A

**REPORT**

Of

**MINI PROJECT OR INTERNSHIP ASSESSMENT**

At

**“PRODESK IT SERVICES”**



On

**“Backend Development”**

Submitted

In the partial fulfillment of Bachelor of Technologies

**Department of Computer Science and Engineering**

****

**Shri Ram Murti Smarak College of Engineering and Technologies, Bareilly**

(2025-26)

Submitted to Submitted by

Ms. Juhi Chaudhary Ms. Anjali Gupta

Assistant Professor(CSE) (2200140100014)

# Certificate (by company)



# Certificate (College)

# Acknowledgement

The successful completion of this industrial training and the subsequent compilation of this report have been the result of the guidance, support, and encouragement of several individuals to whom a debt of gratitude is owed.

First and foremost, profound gratitude is extended to Prodesk IT & Engineering Services for providing the opportunity to undertake a 30-day summer internship in the field of Backend Development. The exposure to a professional and innovative work environment has been an invaluable learning experience.

Sincere appreciation is expressed to the assigned industry mentor, Mr. Rohan Verma, Senior Software Engineer, whose expert guidance, insightful feedback, and constant encouragement were instrumental in navigating the complexities of the project. His mentorship not only enhanced technical skills but also provided a deeper understanding of professional engineering practices.

Special thanks are due to Mr. N. Kumar, Senior HR Manager at Prodesk IT, for facilitating a smooth and enriching internship program. The support from the entire engineering team at Prodesk, who created a welcoming and collaborative atmosphere, is also gratefully acknowledged.

Gratitude is also extended to the faculty and administration of Shri Ram Murti Smarak College of Engineering & Technology, Bareilly, for their foundational academic support which prepared for this practical endeavor.

Finally, this acknowledgement would be incomplete without expressing thanks to family and friends for their unwavering support and encouragement throughout the duration of this training program.

**Table of Contents**

[Certificate (by company) ii](#_Toc207579018)

[Certificate (College) iii](#_Toc207579019)

[Acknowledgement iv](#_Toc207579020)

[List of Figures vii](#_Toc207579021)

[List of Tables viii](#_Toc207579022)

[Chapter 1: Company Profile 9](#_Toc207579023)

[1.1. Introduction to Prodesk IT & Engineering Services 9](#_Toc207579024)

[1.2. Corporate Vision, Mission, and Foundational Values 9](#_Toc207579025)

[1.3. Portfolio of Services and Technical Capabilities 11](#_Toc207579026)

[1.4. Organizational Strength and Global Footprint 12](#_Toc207579027)

[Chapter 2: Industry at a Glance 14](#_Toc207579028)

[2.1. The Indian Information Technology Sector: An Overview 14](#_Toc207579029)

[2.2. Prevailing Technological Paradigms and Future Trajectories 14](#_Toc207579030)

[2.3. Organizational Blueprint of Prodesk IT 15](#_Toc207579031)

[2.4. The Engineering Department: Structure and Workflow 16](#_Toc207579032)

[Chapter 3: Tools and Technology Used in Industry 18](#_Toc207579033)

[3.1. Core Backend Development Ecosystem 18](#_Toc207579034)

[3.2. Data Persistence and Management Systems 19](#_Toc207579035)

[3.3. DevOps, Containerization, and CI/CD Practices 20](#_Toc207579036)

[3.4. Cloud Computing Infrastructure 20](#_Toc207579037)

[3.5. Project Management and Collaboration Platforms 21](#_Toc207579038)

[Chapter 4: Modules Description/ Activities of Industry 22](#_Toc207579039)

[4.1. The Software Development Life Cycle (SDLC) at Prodesk 22](#_Toc207579040)

[4.2. Implementation of the Agile (Scrum) Framework 23](#_Toc207579041)

[4.3. Project Inception: The "CineReserve" Platform 25](#_Toc207579042)

[Chapter 5: Modules of Industry (on which I worked) 27](#_Toc207579043)

[5.1. Architectural Overview of the CineReserve Backend 27](#_Toc207579044)

[5.2. Module 1: Identity and Access Management (IAM) Service 27](#_Toc207579045)

[5.3. Module 2: Catalog Service 28](#_Toc207579046)

[5.4. Module 3: Transactional Service (Booking and Payments) 28](#_Toc207579047)

[5.5. Module 4: Notification Service (Expanded) 29](#_Toc207579048)

[Core Purpose 29](#_Toc207579049)

[Key Responsibilities 29](#_Toc207579050)

[Intern’s Contribution 30](#_Toc207579051)

[Benefits of the Notification Service 31](#_Toc207579052)

[Chapter 6: Details about the modules (on which I worked) with snapshots 32](#_Toc207579053)

[6.1. System Architecture: A Microservices-Based Approach 32](#_Toc207579054)

[6.2. Implementation of Secure Authentication with Spring Security and JWT 34](#_Toc207579055)

[6.3. Development of RESTful APIs for Core Functionalities 35](#_Toc207579056)

[6.4. Database Design, Schema, and ORM Integration 38](#_Toc207579057)

[6.4.1 Implementation with Spring Data JPA and Hibernate 39](#_Toc207579058)

[6.4.2 Data Access Layer 39](#_Toc207579059)

[6.4.3 Benefits of This Approach 40](#_Toc207579060)

[6.5. Real-time Communication with WebSockets 42](#_Toc207579061)

[6.7. Document Generation: PDF E-Tickets 44](#_Toc207579062)

[Chapter 7: Training Outcome 47](#_Toc207579063)

[7.1. Acquisition of Technical Competencies 48](#_Toc207579064)

[7.2. Assimilation of Professional Engineering Practices 50](#_Toc207579065)

[7.3. Development of Analytical and Problem-Solving Skills 52](#_Toc207579066)

[7.4. Enhancement of Collaborative and Communication Abilities 54](#_Toc207579067)

[Chapter 8: Conclusion 57](#_Toc207579068)

[8.1. Synthesis of the Internship Experience 57](#_Toc207579069)

[8.2. Summary of Contributions and Key Achievements 57](#_Toc207579070)

[8.3. Reflections and Future Outlook 59](#_Toc207579071)

[References 59](#_Toc207579072)

# List of Figures

[Figure 5. 1 Software Developent Life Cycle 31](#_Toc207575340)

[Figure 6. 1: CineReserve Microservices Architecture Diagram 33](#_Toc207538464)

[Figure 6. 2: Entity-Relationship Diagram (ERD) for Booking Service 41](#_Toc207538465)

[Figure 6. 3 Assessment: Movie ReservationSystem 45](#_Toc207538466)

[Figure 6. 4 Assessment: Inventory Management System 46](#_Toc207538467)

[Figure 6. 5 Assessment: PDF Merge & Split 46](#_Toc207538468)

[Figure 6. 6 Assessment: Task Management System 47](#_Toc207538469)

# List of Tables

[Table 4. 1 SDLC methodologies at Prodesk 23](#_Toc207536519)

[Table 6. 1: Summary of CineReserve RESTful API Endpoints 35](#_Toc207577449)

[Table 6. 2 Overview for the Module Details 43](#_Toc207577450)

# Chapter 1: Company Profile

## 1.1. Introduction to Prodesk IT & Engineering Services

Prodesk IT & Engineering Services, hereafter referred to as Prodesk IT, is a distinguished information technology and software services company headquartered in Noida, Uttar Pradesh, India. Established in 2012, the organization was founded with a forward-thinking vision to redefine the role of technology in business transformation. Under the leadership of Dr. Amit Maheshwari, whose professional background includes significant experience as an HR Manager at Tech Mahindra, Prodesk IT was conceptualized to deliver tangible, results-driven solutions that empower organizations to excel in the increasingly competitive digital landscape.

The company's origins are rooted in a passion for high-quality software development, a principle that has guided its evolution from a nascent startup to a formidable mid-sized enterprise. This foundation in enterprise-level experience has profoundly shaped Prodesk's internal culture and technical strategy. Rather than operating with the ad-hoc methodologies often seen in smaller firms, Prodesk has institutionalized structured, process-driven development practices. This professional environment ensures that all projects, including those involving interns, are executed with a high degree of engineering rigor, mirroring the standards of much larger corporations.

Over the years, Prodesk IT has achieved significant growth, marked by milestones of innovation and customer success. This trajectory has solidified its reputation as a trusted technology partner for businesses seeking a competitive advantage through bespoke software solutions. The company's commitment to excellence is reflected in its current operational scale, maintaining a dedicated team of 156 full-time employees distributed across five different global locations.

## 1.2. Corporate Vision, Mission, and Foundational Values

The strategic direction and operational ethos of Prodesk IT are anchored in a clearly defined corporate philosophy, encompassing its vision, mission, and a set of core values that guide every aspect of its business.

**Corporate Vision:** Prodesk IT envisions "a world where technology empowers individuals and organizations to achieve their full potential, driving growth, efficiency, and sustainability". This vision extends beyond mere software delivery; it represents a commitment to leveraging technology as a catalyst for positive transformation, enabling clients to overcome complex challenges and unlock new opportunities.

**Corporate Mission:** The company's mission is "to empower businesses to thrive in the digital age by providing cutting-edge technology solutions that drive efficiency, productivity, and growth". Prodesk IT aims to be a trusted partner, translating the needs and goals of its clients into reliable, user-friendly, and efficient software solutions. This mission is realized through the application of state-of-the-art technical approaches designed to minimize future operational costs and mitigate risks for their clientele.

**Foundational Values:** A set of nine foundational values underpins the company's vision and mission, forming the cornerstone of its corporate culture :

* **Innovation:** Positioned as a primary driving force, innovation is viewed not just as the creation of new technologies but as the application of creative solutions to complex problems. The company is committed to exploring emerging technologies such as Artificial Intelligence (AI), Machine Learning (ML), and blockchain.
* **Quality:** A commitment to delivering high-quality products and services that meet or exceed customer expectations. This value encompasses not only bug-free code but also user-friendliness, reliability, and fitness for purpose, supported by robust investments in quality assurance and testing.
* **Customer Focus:** Placing the customer at the heart of all operations. Prodesk IT emphasizes listening to client needs, understanding their challenges, and delivering solutions that provide tangible business value, backed by exceptional customer service.
* **Integrity:** Operating with honesty, transparency, and ethical conduct in all business dealings. Integrity is considered essential for building and maintaining trust with customers, employees, and partners.
* **Collaboration:** A belief that great ideas emerge from collective effort. The company fosters close collaboration with clients, partners, and internal teams to share ideas, solve problems, and achieve common objectives.
* **Continuous Improvement:** A commitment to constantly reviewing and refining processes, technologies, and strategies to enhance efficiency and effectiveness in a rapidly evolving technological landscape.
* **Accountability:** Taking ownership of actions, decisions, and outcomes. This core value is crucial for delivering results that matter and maintaining a reputation as a reliable and trustworthy partner.
* **Diversity and Inclusion:** Embracing diversity as a strength that leads to better ideas and outcomes. Prodesk IT is dedicated to creating an inclusive environment where all individuals feel respected and valued.
* **Sustainability:** Acknowledging the responsibility to minimize environmental impact and support sustainable practices for the benefit of future generations.

## 1.3. Portfolio of Services and Technical Capabilities

Prodesk IT offers a comprehensive suite of services designed to address the diverse technological needs of modern businesses. The company's service portfolio is a testament to its deep expertise across various domains of software engineering and IT management. The primary services offered include :

* **Custom Software Development:** This is the core offering of Prodesk IT. The company specializes in creating tailored software solutions designed to meet the unique requirements of each client. This includes the development of sophisticated web applications, robust mobile apps (for both startups and established enterprises), and other bespoke software products.
* **IT Consulting:** Prodesk IT provides expert guidance and strategic advice to help businesses navigate the complex and ever-evolving technology landscape. This service ensures that a client's IT infrastructure and strategy are fully aligned with their overarching business goals, maximizing return on investment.
* **Cybersecurity:** Recognizing the critical importance of data protection, the company offers comprehensive security measures. These services are designed to protect valuable digital assets and sensitive data against a wide range of evolving cyber threats, thereby safeguarding business continuity and corporate reputation.
* **Cloud Services:** The company delivers scalable and secure cloud solutions that enable businesses to streamline operations, enhance collaboration, and facilitate seamless access to data from any location. This expertise spans major cloud platforms and focuses on creating resilient and efficient cloud architectures.
* **Data Analytics:** Prodesk IT helps businesses harness the power of their data by providing services that derive actionable insights from complex datasets. This enables clients to make informed, data-driven decisions and gain a significant competitive edge in the marketplace.
* **Web and Logo Design:** Complementing its core development services, Prodesk IT also offers professional web development and logo design. The web development services encompass the entire lifecycle, from domain registration and hosting to the final deployment of a fully functional website. Logo design services focus on establishing a strong and memorable brand identity for clients.
* **Project Management:** The company provides dedicated project management services to ensure that engineering projects are delivered on time and within budget. This involves meticulous planning, resource management, and comprehensive oversight of all project phases from inception to completion.

## 1.4. Organizational Strength and Global Footprint

Since its inception in 2012, Prodesk IT has demonstrated a consistent pattern of growth and expansion, evolving into a highly successful IT-Software company with a significant global presence. The organization's strength is rooted in its team of seasoned professionals and its strategic distribution of operations.

