**Mid-Term Report**

Logo, company name

Description automatically generated

**IS621 Agile and DevSecOps**

**Centralised Ticketing System for OCBC Bank**

A diagram of a chatbot

AI-generated content may be incorrect.



1. **PROJECT VISION**
   1. **Problem**

As OCBC expands and adopts a hybrid work model, employees face frustration due to delays and confusion from multiple IT support channels, leading to slow response times of 48-72 hours and unresolved issues affecting productivity. Repetitive IT requests, such as password resets and software installations, overwhelm support staff, further slowing issue resolution. As a result, employee satisfaction with IT support has dropped by 25%, negatively impacting morale and efficiency.

* 1. **Proposed Solution**

We propose a centralized ticketing system integrated with an AI chatbot to streamline IT issue reporting, improve response times, and enhance security and compliance. The chatbot enables AI-powered self-help for common issues, automates ticket categorization, and prioritizes critical problems, reducing IT staff workload. By automating repetitive queries and integrating with collaboration tools, the system ensures seamless issue logging, faster resolution, and improved employee satisfaction.

1. **AGILE PROCESS**
   1. **Kick-Off Meeting**

The kick-off meeting commenced with the team and stakeholders to get buy-in to the MVP timeline, focusing on AI chatbot to enhance ticket registration and resolution. The team structure as shown in Appendix A was established, with clear roles and responsibilities assigned. Our product owner outlined project goals and objectives, emphasizing the mission to deliver an AI-powered ticketing system in five 1-week sprints, with success metrics such as chatbot-driven ticket automation and rapid resolution. The user persona as shown in Appendix B and high-level process flow as shown in Appendix C were reviewed, followed by a discussion on the MVP scope, prioritizing must-have features while debating trade-offs for acceleration. The team aligned on using planning poker (Fibonacci scale) for story estimation. As part of the execution plan, we established a weekly sprint roadmap, tech stack readiness, and risk assessment priorities. JIRA will track sprint progress, completed user stories, and ongoing tasks. The five-sprint timeline was confirmed as feasible. We concluded the meeting with the team’s alignment on immediate action items, assigned ownership, and reinforced the urgency of execution to build lean, iterate fast to maintain the aggressive timeline.

Product Backlog. The backlog consists of two types of user stories:

1. **Functional user stories** - defines the specific features that the Chatbot (Self-service troubleshooting and Chat Escalation) and the centralised ticketing system (including the ticket management, ticket tracking, analytics, and reporting) should deliver from the user’s perspective to address their needs.
2. **Technical user stories** - defines the technical tasks needed to support the features of the product. It focuses on system architecture, performance, and backend components.

User Stories Estimation. The team agreed to use Fibonacci sequence estimation because it provides a structured yet flexible way to estimate effort, particularly for more complex tasks where uncertainty arises. Additionally, we developed a customized scale as shown in **Appendix D** to help us better gauge story points, ensuring consistency and alignment across the team. The team came to a consensus that any story point above 21 must be split into smaller user stories to improve manageability and ensure smoother sprint planning. This exercise helps align assumptions and considerations on user stories and tasks which support team development.

* 1. **Sprint 0**

Sprint 0 focused on setting up the project foundation by defining the overall architecture and system requirements for the centralized ticketing system. The team worked on establishing the necessary development environment, refining the backlog, and preparing for the first development sprint. 4 user stories were identified from the Product Backlog covering 4 epics: **(1) Infrastructure Setup, (2) Version Control Implementation, (3) Continuous Integration/ Continuous Deployment (CI/ CD) Pipeline Setup and (4) Chatbot Framework Selection.** Discussions centered on ensuring the infrastructure was in place, including server configurations, database selections, and API structures. The security framework and compliance requirements essential for operating within the banking environment were finalised. As a team, the acceptance criteria of each user story are agreed upon and the endorsement is received from the Product Owner. At end of Sprint 0, initial prototypes were created, and the backlog was structured to facilitate a smooth transition into development. For Sprint 0 review, the team assessed the completion of foundational tasks, including environment setup, backlog refinement, and chatbot training model initialization.

