A REPORT OF ONE MONTH TRAINING
at
Ryaz.io Technologies

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT
FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

Computer Science and Engineering

JUNE-JULY , 2025

Submitted by : Ravjot Singh

CRN - 2315191

URN - 2302646

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

GURU NANAK DEV ENGINEERING COLLEGE , LUDHIANA

(An Autonomous College Under UGC ACT)

CERTIFICATE

RYAZIO
TECHNOLOGIES
LLP

Training Completion Certificate

REF:
RYAZ/T-128/0523

DATE:
1st August 2025

TO:
Training & Placement
Officer,
Guru Nanak Dev
Engineering College,
Ludhiana.

Dear Sir/Madam,

This is to certify that **Mr. Ravjot Singh** (Roll no.: **2302646**), a student of B. Tech- Computer Science and Engineering of your institute has successfully completed his 1 month Industrial Training at Ryazio Technologies LLP from 23rd June, 2025 to 21st July, 2025.

We acknowledge his dedication and commitment throughout the training period and wish him all the best for his future endeavours.

For more information, you can contact us at **info@ryaz.io**.

Sincerely,
Inderpreet Singh,
Director, Engineering,
Ryazio Technologies LLP

Office Address:
Ryazio Technologies LLP
H. No. 188/2,
Near Raksha Madam,
Water Tank Road, Samrala,
Ludhiana, PUNJAB (141114)
GSTIN: 03AAZFR3160P1ZI
LLPIN: AAO-4043

Declaration

I hereby certify that the work which is being presented in this training report for partial fulfillment of requirements for the award of degree of B.Tech. (Computer Science) submitted to the Department of Computer Science Engineering at **GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA under I.K. GUJRAL PUNJAB TECHNICAL UNIVERSITY** is an authentic record of my own work at Ryaz.io Technologies , Ludhiana, during 4-Week Industrial (Summer) Training - TR-102.

Ravjot Singh

URN-2302646

CRN-2315191

ABSTRACT

This report documents the work carried out during the one-month industrial training at Ryaz.io Technologies, focusing on backend development. Core web-based projects were developed using Node.js, Express, MongoDB, Socket.IO, and CRDTs. These projects included a URL Shortener,, Messaging App, and Collaborative Text Editor. The report highlights the daily progress, key technologies learned, challenges faced, and the outcomes achieved. Emphasis was placed on modular design, secure authentication, real-time communication, and collaborative synchronization. This training helped improve backend development skills and deepened understanding of real-world web application workflows.

Key Learnings :

1. **Backend Development with Node.js and Express**
    - Developed RESTful APIs with route handling, middleware, and modular controller structure.
    - Gained experience in server setup, routing strategies, and request/response lifecycle management.
2. **Authentication and Authorization**
    - Implemented secure user authentication using **JWT**, **bcrypt**, and **cookie-based sessions**.
    - Learned role-based access control and session persistence strategies.
3. **Database Design and MongoDB**
    - Designed schemas using **Mongoose**, used aggregation pipelines, and optimized queries.
    - Understood relational references (e.g., blog-user, messages-sender) and embedded vs referenced document trade-offs.
4. **Real-time Communication**
    - Built a **real-time messaging system** using **Socket.IO**.
    - Implemented public and private chats, message timestamping, and dynamic room handling.
5. **Collaborative Editing with CRDTs**

- Learned about **Conflict-free Replicated Data Types (CRDTs)** for real-time collaboration.
- Integrated **Yjs** for syncing document updates across multiple users using WebSockets.

6. **Frontend Integration**
   - Designed minimal frontends for testing real-time features and data flow between backend and client.
7. **File Handling and Cloud Storage**
   - Used **multer** for handling file uploads and **Cloudinary** for external file storage.
   - Dealt with private/public asset permissions and secure download/preview logic.
8. **Debugging and Problem Solving**
   - Learned to trace errors through console.log(), network tools, and GitHub Copilot suggestions.
   - Understood importance of incremental development, refactoring, and test-first approaches.
9. **Modular Code Architecture**
   - Practiced structuring apps into controllers, routes, services, and models.
   - Ensured maintainability, scalability, and ease of debugging across all projects.
10. **Version Control and Collaboration Tools**

- Used **GitHub** for code management and referencing.
- Understood how to use online developer resources effectively (e.g., GitHub issues, StackOverflow).

Acknowledgement

I take this opportunity to express my sincere gratitude to all those who have supported me throughout the course of my one-month industrial training.