Currently, Prodesk IT maintains a total employee strength of 156 individuals working full-time. This team is distributed across five different operational locations around the globe, allowing the company to serve a diverse international clientele and tap into a global talent pool. This distributed model supports a flexible and resilient operational framework, enabling round-the-clock development and support cycles where necessary.

The company's primary office in the National Capital Region (NCR) of India is located at D-107, 91springboard, Vyapar Marg, Sector-2, Noida, Uttar Pradesh. This facility serves as a central hub for many of the company's development and management activities and was the designated administrative location for the duration of this industrial training.

# Chapter 2: Industry at a Glance

## 2.1. The Indian Information Technology Sector: An Overview

The industrial training was conducted within the context of India's dynamic and globally significant Information Technology (IT) sector. The Indian IT industry stands as one of the country's most vital economic pillars, characterized by robust growth, extensive global integration, and a vibrant innovation ecosystem. According to the National Association of Software and Service Companies (Nasscom), the industry's revenue reached an impressive US$ 227 billion in Fiscal Year 2022, marking a 15.5% year-over-year growth. This trajectory is projected to continue, with IT spending in India expected to rise significantly in the coming years.

India has firmly established itself as a global leader in IT outsourcing, a position built on a foundation of cost-effective service delivery, a large and skilled talent pool, and a commitment to high-quality outcomes. This has made the country a preferred destination for multinational corporations seeking to optimize their technology operations.

Beyond its dominance in outsourcing, India is also home to a thriving startup ecosystem, which has become a hotbed of innovation, particularly in high-growth areas such as Artificial Intelligence (AI), financial technology (fintech), and Software as a Service (SaaS). This ecosystem is fueled by substantial private equity and venture capital investments, which reached a record US$ 36 billion in 2021. Government initiatives like "Digital India" and "Make in India" have further catalyzed this growth by improving digital infrastructure and encouraging domestic and foreign investment in the technology sector.

## 2.2. Prevailing Technological Paradigms and Future Trajectories

The Indian IT industry is currently navigating a period of profound technological transformation. The landscape in 2025 is being shaped by several key trends that are redefining how businesses operate and how software is developed. A forward-looking analysis reveals the following dominant paradigms :

* **Artificial Intelligence (AI) and Automation:** AI adoption is a pervasive trend, with an estimated 75% of Indian enterprises expected to integrate AI into their core processes by 2025. Generative AI, in particular, is being applied to enhance cloud services, improve cybersecurity defenses, and automate repetitive development tasks.
* **Cloud Computing:** Hybrid and multi-cloud environments are becoming the standard. The global expansion of hyperscalers like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud is enabling Indian IT companies of all sizes to migrate their operations to the cloud, optimizing for performance, cost, and compliance.
* **Cybersecurity:** As digital transformation accelerates, so do the associated cyber threats. Consequently, spending on cybersecurity in India is projected to exceed US$ 3.5 billion by 2025. There is a growing emphasis on AI-driven security solutions, regular audits, and comprehensive threat detection strategies.
* **Software as a Service (SaaS):** SaaS ventures are emerging as a high-margin, high-growth segment within the Indian IT landscape. The ability to develop and scale cloud-native products for a global market is a significant opportunity for companies like Prodesk IT.

Despite these opportunities, the industry faces persistent challenges. A significant **skill gap** exists, with a shortage of expertise in emerging technologies like AI and blockchain. Furthermore, companies are experiencing a

**margin squeeze** due to rising employee expenses and high attrition rates, which increases the cost of hiring experienced professionals. These industry-wide pressures provide a compelling rationale for the strategic importance of internship programs. Companies like Prodesk IT utilize internships as a vital mechanism to cultivate a talent pipeline. By training students on their specific technology stacks and corporate culture, they can onboard skilled junior developers in a more cost-effective manner than competing for experienced talent in the open market. This approach directly addresses both the skill gap and cost pressures, making the internship program a strategic business initiative rather than a purely academic exercise.

## 2.3. Organizational Blueprint of Prodesk IT

For a mid-sized software company like Prodesk IT, with a workforce of approximately 156 employees, a well-defined organizational structure is essential for maintaining efficiency, fostering collaboration, and ensuring alignment with strategic goals. Prodesk IT employs a **Functional Organizational Structure**, where the company is organized into departments based on specialized functions. This model promotes deep expertise within each department and provides clear lines of authority and communication.

The primary departments within Prodesk IT are depicted in the organizational chart below (Figure 2.1) and include:

* **Executive Leadership (CEO/Founder):** At the apex of the organization is the Chief Executive Officer, responsible for setting the company's overall vision, strategy, and corporate culture.
* **Engineering Department:** This is the core product and service delivery engine of the company. It is responsible for all aspects of software design, development, and implementation.
* **Quality Assurance (QA) Department:** A dedicated department responsible for all testing activities, ensuring that software products meet the highest standards of quality, reliability, and performance before release.
* **DevOps Department:** This team manages the infrastructure, continuous integration and continuous deployment (CI/CD) pipelines, and overall operational stability of the software systems.
* **Sales & Marketing Department:** Responsible for business development, client acquisition, brand promotion, and market research.
* **Human Resources (HR) & Administration:** This department manages talent acquisition, employee relations, training and development (including the internship program), and general administrative functions.

## 2.4. The Engineering Department: Structure and Workflow

The Engineering Department is the largest and most central department at Prodesk IT, serving as the hub for all software development activities. This internship was situated within the Backend Team of this department. The department operates on a **Matrix Structure** for project execution, which combines the functional hierarchy with project-based teams. While developers report to their functional managers (e.g., Head of Backend Development), they are assigned to cross-functional project teams for the duration of a project.

A typical project team at Prodesk IT is composed of the following roles :

* **Project Manager (PM):** Responsible for the overall project timeline, resource allocation, risk management, and communication with stakeholders.
* **Software Architect:** Designs the high-level technical architecture of the system, selects appropriate technologies, and establishes coding and design standards.
* **Backend Developers:** Responsible for server-side logic, database design and management, and the development of RESTful APIs.
* **Frontend Developers:** Responsible for the client-side of the application, building the user interface and user experience.
* **Quality Assurance (QA) Engineers:** Responsible for designing and executing test plans, identifying defects, and verifying that the software meets all specified requirements.
* **DevOps Engineer:** Works with the project team to automate the build, testing, and deployment processes.

This cross-functional team structure facilitates close collaboration and rapid iteration, which is essential for the Agile development methodology practiced at the company. For the "CineReserve" project, a dedicated team comprising individuals from each of these roles was assembled, creating an integrated and efficient development unit.

# Chapter 3: Tools and Technology Used in Industry

A modern software development organization like Prodesk IT leverages a sophisticated and diverse ecosystem of tools and technologies to build, deploy, and maintain high-quality software solutions. The technology stack is carefully selected to ensure scalability, reliability, and developer productivity. The following sections detail the primary tools and technologies that form the backbone of the development process at the company.

## 3.1. Core Backend Development Ecosystem

The backend development at Prodesk IT is predominantly centered around the Java ecosystem, which is renowned for its robustness, platform independence, and extensive community support.

* **Java:** As the primary programming language, Java provides a stable and mature platform for building enterprise-grade applications. Its object-oriented nature, strong memory management, and rich set of APIs make it an ideal choice for complex, large-scale systems.
* **Spring Boot:** Spring Boot is the cornerstone of backend development at the company. It is a framework built on top of the core Spring Framework that radically simplifies the development of stand-alone, production-grade Spring-based applications. Its key features, which were extensively utilized during the internship project, include:
  + **Auto-Configuration:** Spring Boot automatically configures the application based on the JAR dependencies present on the classpath, drastically reducing the need for boilerplate configuration.
  + **Embedded Servers:** It comes with embedded Tomcat, Jetty, or Undertow servers, allowing applications to be run as standalone JAR files without the need to deploy WAR files to an external web server.
  + **Opinionated Starter Dependencies:** It provides a set of "starter" dependencies that simplify build configuration and get projects up and running quickly.
  + **Microservices Support:** Its lightweight nature and ease of deployment make it exceptionally well-suited for building microservices-based architectures.
* **RESTful APIs:** The standard for communication between services and between the backend and frontend is the REST (Representational State Transfer) architectural style. RESTful APIs, which use standard HTTP methods (GET, POST, PUT, DELETE), provide a stateless, scalable, and flexible way to expose backend functionality.

## 3.2. Data Persistence and Management Systems

The choice of a database is a critical architectural decision that depends on the specific requirements of the application, particularly concerning data structure, scalability, and transactional integrity. Prodesk IT employs a polyglot persistence strategy, using different types of databases for different use cases.

* **PostgreSQL (Primary Relational Database):** For applications requiring strong transactional consistency and data integrity, such as the booking and payment modules of the CineReserve project, PostgreSQL is the preferred relational database management system (RDBMS). It is fully ACID-compliant (Atomicity, Consistency, Isolation, Durability) and offers a rich set of advanced features, including support for complex queries, JSON data types, and robust indexing mechanisms. Its reliability and extensibility make it an excellent choice for mission-critical systems.
* **MongoDB (Primary NoSQL Database):** For use cases that involve unstructured or semi-structured data and require high scalability and schema flexibility, MongoDB is the go-to NoSQL database. As a document-oriented database, it stores data in flexible, JSON-like documents (BSON). This model is highly beneficial for applications with evolving data requirements or those that need to handle large volumes of heterogeneous data. While it offers atomic operations at the document level, it traditionally prioritizes scalability and performance over the multi-record ACID guarantees of relational databases.
* **Spring Data JPA & Hibernate:** To interact with relational databases like PostgreSQL, Prodesk IT utilizes Spring Data JPA. It is a part of the Spring Data family that simplifies the implementation of JPA-based data access layers. Under the hood, it uses Hibernate as the default JPA provider. This combination of Spring Data JPA and Hibernate provides a powerful Object-Relational Mapping (ORM) framework, allowing developers to work with database tables using familiar Java objects and abstracting away much of the boilerplate JDBC code.

## 3.3. DevOps, Containerization, and CI/CD Practices

To support an agile and efficient development workflow, Prodesk IT has adopted a modern DevOps culture, supported by a robust toolchain for automation and collaboration.

* **Git:** Git is the distributed version control system used for managing all source code. It enables multiple developers to work on a project simultaneously without interfering with each other's work, providing powerful features for branching, merging, and tracking the history of every change.
* **Maven:** Maven is the primary build automation and dependency management tool for all Java-based projects. It simplifies the build process by standardizing how projects are built, and it automatically downloads and manages all project dependencies from central repositories.
* **Docker:** Docker is the containerization platform used to package applications and their dependencies into lightweight, portable containers. This ensures that an application runs the same way regardless of the environment it is deployed in, eliminating the "it works on my machine" problem. Containerization is a foundational technology for microservices architectures.
* **Jenkins:** Jenkins is the open-source automation server used to implement Continuous Integration and Continuous Deployment (CI/CD) pipelines. It automates the process of building, testing, and deploying applications whenever new code is committed to the Git repository, enabling rapid and reliable software delivery.

## 3.4. Cloud Computing Infrastructure

Prodesk IT leverages the power and scalability of cloud computing to host its applications and services. While maintaining a cloud-agnostic approach to be flexible for client needs, the company has deep expertise and a preference for Amazon Web Services (AWS) for its internal and many client projects.

Key AWS services utilized include:

* **Amazon EC2 (Elastic Compute Cloud):** Provides scalable virtual servers for running applications.
* **Amazon S3 (Simple Storage Service):** Used for object storage, such as storing static assets, logs, and backups.
* **Amazon RDS (Relational Database Service):** A managed database service that simplifies the setup, operation, and scaling of relational databases like PostgreSQL.
* **Amazon EKS (Elastic Kubernetes Service):** A managed Kubernetes service used for orchestrating and scaling containerized applications built with Docker.

## 3.5. Project Management and Collaboration Platforms

Effective communication and project tracking are critical to the success of any software project, especially within an Agile framework. Prodesk IT uses a suite of industry-standard tools to facilitate this:

* **Jira:** A powerful project management tool used for issue tracking, bug tracking, and agile project management. All project work, from high-level epics to detailed user stories and tasks, is managed within Jira, providing full visibility into the progress of each sprint.
* **Confluence:** A team collaboration and knowledge management tool. It is used to create, share, and collaborate on project documentation, including requirements documents, design specifications, meeting notes, and technical guides.
* **Slack:** The primary platform for real-time team communication. It provides channels for project-specific discussions, team-wide announcements, and direct messaging, fostering a collaborative and responsive work environment.

# Chapter 4: Modules Description/ Activities of Industry

## 4.1. The Software Development Life Cycle (SDLC) at Prodesk

The development of high-quality software at Prodesk IT is not an ad-hoc process but is governed by a structured and systematic framework known as the Software Development Life Cycle (SDLC). The SDLC provides a comprehensive roadmap that defines the distinct phases involved in the conception, creation, and maintenance of a software system. Adherence to an SDLC ensures that projects are delivered in a cost-effective, time-efficient, and predictable manner, meeting all stakeholder requirements and quality benchmarks.