* 1. **Sprint 1**

A screen shot of a graph

AI-generated content may be incorrect.In Sprint 1 the key objective was to integrate the chatbot's core functionality into the staff portal, covering user stories from the epic of **(1) Chatbot and (2) Ticket Creation.**  **Appendix E** shows the 12 user stories worked on in Sprint 1, and also our Progress dashboard. The team focused on ensuring that employees could interact with the chatbot to raise IT support tickets seamlessly. The process involved creating high fidelity mock-up screens as shown in **Appendix F**, building conversational flows, training the chatbot with commonly reported IT issues, and linking it to the ticketing system for automatic ticket generation. Efforts were also made to refine the chatbot’s response accuracy, enabling it to handle basic troubleshooting without human intervention. Once the acceptance criteria have been defined, Sprint 1 commenced on 19 Feb 2025 (refer to Figure 1 above), and daily standup meetings every morning at 10 AM were done to align on progress, address blockers, and ensure the team stays on track toward sprint goals. At the Sprint 1 review meeting, the team successfully demonstrated the chatbot's integration with the staff portal. Employees could now raise IT tickets, with automated logging occurring at the backend. However, initial tests revealed that the chatbot’s accuracy for VPN troubleshooting was lower than expected.

Figure 1: Sprint 1 burndown chart

* 1. **Retrospective Meeting**

At the end of the sprint, we held a Retrospective meeting to reflect on what went well and identified areas for improvement to enhance our ways of working for the next sprint.

|  |  |  |
| --- | --- | --- |
| 1. **What went well** | 1. **Can be better** | 1. **Action Items** |
| **Clear project vision and sprint goals:**  The project's vision was clearly communicated from the start, with well-defined sprint goals. This helped the team to stay aligned and focused. | **Better User Story Estimation:** Some user stories were underestimated in terms of effort due to the assumptions made. It results in last minute rushes to meet the sprint goal | **Pair Programming:**  To facilitate collaboration, pair programming will commence in the next sprint to allow faster issue resolution, better quality code and more shared knowledge across the team. |
| **Effective Ticket Prioritization on Jira:**  Prioritization on Jira helped the team to focus on the most critical issues first, reducing confusion and streamlined our workflow. | **Increased collaboration:**  The developers worked largely in isolation. This led to more time spent on problem solving. | **Adopt an agile mindset:**  Accept that initial story points estimation in the first few sprints will not be perfect, and be willing to learn together |
| **Successful Integration of the Chatbot:**  The integration within the staff portal was successfully completed on time whereby IT employees could raise their IT issues with a linked ticket at the backend. |  | **Frequent check ins:**  Keep updaily stand-ups/sprint reviews to ensure the whole team remains on the same page |
| **Early involvement of testers:**  Testers were involved from sprint 0, and testing was able to happen early and often. Even as development was ongoing, testing cases were being prepared, and continued as features were added. This allowed efficient and effective testing. |  |  |

1. **TECHNICAL DETAILS**

A diagram of a computer

AI-generated content may be incorrect.

Figure 2: High-level architecture diagram of centralized ticketing system

Architecture. The team proposes to use **microservices architecture** to enhance modularity, scalability and maintainability. As illustrated in **Appendix G-1**, each service such as the Chatbot, Ticketing, Notification, User/Auth, Search, Admin and Analytics functions independently, enabling seamless updates, faster deployments and fault isolation. By leveraging dedicated databases (i.e. MongoDB, PostgreSQL, MariaDB and Elasticsearch), we can optimize data management and performance. Additionally, the API Gateway will serve as a centralized entry point, improving efficiency and security. This loosely coupled architecture provides flexibility which allows teams to deploy, develop and scale services independently, ensuring a robust and future-proof system. We are adopting **on-prem infrastructure** for enhanced data security, regulatory compliance, and seamless integration with legacy banking systems. This approach ensures data sovereignty, long-term cost control, and keeps sensitive financial data within the bank’s physical control, addressing residency concerns crucial to the banking sector. We adopted X and Y-scaling to duplicate our services and decompose our microservices. With this, we can handle increased load efficiency, high scalability and maintain high availability while enabling rapid development and deployment of new features on the go.

