satisfactory, mentor can request revisions. Mentors may fill an evaluation rubric or checklist for each milestone to guide student learning. | Set up the milestone structure for projects (if using a template, e.g., Milestone 1: Project Charter, Milestone 2: Prototype, etc.). Admin can view the status of all teams’ milestones (e.g., how many have completed Milestone 3). They can override approvals or extend deadlines in special cases. |
| **Resource Repository** | A section where all project-related files and resources are stored. Students can upload files (design documents, code archives, etc.) or link to external repositories (like GitHub). This acts as a **project library** for the team. Students can browse and download any team files. | Access to the team’s repository of files. Mentors might upload additional resources or templates for the team (e.g., a sample business plan document). They review files for correctness. If integrated with an external Git repo, the mentor can view commit history and code to give technical feedback. | Ensure storage limits and access controls. Admin may configure integration with external tools (like authorizing a Git integration for the platform). They can access any project’s files (for audit or demo day purposes). |
| **Team Member Profile & Roles** | List of team members with assigned roles (e.g., Team Lead, Developer, Designer, etc.). Students can view each other’s profiles and skills from here, fostering team familiarity. Possibly includes a small bio or strengths for each member. Some platforms allow peers to give each other feedback or endorsements at the end. | Mentor can see each member’s role and engagement level. They might be able to adjust roles or encourage rotation of responsibilities (learning different skills). Mentors might also have a say in peer evaluations or give a brief comment on each member’s performance for final assessment. | Assign roles when forming the team (or approve roles if students select). Admin sees overall user involvement – e.g., ensure one student isn’t in too many projects at once. At phase end, admin may compile evaluation reports for each student (including mentor feedback) to determine if the student “passes” the incubator phase. |
| **Project Evaluation/Outcome** | At the end of the project, students can view their **final evaluation**. This could be a pass/fail or a graded outcome for the project. They might receive a detailed feedback report from the mentor, and possibly a **project completion certificate** or badge. Students can also likely **showcase** their project output (link to Earn phase) – e.g. mark it ready for the marketplace. | Complete an evaluation form or write a summary for the team and individual members. Mentor recommends whether the project is launch-ready or suggests next steps. They might also nominate outstanding projects for featuring in the Earn marketplace. Once evaluation is submitted, the mentor’s active duties end for that project. | Review all project evaluations for consistency. Admin may moderate the results (especially if mentors give scores – ensure fairness across different mentors). They can decide which projects get highlighted for the Earn phase. Admin could trigger the creation of a **portfolio entry** in the Earn marketplace for completed projects, pending student confirmation. |

*Notes:* In the Churn phase, **navigation** for students and mentors will center on the project workspace. For example, a student’s menu might show *My Project* (leading to the dashboard), *Task Board*, *Team Chat*, *Milestones*, etc., possibly grouped under a “Simulation Company” section. Mentors will have a menu listing the projects they supervise and similar sub-pages for each project. The **functional flow** here: a student accepts or is assigned to a project, collaborates through the tools above, and upon completion, moves toward monetization. Mentors guide the project from inception to completion. Admins facilitate the setup and ensure each team has what they need.

## Phase 3: Earn (Monetization & Marketplace Phase)

In the **Earn** phase, the platform shifts to an entrepreneurial and professional development focus. Students (now potentially graduates of the first two phases) can monetize their work or skills. There are two primary avenues in this phase:

* **Product Launch/Showcase:** Students can turn their Churn-phase project into a real product. The platform provides a **Product Showcase page** for each project, where the team can publish a description, screenshots/demo, and even a downloadable or live version if applicable. This acts as a portfolio piece that can be viewed by potential employers, investors, or platform peers. If the product is something that can be sold (e.g., a mobile app or a digital service), the platform could facilitate a **launch** – for example, allowing the team to list it in an **internal marketplace** or app store. Even if not directly sold, this showcase is a way to attract opportunities or feedback.
* **Freelance/Marketplace:** The platform includes an **internal marketplace** for projects and gigs. External companies or entrepreneurs (or the platform’s partner network) can post **freelance tasks, internships, or project contracts** for which these students can apply. Students browse listings, which could range from small paid tasks (e.g., "design a logo", "develop a prototype feature") to larger project collaborations. This helps students earn money and real experience. Students can apply to tasks, communicate with the task poster, and on completion, possibly get rated or reviewed. The platform may manage payments to ensure safe transactions.