The SDLC methodology employed at Prodesk IT encompasses seven essential phases, each with specific objectives and deliverables :

1. **Phase 1: Planning and Requirements Analysis:** This is the foundational phase where the project's purpose, scope, and feasibility are determined. It involves extensive collaboration with stakeholders—including clients, business analysts, and end-users—to gather and analyze requirements. The primary output of this phase is a Software Requirement Specification (SRS) document, which details the software's intended functions, necessary resources, potential risks, and a preliminary project timeline.
2. **Phase 2: Defining Requirements:** Building upon the initial analysis, this phase involves formalizing and documenting the requirements in a clear, detailed, and unambiguous manner. The SRS is refined to serve as a definitive guide for the design and development teams.
3. **Phase 3: Design:** In this phase, the system architecture and software design are created. The development team translates the requirements from the SRS into a technical blueprint. This includes defining the high-level architecture (e.g., monolithic vs. microservices), designing the database schema, specifying API contracts, and creating detailed designs for individual components. The output is a Software Design Document (SDD).
4. **Phase 4: Development (Coding):** This is the implementation phase where the actual coding takes place. Developers write the source code based on the specifications laid out in the SDD. The project is often broken down into smaller modules, which are developed and unit-tested individually.
5. **Phase 5: Testing:** The testing phase is crucial for ensuring the quality of the software. It involves a variety of testing activities, including unit testing, integration testing, system testing, and user acceptance testing (UAT). The Quality Assurance (QA) team rigorously checks the software for bugs, defects, and deviations from the requirements.
6. **Phase 6: Deployment:** Once the software has passed the testing phase, it is deployed to a production environment where it becomes accessible to end-users. This phase includes packaging the application, configuring the production environment, and executing the release plan.
7. **Phase 7: Maintenance:** After deployment, the software enters the maintenance phase. This involves ongoing support, monitoring, bug fixing, and the implementation of enhancements or new features in subsequent release cycles.

Table 4. 1 SDLC methodologies at Prodesk

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Name** | **Key Focus** | **Main Deliverables** |
| **1** | Planning & Requirements Analysis | Define purpose, scope, feasibility; gather requirements with stakeholders | Software Requirement Specification (SRS) draft |
| **2** | Defining Requirements | Refine & document clear, detailed requirements | Finalized SRS (definitive guide) |
| **3** | Design | Create system architecture & detailed design (database, APIs, components) | Software Design Document (SDD) |
| **4** | Development (Coding) | Implement features; write & unit-test code modules | Source code, tested modules |
| **5** | Testing | Ensure quality via unit, integration, system, and UAT testing | QA test reports, validated software |
| **6** | Deployment | Release to production; configure environment & execute release plan | Deployed software in production |
| **7** | Maintenance | Provide support, bug fixes, monitoring, and enhancements | Updates, patches, new features |

## 4.2. Implementation of the Agile (Scrum) Framework

While the SDLC provides the overall structure, Prodesk IT adopts the **Agile methodology** for the practical execution of its software development projects. Specifically, the company implements the **Scrum framework**, which is an iterative and incremental approach designed to deliver value to customers quickly and efficiently. The choice of Agile is a strategic one, as it provides the flexibility and responsiveness needed to thrive in the fast-paced technology industry.

The Agile philosophy at Prodesk IT is guided by the four core values of the Agile Manifesto :

* **Individuals and interactions** over processes and tools.
* **Working software** over comprehensive documentation.
* **Customer collaboration** over contract negotiation.
* **Responding to change** over following a plan.

These values are put into practice through the Scrum framework, which organizes work into time-boxed iterations called **sprints**, typically lasting two weeks. The key ceremonies and artifacts of Scrum as implemented at Prodesk are:

* **Scrum Roles:**
  + **Product Owner:** Represents the stakeholders and is responsible for defining features and managing the product backlog.
  + **Scrum Master:** A servant-leader who facilitates the Scrum process, removes impediments, and ensures the team adheres to Agile principles.
  + **Development Team:** A cross-functional, self-organizing team of developers, QA engineers, and other specialists who are responsible for delivering a potentially shippable increment of the product at the end of each sprint.
* **Scrum Events (Ceremonies):**
  + **Sprint Planning:** Held at the beginning of each sprint, where the team selects a set of high-priority items from the product backlog to work on during the sprint.
  + **Daily Stand-up:** A short, daily meeting where team members synchronize their activities and plan for the next 24 hours.
  + **Sprint Review:** Held at the end of the sprint, where the team demonstrates the completed work to stakeholders and gathers feedback.
  + **Sprint Retrospective:** A meeting where the team reflects on the past sprint to identify what went well and what could be improved, fostering a culture of continuous improvement.
* **Scrum Artifacts:**
  + **Product Backlog:** A prioritized list of all desired features and functionalities for the product.
  + **Sprint Backlog:** A subset of the product backlog that the team commits to completing during a sprint.
  + **Increment:** The sum of all the product backlog items completed during a sprint, which should be in a usable condition.

The adoption of the Agile framework is deeply interconnected with the company's choice of a microservices architecture for new projects. Agile's emphasis on breaking down large projects into small, manageable increments aligns perfectly with the microservices principle of decomposing a large application into small, independently deployable services. This symbiotic relationship allows a development team to focus on building, testing, and deploying a single microservice within a single sprint, enabling rapid, independent releases and fostering a highly efficient and parallel development workflow.

## 4.3. Project Inception: The "CineReserve" Platform

During the internship period, the primary project of focus was the development of a new, state-of-the-art movie ticket booking platform, codenamed **"CineReserve"**. This project was initiated by Prodesk IT to enter the lucrative online entertainment market with a modern, scalable, and feature-rich solution.

The project began with the **Planning and Requirements Analysis** phase of the SDLC. The business goals for CineReserve were defined as follows:

* To create a highly available and scalable platform capable of handling high traffic volumes, especially during peak booking times for popular movies.
* To provide a seamless and intuitive user experience for browsing movies, selecting seats, and making payments.
* To build a flexible architecture that would allow for the easy addition of new features in the future, such as food and beverage ordering, loyalty programs, and personalized recommendations.

Following the initial planning, the project team, including the intern, participated in several requirements gathering sessions. These sessions led to the creation of a high-level **Product Backlog** in Jira. The backlog was structured into epics, which represented major features of the platform. The initial epics identified for the Minimum Viable Product (MVP) included:

* User Authentication and Profile Management
* Movie and Cinema Catalog Management
* Showtime and Seating Plan Display
* Ticket Booking and Reservation Flow
* Payment Gateway Integration
* Real-time Booking Confirmation and Notifications

This product backlog served as the foundation for the subsequent design and development sprints, with the intern being assigned tasks related to several of these core functional areas.

# Chapter 5: Modules of Industry (on which I worked)

The CineReserve platform was designed using a modern architectural style to meet its goals of scalability, maintainability, and rapid evolution. The intern was assigned to the backend development team and contributed to several key modules, or microservices, that form the core of the system.

## 5.1. Architectural Overview of the CineReserve Backend

To achieve the desired levels of scalability and team autonomy, a **Microservices Architecture** was chosen for the CineReserve backend. In this architectural style, the application is decomposed into a collection of small, loosely coupled, and independently deployable services. Each service is responsible for a specific business capability and communicates with other services over a network, typically via lightweight protocols like REST APIs.

This approach offers several key advantages over a traditional monolithic architecture:

* **Scalability:** Individual services can be scaled independently based on their specific load, optimizing resource utilization.
* **Team Autonomy:** Small, focused teams can develop, deploy, and maintain their respective services independently, leading to faster development cycles.
* **Technology Flexibility:** Each service can be built using the technology stack best suited for its specific task.
* **Fault Isolation:** A failure in one service is less likely to bring down the entire application, improving overall system resilience.

The CineReserve backend was structured around a set of core microservices, each with a well-defined responsibility. The intern's work was distributed across four of these fundamental services.

## 5.2. Module 1: Identity and Access Management (IAM) Service

This service is the gatekeeper of the entire platform, responsible for all aspects of user identity and security. Its primary functions include:

* **User Registration:** Providing endpoints for new users to create an account.
* **User Authentication:** Verifying user credentials (e.g., username and password) during login.
* **Authorization:** Managing user roles and permissions to control access to different parts of the application.
* **Token Management:** Generating and validating JSON Web Tokens (JWTs) for secure, stateless authentication of API requests.

The intern's contribution to this module involved assisting in the development of the REST endpoints for user registration (/auth/register) and login (/auth/login). A key task was to implement the logic for generating a JWT upon successful authentication, which would then be used by the client to access protected resources.

## 5.3. Module 2: Catalog Service

The Catalog Service acts as the central repository for all information related to movies, cinemas, and their schedules. It is primarily a read-heavy service that provides data to the frontend for display to users. Its responsibilities include:

* Managing movie details (title, genre, director, cast, etc.).
* Managing cinema details (name, location, screens).
* Managing showtimes for each movie at each cinema.
* Providing APIs to search and filter movies and cinemas.

The intern was tasked with assisting in the design of the database schema for the movies, theaters, and shows tables within this service's dedicated database. Additionally, the intern helped create several REST API endpoints for fetching this catalog data, such as GET /api/movies and GET /api/theaters/{theaterId}/shows.

## 5.4. Module 3: Transactional Service (Booking and Payments)

This is the most critical service in the CineReserve platform, as it handles the core business logic of booking tickets and processing payments. This service must be highly reliable and ensure transactional integrity. Its key functions are:

* Managing seat layouts for each cinema screen.
* Handling real-time seat selection and locking to prevent double-booking.
* Creating and managing booking records.
* Integrating with a third-party payment gateway to process financial transactions.

The intern's primary focus during the training period was on this module. The assigned tasks included implementing the core booking logic, which involved checking seat availability, creating a booking record with a PENDING status, and then confirming the booking upon successful payment. A significant part of this work was the integration of the Razorpay payment gateway to handle online payments.

## ****5.5. Module 4: Notification Service (Expanded)****

The **Notification Service** is a critical supporting module within the CineReserve platform, designed to provide **real-time communication** and ensure users are consistently updated about the status of their interactions with the system. Unlike core transactional services, which directly handle bookings and payments, this service focuses on **asynchronous event-driven communication**, thereby **decoupling notifications from business logic**. This separation improves system performance, scalability, and maintainability.

#### ****Core Purpose****

* To deliver timely, reliable, and relevant updates to users.
* To ensure smooth user experiences without manual intervention (e.g., refreshing pages or checking emails repeatedly).
* To offload non-core responsibilities (like messaging) from core business services, thus reducing latency and failure risks.

#### ****Key Responsibilities****

1. **Booking Confirmation Messages**
   * Automatically sends **emails** and **SMS notifications** to users immediately after a booking is confirmed.
   * These notifications include booking details such as seat number, showtime, theater name, and payment status.
   * Provides a digital acknowledgment to users, serving as proof of transaction.
2. **Real-Time Status Updates**
   * Implements **push notifications** to instantly inform users about critical updates (e.g., booking confirmation, payment status, or system alerts).
   * Utilizes **WebSockets** for continuous, two-way communication between the server and the client’s browser or mobile app.
   * Ensures that users do not need to refresh the application manually to check booking status.
3. **User Notification Preferences**
   * Manages user-specific preferences for receiving notifications (e.g., opting in/out of email, SMS, or push messages).
   * Allows flexible configuration so users can choose their preferred communication channel.

#### ****Intern’s Contribution****

During the internship, a major contribution was made in the area of **real-time notifications** through:

* **WebSocket Endpoint Development**
  + A dedicated WebSocket endpoint was implemented to handle instant communication between the Notification Service and client applications.
  + This feature ensured that as soon as the Transactional Service confirmed a booking, the Notification Service would immediately trigger and push the update to the user’s interface.
  + For example, once a payment is validated, the user instantly sees a “Booking Confirmed” message in their browser, enhancing the **user experience with zero delay**.
* **Integration with Transactional Service**
  + Worked on integrating notification triggers with the booking workflow, ensuring that only successful bookings generated confirmations.
  + Helped maintain consistency and avoid duplicate or incorrect notifications.

#### ****Benefits of the Notification Service****

* **Improved User Experience**: Real-time updates keep users engaged and informed.
* **System Decoupling**: By separating notifications from booking logic, failures in one service do not affect the other.
* **Scalability**: The notification module can be scaled independently to handle large volumes of messages during peak usage (e.g., blockbuster movie releases).
* **Flexibility**: Supports multiple channels (email, SMS, push) and user preferences.