First and foremost, I am deeply thankful to **Ms. Uttamdeep Kaur**, my industry mentor at **Ryaz.io Technologies, Ludhiana**, for her constant guidance, valuable insights, and support during the training. Her mentorship was instrumental in enhancing my technical skills and understanding of real-world software development practices.

I would also like to extend my heartfelt thanks to the **Department of Computer Science and Engineering, Guru Nanak Dev Engineering College, Ludhiana**, for providing me with the opportunity to undergo this training and apply my academic knowledge in a practical environment.

I am especially grateful to the entire team at Ryaz.io Technologies for their welcoming environment, constructive feedback, and collaborative spirit. Working alongside professionals in a dynamic setting has been an enriching experience.

Lastly, I would like to thank my family and friends for their continuous motivation and encouragement throughout this journey.

This training has not only helped me strengthen my programming and problem-solving abilities but also provided exposure to the development process, team collaboration, and industry-standard tools and practices.

Ravjot Singh

CRN-2315191

URN-2302646

About Company

Ryaz.io Technologies is a reputed technology-driven company based in **Ludhiana, Punjab**, specializing in software development, web technologies, and IT training services. It is an online-based software development and training company that operates remotely, providing project-based learning and mentorship through virtual platforms without a physical office setup. The organization offers a range of services that include web application development, backend system architecture, API integration, cloud-based solutions, and full-stack training programs for students and professionals.

Founded with the vision to bridge the gap between academic knowledge and industrial requirements, it focuses on practical learning experiences, hands-on projects, and mentoring to equip students with industry-relevant skills. The company actively works with emerging technologies such as **React , next.js , node.js , fast Api , postgressSOL , mongoDB etc.**

Key highlights of the company include:

- Offering professional training in various fields like Web-Technologies , backend development etc .
- Hosting regular industrial training and internship programs for engineering and computer science students.
- Mentoring project-based learning to help trainees build complete end-to-end applications.
- Maintaining a team of experienced software developers, mentors, and industry professionals.
- Encouraging open-source contribution, agile practices, and peer-reviewed code.

The organization promotes a collaborative and innovative environment where learners are encouraged to ask questions, explore different solutions, and grow both technically and professionally.

Contents

1. Cretificate 1

2. Declaration 2

3. Abstract 3

4. Acknowledgement 5

5. About Company 6

6. Content 7

7. Chapter 1 - Introduction 9

1.1 – Background……. 9

1.2 – Objective……. 9

1.3 – Scope of training……. 9

1.4 – Tools and technologies used……. 10

1.5 – Relevance to academic curriculum……... 10

8. Chapter 2 – Training work undertaken 11

2.1 – Backend application architecture…….. 11

2.2 – Database design and management…… 11

2.3 – Authentication and authorization…….. 12

2.4 – Real-time communication and websockets……... 12

2.5 – Session and state management…….. 12

2.6 – File upload and cloud integration……. 13

2.7 - Error handling and debugging…….. 13

2.8 – Frontend integration…….. 13

2.9 – Deployment and testing…… 13

2.10 – URL shortener api……... 14

2.11 – Real time message application…… 15

9. Chapter 3 – Result and discussion 18

3.1 - Functional application developed……. 18

3.2 – Skills and technologies applied…… 18

3.3 – Key challenges faced……... 18

3.4 – Overall outcome……... 19

10. Chapter 4 – Conclusion and future scope 20

4.1 – Conclusion……... 20

4.2 – Future scope……. 20

CHAPTER 1: INTRODUCTION

1.1 Background

The rapid growth of digital technologies has transformed the way people interact, communicate, and access information. Web-based platforms, collaborative tools, and real-time applications have become central to various domains—from education and healthcare to business and entertainment. As a student pursuing a degree in computer science and engineering, gaining hands-on experience in these areas is crucial to bridge the gap between academic concepts and real-world application.

The one-month industrial training at Ryaz.io Technologies, Ludhiana was focused on full-stack web development using modern technologies such as Node.js, Express.js, MongoDB and Socket.IO. This training provided a practical platform to develop scalable and real-time applications.

1.2 Objectives

The major objectives of the industrial training were:

- To gain hands-on experience in backend development using JavaScript-based technologies.
- To understand the principles of RESTful API design and implementation.
- To develop real-time features using WebSockets for instant communication.
- To learn secure authentication practices using JWT and session tokens.
- To debug, modularize, and deploy code effectively.
- To enhance problem-solving skills through project-based learning and code refactoring.

