OTMT Platform Final Report & Legacy Documentation

A comprehensive overview of the project's development, architecture, and operational guidelines.

Amartya Singh
amartya22062@iiitd.ac.in

Anish
anish22075@iiitd.ac.in

June 19, 2025

Contents

1 Introduction

This document serves as the final technical report and legacy documentation for the Office of Technology Management and Transfer (OTMT) digital platform project. The primary objective of this initiative was to design, develop, and deploy a modern, robust, and scalable digital ecosystem to manage, showcase, and facilitate the institution's innovations, research projects, and intellectual property.

The project was undertaken by Anish and Amartya Singh. The delivered solution is a multi-faceted platform comprising several interconnected components:

- **A public-facing website** serving as the primary information hub.
- **A user portal and administrative panel** for managing platform content and user-submitted technologies.
- **A native Android mobile application** for on-the-go access.
- **A custom brochure generation pipeline** to create professional marketing materials.
- **An AI-powered chatbot** to assist users with technology inquiries and TRL (Technology Readiness Level) assessments.

This report details the project's entire lifecycle, from the initial proposal to the final, deployed architecture. It is intended for two primary audiences: our supervising professor, as a comprehensive summary of the work accomplished, and the OTMT office, as a technical handover document to guide future maintenance, operation, and development.

2 The Initial Plan

The project commenced with a structured plan that outlined the core objectives and a phased development timeline. This initial roadmap served as a foundational blueprint, establishing the project's scope and key deliverables, which would later evolve based on technical discoveries and stakeholder feedback.

2.1 Proposed Roadmap and Milestones

The initial development was mapped out in a week-by-week schedule. The strategy was to build the platform incrementally, with a logical progression through different technology layers:

1. **Weeks 1-4: Frontend Development.** The first phase was dedicated to addressing existing frontend issues, fixing bugs, and refining the User Interface (UI) and User Experience (UX) to align with modern design and accessibility standards.
2. **Weeks 5-7: Backend Development.** Following the frontend, the focus was to shift to setting up the server-side environment, designing the database schema, and developing core REST APIs, including a user authentication system.
3. **Weeks 8-9: Admin Panel.** This phase involved the creation of an administrative dashboard for content and user management.

4. **Weeks 10-12: Mobile Application.** The plan included the development of a mobile app to extend the platform's reach, followed by end-to-end testing of the entire integrated system.

5. **Week 13 onwards: Deployment.** The final weeks were allocated for security hardening, performance optimizations, and the deployment of the web platform and mobile application.

2.2 Initial Technology Stack

The proposed technology stack was based on modern, widely-adopted frameworks and tools to ensure maintainability and scalability:

- **Frontend:** React.js, Tailwind CSS, Material-UI
- **Backend:** Node.js with Express.js
- **Database:** MongoDB (NoSQL)
- **Mobile App:** React Native (cross-platform)
- **Authentication:** Firebase Authentication

This initial plan provided the necessary structure to begin development, while remaining flexible enough to accommodate the iterative and adaptive approach that ultimately defined the project's successful execution. The subsequent sections of this report will detail the journey from this plan to the final, delivered system.

3 The Development Journey: Challenges and Solutions

The development process was highly iterative and adaptive. While the initial plan provided a strong foundation, the final product was shaped by continuous stakeholder feedback, proactive technical improvements, and pragmatic solutions to unforeseen challenges. This section chronicles the evolution of each major component, detailing the problems encountered and the solutions implemented.

3.1 The Public Website: From Replication to Refinement

- **Initial Task:** The project began with the goal of replicating and completing a pre-existing, half-built website. Initial designs were created to match this existing structure.
- **Challenge:** Upon review, we identified significant deficiencies in the original design concerning User Experience (UX) principles and accessibility standards. A direct replication would have resulted in a suboptimal and difficult-to-use platform.
- **Solution:** A strategic decision was made to perform a complete UI/UX overhaul.
  - We first presented multiple new design mockups to stakeholders, finalizing a clean, professional, white-based theme.

  - The technology stack was refined; we moved away from Tailwind CSS in favor of **Material-UI (MUI) with custom CSS**. This provided greater control over component styling and ensured we could meet the specific, custom design requirements. * The entire website structure was re-imagined. We restructured layouts to prioritize important information, improved button accessibility, and added fluid interactions and animations to enhance the user journey.