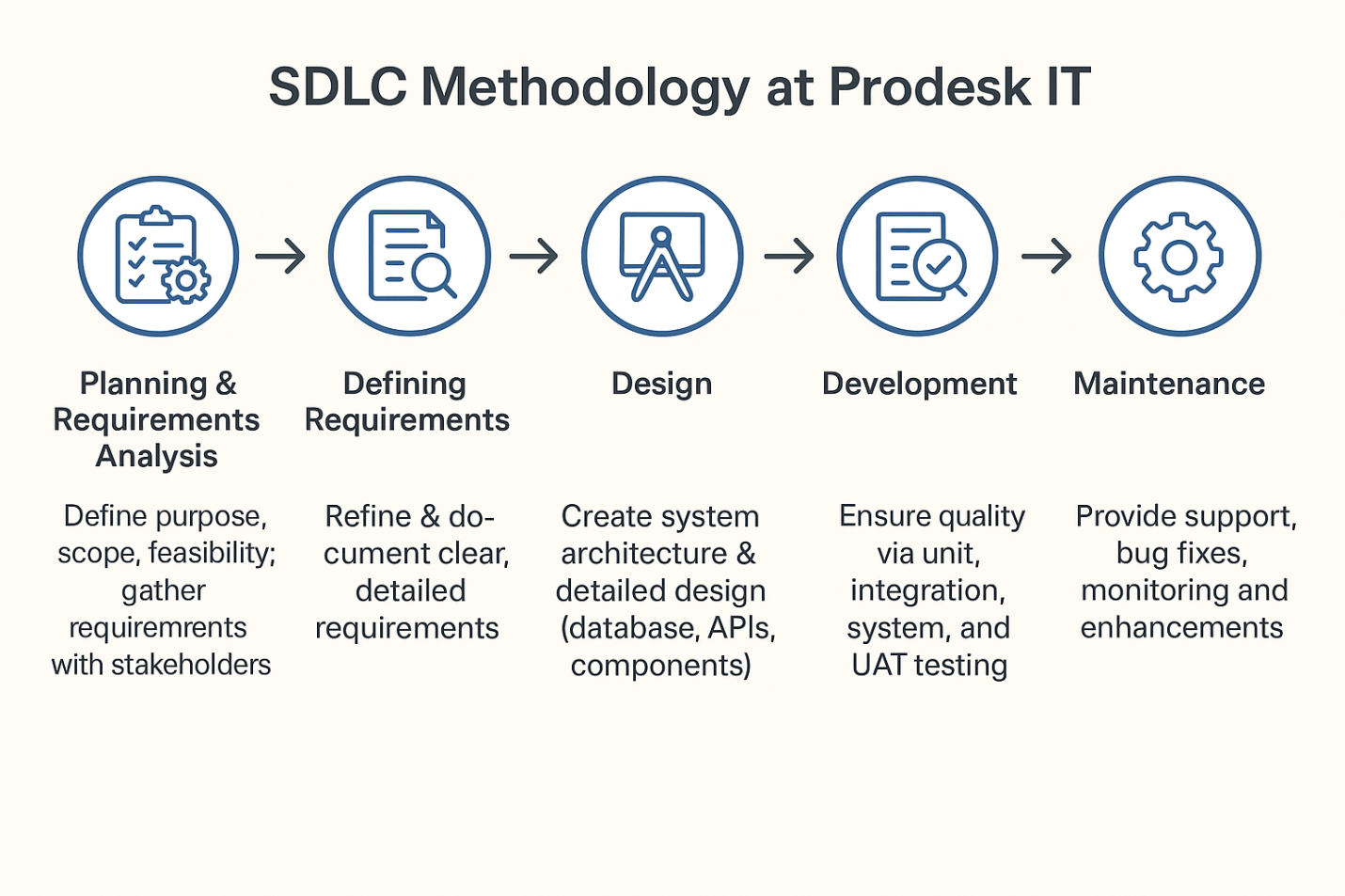


Figure 5. 1 Software Developent Life Cycle

# Chapter 6: Details about the modules (on which I worked) with snapshots

This chapter provides a detailed technical exposition of the work undertaken on the CineReserve project. It delves into the architectural design, implementation details of key features, database schema, and integration with third-party services, accompanied by illustrative diagrams, tables, and code snippets.

## 6.1. System Architecture: A Microservices-Based Approach

The CineReserve platform is architected as a distributed system composed of several independent microservices. This design choice was made to ensure high scalability, fault tolerance, and ease of maintenance. The overall architecture, as depicted in Figure 6.1, consists of the following key components :

* **Client Applications:** These are the user-facing applications (e.g., a web browser or mobile app) that interact with the backend system.
* **API Gateway:** This component acts as a single entry point for all client requests. It is responsible for request routing, authentication, rate limiting, and other cross-cutting concerns. By abstracting the underlying microservices, it simplifies the client-side implementation.
* **Service Registry (Eureka):** In a dynamic microservices environment, services need a way to find each other. The Service Registry (using Netflix Eureka in this implementation) serves this purpose. Each microservice registers itself with the registry upon startup, allowing other services and the API Gateway to discover its location.
* **Microservices:** These are the individual services that implement the core business logic, as described in Chapter 5. Each service has its own dedicated database, following the "database per service" pattern to ensure loose coupling.
* **Message Broker:** Used for asynchronous communication between services. For example, the Booking Service can publish a "Booking Confirmed" event to a message queue, which the Notification Service then consumes to send an email.

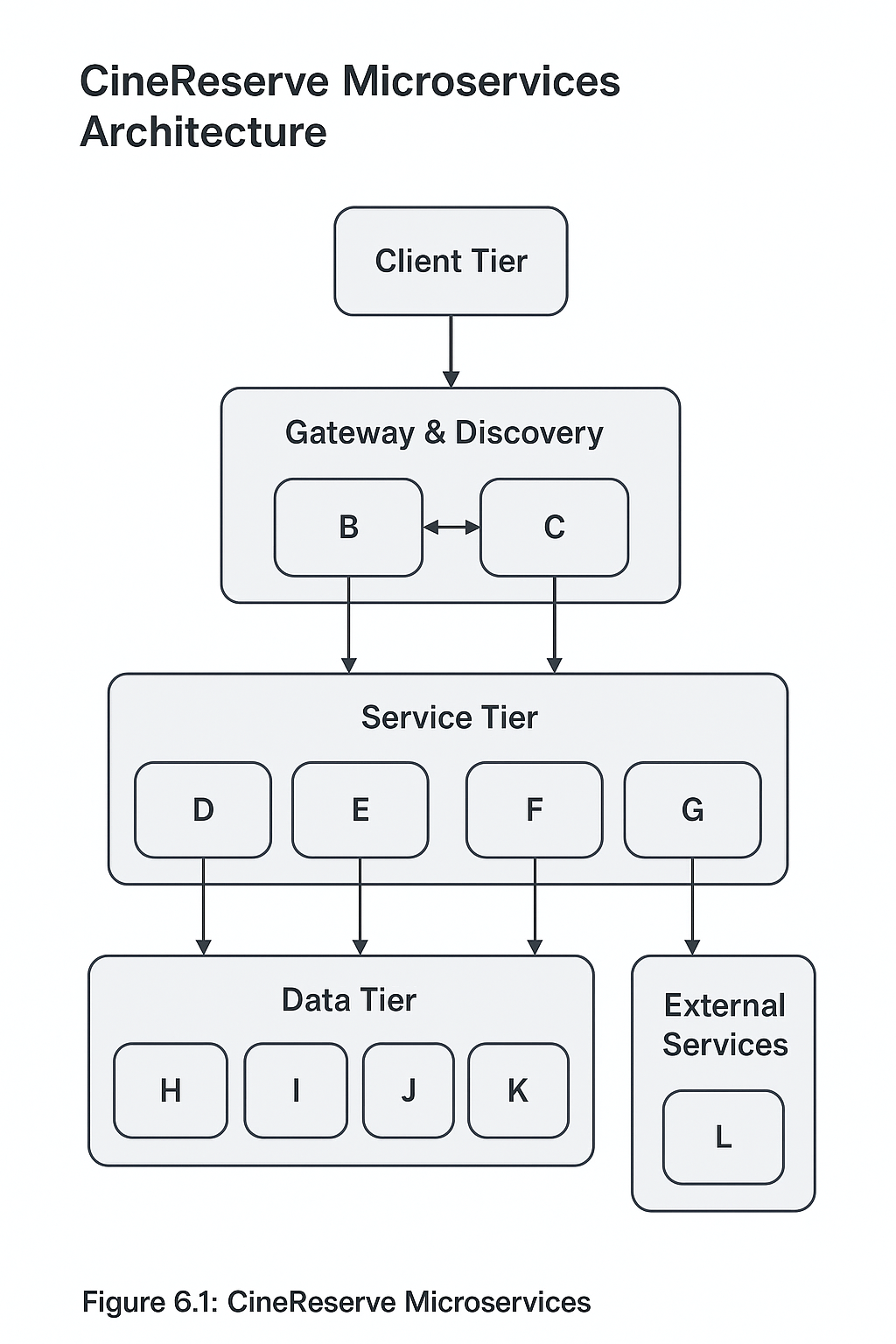


Figure 6. 1: CineReserve Microservices Architecture Diagram

The typical flow for a user booking a ticket is as follows:

1. The client sends a login request to the API Gateway.
2. The Gateway routes the request to the IAM Service, which authenticates the user and returns a JWT.
3. The client makes subsequent requests to browse movies (routed to the Catalog Service) and book a ticket (routed to the Booking Service), including the JWT in the Authorization header.
4. The API Gateway validates the JWT before forwarding the requests to the respective services.
5. The Booking Service communicates with the external Payment Gateway to process the payment.
6. Upon successful payment, the Booking Service confirms the booking and publishes an event.
7. The Notification Service consumes this event and pushes a real-time confirmation to the client via a WebSocket connection.

## 6.2. Implementation of Secure Authentication with Spring Security and JWT

Securing the CineReserve platform was a top priority. The authentication mechanism was implemented using Spring Security 6 and JSON Web Tokens (JWT), providing a robust and stateless solution for protecting API endpoints.

A JWT is a compact, URL-safe means of representing claims to be transferred between two parties. A JWT consists of three parts separated by dots (.): Header, Payload, and Signature.

* **Header:** Contains metadata about the token, such as the signing algorithm used (e.g., HS256).
* **Payload:** Contains the claims, which are statements about an entity (typically the user) and additional data, such as the user's ID, roles, and token expiration time (exp).
* **Signature:** Used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

The authentication flow is as follows:

1. A user submits their credentials (username/password) to the POST /auth/login endpoint on the IAM Service.
2. Spring Security's AuthenticationManager validates the credentials against the user data stored in the database.
3. If authentication is successful, a JwtService utility class generates a JWT containing the user's details and an expiration date.
4. This token is returned to the client.
5. The client must include this JWT in the Authorization header (as a Bearer token) for all subsequent requests to protected endpoints.
6. A custom JwtAuthFilter intercepts each incoming request, extracts the token, validates its signature and expiration, and sets the user's authentication context in Spring Security's SecurityContextHolder.

Below is a snippet of the Spring Security configuration class (SecurityConfig.java), demonstrating how endpoints are secured and the custom JWT filter is integrated :

This Java class SecurityConfig is a Spring Security configuration that sets up authentication and authorization for a web application. It disables CSRF protection, defines which endpoints are accessible without authentication (/auth/register and /auth/login), and restricts access to other endpoints based on user roles—allowing both ROLE\_USER and ROLE\_ADMIN for movie APIs, but only ROLE\_USER for booking APIs. All other requests require authentication. The class configures the application to use stateless session management, integrates a custom JwtAuthFilter before the UsernamePasswordAuthenticationFilter, and provides a DaoAuthenticationProvider with a UserDetailsService and a BCryptPasswordEncoder for password handling. It also defines beans for PasswordEncoder, AuthenticationProvider, and AuthenticationManager, ensuring secure user authentication using JWT and role-based access control.

## 6.3. Development of RESTful APIs for Core Functionalities

In a microservices-based architecture, **RESTful APIs** form the essential backbone that enables smooth communication between independent services as well as between the backend and external clients such as web or mobile applications. These APIs act as formal contracts that define the interaction rules and ensure consistency, scalability, and interoperability across the system.

For the CineReserve platform, the APIs were designed following **REST principles**, ensuring statelessness, resource-based design, and predictable URL structures. Each endpoint was aligned with standard HTTP methods such as **GET** (retrieve data), **POST** (create resources), **PUT/PATCH** (update resources), and **DELETE** (remove resources). This adherence to conventions simplified both client-side consumption and long-term maintenance of the services.

The intern actively contributed to the **design and implementation** of these APIs, focusing on the core functionalities required by the platform. The major areas covered included:

1. **User Management APIs**
   * Endpoints for user registration, authentication, and profile management.
   * Designed to handle secure login/logout processes, password updates, and retrieval of user-specific booking history.
2. **Movie and Show APIs**
   * Endpoints to list movies currently available for booking, retrieve movie details, and access metadata such as duration, language, genre, and release date.
   * Show-related APIs allowed clients to fetch all available shows for a specific movie, including showtime and auditorium information.
3. **Seat Management APIs**
   * Endpoints to check the availability of seats for a given show in real-time.
   * Provided detailed seat maps with availability status to support user interactions during seat selection.
4. **Booking APIs**
   * Core endpoints that allowed users to create new bookings by selecting a movie, show, and seats.
   * Supported features such as retrieving booking details, viewing payment status, and canceling or modifying bookings under defined conditions.
   * These endpoints ensured transactional integrity by verifying seat availability at the time of booking and preventing double-booking.
5. **Payment Integration APIs**
   * Endpoints were designed to initiate, confirm, and verify payment transactions.
   * Ensured secure communication with third-party payment gateways and proper synchronization of booking status with payment outcomes.

