Absolutely! Here's the **LMS Project Architecture Deep Dive** we prepared earlier, now formatted and copy-paste ready for your **presentation** or **Google Docs** use:

📚 LMS Project Architecture Deep Dive

🎤 **Presenter:** Gift Ndlala

📅 **Date:** April 23, 2025

🔍 **Purpose:** Internal & external stakeholder presentation

🎯 Overview

This document outlines the full-stack architecture of the Learning Management System (LMS), focusing on **modularity**, **scalability**, **maintainability**, **security**, and **future readiness**. It highlights the technologies used, how components interact, and the current project status.

⚛️ Frontend Architecture – *React.js + MUI*

☑️ Core Technologies

- **React 18.2.0** – Component-based UI with hooks for state and lifecycle
- **React Router 6.23.0** – Nested routing and role-based route protection
- **Material-UI (MUI)** – Prebuilt UI components, theme customization, responsiveness
- **Axios** – API calls with interceptors (JWT token injection & auto-refresh)
- **React Context + Custom Hooks** – Global state (auth, user role, notifications)
- **Jest + React Testing Library** – Frontend testing
- **ESLint + Prettier** – Code consistency and formatting

🗂 Directory Structure

```
/src
├── components/         → Reusable UI (buttons, modals, loaders)
│   ├── common/
│   ├── layout/
│   └── forms/
├── pages/              → Student & instructor pages
│   ├── student/
│   ├── instructor/
│   └── shared/
├── services/           → Axios config, API handlers
├── hooks/              → Custom hooks (useAuth, useRole)
├── context/            → Auth & notification providers
├── styles/             → CSS & MUI themes
├── assets/             → Icons & images
└── tests/              → Frontend unit/integration tests
```

⚙ State Management & UX

- **React Context** – Auth state and notifications
- **Storage** – JWT persisted with LocalStorage or SessionStorage
- **Optimistic UI** – Frontend updates instantly before server confirmation
- **Error Boundaries** – Component-level fallback UIs

🔐 Security & Routing

- **Protected Routes** – Role-based redirection (Student, Instructor, Admin)
- **JWT via Axios Interceptors** – Secure, auto-refresh on token expiry
- **Validation** – Yup + server-side DRF validations

💡 UX Features

- Responsive layout (mobile-first)
- Form validation with inline feedback
- Skeleton loaders & progress indicators
- Accessibility (ARIA, keyboard navigation)
- Theme toggle (Dark/Light)

🧬 Backend Architecture – *Django + DRF*

☑ Core Technologies

- **Django 5.2 / DRF** – Modular backend with ViewSets and Serializers
- **PostgreSQL** – Production-grade relational database
- **Celery + Redis** – Background tasks, async email, scheduled reports
- **Django Channels** – Real-time WebSocket support
- **SimpleJWT** – JWT-based access & refresh token authentication
- **Djoser** – Built-in login, logout, password reset endpoints

🧱 App Structure

```
/backend
├── accounts/          → User model, role management
├── courses/           → Courses, modules, lessons
├── assignments/       → Assignment creation, submission, grading
├── quizzes/           → Question banks, auto-grading, quiz attempts
├── notifications/     → Announcements, messages, email alerts
├── analytics/         → Student activity & course reports
└── core/              → Global settings, permissions, logging
```

🔑 Authentication & Authorization

- **JWT (SimpleJWT)**: Short-lived access tokens, long-lived refresh
- **Role-Based Access Control (RBAC)**: Only Students/Instructors/Admins can access assigned routes and data
- **Djoser**: Handles registration, password reset, email confirmation
- **Audit Logs**: Track critical activity like grading or wallet updates

🌐 API Design & Data Flow

- **RESTful API (v1)** – Clean, consistent, versioned endpoints
- **Pagination** – For student lists, submissions, course content
- **Optimized Queries** – `select_related` and `prefetch_related` for N+1 query issues
- **Secure File Uploads** – Stored in `/media/`, accessible with token
- **Bulk Operations** – Batch enrollments, multi-file uploads planned

🟦 Real-Time Features

WebSockets (via Django Channels)

- Chat between students and instructors
- Live announcement feeds
- Connection presence tracking

Celery + Redis

- Background email notifications
- Assignment due reminders
- Automated analytics reports

🛡️ Security Measures

- **CORS Whitelisting** – Only frontend origins allowed
- **Throttling** – Prevents brute force abuse
- **Input Sanitization** – Strict serializers + form validation
- **CSRF Protection** – Enabled for session-based endpoints
- **GDPR Compliance** – Planned support for data deletion/export
- **Backup & Logging** – Automated DB dumps and centralized error logs

🚀 DevOps & Deployment Strategy

- **Dockerized Setup** – Full containerization for backend and frontend
- **CI/CD with GitHub Actions** – Run tests, linting, build pipeline
- **`.env` Config** – Secure use of secrets, DB configs
- **Production Server Prep** – SSL, reverse proxy, CDN
- **Monitoring** – Sentry for error logging, Prometheus for health checks

🔄 API & Data Flow

- **REST API** → All frontend data handled via JSON
- **Axios** → Automatically adds token to headers and refreshes if expired
- **WebSocket Channels** → Enables live chats & notifications

📈 Project Observations & Incomplete Work

☑️ Working Features

- User registration & JWT login
- Student dashboard layout
- Instructor can:

    - Create modules
    - Assign students
    - Post announcements
- File upload system for assignments

⚠️ Known Issues / In Progress

- 🚫 Some API data isn't fetched on frontend (e.g., modules)
- 🚫 Login/token refresh isn't 100% reliable
- 🚫 Role-based route guards need final testing
- 🔐 Security improvements pending (upload validation, throttling)

👥 Professional Skills Needed Going Forward

- Backend Django Developer (DRF + model refinement)
- React Frontend Developer (Context API, routing)
- DevOps Engineer (SSL, Docker, CI/CD)
- UI/UX Designer (modern dashboard themes, usability testing)
- QA Tester (end-to-end + integration testing)

💬 Common Questions You Might Be Asked (with Answers)

💬 Question	✅ Suggested Answer
What stack did you use?	React + Django REST Framework + PostgreSQL + JWT
How do students get assigned modules?	Instructors assign modules using the backend interface
How does login authentication work?	Using JWT (SimpleJWT) with Djoser and Axios interceptors
Can students upload PDFs and assignments?	Yes, through the `AssignmentSubmission` endpoint

Can instructors grade assessments?	Yes, assessments and feedback are tied to instructor ID
Can the LMS send notifications or reminders?	Yes, via Celery tasks and WebSocket announcements
What security measures are in place?	CORS, token auth, CSRF, RBAC, DB logging, secure uploads
How scalable is this LMS?	Designed modularly, can be containerized and deployed to cloud (Docker ready)

🧠 Closing Thoughts

"This LMS was built from the ground up with a team of interns under tight constraints. I've personally seen it evolve from an idea into a fully operational full-stack project. Toward the final stages, I was joined by Siphamandla who helped solve some of the authentication bugs. We're proud of what we've built and even more excited for what's to come."