Tools Used. We used the following tools: (1) **JIRA Service Management** for IT Ticketing functions along with problem, incident management and change management tools (2) **Spring Boot** (Java) for most backend microservices. (3) **OpenAI API or other LLM models** for Chatbot Service. (4) **Microsoft Teams API** for connection with SMTP servers for notifications via SMS/Email/chat. (5) **React.js/Node.js** for web/mobile UI. Relational databases (**MariaDB/PostgreSQL/MySQL**) & Non-Relational databases (**NoSQL, MongoDB**). (6) **Elasticsearch** for quick searches, (7) **Kibana** for visualization and log management, (8) **OpenShift (Enterprise Kubernetes by Red Hat)** Built on Kubernetes but with added security, monitoring and CI/CD tools for containerization, (9) **Git with GitHub or GitLab** for Version Control and (10) **Jenkins/GitLab CI** for CI/CD Platform.

Security Considerations. Confidentiality, Integrity and Availability are integrated throughout our microservices design to ensure reliable and secure operations. In overall, **Confidentiality** is at risk due to unauthorized access to sensitive data, API interception, and insider threats targeting restricted services like Admin services/UserAuth. Encryption protocols (i.e. TLS) will be used for secure communications, role-based access control (RBAC) to be enforced and access logs to be regularly audited to detect anomalies. Critical data to protect includes user credentials, chatbot interactions, authentication tokens and business analytics data. **Integrity** can be compromised by data tampering of Ticketing, Notification and Admin services, SQL injection and unauthorized modifications to APIs or our microservices. To prevent this, input validation, cryptographic hashing for data integrity and code signing for service updates as essential data (i.e. ticketing records, admin settings and notification logs) must remain accurate. **Availability** threats such as DDoS, ransomware and hardware failures can disrupt services. Ensuring system resilience requires firewalls on redundancy, load balancers, failover strategies, rate limiting, anti-DDoS protections and regular database backups. More information about the security considerations for each of the technical architecture components can be found in **Appendix G-2 and G-3**. Further elaborations on backup integrity, offsite backup security, backup retention policies and disaster recovery testing are shown at **Appendix H**.

Presentation Tier. Our JIRA centralized ticketing system UI allows different roles to view their dashboards as explained below:

|  |  |
| --- | --- |
| **User Interface (UI) Types** | **Features** |
| End User Portal | Web/mobile interface where users can easily interact with chatbots, submit tickets and view tickets. For user login, MFA with biometrics to be added for privacy assurance |
| IT Admin Dashboard | Centralized interface for IT admin to manage, prioritize, resolve tickets, with ability to analyze trends, filtering and reporting |
| Chatbot Interface | Interactive AI module integrated within the user portal, to allow the chatbot to answer queries and provide IT assistance |