1.3 Scope of Training

The scope of the training encompassed:

- Designing and implementing backend logic using Node.js and Express.
- Creating and managing databases using MongoDB and Mongoose ORM.
- Integrating authentication and session management for secure access.
- Handling file uploads and external cloud storage with Multer and Cloudinary.

- Developing and testing real-time communication using Socket.IO.
- Creating minimal front-end templates using EJS and integrating with backend APIs.
- Hosting, testing, and debugging the applications locally.

1.4 Tools and Technologies Used

Category	Tools/Technologies
Backend Framework	Node.js, Express.js
Database	MongoDB, Mongoose
Authentication	JSON Web Tokens (JWT), bcrypt, cookie-session
Real-time Communication	Socket.IO, WebSocket
File Upload/Storage	Multer, Cloudinary
Version Control	Git, GitHub
Development Tools	VS Code, Postman, Nodemon, Chrome DevTools

1.5 Relevance to Academic Curriculum

This training aligns closely with the core subjects of Information Technology such as:

- Web Technologies
- Data Structures and Algorithms
- Database Management Systems
- Software Engineering

By practically applying concepts from these subjects, the training helped enhance the learning outcomes of my academic curriculum and also prepared me for real-world industry challenges.

CHAPTER 2 – TRAINING WORK UNDERTAKEN

During my one-month industrial training at Ryaz.io Technologies, I explored and implemented multiple core aspects of backend web development. The training was structured around solving real-world problems using modern development stacks. The entire experience was divided across several focused themes, allowing me to develop deep insights into how various components of a full-stack application operate together.

2.1 Backend Application Architecture

- Designed and implemented backend services using Node.js and Express.js.
- Established modular file structures to manage routes, controllers, middlewares, and models.
- Followed RESTful API conventions for designing endpoints for CRUD operations.
- Utilized Mongoose to model and interact with MongoDB collections efficiently.
- Integrated dotenv to manage environment variables securely.
- Implemented server-side rendering using EJS for dynamic web pages in some modules.

2.2 Database Design and Management

- Created multiple MongoDB schemas using Mongoose for entities like URLs, blogs, messages, users, and documents.
- Handled schema validations, default values, timestamps, and relationships across collections.
- Implemented advanced MongoDB operations such as aggregation pipelines, pagination, and filtering.
- Stored session data and user information securely in MongoDB to enable persistent login.

2.3 Authentication and Authorization

- Implemented user registration and login flows with proper validation.
- Secured APIs using JWT (JSON Web Tokens) for stateless authentication.
- Stored tokens in cookies and created middleware to verify tokens on protected routes.
- Used bcrypt to securely hash and store user passwords.
- Controlled access to routes and features based on user roles and session states (e.g., message ownership).

2.4 Real-Time Communication and WebSockets

- Built real-time messaging and collaboration features using Socket.IO.
- Implemented public chat functionality and extended it to support Direct Messaging (DM) using private rooms.
- Managed socket events like connection, disconnection, message broadcast, and room joining.
- Applied event-based architecture to decouple front-end and back-end interactions.
- Debugged challenges in event synchronization and route isolation for DMs.

2.5 Session and State Management

- Used express-session and connect-mongo to manage server-side sessions.
- Persisted sessions across reloads and interactions using secure cookies.
- Debugged issues related to session inconsistency across views, especially for rendering dynamic content.

2.6 File Upload

- Integrated Multer to support file uploads in chat and blog features.
- Configured Cloudinary for cloud storage of media files.
- Resolved bugs related to duplicate file deletions and incorrect upload paths.

2.7 Error Handling and Debugging

- Developed consistent error-handling middleware across all Express apps.
- Utilized tools like console tracing, Postman, and GitHub Copilot to detect logic errors.
- Learned to handle client/server communication failures, including file not found, session timeouts, and token expiration.
- Refactored and debugged controller logic to resolve authentication bugs and update failures.

2.8 Frontend Integration

- Built basic interfaces using HTML, CSS, and JS templates to test backend APIs.
- Rendered data conditionally using values passed from backend routes (e.g., user sessions, messages).
- Integrated features like dashboard lists, and real-time message rendering.
- Connected backend logic to user interaction components such as buttons, input fields, and status indicators.