To ensure clarity and usability, all APIs were thoroughly **documented** using **Postman**, a widely used API testing and documentation platform. The documentation provided detailed descriptions of each endpoint, required input parameters, expected responses, and possible error codes. This not only facilitated collaboration within the development team but also streamlined the onboarding of frontend developers who relied on these APIs.

Additionally, **Postman collections** were created to automate testing and validate endpoint behavior. These collections enabled regression testing after updates and ensured that all APIs consistently returned the expected results.

The design emphasized **robust error handling** and **consistent response formats**, allowing clients to manage different scenarios gracefully (e.g., unavailable seats, expired sessions, or failed payments). Clear status codes such as 200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, and 404 Not Found were returned in alignment with REST best practices.

By carefully structuring and documenting these APIs, the CineReserve platform achieved:

* **Scalability**: Independent services could be updated or extended without disrupting others.
* **Flexibility**: APIs could be consumed by multiple types of clients (web, mobile apps, or third-party systems).
* **Reliability**: Rigorous testing ensured robustness and minimized downtime.
* **Developer Efficiency**: Well-documented endpoints accelerated frontend development and integration.

In summary, the development of RESTful APIs provided a solid communication framework for CineReserve. The APIs not only supported the platform’s **core functionalities**—from user management to booking workflows—but also laid the groundwork for future scalability and integration with external services.

Table 6. 1: Summary of CineReserve RESTful API Endpoints

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Endpoint | HTTP Method | Description | Roles Allowed | Sample Request Body | Sample Success Response |
| /auth/register | POST | Registers a new user account. | Public | {"username": "user1", "email": "[user1@example.com](mailto:user1@example.com)", "password": "password123"} | {"message": "User registered successfully"} |
| /auth/login | POST | Authenticates a user and returns a JWT. | Public | {"username": "user1", "password": "password123"} | {"token": "eyJhbGciOiJIUzI1NiJ9..."} |
| /api/movies | GET | Retrieves a list of all currently showing movies. | USER, ADMIN | N/A | [{"id": 1, "title": "Inception"},...] |
| /api/theaters/{theaterId}/shows | GET | Retrieves all shows for a specific theater. | USER, ADMIN | N/A | `` |
| /api/bookings | POST | Creates a new ticket booking. | USER | {"showId": 101, "seatIds": , "totalAmount": 500.00} | {"bookingId": "b-123", "status": "PENDING", "paymentOrderId": "order\_abc"} |
| /api/payments/verify | POST | Verifies the payment signature from the gateway. | USER | {"razorpay\_order\_id": "order\_abc", "razorpay\_payment\_id": "pay\_def", "razorpay\_signature": "..."} | {"bookingId": "b-123", "status": "CONFIRMED"} |

## 6.4. Database Design, Schema, and ORM Integration

A robust and well-normalized database schema is the foundation of the Booking Service. The schema was carefully designed to capture all relevant information about users, movies, shows, seats, and bookings. By applying normalization principles, redundancy was minimized, data consistency was enforced, and integrity was maintained across all relationships.

The database schema was represented through an **Entity-Relationship Diagram (ERD)**, which outlines the major entities and their associations. Key entities in this design include:

* **User**: Stores essential details about customers, such as personal information and login credentials.
* **Movie**: Represents films available for booking, along with attributes like title, duration, genre, and description.
* **Show**: Represents specific screenings of a movie, linked to a date, time, and theater or screen.
* **Seat**: Captures seating information within a particular screen, including seat number, row, and availability.
* **Booking**: Acts as a transactional record linking users, shows, and reserved seats.

Relationships were defined to mirror real-world interactions. For instance:

* A **User** can make multiple **Bookings**.
* A **Movie** can be associated with multiple **Shows**.
* Each **Show** contains many **Seats**, and each seat can either be free or booked.
* A **Booking** links a user to a specific show and one or more reserved seats.

This structure enforces referential integrity using foreign keys and ensures that all data is logically connected. For example, a booking cannot exist without a corresponding user, show, and seat.

### 6.4.1 Implementation with Spring Data JPA and Hibernate

To translate this schema into the Spring Boot application, the **Object-Relational Mapping (ORM)** approach was applied using **Spring Data JPA** with **Hibernate** as the persistence provider. Each table in the schema was mapped to a corresponding Java class. These classes were annotated to identify them as persistent entities. Each attribute of the class corresponded to a column in the table, while relationships such as one-to-many, many-to-one, or many-to-many were explicitly defined using annotations. This ensured that object-level associations matched the underlying relational model.

For example, the **Booking** entity contained fields referencing the user who made the booking, the show being booked, and the seats reserved. Instead of manually managing foreign keys, these associations were defined as object relationships, which Hibernate automatically translated into the appropriate joins when queries were executed.

### 6.4.2 Data Access Layer

The data access layer was implemented using **repository interfaces**. By extending the JpaRepository interface provided by Spring Data JPA, the application gained access to a wide set of CRUD operations without requiring explicit implementation. This included common tasks such as saving new records, retrieving entities by their identifiers, updating existing records, and deleting entries.

In addition, Spring Data JPA offered the capability to define custom query methods by following naming conventions or by using query annotations. This allowed the Booking Service to execute domain-specific queries, such as retrieving all available seats for a given show or listing all bookings made by a particular user, without the need for verbose SQL statements.

### 6.4.3 Benefits of This Approach

This integration of schema design with ORM brought several advantages:

1. **Reduced Boilerplate Code**: Developers did not need to write repetitive SQL or JDBC logic, as most queries were automatically generated.
2. **Maintainability**: Changes to the database schema or entity classes were automatically reflected in the application with minimal modifications.
3. **Portability**: Since Hibernate abstracts database-specific details, the system can be adapted to different relational databases with little effort.
4. **Data Integrity**: The combination of database constraints and ORM-level validation ensured consistency across all operations.
5. **Productivity**: Developers could focus on business logic rather than low-level data management tasks.

In summary, the combination of a well-structured relational schema and ORM integration through Spring Data JPA and Hibernate provided a highly efficient, maintainable, and scalable data management solution for the Booking Service.

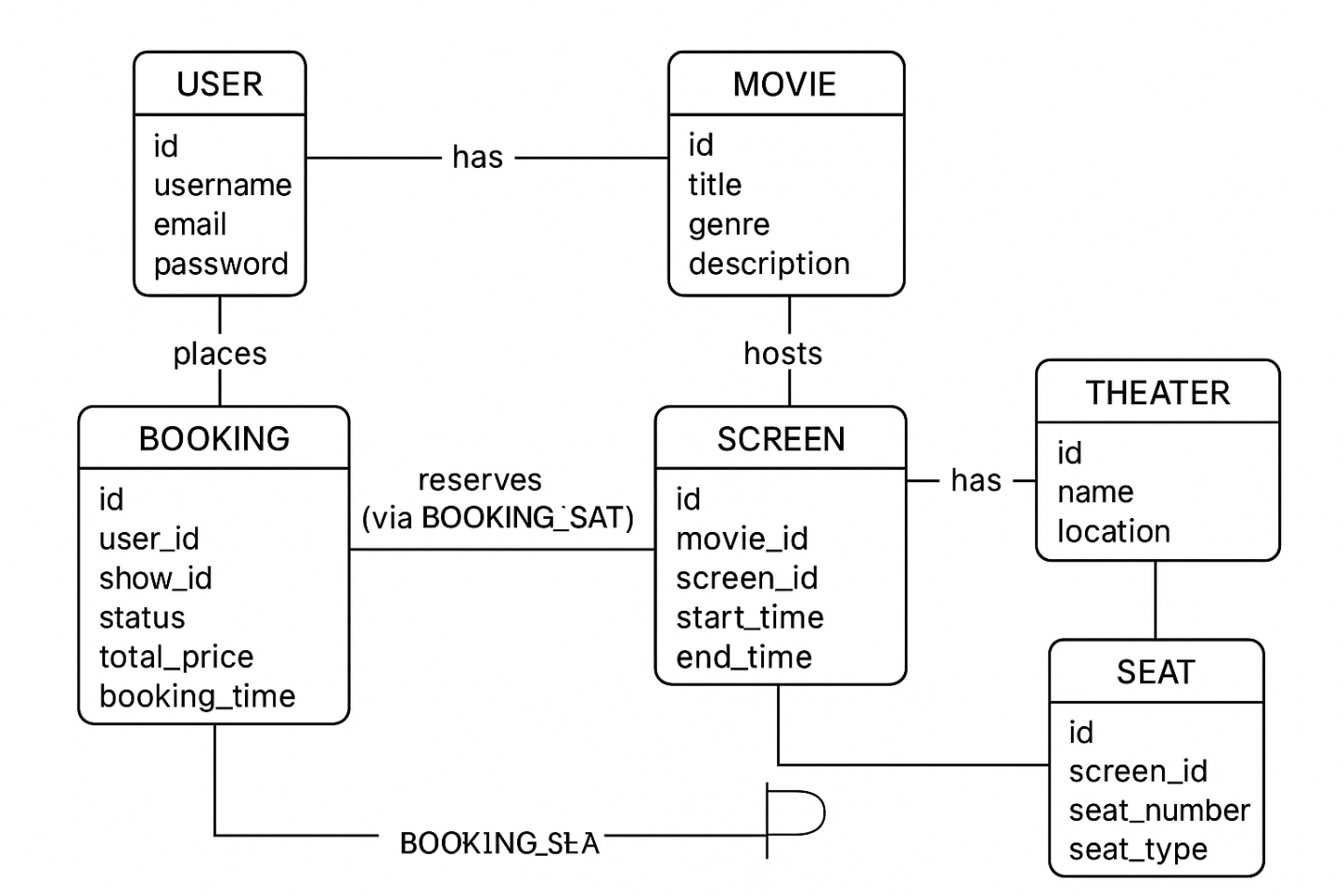


Figure 6. 2: Entity-Relationship Diagram (ERD) for Booking Service

The implementation of this schema in the Spring Boot application was accomplished using Spring Data JPA and Hibernate. Each table was mapped to a Java class annotated with @Entity. For example, the Booking entity class is shown below :

The data access layer was implemented by creating repository interfaces that extend JpaRepository. This approach significantly reduces boilerplate code, as Spring Data JPA provides implementations for common CRUD operations at runtime.

This Java class Booking is a JPA entity mapped to the bookings table in the database, representing a movie ticket booking. Each booking has a unique id generated automatically and is linked to a User and a Show through @ManyToOne relationships, ensuring that every booking belongs to a specific user and show. It also has a @ManyToMany relationship with Seat, managed via a join table booking\_seat, which allows multiple seats to be reserved under a single booking. The booking’s current state is tracked using the BookingStatus enum, stored as a string. Additional details include totalPrice to record the amount paid and bookingTime to store when the booking was made. Overall, this entity models the essential attributes and relationships needed to manage bookings in a movie ticketing system.

## 6.5. Real-time Communication with WebSockets

To provide an immediate and interactive user experience, the Notification Service was implemented using WebSockets. WebSocket is a communication protocol that provides a full-duplex communication channel over a single TCP connection, allowing the server to push data to the client in real-time without the client having to poll for updates.

Spring Boot provides excellent support for WebSockets, particularly when used with the STOMP (Simple Text Oriented Messaging Protocol) sub-protocol, which defines a simple wire protocol for clients and brokers to communicate with.

The implementation involved the following steps:

1. **Configuration:** A configuration class was created to enable the WebSocket message broker and register a STOMP endpoint.
2. **Controller:** A controller was created to handle messages. However, for server-to-client push notifications, the SimpMessagingTemplate is used to send messages directly to a topic.
3. **Client-Side Implementation:** A simple JavaScript client using SockJS and StompJS was created to connect to the WebSocket endpoint and subscribe to the user-specific topic.

**6.6. Third-Party Service Integration: The Payment Gateway**

A critical part of the intern's work on the Booking Service was the integration of a payment gateway to handle online transactions. **Razorpay**, a popular payment gateway in India, was chosen for this purpose due to its comprehensive documentation and developer-friendly APIs.

The integration process followed these steps :

1. **Account Setup and API Keys:** A Razorpay test account was created, and API key\_id and key\_secret were obtained and configured in the Spring Boot application's application.properties file.
2. **Order Creation (Backend):** When a user proceeds to pay, the client sends the booking details to the backend. The backend PaymentService uses the Razorpay Java SDK to create an order with Razorpay's servers. This order includes the amount (in the smallest currency unit, e.g., paise) and a unique receipt ID. Razorpay returns a unique order\_id.
3. **Checkout (Frontend):** The backend returns the order\_id to the frontend. The frontend then uses Razorpay's Checkout.js library to open a secure payment modal, passing the order\_id and other details.
4. **Payment Callback and Verification (Backend):** After the user completes the payment, Razorpay's checkout library sends a callback to the frontend with the razorpay\_payment\_id, razorpay\_order\_id, and razorpay\_signature. The frontend forwards this information to a verification endpoint on the backend.
5. **Signature Verification:** The backend must verify the razorpay\_signature to confirm that the payment callback is authentic and has not been tampered with. This is done using a utility function that generates an HMAC-SHA256 signature from the order\_id and payment\_id using the key\_secret and compares it with the signature received from the client.