- **Scope Expansion:** Following a demonstration to our supervisor, Mr. Alok Nikhil Jha, the scope was expanded to include new pages such as 'Our Research' and a redesigned 'Our Partners' page. Concurrently, the 'Tech Finder' module—a database showcasing institutional technologies—was integrated seamlessly into the revamped UI, transforming the website into a unified platform.

3.2 Backend Architecture and Mobile Application

- **Backend Development:** We designed and developed a robust backend using **Node.js with Express.js** and a **MongoDB** database.
  - **Challenge:** The backend needed to serve data to multiple frontends (web, mobile app, chatbot) securely and efficiently.
  - **Architectural Solution:** We implemented a **microservices-inspired, two-backend architecture**. A primary, general-use backend was created with **read-only permissions** to the database. Its sole purpose is to expose data securely via REST APIs to all public-facing clients. This design minimizes the attack surface, as public clients can never execute write operations.

- **Mobile Application Development:**
  - **Challenge:** The initial plan suggested React Native. However, to ensure optimal performance, a native user experience, and full access to platform-specific features, a different approach was required.
  - **Solution:** We developed a fully native Android application using **Kotlin and XML**. Development began after the website's UI revamp, which allowed us to maintain perfect brand and design consistency. The application features all key informational pages and a custom-integrated Tech Finder, all powered by the secure, read-only general backend.

3.3 The Admin Panel: A Three-Stage Evolution

The admin panel underwent the most significant transformation, evolving through three distinct versions in response to changing requirements.

1. **Version 1: The Secure Internal Tool**
   - **Purpose:** An internal tool for OTMT staff to perform CRUD (Create, Read, Update, Delete) operations on technologies and events.
   - **Technology:** Built on the MERN stack (MongoDB, Express.js, React, Node.js) with a Vite-powered frontend.

- **Key Security Feature:** This panel was powered by its own dedicated, **write-enabled backend**. This backend was the only service with permissions to modify the database. It communicated with the frontend via a separate set of API endpoints secured with JWT (JSON Web Token) for session management. This architecture completely isolated write operations from the public internet.

2. **Version 2: The ERP-Style Office Panel**

   - **Trigger:** Feedback indicated the need for a more formal, role-based system for internal office use.
   - **Solution:** The panel was rebuilt to incorporate **Firebase Authentication** for secure login via institutional Google accounts. We implemented a three-tier Role-Based Access Control (RBAC) system:

     Super Admin: Full control, including managing user permissions.

     Admin: Full CRUD control over technologies and events, but cannot manage users.

     Employee: Can only add new technologies and events.

3. **Version 3: The Public-Facing User Portal**

   - **Trigger:** A final requirement shift to empower individual researchers and students to manage their own creations.
   - **Solution:** The panel was re-architected into its final form. Now, any authenticated user (professor, student) can log in to view, edit, and manage **only their own submitted technologies**. The OTMT Admin role retains its global oversight.
   - **New Feature - The Recycle Bin:** To prevent accidental data loss, a data purging system was introduced. When a user ”deletes” a technology, it is moved to a separate collection and remains recoverable for 30 days before being permanently deleted by an automated process.

3.4 The Brochure Generation Pipeline: A Custom Solution

- **Challenge:** The initial plan to use the Canva API proved unsuitable for our specific workflow and customization needs.
- **Solution:** We engineered a completely bespoke brochure generation pipeline as a standalone **Vite** application.
- **Workflow and Features:**

  1. Users upload technology data via a predefined CSV or Excel template (exported from Google Forms).
  2. The application automatically populates a professionally designed, modular PDF template with this data.
  3. **Real-Time PDF Editor:** The system provides an in-browser interface where users can directly edit the text of the generated PDF in real-time before finalizing.

4. **QR Code Integration:** For each brochure, the system generates a unique QR code, by the URL given, that links directly to that technology's detailed page on the main OTMT website, seamlessly bridging physical and digital materials.