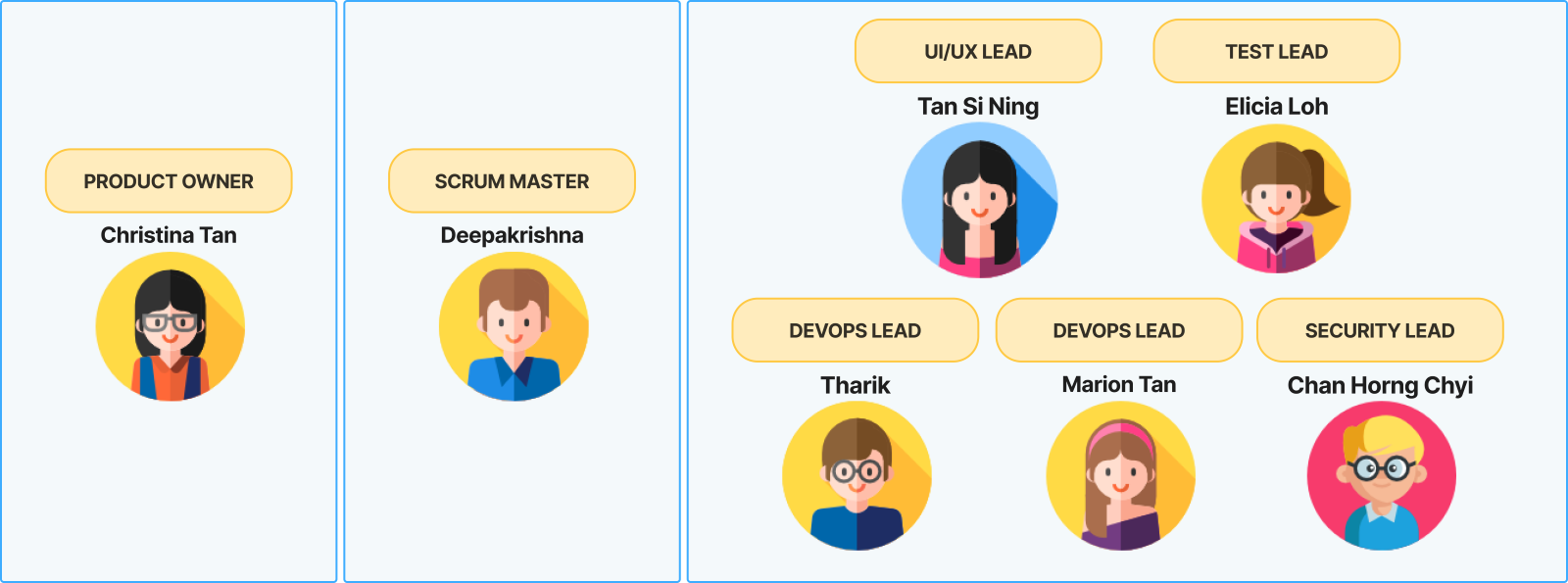
Business Logic Tier. The business logic tier comprises of four key components (1) The **Ticket Management Engine** handles the lifecycle of support tickets, including creation, updates, deletion, and workflow integration for routing or escalation. (2) The **Chatbot Engine** uses NLP to interpret user queries, providing automated responses or escalating when necessary. (3) The **Integration Layer** connects with external systems such as email, notifications, and data exchange platforms to facilitate seamless communication. (4) Lastly, the **Business Logic Modules** enforce authentication, authorization, and Role-Based Control (RBC) to manage secure access for different users within the portal.

Data Tier. In our Data Tier, a **Relational Database** stores structured data, including user profiles, ticket details, history, and admin actions, while a **Non-Relational Database** handles unstructured data such as user queries and conversation logs. Elasticsearch is used for log searching and analytics. Data Storage & Management ensures security through encryption, regular backups, and data recovery protocols to maintain integrity. Secure APIs exposes endpoints for the logic tier to access data, and an Event-Driven Architecture like Kafka synchronizes related data across services in real time.