2.9 Testing

- Tested all applications locally with Nodemon, Postman, and socket clients.
- Refactored codebases into logical modules for reusability and deployment readiness.
- Emulated live user behavior through test UIs and simultaneous client connections.
- Deployed a few components to local production setups to simulate live environments.

2.10 URL Shortener Apis

The URL shortener project aimed to reduce long URLs into compact, shareable links and handle redirection to the original addresses.

Key Tasks and Learnings:

- **Backend Setup with Node.js, Express, and MongoDB**
  The backend was built using Node.js for its asynchronous, event-driven model, along with Express.js to manage HTTP routing efficiently. MongoDB was used as the database due to its flexible schema design, making it ideal for storing URL mappings. Express routes handled incoming requests for generating short links and redirecting to the original ones, while MongoDB stored long URL, short code, and metadata in separate collections.
- **Unique Short Code Generation using nanoid**
  To generate short and unique identifiers, the nanoid package was implemented. It produces random alphanumeric strings with very low collision probability, ensuring each short link was unique. For example, a 7–10 character code generated by nanoid provides billions of unique combinations, making it highly scalable.
- **Schema Design for URLs**
  A Mongoose schema was created to store the original URL, its shortened counterpart, the creation date, and a counter to track visits. Indexes were added on the shortCode field to improve query performance since redirection requests often depended on fast lookups.
- **Exploring the Crypto Module for Custom Shortener**
  Apart from third-party libraries, Node's built-in crypto module was explored to hash URLs using algorithms like SHA-256 or MD5. Although this produced consistent hashes, they were longer than needed, so truncation and base encoding techniques were tested. This experiment provided insights into building URL shorteners without external dependencies.
- **Redirect Logic Implementation**
  A dedicated Express route was written to accept incoming requests with short codes, query MongoDB for the original URL, and perform a 301 redirect. Middleware ensured proper error handling if the short code did not exist.
- **Analytics Tracker**
  Each time a short URL was accessed, the database counter for that link was incremented. This allowed the system to maintain **usage statistics**, giving insights into the popularity and frequency of link visits. Such analytics can be extended to include geolocation and device details using request headers.

- **Modular Code Refactoring**
  The project was restructured into routes, controllers, models, and middlewares for scalability. This modular design adhered to the MVC (Model-View-Controller) pattern, improving readability and maintainability.

2.11 Real-Time Messaging Application

This application was designed to facilitate real-time public and private communication between users.

Key Tasks and Learnings:

- **Socket.IO-based Chat System**
  The backbone of the application was built on Socket.IO, which abstracts WebSockets and provides fallback mechanisms like long polling. This ensured compatibility across browsers. Events such as connection, message, disconnect, and custom events like joinRoom were created. Each event listener handled client-server communication, and acknowledgment callbacks were used to confirm message delivery.
- **Message Schema and Persistence**
  Messages were not only exchanged in real time but also persisted in MongoDB using Mongoose. Each message document contained details such as senderId, senderName, content, timestamp, and optional flags like isEdited or isDeleted. This schema design ensured reliability, so even if the server restarted, message history could be retrieved and displayed to users.
- **Session Handling with express-session and MongoDB Store**
  User sessions were handled using express-session combined with connect-mongo. This enabled persistent sessions, meaning users didn't need to re-login after refreshing their browser. Sessions stored user IDs and usernames, which were critical for associating messages with the correct user, enforcing permissions, and maintaining consistent identity across socket connections.
- **Edit and Delete Functionalities**
  Users could edit or delete only the messages they authored. This was enforced by validating the senderId stored in session against the message's author. When a message was edited, an (edited) label was appended dynamically. Deleted messages were replaced with a placeholder such as *"This message has been deleted"*. These actions were also synchronized across all connected clients through broadcasted socket events.
- **Direct Messaging with Private Rooms**
  To implement private communication, Socket.IO's room mechanism was leveraged. Each DM session was assigned a unique room ID, typically based on the concatenation of user IDs (e.g., room_user1_user2). When two users started a

DM, both sockets were joined into the same room, ensuring that messages were only visible to the intended recipients. This approach scaled well as more users could join public rooms while still maintaining private interactions.

- **Route Restructuring and Modularization**
  As the project grew, routes became difficult to manage. To solve this, the code was refactored into modular layers:

  Routes: Defined endpoints for chat features (/messages, /edit, /delete).

  Controllers: Handled business logic like saving messages, updating/deleting messages, and fetching chat history.