Java

// In PaymentService.java  
public boolean verifyPaymentSignature(String orderId, String paymentId, String signature) {  
 try {  
 String payload = orderId + "|" + paymentId;  
 Utils.verifyPaymentSignature(payload, signature, keySecret);  
 return true;  
 } catch (RazorpayException e) {  
 // Signature mismatch  
 return false;  
 }  
}

Upon successful verification, the booking status in the database is updated from PENDING to CONFIRMED.

Table 6. 2 Overview for the Module Details

|  |  |  |
| --- | --- | --- |
| **Section** | **Topic** | **Description** |
| **6.1** | System Architecture | Microservices-based architectural approach for modular, scalable design |
| **6.2** | Secure Authentication | Implementation of Spring Security with JWT for authentication and authorization |
| **6.3** | RESTful APIs | Development of APIs to support core functionalities of the system |
| **6.4** | Database Design | Schema design and ORM (Object Relational Mapping) integration |
| **6.5** | Real-time Communication | Use of WebSockets for live, bidirectional communication |
| **6.7** | Document Generation | Automated PDF E-Ticket generation for users |

## 6.7. Document Generation: PDF E-Tickets

To provide a complete, end-to-end feature, the intern was tasked with implementing a mechanism to generate a downloadable PDF e-ticket upon successful booking confirmation. This functionality was built using **Apache PDFBox**, a powerful open-source Java library for working with PDF documents. The inclusion of this feature demonstrates a comprehensive approach to problem-solving, moving beyond standard API and database interactions to produce a tangible, user-facing output.

The process of generating an e-ticket involved the following steps :

1. **Add Maven Dependency:** The pdfbox dependency was added to the pom.xml file.
2. **Create PDF Document:** A new PDDocument object was created in memory.
3. **Add Page:** A new PDPage was added to the document.
4. **Write Content:** A PDPageContentStream was used to write content to the page. This included:
   1. Movie title, theater name, showtime, and seat numbers.
   2. Booking ID and total price.
   3. A dynamically generated QR code (using another library like ZXing) representing the booking ID for easy scanning at the cinema.
5. **Save Document:** The PDDocument was saved to a ByteArrayOutputStream.
6. **Serve as API Response:** The byte array was then returned from a REST endpoint with the appropriate Content-Type (application/pdf) and Content-Disposition headers, prompting the user's browser to download the file.

This functionality added significant value to the project, showcasing the ability to integrate and utilize third-party libraries to deliver a complete and professional user experience.