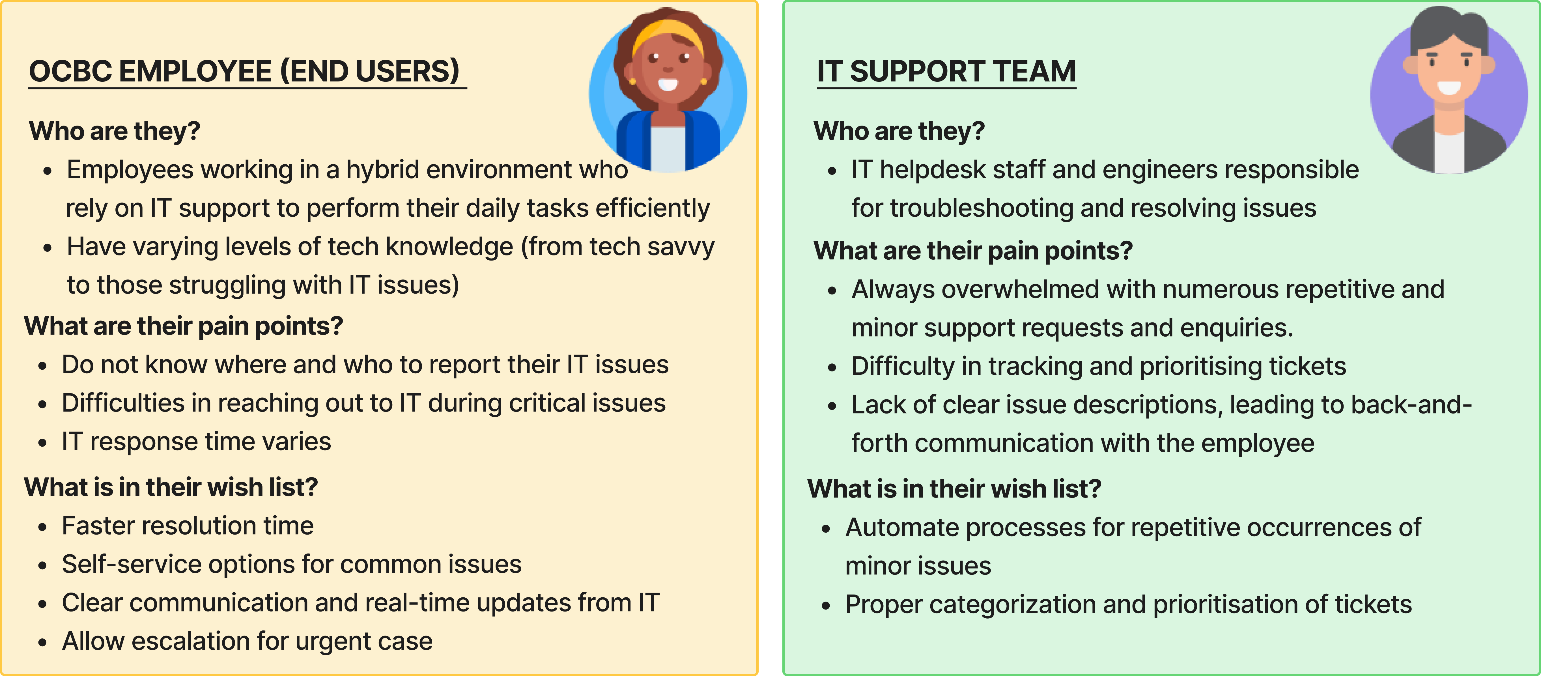
Deployment. The deployment of our system will leverage a CI/CD pipeline to ensure security, efficiency and compliance. Developers commit code to a Git-based repository, triggering an automated CI process via tools. The system undergoes build, static code analysis, and automated testing (unit, integration, and functional) before creating deployable artifacts stored in Nexus or Artifactory. Staging deployment allows further testing and manual/automated approval gates ensure regulatory compliance before deployment to production using Blue-Green or Canary strategies. Post-deployment, real-time monitoring and feedback mechanisms ensure system stability and continuous improvement. This automated, secure, and scalable deployment method will enhance operational efficiency, reduce downtime, and align with banking compliance standards. Please see **Appendix I for more details.**