Throughout the Earn phase, mentors might step back, but could still play a role as advisors or references. For instance, a mentor might help a student polish their product listing or prepare for a client interview, though these interactions may happen off-platform or informally. Admins manage the marketplace environment: approving new product listings and freelance postings to maintain quality, handling any financial transactions (escrow, payouts), and featuring successful student stories. This phase closes the loop of the platform – a student who *learned* skills and *built* a project can now *earn* real rewards, all within one ecosystem.

### Earn Phase – Sitemap & Feature Mapping

Key pages and features in the Earn phase are mapped out below:

| **Page / Section** | **Student (Earner)** | **Mentor** | **Admin** |
| --- | --- | --- | --- |
| **Marketplace Home** | Overview of the Earn marketplace. Students can toggle between viewing **“Products”** (launched projects by peers) and **“Tasks”** (freelance opportunities). The page features search and filter (e.g., by category, technology, paid/free). Students see highlights like top-rated projects or newest job postings. | Mentors might browse the marketplace out of interest or to see their mentees’ listings, but they typically do not actively participate here. (If mentors have their own companies, they *could* post tasks, but that would be as an external poster, not in mentor role). | Oversee all marketplace content. Admin can feature certain projects on the homepage, and ensure categories and filters are up to date. They also monitor for any inappropriate listings. |
| **Product/Project Showcase** | A profile page for each project that students choose to publish. If a student (or team) from Churn phase makes their project public, it appears here. The page includes project description, team members or creators, screenshots/videos, and possibly a link to a live demo. If the product is for sale, a “Purchase” or “Get in Touch” button is available. Otherwise, a “Like” or “Comment” section could allow feedback from other users. Students can edit their project listing, update media, or view analytics (views, likes). | Mentors can view these pages and possibly leave comments or endorsements for their mentees’ projects (if the platform allows comments/ratings). They might also guide students on improving their listing content. Otherwise, mentors aren’t owners of these pages unless they were directly involved in the project creation. | Approve product listings before they go live (to avoid any offensive or low-quality content). Admin can edit or take down listings that violate terms. They might also have an **Analytics Dashboard** to see which projects are attracting interest or which categories are most active. In case the platform allows transactions, the admin manages the selling mechanism (setting revenue share or handling payout to students when a sale occurs). |
| **Freelance/Task Listings** | A listing board for gigs and tasks. Each listing shows the task title, description, required skills, budget or reward, and deadline or expected duration. Students can click a listing to see full details and then **Apply** or **Bid**. They will typically submit a proposal or cover letter through the platform. Students may track their applications in an *Applications* section (seeing status: pending, accepted, etc.). If hired/accepted for a task, the listing moves to their “Active Contracts”. They then see project details and deliverables submission area (similar to how Churn projects were managed, but likely simpler). After completing the task, they mark it complete and might receive a rating/review. | Mentors do not directly apply to tasks in their mentor capacity. However, mentors could be **task posters** if the system allows companies/partners to have accounts – e.g., a mentor working at a company could post an internship. In that case, they’d use an **organization/partner account** rather than their mentor account. So under mentor role, not much direct action. They might continue to advise students on how to handle freelance work (outside the system). | Approve new task postings (especially if external clients post tasks – ensure they are legitimate opportunities). Admin manages user roles for task posters (could be separate from normal students/mentors). They may also implement an escrow system: when a task is accepted, payment is held by the platform and released on completion. Admin resolves disputes if a client and student disagree on completion. They also ensure the freelance marketplace runs smoothly and safely (trust & safety team responsibilities). |
| **Earnings & Wallet** | A personal earnings dashboard. Students can see the money earned from completed tasks or product sales. It shows a balance (wallet) and history of transactions (with dates and project/task names). Students can initiate a **withdrawal** to their bank or payment account once they have a certain balance (if real money is transacted through the platform). Also includes tax or invoice info if needed. | Not applicable (mentors are not earning via this platform, unless separately participating as task posters which is outside mentor role). | Oversee financial transactions. Admin can process payouts, view aggregate earnings statistics, and charge any platform fees. There will be an admin interface for managing payout requests, refunding a client if needed, and generating reports (for accounting or for showing platform impact, e.g., total earned by students via the platform). |
| **Application/Communication** | After a student applies for a freelance task, a **communication channel** might open between the student and the task poster (similar to Upwork style messaging). Students can discuss project requirements, send work updates, and deliver final files through this chat or a submission form. This is typically one-to-one and private. For product sales or inquiries, if a visitor is interested in a student’s product, there may be a **contact form** or comment thread on the product page for inquiries. Students will receive these messages and can respond. | Not directly used by mentors. (Mentors might be involved as a third-party reference if a student wants, but the platform doesn’t require mentor intervention here.) | Monitor communications to prevent fraud or misconduct. The admin panel may flag certain keywords or allow admins to step into a conversation if a dispute is reported. Admin can also see all open contracts and their status. |