3.5 The Chatbot: Adapting to Infrastructure Constraints

- **Version 1: Self-Hosted RAG Bot**
  - **Initial Design:** A RAG (Retrieval-Augmented Generation) pipeline using a hybrid retrieval model (BM25 keyword search + semantic vector search). It was designed to run on a local GPU server using open-source models via LM Studio or Ollama.
- **Challenge:** A critical institutional IT policy prohibited the use of dedicated GPU servers for this project, making the initial design unfeasible for deployment.
- **Version 2 (Final): The Cloud-Based Agentic Pipeline**
  - **Solution:** We re-architected the entire chatbot into a more sophisticated and scalable agentic system that relies on a third-party LLM API.
  - **Technology:** The backend was rebuilt using **FastAPI** and **LangChain**, connecting to the **Google Gemini 1.5 Flash API**.
  - **Agentic Workflow:**
    1. The agent first fetches and caches live technology data from our general-purpose backend API.
    2. When a user query is received, an LLM call is made to perform **intent classification** (is the user asking for 'tech-info', 'TRL-assessment', or 'general' info?) and extract relevant entities.
    3. Based on the intent, the agent decides its next action: retrieve specific documents from its cache for 'tech-info' queries, or proceed directly for other query types.
    4. A final LLM call generates the response, using the chat history, system prompt, and any retrieved data as context.
  - **UI/UX:** The chatbot was implemented as a simple, non-persistent pop-up window on the homepage, positioning it as an efficient, on-demand help tool rather than a conversational companion.

4 Final System Architecture and Workflow

The final deployed system is a robust, decoupled, and secure ecosystem. This section provides a technical overview of the complete architecture, from the server infrastructure to the application-level workflows.

4.1 Infrastructure and Deployment

The entire platform is deployed on a Linux server, configured for security, performance, and high availability.