**Snapshots :**

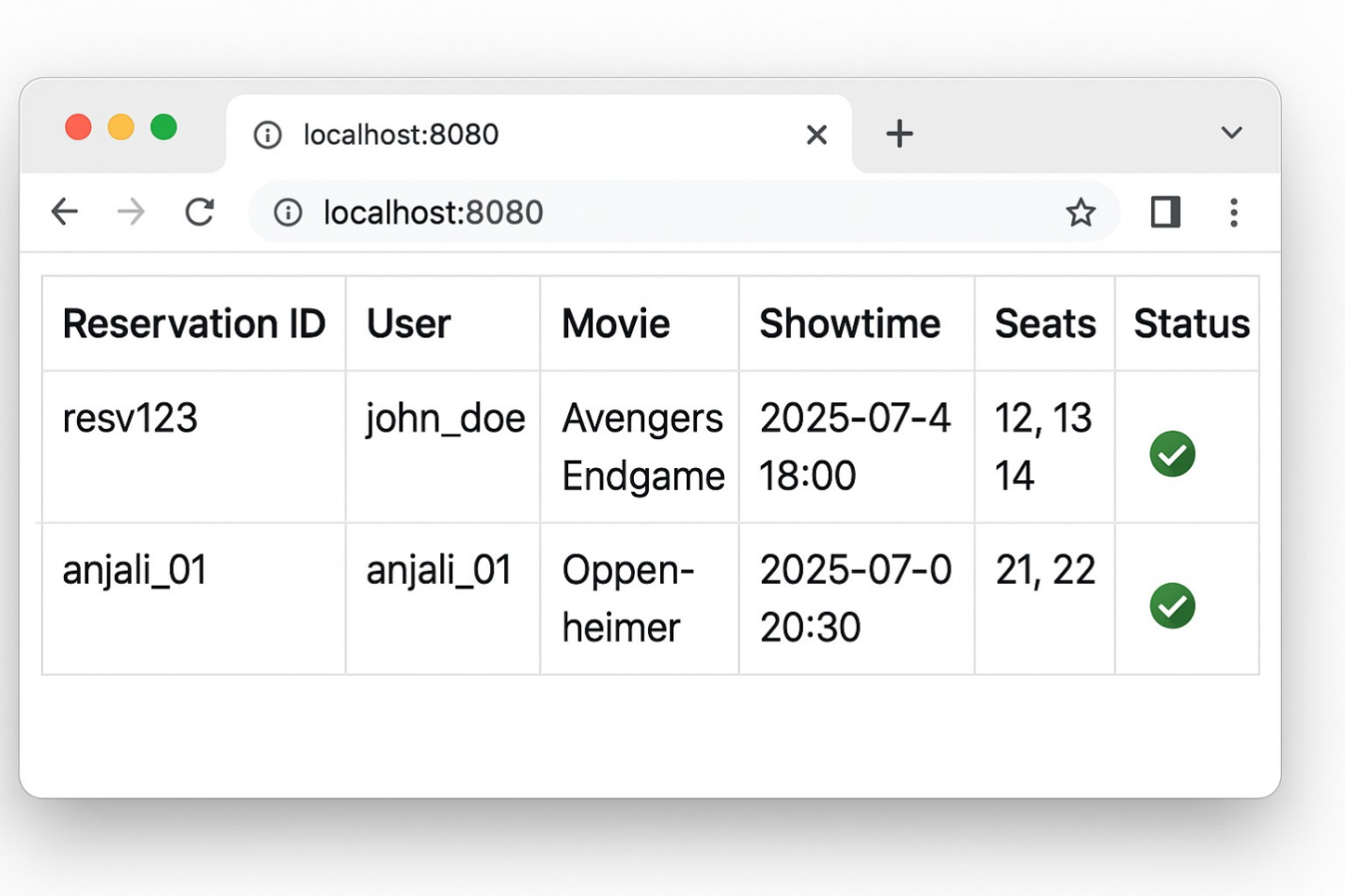


Figure 6. 3 Assessment: Movie ReservationSystem

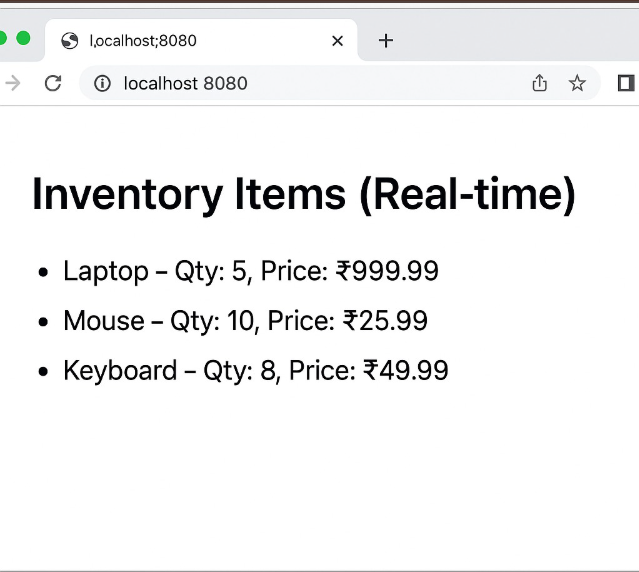


Figure 6. 4 Assessment: Inventory Management System

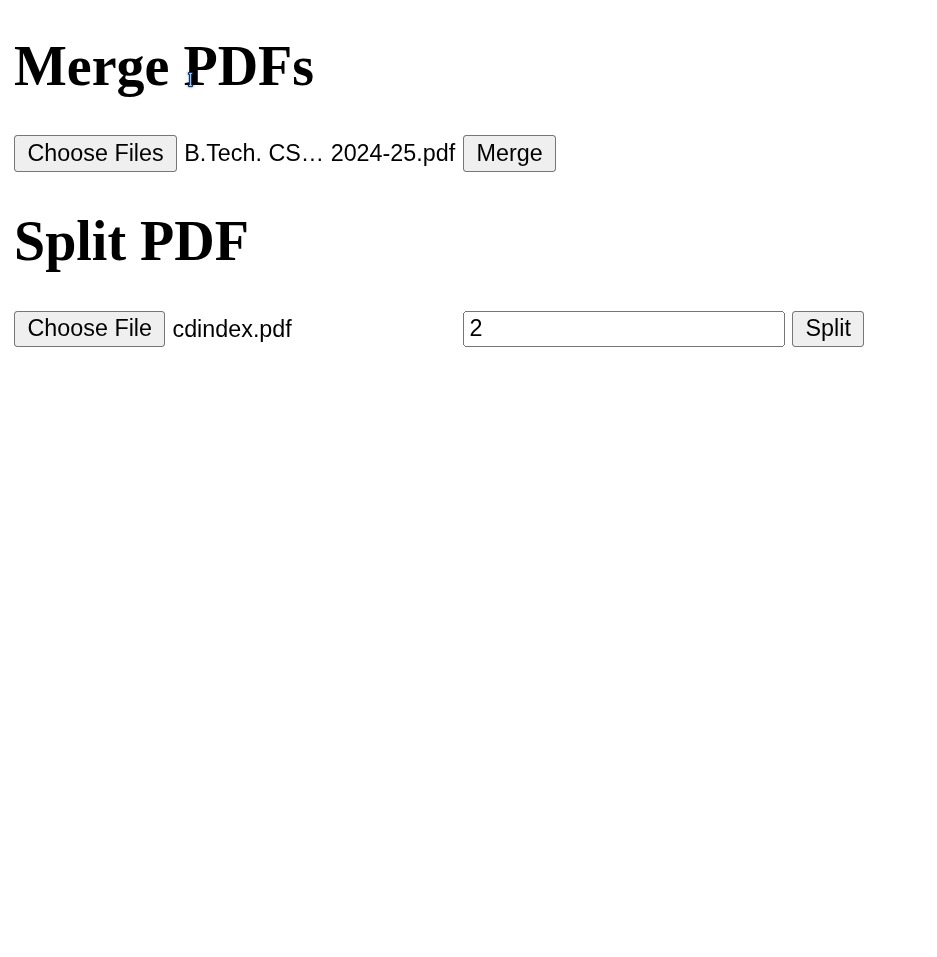


Figure 6. 5 Assessment: PDF Merge & Split

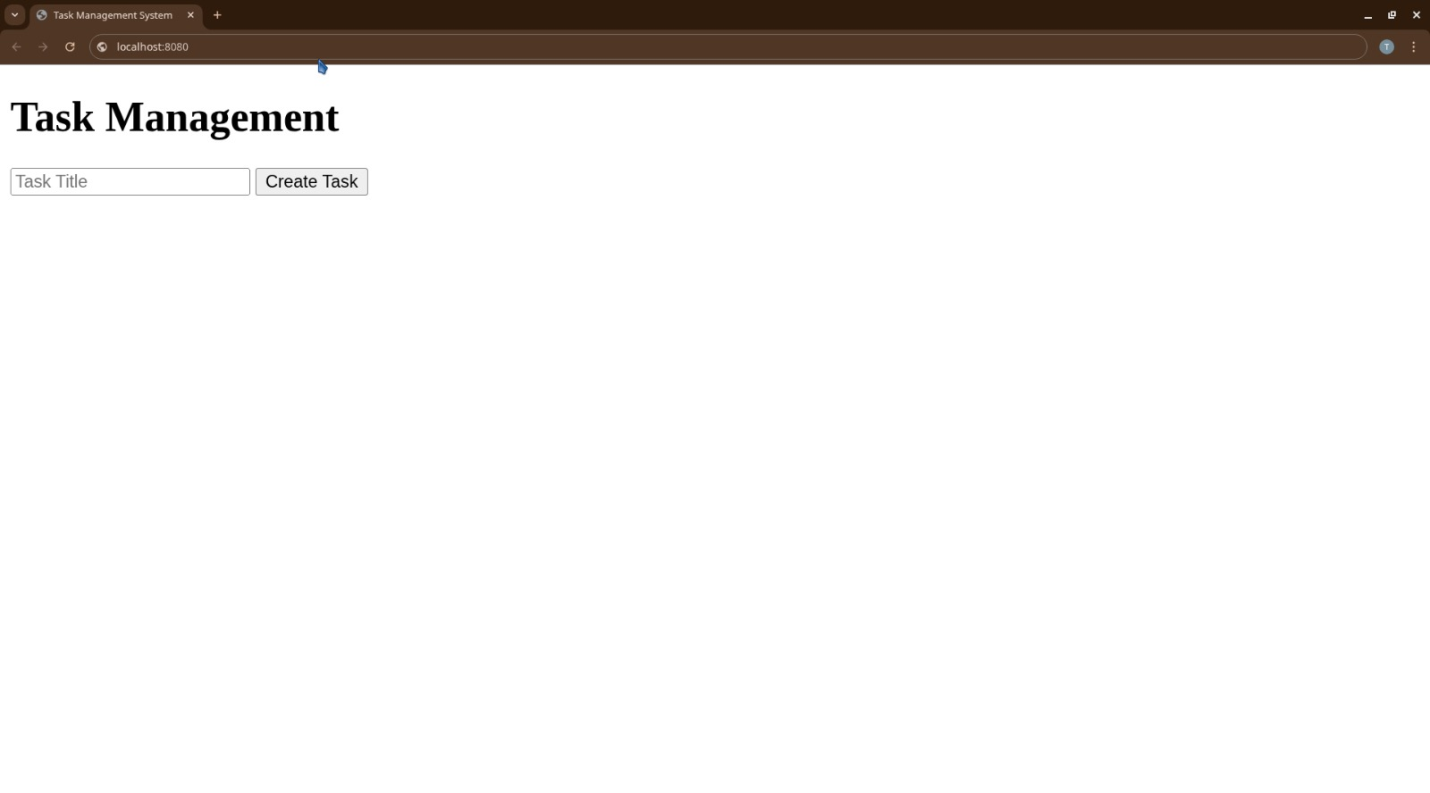


Figure 6. 6 Assessment: Task Management System

# Chapter 7: Training Outcome

The 30-day industrial training program at Prodesk IT & Engineering Services was designed as a truly comprehensive and immersive learning experience, strategically crafted to accelerate professional growth and deepen technical understanding. This intensive period offered more than just exposure; it provided hands-on, in-depth engagement with the realities of modern software development. The invaluable opportunity to work on a real-world project, specifically the intricate "CineReserve" platform, under the direct guidance of experienced professionals, proved instrumental in effectively bridging the gap between theoretical academic knowledge and its practical, nuanced application in an industrial setting. This collaborative and project-centric approach fostered an environment where abstract concepts were immediately put into practice, debugged, and refined. As a result, the training yielded significant advancements in several critical areas, and the holistic outcome can be categorized into four interconnected key areas of development, each contributing substantially to a well-rounded professional skillset.

## 7.1. Acquisition of Technical Competencies

* The hands-on experience gained during the internship was not merely theoretical but deeply practical, leading to a significant and measurable enhancement of technical skills across a wide spectrum of modern backend technologies. The key competencies acquired are detailed as follows:
* **Java and Spring Boot:** I gained comprehensive proficiency in developing robust, secure, and scalable backend services using Java and the Spring Boot framework. This went far beyond basic application setup. It included a deep, functional understanding of core Spring concepts like **Dependency Injection (DI)** and Inversion of Control (IoC), which I used daily to write loosely coupled, modular, and easily testable components (e.g., injecting service classes into controllers). I leveraged Spring Boot's powerful **Auto-Configuration** to rapidly bootstrap applications with minimal boilerplate code for features like database connectivity and web server setup. A significant part of my work involved the **creation of RESTful controllers** using annotations like @RestController, @RequestMapping, @PostMapping, and @GetMapping to define API endpoints, handle various HTTP methods, and manage the serialization of Java objects to JSON responses and the deserialization of JSON request bodies into Java objects.
* **Microservices Architecture:** The "CineReserve" project provided a practical sandbox for understanding the principles and patterns of a microservices architecture. I developed a tangible understanding of **service decomposition**, seeing firsthand how a large, monolithic application was broken down into smaller, independently deployable services, each responsible for a specific business domain (e.g., Identity, Catalog, Booking, Notifications). I implemented **inter-service communication via REST APIs**, for example, making synchronous HTTP calls from the Booking Service to the Catalog Service to retrieve movie details when a user initiates a booking. This work reinforced the "database per service" pattern, where each microservice owned its own dedicated database, ensuring loose coupling and preventing one service from directly accessing the data store of another, which is a cornerstone of this architectural style.
* **API Development and Security:** A primary focus of the internship was acquiring skills in designing, implementing, and, critically, securing RESTful APIs. This included adhering to REST principles by using proper HTTP verbs, meaningful resource URIs, and appropriate HTTP status codes. I gained extensive hands-on experience with the **Spring Security** framework for implementing robust authentication and authorization. I configured security filter chains to protect specific endpoints, requiring valid user credentials for access. A key achievement was implementing a **stateless session management** system using **JSON Web Tokens (JWT)**. This involved creating an authentication endpoint that, upon successful login, would generate a signed JWT containing the user's identity and roles. This token was then sent with every subsequent API request in the Authorization header, where it was validated by the receiving microservice to authorize the requested action.
* **Data Persistence:** The internship provided practical experience with both relational and NoSQL database paradigms, using **PostgreSQL** for its transactional integrity. I developed a strong proficiency in using **Spring Data JPA** and its underlying ORM (Object-Relational Mapping) provider, **Hibernate**. This workflow involved defining Java objects as @Entity classes that mapped directly to database tables. I extensively used **Spring Data Repositories** by creating interfaces that extended JpaRepository, which automatically provided a full set of CRUD (Create, Read, Update, Delete) operations, significantly reducing the amount of boilerplate data access code. Furthermore, I learned to effectively **manage database transactions** using the declarative @Transactional annotation on service-layer methods to ensure that a series of related database operations would either all succeed or all fail together, thus maintaining data consistency and integrity, which was especially critical in the booking process.
* **Real-time Communication:** I learned to implement modern, real-time features using **WebSockets**, a technology that enables full-duplex communication between a client and a server. Specifically, I used Spring Boot's built-in support to create a WebSocket endpoint and utilized the **STOMP (Simple Text Oriented Messaging Protocol)** over WebSockets. This allowed me to build a simple publish-subscribe messaging system. For instance, after a booking was successfully confirmed, the backend Booking Service would publish a notification message to a specific topic (e.g., /topic/confirmations/user123). The frontend client, subscribed to this topic, would receive this message as an instantaneous **server-to-client push notification**, enabling the UI to be updated in real-time without the need for the user to manually refresh the page.
* **Third-Party Integrations:** The project provided valuable, real-world experience in integrating external services and libraries into a backend application, a common requirement in software development. This included a deep integration with the **Razorpay payment gateway SDK**. The process involved not just invoking the API to create a payment order, but also building a secure webhook endpoint to receive asynchronous callbacks from Razorpay, cryptographically verifying the signature of these callbacks to prevent fraud, and then updating the booking status in the database. In a separate feature, I worked extensively with the **Apache PDFBox library** for dynamic document generation. I wrote Java code that, upon a successful booking, would create a new PDF document from scratch, programmatically add text, images (like QR codes), and formatting to generate a complete e-ticket, and then serve this dynamically generated PDF file to the user over a REST API endpoint.

## 7.2. Assimilation of Professional Engineering Practices

* Beyond the mastery of specific technologies, the internship provided invaluable and deep-seated insight into the professional practices, methodologies, and philosophies that underpin modern, high-functioning software development teams.
* **Agile and Scrum:** My participation in the Agile (Scrum) development process was not passive but active and integral. The daily stand-up meetings were a disciplined forum for synchronizing the team, where I articulated my previous day's accomplishments, outlined my current day's objectives, and, most importantly, raised any blockers that were impeding my progress. In the **sprint planning sessions**, I engaged with the team to analyze user stories from the product backlog, break them down into smaller, manageable technical tasks, and participate in estimation exercises to collectively agree on the effort required. This provided a clear roadmap for the upcoming sprint. The **sprint reviews** were an opportunity to demonstrate the working software I had developed to the project manager and other stakeholders, gathering crucial feedback that would inform the next iteration. Finally, the **sprint retrospectives** were a candid and constructive environment where the team reflected on the past sprint—discussing what went well, identifying what could be improved, and creating actionable steps to enhance our process in the future. This cyclical process provided a profound appreciation for the power of iterative development, the necessity of continuous feedback loops, and the flexibility of adaptive planning to respond to changing requirements.
* **Version Control:** I became proficient in leveraging Git as a powerful tool for version control in a collaborative, multi-developer environment. The workflow became second nature. All new development was done on isolated **feature branches**, a practice that protected the integrity and stability of the main development branch. I learned to make atomic, logical **commits** with clear and descriptive messages that created a transparent and understandable history of the project's evolution. A core part of the daily routine was creating **pull requests (PRs)**, which served as the formal gateway for code review. These PRs were not just code dumps; they were well-documented requests for feedback, often linked to a specific Jira ticket, that initiated a technical dialogue with my peers. This process also involved managing local and remote branches and occasionally resolving merge conflicts, ensuring that my contributions integrated seamlessly with the work of others before being **merged** into the main development line.

* **CI/CD Principles:** While direct management of the Jenkins pipeline was outside my intern responsibilities, I gained a practical and functional understanding of the immense importance of Continuous Integration and Continuous Deployment (CI/CD). I saw this in action every time I pushed code or created a pull request. This would automatically trigger a **Continuous Integration (CI)** pipeline, which would build the service, execute the full suite of unit and integration tests, and perform static code analysis. The immediate feedback—a green checkmark or a red 'x' next to my PR—was a powerful mechanism for catching bugs early and ensuring that my changes didn't break existing functionality. I also understood that once a PR was approved and merged, a **Continuous Deployment (CD)** pipeline would take over, automatically deploying the latest version of the service to a staging environment for further testing. This firsthand observation made it clear how CI/CD automation is essential for increasing development velocity, improving code quality, and reducing the risk of manual deployment errors.
* **Code Quality and Reviews:** The code review process was one of the most impactful learning experiences of the internship. I learned to approach coding not just as the act of making something work, but as the craft of writing **clean, maintainable, and well-documented code** that other engineers could easily understand and extend. I actively participated in the code review process, carefully reading the feedback provided by senior developers on my pull requests. This constructive criticism was an invaluable, personalized lesson in adhering to the company's established coding standards and industry best practices. Whether it was a suggestion to refactor a complex method for better readability, a recommendation to use a more efficient algorithm, or a correction in naming conventions, each piece of feedback was a building block. This rigorous review loop was crucial in elevating the quality of my code and instilling a professional discipline that will be foundational to my future work.

## 7.3. Development of Analytical and Problem-Solving Skills

* Working on a complex, multi-faceted project like CineReserve consistently presented numerous technical challenges that were instrumental in honing my analytical and problem-solving abilities. The process of taking a problem from symptom to resolution required a methodical and logical approach. Key experiences in this area include:
* **Debugging Complex Issues:** I frequently encountered and resolved a variety of challenging issues that required deep dives into the application's runtime behavior. This involved more than just fixing syntax errors; it meant untangling logical flaws in a complex system. For instance, I spent considerable time **debugging Spring Security filter chain configurations** when certain API endpoints were unexpectedly returning "403 Forbidden" errors. The resolution involved meticulously enabling debug-level logging for Spring Security modules to trace the entire request flow through the sequence of filters, identifying the exact point of failure, and correcting the security rule configurations. Another significant challenge was **troubleshooting intermittent database transaction rollbacks in the booking process.** A booking operation involved multiple database writes, and if any single step failed, the entire transaction had to be rolled back to maintain data integrity. Identifying the root cause required analyzing stack traces, configuring JDBC logging to inspect the exact SQL queries being executed, and using the debugger to examine the state of objects at the moment of failure. Finally, I worked on **ensuring the correct and robust handling of asynchronous payment gateway callbacks (webhooks).** This was particularly complex because the callback is an external event that could arrive at any time, potentially creating race conditions or idempotency issues. Debugging this involved using tools to tunnel the webhooks to my local machine and implementing comprehensive logging to capture every callback payload, ensuring that the system could reliably handle duplicate or out-of-order notifications without corrupting the booking status.
* **Architectural Trade-offs:** The project provided a practical education in the critical trade-offs involved in modern software design and architecture. I gained a tangible appreciation for the "it depends" nature of engineering decisions. A prime example was **understanding the data storage strategy** across different microservices. I came to comprehend why a relational database like PostgreSQL was the clear choice for the transactional **Booking Service**. Its strict adherence to **ACID (Atomicity, Consistency, Isolation, Durability) properties** was non-negotiable for a system handling financial transactions and seat inventory, where data consistency is paramount. In contrast, I understood how a NoSQL database like MongoDB might be a more suitable choice for a service that generates less structured, high-volume data, such as **notification logs or user activity streams.** In that context, the benefits of schema flexibility, horizontal scalability, and high write throughput would likely outweigh the need for the strict transactional guarantees offered by a relational system. This experience solidified my understanding that choosing the right technology requires a careful analysis of the specific problem's requirements and constraints.
* **Algorithmic Thinking:** Beyond using framework features, I had to apply structured, logical thinking to implement key business logic. This was most evident in the implementation of the **core booking algorithm.** The task required breaking down a complex user action into a sequence of discrete, verifiable steps. This included first querying the database to check the current availability of the requested seats, ensuring they were not already booked or locked by another user's pending transaction. A critical aspect was **conceptually handling concurrent requests.** I learned about the dangers of race conditions—where two users could simultaneously see a seat as available and both attempt to book it. This led to an understanding of concurrency control mechanisms like pessimistic locking (e.g., using SELECT ... FOR UPDATE in SQL) to exclusively lock the seat records during a transaction. Finally, I implemented the logic for **managing the state transitions of a booking.** A booking wasn't simply "created"; it moved through a defined lifecycle, from PENDING (when the user starts the payment process) to CONFIRMED (upon successful payment) or FAILED/EXPIRED (if the payment fails or the user abandons the process). Implementing this state machine required careful algorithmic thinking to ensure that each state transition was valid and that the system could not enter an inconsistent state, thus preventing issues like double bookings or orphaned seat reservations.

