Slide 1 – Title Slide

Script: This project, *Code-A-Thon: Coding Interface With Secured Features*, is a complete enhancement of an already functional MERN-based online examination system.
While the existing system supported MCQs and basic coding questions, it lacked advanced admin control, security, and flexibility.
My contribution focuses on **building a secure coding interface, designing a dynamic exam creation system, and implementing a dedicated Admin Panel**, all without breaking or modifying the existing components used by students.
This ensures stability, scalability, and future-proofing of the platform.

Slide 2 – Introduction

Script: Online examinations today demand three major things: **flexibility**, **security**, and **scalability**.
Many platforms fail because:

- exams are hard-coded,
- no dynamic sections exist,
- coding tests lack security,
- and admin controls are limited.

The original MERN system was working but static.
Therefore, this project introduces **secure coding, dynamic sections, admin-driven exam creation, and strict anti-cheating mechanisms**, forming a complete modern assessment ecosystem.

Slide 3 – Problem Statement

Script: The existing system and traditional exam platforms suffer from:

1. **Rigid Structure:**
   Exams were fixed. Admins could not create new exams or restructure them.
2. **No Section Handling:**
   Exams couldn't be split into Section A, B, C with different time limits. This is essential for academic and competitive exams.
3. **Manual Question Entry:**
   Questions were manually inserted into the code or database, making it prone to errors and hard to maintain.
4. **Limited Coding Support:**
   Coding questions lacked secure execution, proper output handling, and multi-language support.
5. **Security Issues:**
   No tab-switch detection, no full-screen mode, no monitoring—making cheating easier.
6. **Fragile Architecture:**
   Any new feature risked breaking existing working components, making it difficult to experiment or scale.

These challenges highlight the need for a **secure, dynamic, admin-controlled, scalable online assessment solution**.

Slide 4 – Objectives

Script: The project sets out several technical and operational objectives:
Functional Objectives

- Build a **dynamic exam creation system** where admins can create exams anytime without code changes.
- Allow **multiple sections** with individual timers, question sets, and difficulty levels.
- Support **MCQ and coding questions** in the same exam.

Security Objectives

- Introduce a **secure coding interface** with real-time code execution.
- Implement **full-screen lock, tab-switch detection, minimize detection**, and warning system.

- Prevent cheating using browser activity monitoring.

System Objectives

- Ensure **no breaking changes** to existing user modules.
- Achieve **modular and scalable architecture**.
- Store every submission reliably in **MongoDB with timestamps**.
- Provide **admin analytics, dashboards, and CSV exports**.

These objectives ensure the system is secure, flexible, and long-term sustainable.

Slide 5 – Hardware & Software Requirements

Script:Hardware Requirements

- A basic Intel i3 or higher processor ensures Node.js and MongoDB run smoothly.
- 8 GB RAM recommended for running multiple services: backend server, frontend dev server, and database.
- Minimum 20 GB storage for project files, logs, node modules, and MongoDB data.
- Internet for API calls, code execution using Piston API, and online exam operations.
  - ❖ **Piston API (Brief Explanation)**
  - ❖ Piston API is an **open-source code execution engine** that allows you to **run code in multiple programming languages through an API**. It executes the code inside **secure sandbox containers**, so the user's code never touches or harms your server.
  - ❖ It is mainly used in **coding exams, online judges, and learning platforms** because it provides:
  - ❖ **Multi-language support** (C, C++, Java, Python, JavaScript, etc.)
  - ❖ **Real-time code output**
  - ❖ **Safe, isolated execution**
  - ❖ **Fast compile + run performance**
  - ❖ In your project, Piston API is used to power the **coding section**, allowing students to type code, execute it instantly, and view results securely without risking the backend system.

Software Requirements

- **React.js** for modular UI.
- **Node.js + Express.js** for API handling.
- **MongoDB** for dynamic and document-based storage.
- **VS Code** for development.
- **Git & GitHub** for version control.
- **Chrome/Edge** for running and testing the exam.

The requirements ensure stability, faster development, and smooth deployment.

Slide 6 – Existing System vs Proposed System

Script:

Existing System Limitations:

- Exams were pre-defined; no admin creation.
- No sections like A, B, C.
- Coding tests were basic and insecure.
- No security or anti-cheating measures.
- Hard-to-scale system due to tight coupling.

Proposed System Improvements:

- Fully functional **Admin Panel** for exam creation.
- **Section-wise structure** enabling multi-part exams.
- **Secure coding interface** with compiler integration.
- **Real-time question storage** in MongoDB.
- **Anti-cheating features** to maintain exam integrity.
- **Modular structure** to ensure high scalability.

This transformation upgrades the platform from a simple quiz tool to a **complete examination ecosystem**.

Slide 7 – Literature Survey

Script: Reviewed research papers, online assessment platforms like HackerRank, CodeChef, and university LMS systems.

Key findings:

- MCQ-only systems dominate the market.
- Coding examination platforms require heavy infrastructure.
- Secure coding environments are rare in academic tools.
- Existing tools often lack dynamic configuration and section-based exams.
- Only a few platforms offer browser activity monitoring.