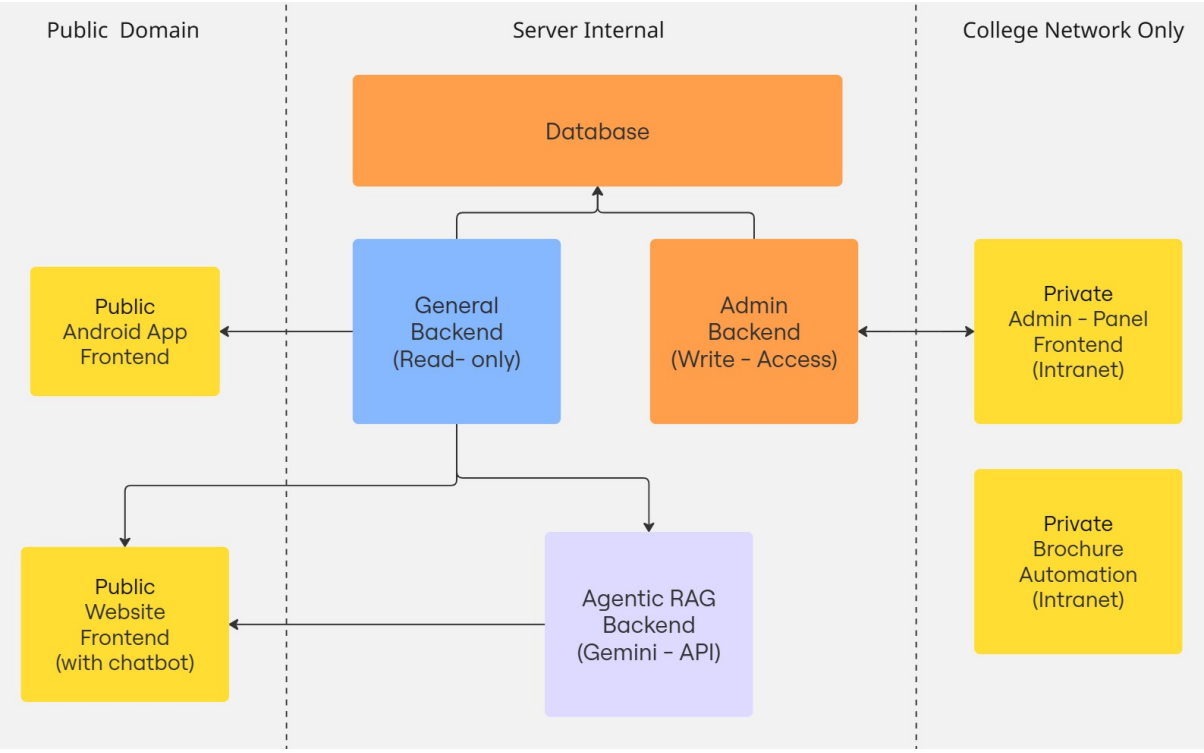

Figure 1: High-Level System Architecture Diagram

- **Reverse Proxy:** **NGINX** is used as a reverse proxy to manage all incoming HTTP/S traffic. It routes requests to the appropriate backend service and handles SSL/TLS termination.
- **Security:** SSL certificates are installed to enforce **HTTPS** across all services. A critical security measure is the network segregation of services:
  - The public website and general APIs are accessible via a public domain.
  - The Admin/User Portal and its write-enabled backend are deployed on an **intranet-only accessible domain**, making them unreachable from the public internet.
- **Process Management:** All Node.js applications are managed by **PM2**, a production process manager that provides automatic restarts on failure, load balancing across CPU cores, and continuous uptime.
- **Builds:** All frontend applications are served as highly optimized, minified **production builds** to ensure minimal load times for end-users.

4.2 Backend Architecture: The Two-Backend Model

The core of the system's security and scalability lies in its two-backend design.

General Backend (Read-Only)

- **Stack:** Node.js, Express.js.
- **Purpose:** Exposes data from the MongoDB database via REST APIs. It has read-only credentials and serves as the single source of truth for the public website, the mobile app, and the chatbot's data caching mechanism.

Admin Backend (Write-Enabled)

- **Stack:** Node.js, Express.js.
- **Purpose:** Handles all CRUD operations. It is exclusively used by the Admin/User Portal and is the only service with write permissions to the database.
- **Security:** Access is restricted by JWT-based session authentication and network-level isolation (intranet-only).

4.3 Application and Service Workflows

Admin/User Portal Workflow

1. A user navigates to the portal's intranet URL.
2. They authenticate via Google using the Firebase Authentication integration.
3. Firebase returns a token, which is used to create a session with our write-enabled admin backend.
4. The React frontend makes authenticated API calls to the admin backend to view, add, or edit technologies, respecting the user's role and permissions.

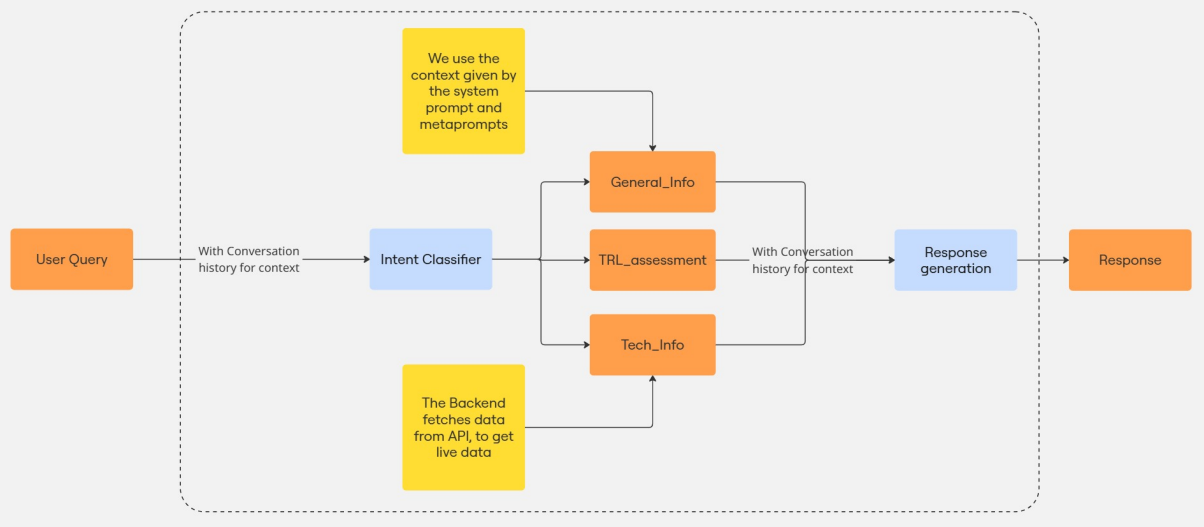

Figure 2: High-Level diagram of the chatbot pipeline

Chatbot Agentic Workflow

1. The FastAPI service starts and caches the latest technology data from the General Backend API.
2. A user opens the chatbot UI and sends a message.
3. The message and chat history are sent to the Gemini API for intent classification and entity extraction.
4. The FastAPI application analyzes the intent:
   - If 'tech-info', it performs a search on the cached JSON data to find relevant context.
   - If 'TRL-assessment' or 'general', it proceeds without retrieval.
5. A final, comprehensive prompt (including system instructions, history, and retrieved context) is sent to the Gemini API to generate the final response.