## 7.4. Enhancement of Collaborative and Communication Abilities

* The internship experience underscored the principle that technical proficiency alone is insufficient in a professional engineering environment; the ability to collaborate and communicate effectively is equally crucial. The development in this area was significant and multifaceted:
* **Teamwork:** The daily operations were a practical lesson in effective teamwork within a cross-functional Agile team. This involved more than just completing individual tasks; it required constant collaboration with backend and frontend developers to define and integrate API contracts, ensuring a seamless flow of data between the server and the client. It also meant working closely with Quality Assurance (QA) engineers to understand bug reports, replicate issues, and provide timely fixes. Engaging with the project manager during sprint planning, daily stand-ups, and review meetings was essential to align on priorities, report progress, and collectively work towards achieving the common sprint goals. This dynamic environment fostered a shared sense of ownership and collective problem-solving.
* **Technical Communication:** A significant area of growth was the ability to articulate complex technical concepts, challenges, and proposed solutions with clarity and precision. During daily stand-up meetings and technical brainstorming sessions, I learned to present my progress and impediments concisely. Furthermore, this skill was honed through written communication. Crafting detailed yet understandable descriptions for pull requests became a standard practice, explaining not just *what* changes were made, but *why* they were necessary and how they fit into the broader architecture. Similarly, updating and commenting on Jira tickets required clear documentation of issues, progress, and resolutions, ensuring that all team members had a transparent view of the project's status.
* **Use of Collaboration Tools:** Proficiency was gained in using a suite of industry-standard collaboration tools that form the backbone of a modern software development workflow. My experience with **Jira** went beyond simple task tracking; it involved participating in the entire lifecycle of a task, from understanding user stories and estimating effort with story points to moving tickets across the Scrum board and documenting progress. **Confluence** served as the central knowledge repository, and I learned to effectively navigate and utilize it to access critical technical documentation, architectural diagrams, and team best practices. **Slack** was the hub for real-time and asynchronous communication, and I learned to use it efficiently for quick queries in dedicated channels, sharing code snippets for feedback, and staying updated on automated notifications from CI/CD pipelines and version control systems, understanding how these integrated tools foster a responsive and efficient team dynamic.

# Chapter 8: Conclusion

## 8.1. Synthesis of the Internship Experience

The 30-day industrial training in Backend Development at Prodesk IT & Engineering Services was a profound and immersive experience that provided an exceptional opportunity to translate abstract academic knowledge into tangible, real-world software engineering solutions. This well-structured and intensive program was meticulously designed to offer deep and practical insights into the cutting-edge technologies, agile processes, and collaborative culture that define a modern IT services company. The hands-on involvement with the "CineReserve" project was the cornerstone of this learning journey. It offered a holistic and comprehensive view of the entire software development lifecycle, beginning with the critical phase of requirements analysis and high-level architectural design, and progressing through the intricate stages of backend implementation, robust unit and integration testing, and ultimately, deployment to a production-like environment.

The training program was instrumental in successfully bridging the often-significant gap between the theoretical concepts taught in the classroom and their practical, nuanced application in an industrial context. At Prodesk IT, the professional environment was characterized by a steadfast adherence to Agile methodologies, fostering a dynamic of iterative progress and constant feedback. This, combined with a sharp focus on delivering high-quality, innovative solutions, created an atmosphere that was exceptionally conducive to accelerated learning and significant professional growth. A critical component of this success was the invaluable mentorship provided by senior engineers. Their guidance was consistently both technically sound—offering elegant solutions to complex coding problems—and professionally insightful, providing context on industry best practices, code maintainability, and career development.

## 8.2. Summary of Contributions and Key Achievements

Throughout the internship, a series of concrete and impactful contributions were made to the CineReserve project. These achievements are not just a list of completed tasks but a clear demonstration of the practical application of the skills learned and honed during the training. The key achievements can be detailed as follows:

* **Implemented User Authentication Endpoints**: I assisted in the development of the foundational user registration and login REST APIs for the Identity and Access Management (IAM) service. This involved designing and building secure endpoints to handle user credentials, perform data validation, and interface with the database to create and retrieve user profiles, forming the primary gateway for all user interactions.
* **Developed JWT Generation Logic**: Taking a key role in the system's security, I successfully implemented the logic for generating JSON Web Tokens (JWTs) upon a user's successful authentication. This involved creating stateless, digitally signed tokens containing user claims, which are then used to authorize access to protected resources across the microservices ecosystem, a core component of modern, secure application architecture.
* **Designed Database Schema for Catalog Service**: I contributed significantly to the logical and physical database design for the core entities of the application: movies, theaters, and shows. This work involved defining tables, specifying data types, establishing relationships (such as one-to-many and many-to-many), and creating indices to ensure data integrity and optimize query performance, thereby laying a robust data foundation for all catalog-related functionality.
* **Integrated Razorpay Payment Gateway**: I led the complex task of integrating the Razorpay payment gateway into the Booking Service. This required successfully implementing the complete end-to-end payment flow, which included initiating the transaction by creating an order via the Razorpay API, handling the callback from the payment gateway, and crucially, implementing the cryptographic payment signature verification on the backend to securely confirm payment success.
* **Implemented Real-time Notifications**: To significantly enhance the interactive user experience, I developed a WebSocket endpoint within the Notification Service. This implementation allows the server to push real-time booking confirmation updates directly to the client's browser, providing instant feedback without the need for traditional, less efficient methods like HTTP polling.
* **Created PDF E-Ticket Generation**: I implemented a complete, user-facing feature to dynamically generate and serve PDF e-tickets upon a successful booking. This was accomplished by utilizing the Apache PDFBox library to create a professional-looking ticket, populated with dynamic data from the booking, including movie details, seat numbers, and a scannable QR code, delivering a tangible and valuable asset to the end-user.

## 8.3. Reflections and Future Outlook

This internship has been a truly transformative experience, profoundly influencing my technical understanding and shaping my future career aspirations. The direct exposure to a distributed microservices architecture, cloud-native principles, and modern DevOps practices has provided a clear and exciting perspective on the future trajectory of software development. It has cemented my interest in pursuing a career dedicated to backend systems engineering, with a particular focus on the unique challenges involved in building highly scalable, resilient, and distributed applications.

The practical, hands-on skills gained in Java and the Spring Boot framework have provided a strong and durable foundation upon which to build future expertise. Furthermore, the experience of collaborating within a professional Agile/Scrum team has vividly highlighted the critical importance of effective communication, seamless collaboration, and a commitment to continuous improvement as the pillars for achieving successful and timely project outcomes.

Moving forward, my immediate goal is to continue exploring the advanced concepts introduced during this training in greater depth. I am particularly eager to delve into the areas of cloud computing platforms like AWS and Azure, master container orchestration with tools like Kubernetes, and explore advanced database management strategies for performance and scale. The internship at Prodesk IT has transcended its role as a mere academic requirement; it has served as a powerful catalyst, igniting a genuine passion for the craft of software engineering and providing a clear, well-defined direction for my future career in the ever-evolving technology industry.

# References

1. Prodesk IT. (2025). *Our Vision*. Retrieved from <https://prodesk.in/our-vision-1>
2. Prodesk IT. (2025). *Home*. Retrieved from <https://prodesk.in/home>
3. Scribd. (2025). *Prodesk IT Engineering Service | PDF*. Retrieved from <https://www.scribd.com/document/863984725/Prodesk-IT-Engineering-Service>
4. Prodesk IT. (2025). *IT SOFTWARE*. Retrieved from <https://prodesk.in/it-software>
5. Upforce Tech. (2025). *The Future of India's IT Industry in 2025*. Retrieved from <https://upforcetech.com/the-future-of-indias-it-industry-in-2025/>
6. Grip Invest. (2025). *The Information Technology Industry in India*. Retrieved from <https://www.gripinvest.in/blog/the-information-technology-industry-in-india>
7. Functionly. (2025). *The Optimal SaaS Company Organizational Structure*. Retrieved from <https://www.functionly.com/orginometry/industry-org-charts/the-optimal-saas-company-organizational-structure>
8. University of San Diego. (2025). *IT Organizational Structure*. Retrieved from <https://onlinedegrees.sandiego.edu/it-organizational-structure/>
9. GeeksforGeeks. (2025). *Backend Development*. Retrieved from <https://www.geeksforgeeks.org/blogs/backend-development/>
10. Mobisoft Infotech. (2025). *Essential Principles of Spring Boot Microservices Architecture*. Retrieved from <https://mobisoftinfotech.com/resources/blog/essential-principles-spring-boot-microservices>
11. Spring.io. (2025). *Microservices*. Retrieved from <https://spring.io/microservices/>
12. Kitrum. (2025). *Difference Between MongoDB, MySQL, and PostgreSQL*. Retrieved from <https://kitrum.com/blog/difference-between-mongodb-mysql-and-postgresql/>
13. Amigoscode. (2025). *Spring Boot Roadmap 2025*. Retrieved from <https://amigoscode.com/blogs/spring-boot-roadmap-2025>
14. GeeksforGeeks. (2025). *Introduction to Spring Boot*. Retrieved from <https://www.geeksforgeeks.org/springboot/introduction-to-spring-boot/>
15. Harness. (2025). *Software Development Life Cycle (SDLC) Phases*. Retrieved from <https://www.harness.io/blog/software-development-life-cycle-phases>
16. Agile Manifesto. (2001). *Principles behind the Agile Manifesto*. Retrieved from <https://agilemanifesto.org/principles.html>
17. Simplilearn. (2025). *What is Agile?*. Retrieved from <https://www.simplilearn.com/tutorials/agile-scrum-tutorial/what-is-agile>
18. GeeksforGeeks. (2025). *Spring Boot 3.0 JWT Authentication with Spring Security using MySQL Database*. Retrieved from <https://www.geeksforgeeks.org/springboot/spring-boot-3-0-jwt-authentication-with-spring-security-using-mysql-database/>
19. Xoriant. (2025). *Microservices Security using JWT Authentication Gateway*. Retrieved from <https://www.xoriant.com/blog/microservices-security-using-jwt-authentication-gateway>
20. CodeJava.net. (2025). *User Registration and Login Tutorial*. Retrieved from <https://www.codejava.net/frameworks/spring-boot/user-registration-and-login-tutorial>
21. GitHub. (2025). *Book-My-Show APIs Spring Boot Project*. Retrieved from <https://github.com/Amit8127/Book-My-Show>
22. HelloInterview. (2025). *System Design: Ticketmaster*. Retrieved from <https://www.hellointerview.com/learn/system-design/problem-breakdowns/ticketmaster>
23. GeeksforGeeks. (2025). *Design Movie Ticket Booking System like BookMyShow*. Retrieved from <https://www.geeksforgeeks.org/system-design/design-movie-ticket-booking-system-like-bookmyshow/>
24. Vertabelo. (2025). *A Database Model for a Movie Theater Reservation System*. Retrieved from <https://vertabelo.com/blog/a-database-model-for-a-movie-theater-reservation-system/>
25. Razorpay. (2025). *Payment Gateway Integration in India*. Retrieved from <https://www.cashfree.com/blog/integrate-payment-gateway-india/>
26. SheTechSavant. (2025). *Payment Gateway Integration*. Retrieved from <https://shetechsavant.medium.com/payment-gateway-integration-a711f68ab31f>
27. Tutorialspoint. (2025). *PDFBox - Splitting a PDF Document*. Retrieved from <https://www.tutorialspoint.com/pdfbox/pdfbox_splitting_a_pdf_document.htm>
28. Apache PDFBox. (2025). *Apache PDFBox® - A Java PDF Library*. Retrieved from <https://pdfbox.apache.org/>
29. Spring.io. (2025). *Messaging with STOMP over WebSocket*. Retrieved from <https://spring.io/guides/gs/messaging-stomp-websocket/>
30. GeeksforGeeks. (2025). *Building Real-Time Applications with Java Spring Boot and WebSocket*. Retrieved from <https://www.geeksforgeeks.org/advance-java/building-real-time-applications-with-java-spring-boot-and-websocket/>
31. Microsoft Learn. (2025). *Microservices architecture style*. Retrieved from <https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices>
32. Microservices.io. (2025). *Pattern: Microservice Architecture*. Retrieved from <https://microservices.io/patterns/microservices.html>
33. Postman. (2025). *How to Generate API Documentation using Postman*. Retrieved from <https://www.geeksforgeeks.org/node-js/how-to-generate-api-documentation-using-postman/>
34. DrawSQL. (2025). *Database schema diagrams*. Retrieved from <https://drawsql.app/>
35. dbdiagram.io. (2025). *Database Relationship Diagrams Design Tool*. Retrieved from <https://dbdiagram.io/>