  Socket Handlers: Managed real-time events independently of REST endpoints.
  This separation adhered to best practices like the MVC (Model-View-Controller) pattern.

- **Typing Indicators and Presence Awareness**
  To improve user experience, a typing indicator was introduced. When a user started typing, a typing event was emitted and broadcasted to other users in the same room. Similarly, user presence (online/offline status) was managed by listening to connection and disconnect events, and the frontend updated the user list dynamically.
- **Error Handling and Reliability**
  The system included socket-level error handling. For example:
  If a user tried to send a message without a valid session, the server returned an error event. If a room did not exist, the user was redirected to a default lobby.
  Additionally, reconnection logic in Socket.IO ensured that if a user lost network connectivity, they would rejoin automatically and recover missed messages from MongoDB.
- **Scalability and Performance Considerations**
  The architecture was designed with scalability in mind. Socket.IO can be scaled horizontally using Redis Adapter, which synchronizes socket events across multiple server instances. Even though this was not implemented during training, the design choices (modular routes, room-based messaging, and event-driven architecture) prepared the system for future expansion to support larger user bases.
- **Testing Interface for Real-Time Features**
  A lightweight testing interface was developed using plain HTML, CSS, and the Socket.IO client library. This interface allowed the simulation of multiple users on different browser tabs. It was particularly helpful for testing features like public chat broadcasting, private messaging, editing, deleting, and real-time

typing indicators. This minimal setup ensured faster debugging without requiring a fully styled UI.

CHAPTER 3: RESULTS AND DISCUSSION

3.1 Functional Applications Developed

- **URL Shortener Service:**
  A fully functional backend system that accepts long URLs and returns shortened versions. It supports custom code generation and redirection. The final system includes click tracking and analytics, along with a simple frontend form.

- **Messaging Application (Public & Private Chat):**
  A real-time chat app developed using Socket.IO, with support for public channels, direct messaging (DM), session-based message access control, and file uploads via Cloudinary. Users could send, edit, and delete messages with proper permission control.

3.2 Skills and Technologies Applied

- **Backend-Development:**
  Gained expertise in Node.js and Express.js by building scalable APIs, organizing code into modular components, and integrating middleware for validation and error handling.

- **Database-Management:**
  Designed and implemented MongoDB schemas using Mongoose. Understood data modeling, relationship handling, and advanced queries like aggregation and pagination.

- **Authentication.and.Security:**
  Learned and implemented secure authentication using JWTs and password hashing with bcrypt. Also handled token verification in real-time systems (sockets), which was a crucial learning experience.

- **Real-Time Communication:**
  Built real-time application using WebSockets and Socket.IO. Understood event-driven programming and how client-server synchronization works in practice.

- **File Uploads and Cloud Integration:**
  Integrated Multer for file handling and Cloudinary for storage and preview.

3.3 Key Challenges Faced

- **Session and State Handling:**
  Managing session data between backend and frontend, especially when rendering

dynamic content or updating socket connections, led to multiple debugging cycles.

- **Asynchronous Bugs and Token Handling:**
  Several issues occurred due to async logic errors, token expiration, and failure to attach auth headers to WebSocket clients. These were resolved using log tracing and tool-based diagnosis.

- **File Handling Errors:**
  In the messaging app, the uploaded files initially returned base64 strings or failed to preview due to incorrect pathing and configuration.

- **Frontend and Backend Sync:**
  Problems arose when the frontend logic didn't match backend expectations—particularly in socket rooms and dashboard rendering. These were fixed by revisiting API contracts and reviewing state flow.

3.4 Overall Outcome

By the end of the training period, I successfully delivered backend-driven application with production-ready features. The experience helped solidify my understanding of web development fundamentals, real-time communication, authentication systems, and collaborative technologies. Beyond technical skills, I learned how to debug effectively, modularize large codebases, and document my work systematically.

CHAPTER 4: CONCLUSION AND FUTURE SCOPE

4.1 Conclusion

Over this one-month training , I developed a deeper understanding of backend development, real-time communication, and collaborative systems. By completing four core projects, I explored multiple layers of application architecture including RESTful APIs, session handling and file storage. The experience helped me strengthen not only my technical proficiency but also my debugging and problem-solving capabilities.

4.2 Future Scope

- Add roles and dashboards to chat apps.
- Deploy apps using Docker and CI/CD pipelines.
- Enhance editor with rich text formatting and document history.
- Implement E2E encryption for messaging.
- Explore P2P sync and offline storage for collaborative tasks.