*Notes:* The **navigation** for the Earn phase might be somewhat separate from the Learn/Churn phases. For example, a student who has completed a project would unlock an “Earn” section in their menu, containing links like *Marketplace*, *My Project Listing*, *Freelance Opportunities*, *Earnings*. It’s possible the platform surfaces Earn-phase items in the general dashboard as well (e.g., “New freelance projects available” or “Your project got 100 views”). The **functional flow** for a student here is: set up their project portfolio page, then optionally browse and apply for gigs. If they secure a gig, they complete it with the client and earn money which they can withdraw. Admins maintain the integrity of the marketplace throughout.

## Platform Architecture

The platform’s architecture uses a **React** front-end and a **Node.js** back-end arranged in a scalable, modular fashion. The React front-end is a single-page application that communicates with the back-end via RESTful APIs (or GraphQL) for dynamic data (course content, user data, etc.)[[1]](https://github.com/harshagrawal2503/Learnhub#:~:text=%E2%9A%99%EF%B8%8F%20Back,course%20content%20and%20user%20data). The choice of React allows a fluid, interactive UI for features like video playback, quizzes, drag-and-drop task boards, and real-time chat. The Node.js back-end is organized following a **microservices-friendly architecture**, meaning the system is divided into multiple distinct services that can be developed, deployed, and scaled independently[[4]](https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture#:~:text=Separating%20Services)[[2]](https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture#:~:text=Technically%20speaking%2C%20the%20key%20to,accessible%20and%20open%20as%20feasible). Each service encapsulates a specific domain of the platform’s functionality. Key microservice components might include:

* **Authentication & User Service:** Handles user accounts, authentication (login/registration, OAuth if used), and authorization roles (student/mentor/admin). Issues JWT tokens for client requests. Manages user profiles and preferences. This service ensures role-based access control so that, for example, admin APIs can only be called by admins, etc.
* **Course Service (Learn microservice):** Manages courses, modules, and content delivery. Stores course metadata, module structure, quiz questions, and handles logic for progress tracking (e.g., marking lessons complete, grading quizzes). Also issues certificates upon completion. This service interacts with a **Content Storage** subsystem for serving videos and files (which might be on a cloud storage/CDN). It could also include a sub-component for a **Quiz Engine** to handle quiz logic (grading, randomization, question banks). *In a microservices design, the Course service is independent:* it can be updated or scaled (e.g., more instances for many concurrent video viewers) without affecting other parts of the system[[5]](https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture#:~:text=In%20order%20to%20create%20a,rearranged%20to%20satisfy%20shifting%20requirements).
* **Mentorship/Communication Service:** Focused on communications and mentoring interactions. It powers features like the Q&A forums, direct chat messaging, notifications, and scheduling of live sessions. It may utilize WebSocket connections or similar for real-time chat and notifications (e.g., using libraries like Socket.io on Node). Live session scheduling might integrate with third-party APIs (Zoom/Teams) or a custom webinar solution. This service ensures that messages and forum posts are stored and delivered properly, and could incorporate a moderation filter for content. It also tracks which mentor is assigned to which student or project.
* **Project Management Service (Churn microservice):** Handles the simulation company projects. This includes storing project info, team memberships, task boards (which could be implemented using an internal representation of tasks, or even integrate with something like Trello API if desired), milestone tracking, and file repositories. It might have its own database for project data, separate from the course data. This service ensures integrity of project workflows – for instance, preventing unauthorized users from accessing a team’s project data. It may also incorporate a lightweight **version control integration** (links to external GitHub repos or an internal Git service) for code submissions. Given the complexity, this service could even be broken down further: e.g., a **Task Service** for tasks and a **Collaboration Service** for chats/forums specifically within projects, depending on scaling needs.
* **Marketplace Service (Earn microservice):** Manages the product listings and freelance tasks marketplace. It stores listings for gigs and product pages, including metadata like descriptions, prices, etc. It handles applications to tasks, messaging between students and task posters, and records of transactions. If payments are involved, this service integrates with a **Payment Gateway** (like Stripe/PayPal) to handle escrow and payouts. It also keeps track of reviews/ratings for completed tasks. The Product Showcase portion might interface with the Project Management service (to import project info upon completion) or simply allow students to input details manually. This service ensures that only authorized users (students or verified partners) create listings, and that data is isolated per listing.
* **Admin & Analytics Service:** Provides administrative functions and aggregates data from other services. Admins might have a separate front-end interface (or just additional pages in the React app protected by admin role) that calls admin endpoints. These endpoints allow managing users (CRUD users, assign roles), viewing platform-wide analytics (e.g., number of active learners, conversion from Learn to Churn, revenue from Earn, etc.), content moderation (approve/reject courses, projects, tasks), and configuration settings. Analytics might also be handled by a separate pipeline or third-party integration (e.g., using tools or a data warehouse), but from a platform viewpoint, this service compiles information across microservices to present to admins. For example, it might call the Course service for enrollment stats, the Project service for milestone completion rates, etc., or subscribe to events from each.