From this study, our system adopts:

- Multi-language coding support
- Secure, monitored test-taking
- Section-based exam structure
- Real-time code execution
- Dynamic admin-driven exam creation

Slide 8 – Process Flow

Script:

Admin Workflow:

1. Admin logs in → authenticated using role-based login.
2. Creates exam → adds title, description, duration.
3. Adds sections → each with unique timing and question type.
4. Adds questions → MCQs or coding.
5. Publishes exam → instantly available to students.
6. Monitors real-time results → performance analytics.

Candidate Workflow:

1. Candidate logs in → verified by backend.
2. Reads instructions → agrees and enters full-screen secure mode.
3. Selects section → each section handled separately.
4. MCQ section → timer, navigation, monitoring.
5. Coding section → secure editor with real-time compiler support.
6. Submits exam → data stored instantly in MongoDB.
7. System exits full-screen → prevents further access.

This workflow ensures smooth functioning for both admin and candidates.

Slide 9 – Project Design

Script:

The architecture follows a **modular component-based design**:

Frontend (React)

- Pages: Login, Instructions, MCQ, Coding, Progress, Submission
- Admin Pages: Dashboard, Create Exam, Add Section, Add Questions
- Components: Timer, Warning Popup, Code Editor, Section Navigator
  This makes it easy to expand and maintain.

Backend (Node.js + Express)

- Routes for: authentication, exams, sections, MCQs, coding questions, submissions
- Middleware for: role verification, error handling, security
- Services: Piston integration, result processing

Database (MongoDB)

Collections include:

- users

- exams
- sections
- MCQs
- codingQuestions
- submissions
- codingOutputs
- securityLogs

Schemas were **extended**, not replaced—so legacy data stays safe.

Slide 10 – Project Implementation
Script:
Backend Implementation:

- Created REST API endpoints for exam creation
- Added JWT-based authentication
- Added APIs for storing coding outputs
- Added tab-switch logs and warning counts

Frontend Implementation:

- Reusable UI components for MCQs, coding, timers
- Full-screen lock using browser APIs
- Warning popup triggered by event listeners
- Code editor integrated with Piston API
- CSV export using admin dashboard button

Security Integration:

- Detect minimize, tab-switch, window blur
- Lock navigation
- Warn users on every violation

This implementation ensures a high-security, dynamic examination platform.

Slides 11–24 – Screenshots and Detailed Explanation
Below are **deep explanations** for each interface.

Slide 11 – Intro Screen
Purpose: branding, identity, event name, department details.
This creates a professional impression and separates user-level content from system-level content.

Slide 12 – Admin Login
Secure login using hashed passwords and JWT tokens.
Only admins can configure exams.
Prevents unauthorized exam manipulation.

Slide 13 – Admin Dashboard
Displays analytics:

- number of users
- exam count
- average marks
- section performance
- real-time submissions
  Allows admin to take decisions based on live data.

Slide 14 – CSV Export

Admin clicks “Export CSV”.
System converts MongoDB data into CSV format.
Useful for:

- academic record keeping
- offline evaluation
- research purposes

Slide 15 – User Login
Simple and secure login for candidates.
Redirects based on role—admin or student.

Slide 16 – Exam Instructions
Contains rules, time limits, security conditions.
User must agree, ensuring legal and academic compliance.

Slide 17 – Section Selection
Sections appear with timers and statuses.
Full-screen mode begins here.
Users cannot skip or return after submitting a section.

Slide 18 – MCQ Screen
Displays:

- current question
- options
- timer
- next/previous buttons
  Also tracks answered and unanswered questions.

Slide 19 – Security Warning Popup
Triggered when user:

- switches tabs
- minimizes window
- exits full-screen
  System logs warnings stored in MongoDB.
  Ensures exam integrity.

Slide 20 – Section B Screen
Different set of MCQs.
Shows modularity of sections.
Each section functions independently.

Slide 21 – Coding Assessment Interface
The heart of the system.
Features:

- multi-language support
- code execution using Piston API
- run output
- error messages
- final submit button
  Also monitored for security violations.

Slide 22 – Section Progress
Shows completed and pending sections.
Helps candidates manage time and priority.

Slide 23 – Assessment Completion
User sees all sections completed.
Prevents entering any section again.
Moves user to final submission.

Slide 24 – Final Submission Confirmation
Once submitted:

- results stored in database
- timestamps recorded
- full-screen mode exited
- session ended
- results available for admin monitoring

Slide 25 – Tools & Technologies
Script:
Frontend:
React.js, HTML, CSS, JavaScript
Backend:
Node.js, Express
Database:
MongoDB
Code Execution:
Piston API
Utilities:
Axios, VS Code, GitHub, Chrome/Edge
Together these tools ensure rapid development, modularity, and stable performance.

Slide 26 – Conclusion
Script: The project successfully upgrades the existing system into a modern, secure, scalable online assessment platform.
It introduces:

- dynamic exam creation
- secure coding interface
- full-screen enforcement
- anti-cheating monitoring
- real-time result storage
- analytics and CSV export
- modular MERN architecture

This system is suitable for **academic exams, competitive tests, coding competitions, and recruitment drives**.
The architecture allows future features like AI-based evaluation, plagiarism detection, or a proctored video system.