Brochure Generator Workflow

1. A user opens the standalone Vite application.

2. They upload a CSV/Excel file containing technology information.
3. The application parses the file and dynamically renders a PDF preview in the browser.
4. The user can click on text fields within the preview to edit the content live.
5. Upon finalization, the system generates a high-quality PDF file with an embedded QR code linking to the technology's web page.

4.4 Database Schema

The platform utilizes a MongoDB database with two primary collections: 'technologies' and 'events'. The schemas are designed to be comprehensive and flexible.

4.4.1 The 'technologies' Collection

This collection stores all information related to a single technology or innovation.

Field	Type	Description & Constraints
id	String	A unique identifier for the technology. **Required, Unique**.
name	String	The official name of the technology. **Required**.
description	String	A brief, one-line summary.
overview	String	A paragraph-level overview.
detailedDescription	String	A full, detailed description.
genre	String	The category or domain of the technology (e.g., Software, Biotech).
docket	String	The unique docket number for internal tracking. **Required, Unique**.
createdBy	Object	Information about the user who submitted the technology.
createdBy.userId	String	The unique ID of the creating user. **Required, Indexed**.
createdBy.name	String	Name of the creating user.
createdBy.email	String	Email of the creating user.
innovators	Array of Objects	A list of the official innovators. Each object contains ‘name‘ (String) and ‘mail‘ (String).
advantages	Array of Strings	A list of key advantages.
applications	Array of Strings	A list of potential applications.
useCases	Array of Strings	A list of specific use cases.
relatedLinks	Array of Objects	Each object contains a ‘title‘ (String) and ‘url‘ (String).
technical Specifications	String	Detailed technical specs.
trl	Number	Technology Readiness Level. **Default: 1**.
spotlight	Boolean	Flag to feature the technology on the homepage. **Default: false**.
images	Array of Objects	Each object contains a ‘url‘ (String) and ‘caption‘ (String).
patent	String (Enum)	Patent status. Must be one of: ”Not Filed”, ”Application Filed”, ”Under Examination”, ”Granted”, ”Abandoned/Lapsed”. **Required**.
patentId, ...FilingDate, etc.	String, Date	Additional fields for detailed patent tracking.
brochures	Array of Objects	A list of associated brochure files. Each object contains ‘url‘ (String) and ‘originalName‘ (String).
timestamps	Timestamps	Automatically managed ‘createdAt‘ and ‘updatedAt‘ fields.

4.4.2 The 'events' Collection

This collection stores information about OTMT-hosted events.

Field	Type	Description & Constraints
`title`	String	The name of the event. **Required**.
`month`	String	Event month (e.g., "JAN"). **Required**.
`day`	String	Event day (e.g., "25"). **Required**.
`location`	String	The venue or location of the event.
`time`	String	The start time of the event.
`description`	String	A detailed description of the event.
`registration`	String	A URL for the registration page.
`isActive`	Boolean	A flag to control event visibility. **Default: false**.

4.5 Local Development and Deployment Guide

This guide provides the necessary steps to set up and run the entire platform in a local development environment. It assumes you have Node.js, npm, and Python/pip installed.

1. **Download Repositories and Install Dependencies:**

   - Clone all repositories listed in Section 6.2 to your local machine.
   - For all **Node.js-based repositories** (frontends, backends, brochure generator), navigate into each directory and run:

     ```
     npm install --force
     ```

     Note: The `--force` flag is used to bypass potential peer dependency conflicts. Use with caution and review any warnings.
   - For the **Python-based repository** (`Tech-Transfer-Pal`), navigate into its directory and run:

     ```
     pip install -r requirements.txt
     ```

2. **Configure Environment Variables (CRITICAL STEP):**

   - Each repository's source code contains hardcoded API endpoints and database connection strings pointing to the live production environment. For local development, these **must** be changed.
   - In each frontend and backend repository, locate the configuration files.
   - **Change all server URLs** from the production domain (e.g., 'https://api.otmt.iiitd.ac.in') to your local addresses (e.g., 'http://localhost:5000'). Ensure the ports match the ports your local backend services will run on.

- You will also need to provide your own local MongoDB connection string and API keys (e.g., for Firebase and Gemini) in the respective backend configuration files.