Other supporting components include a **Notification/Email Service** (to send emails or push notifications for events like “assignment graded” or “new task posted”), and possibly a **Search Service** (to index courses, tasks, products for efficient searching). Each microservice typically has its own database (e.g., a SQL or NoSQL database depending on data type) to remain loosely coupled and scalable. For instance, the Course service might use a relational DB for structured progress data, while the Project service might use a document database to flexibly store various project artifacts.

All these Node.js services are deployed on the back-end behind an API Gateway or load balancer. The **API Gateway** unified endpoint can route requests to the appropriate service, handle authentication verification (checking tokens), and potentially do request aggregation. The React front-end thus generally interacts with the gateway, which then calls microservice APIs. This makes the front-end simpler and the system more secure (the gateway can also simplify CORS and endpoints for the client). To the end-user, the application feels like one cohesive platform, even though behind the scenes multiple services work in concert[[6]](https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture#:~:text=The%20answer%20was%20to%20create,app%20stability%20significantly%20during%20upgrades). For example, when a student views their Dashboard: the front-end might call a unified /dashboard API, and the gateway or a composite service gathers info from the Course service (progress data), Project service (current project status), and Marketplace service (any new messages or earnings) before returning the data. This approach keeps services decoupled yet allows rich cross-phase experiences.

**Tech Stack Details:** The Node.js back-end can be implemented with frameworks like Express.js or NestJS for structured development[[7]](https://code-b.dev/blog/microservices-with-nodejs-and-react#:~:text=,Node%20JS%20for%20Microservices%20development)[[8]](https://code-b.dev/blog/microservices-with-nodejs-and-react#:~:text=). Data is likely stored in a combination of databases: a **SQL database** (e.g., PostgreSQL) for structured data like user accounts, transactions, and a **NoSQL database** (e.g., MongoDB) for flexible data like course content or chat logs – or vice versa, depending on design preferences. The platform might also use Redis for caching (e.g., caching course catalog or user sessions) to improve performance, and an Elasticsearch service if full-text search across courses or projects is needed. Static assets (videos, images) would be stored on a cloud storage (like AWS S3 or Cloudinary), with CDN delivery for efficiency[[9]](https://github.com/harshagrawal2503/Learnhub#:~:text=%EF%B8%8F%20Database%20The%20database%20for,information%20related%20to%20the%20platform).