3. **Run the Development Servers:**

   - For each **Node.js repository**, run the following command to start its local development server:

     ```
     npm run start
     ```

   - For the **FastAPI repository** (`Tech-Transfer-Pal`), activate the python environment there and run the following command using Uvicorn:

     ```
     uvicorn main:app --reload
     ```

   - Once all services are running, you can access the frontends in your browser via their respective 'localhost' URLs, and they will communicate with your local backend services.

5 Limitations and Future Improvements

This platform has been engineered to be robust and scalable. However, like any system, it operates under certain constraints and offers opportunities for future enhancement. This section transparently outlines the current limitations and provides recommendations for future development.

5.1 API Rate and Cost Limitations

The most significant limitations are related to the use of third-party services operating on free or developer-tier plans.

- **Chatbot (Critical Limitation):**
  - **Constraint:** The AI chatbot currently utilizes the free tier of the **Google Gemini 1.5 Flash API**. This tier is subject to strict rate limits (requests per minute) and a total usage cap.
  - **Impact:** During periods of high user traffic, the chatbot may become unresponsive or return errors once these limits are exceeded. This makes it unsuitable for a full-scale public launch without changes.
  - **Recommendation:** For the chatbot to function reliably in a production environment, the OTMT office **must upgrade to a paid Google AI Platform plan**. This involves creating a billing account and replacing the current free-tier API key with a production key in the chatbot's backend environment configuration.
- **User Portal Authentication (Monitored Limitation):**
  - **Constraint:** The user authentication system relies on a free **Firebase** project. This plan includes generous limits (e.g., up to 50,000 monthly authentications for Google Sign-In), but they are not infinite.

- **Impact:** Given the current target user base of internal faculty and students (estimated at 100-200 active users), these limits are highly unlikely to be reached. The system is therefore stable for its intended purpose.
- **Recommendation:** No immediate action is required. However, if the platform's user base expands significantly in the future, usage should be monitored via the Firebase console. An upgrade to a paid plan would be a straightforward process if needed.

5.2 Areas for Future Enhancement

The following are suggestions for new features and improvements that could further enhance the platform's value.

- **Configurable Data Purging:** The "Recycle Bin" in the user portal currently has a fixed 30-day retention period for deleted technologies. A future improvement could be to make this duration a configurable setting in the Super Admin dashboard.
- **Persistent Chatbot History:** The chatbot is currently stateless, meaning conversations are lost upon page refresh. This was a deliberate design choice to position it as a quick-help tool. A potential feature for logged-in users could be an option to save and review their chat history.
- **Advanced Brochure Customization:** The brochure generation pipeline uses a single, fixed template. Future versions could allow users to choose from a library of different templates or even offer a simple drag-and-drop interface for layout customization.
- **Analytics and Reporting Dashboard:** The admin panel could be expanded with a dedicated analytics section, providing visualizations on user engagement metrics and popular technology categories.

6 Closing Note and Resources

The successful completion of the OTMT digital platform marks a significant step forward in providing a centralized, modern, and efficient ecosystem for managing the institution's intellectual assets. Through an agile and adaptive development process, we have delivered a suite of tools that not only meet the initial project requirements but also provide substantial added value through features like the custom brochure generator and the AI-powered chatbot.

This project was a journey of continuous learning and problem-solving, and we are confident that the resulting platform will serve the OTMT office, its faculty, and its students well into the future.

For any further development, maintenance, or technical queries, please refer to the resources below or contact us directly.

6.1 Project GitHub Repositories

The source code for each component of the project is hosted on GitHub in separate repositories for clear modularity and maintenance.

- **Public Website Frontend:** https://github.com/devan1shX/TMTO

- **Admin/User Portal Frontend:** https://github.com/devan1shX/Admin-Frontend
- **General Backend (Read-Only):** https://github.com/devan1shX/TMTO-Backend
- **Admin Backend (Write-Enabled):** https://github.com/devan1shX/Admin-Backend
- **Android Mobile Application:** https://github.com/devan1shX/OTMT-App
- **AI Chatbot Backend:** https://github.com/Beingstupid4me/Tech-Transfer-Pal
- **Brochure Generator:** https://github.com/devan1shX/Brochure-Automation

6.2 Contact Information

For follow-up questions regarding this report or the project, feel free to reach out to us.

Amartya Singh
Email: amartya22062@iiitd.ac.in

Anish
Email: anish22075@iiitd.ac.in

— End of Report —