**Scalability & Microservices Benefits:** This architecture ensures that each phase of the platform can scale independently. For instance, the Learn phase might have heavy video streaming usage – one could scale up the Course Content service and the CDN bandwidth without touching the Project or Marketplace services. Similarly, if the Earn phase grows with lots of transactions, that service and its database can scale (and be subject to tighter security for payments) without impacting the learning functions. It also allows the development team to work in parallel – one team can build new quiz features in the Course service while another improves the marketplace, with minimal interference. As a microservices-based design, the system is **modular and maintainable**: updates or fixes to one service (like the quiz grading engine) do not require redeploying the entire application[[10]](https://code-b.dev/blog/microservices-with-nodejs-and-react#:~:text=Microservices%20is%20an%20architectural%20approach,for%20a%20specific%20business%20function)[[11]](https://code-b.dev/blog/microservices-with-nodejs-and-react#:~:text=These%20services%20communicate%20with%20each,service%20communication%20and%20system%20complexity). This aligns with modern SaaS e-learning architecture practices that prioritize independent services and shared APIs over monolithic LMS systems[[4]](https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture#:~:text=Separating%20Services).

Finally, the platform can be deployed in a cloud environment using containerization (Docker/Kubernetes) to manage the multiple services. Each microservice can run in its own container, enabling easy deployment and scaling. Continuous integration and delivery (CI/CD) pipelines would allow frequent updates to each service. The React front-end can be deployed as a static bundle served via CDN or via a Node server (or even using Next.js for server-side rendering if SEO for public pages is needed).

In summary, this B2C education platform is designed with a **phase-wise structure** that maps to the user’s learning journey (Learn → Churn → Earn), and an underlying **microservices architecture** that supports this journey in a flexible, scalable way. The sitemap and features ensure that **Students** have a clear path from acquiring knowledge to applying it and finally capitalizing on it, **Mentors** can effectively support and evaluate learners at each step, and **Admins** can oversee and grow the ecosystem with fine-grained control. This structured approach provides an engaging experience akin to progressing from a course to an internship to earning in the real world – all within one integrated platform.

**Sources:** The design draws on e-learning platform conventions and modern architecture principles[[3]](https://github.com/harshagrawal2503/Learnhub#:~:text=,Some%20of%20these%20pages%20are)[[4]](https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture#:~:text=Separating%20Services), ensuring a user-friendly interface with rich functionality backed by a robust, scalable technical foundation.

[[1]](https://github.com/harshagrawal2503/Learnhub#:~:text=%E2%9A%99%EF%B8%8F%20Back,course%20content%20and%20user%20data) [[3]](https://github.com/harshagrawal2503/Learnhub#:~:text=,Some%20of%20these%20pages%20are) [[9]](https://github.com/harshagrawal2503/Learnhub#:~:text=%EF%B8%8F%20Database%20The%20database%20for,information%20related%20to%20the%20platform) GitHub - harshagrawal2503/Learnhub: E-Learning Platform using React.js, Node.js and MongoDB along with power of Machine Learning and Natural Language Processing to provide personalized learning experience

<https://github.com/harshagrawal2503/Learnhub>

[[2]](https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture#:~:text=Technically%20speaking%2C%20the%20key%20to,accessible%20and%20open%20as%20feasible) [[4]](https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture#:~:text=Separating%20Services) [[5]](https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture#:~:text=In%20order%20to%20create%20a,rearranged%20to%20satisfy%20shifting%20requirements) [[6]](https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture#:~:text=The%20answer%20was%20to%20create,app%20stability%20significantly%20during%20upgrades) Microservices Architecture And Migration - eLearning Industry

<https://elearningindustry.com/migrating-monolithic-learning-management-system-to-microservice-architecture>

[[7]](https://code-b.dev/blog/microservices-with-nodejs-and-react#:~:text=,Node%20JS%20for%20Microservices%20development) [[8]](https://code-b.dev/blog/microservices-with-nodejs-and-react#:~:text=) [[10]](https://code-b.dev/blog/microservices-with-nodejs-and-react#:~:text=Microservices%20is%20an%20architectural%20approach,for%20a%20specific%20business%20function) [[11]](https://code-b.dev/blog/microservices-with-nodejs-and-react#:~:text=These%20services%20communicate%20with%20each,service%20communication%20and%20system%20complexity) Building Microservices using NodeJS + React [Detailed Guide]

<https://code-b.dev/blog/microservices-with-nodejs-and-react>