**APPENDIX**

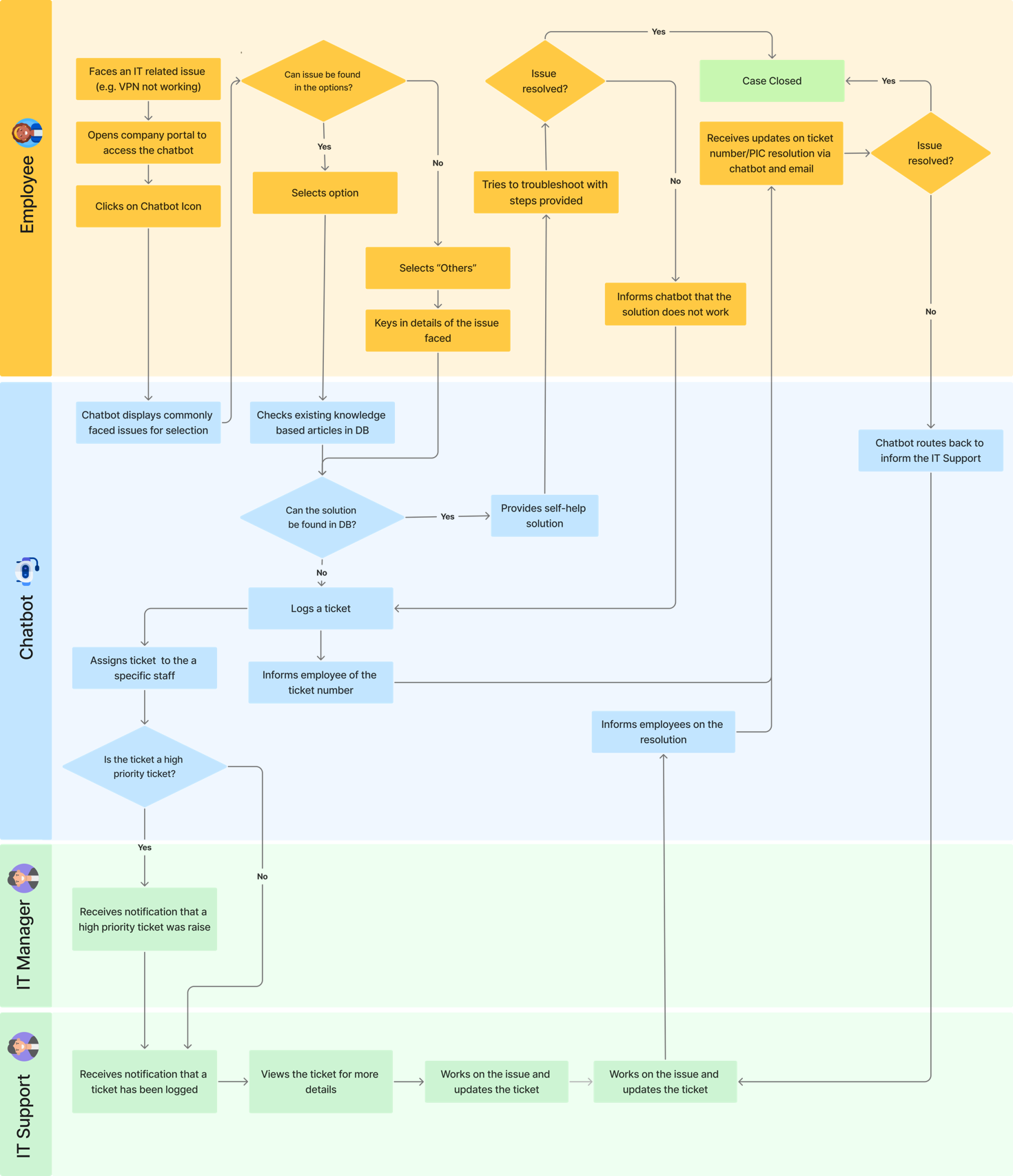
**Appendix A – Team Structure**



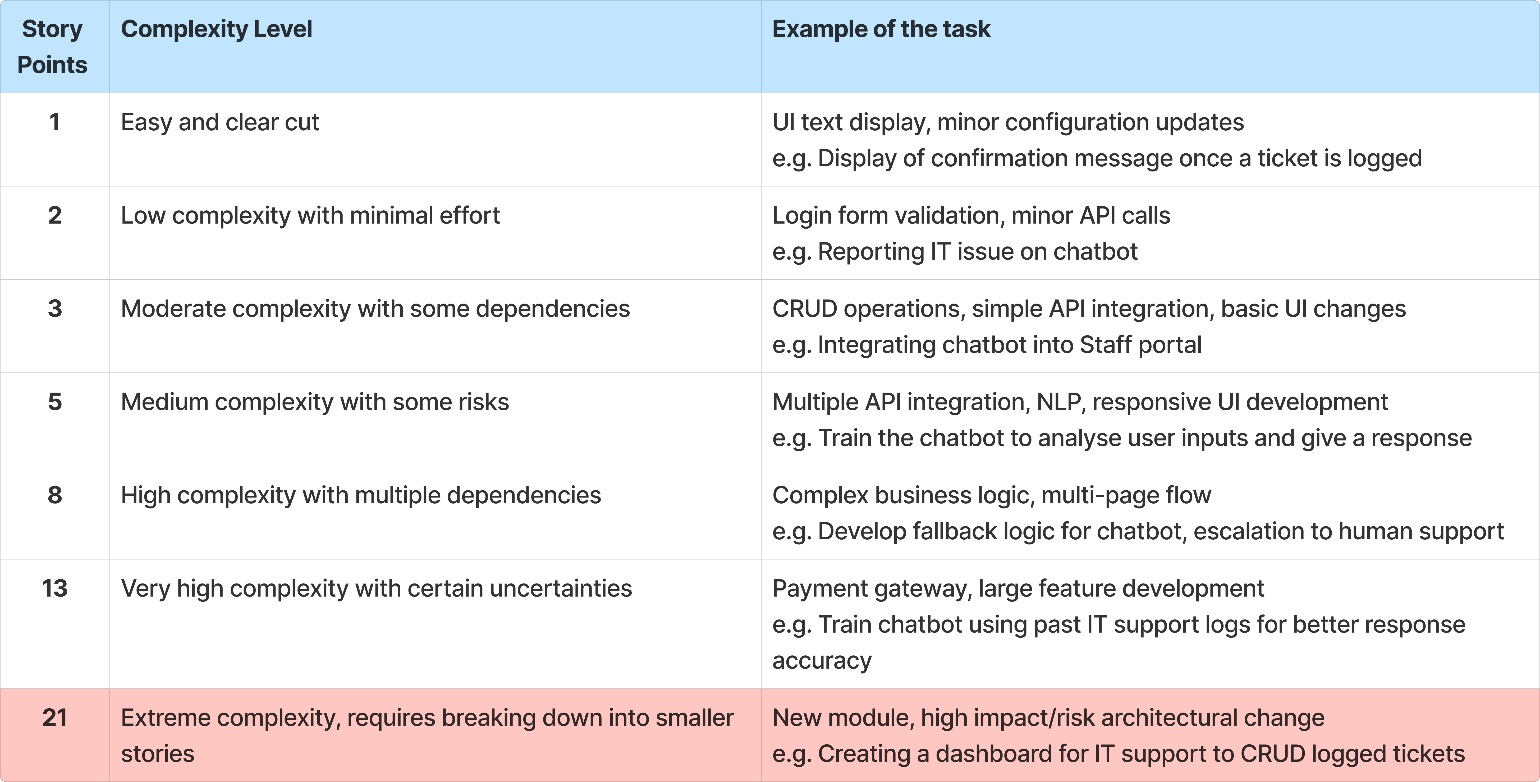
**Appendix B – User Personas**



**Appendix C – Chatbot Process flow**



**Appendix D – Fibonacci customized scale**



**Appendix E - User Stories Backlog**

A screenshot of a computer

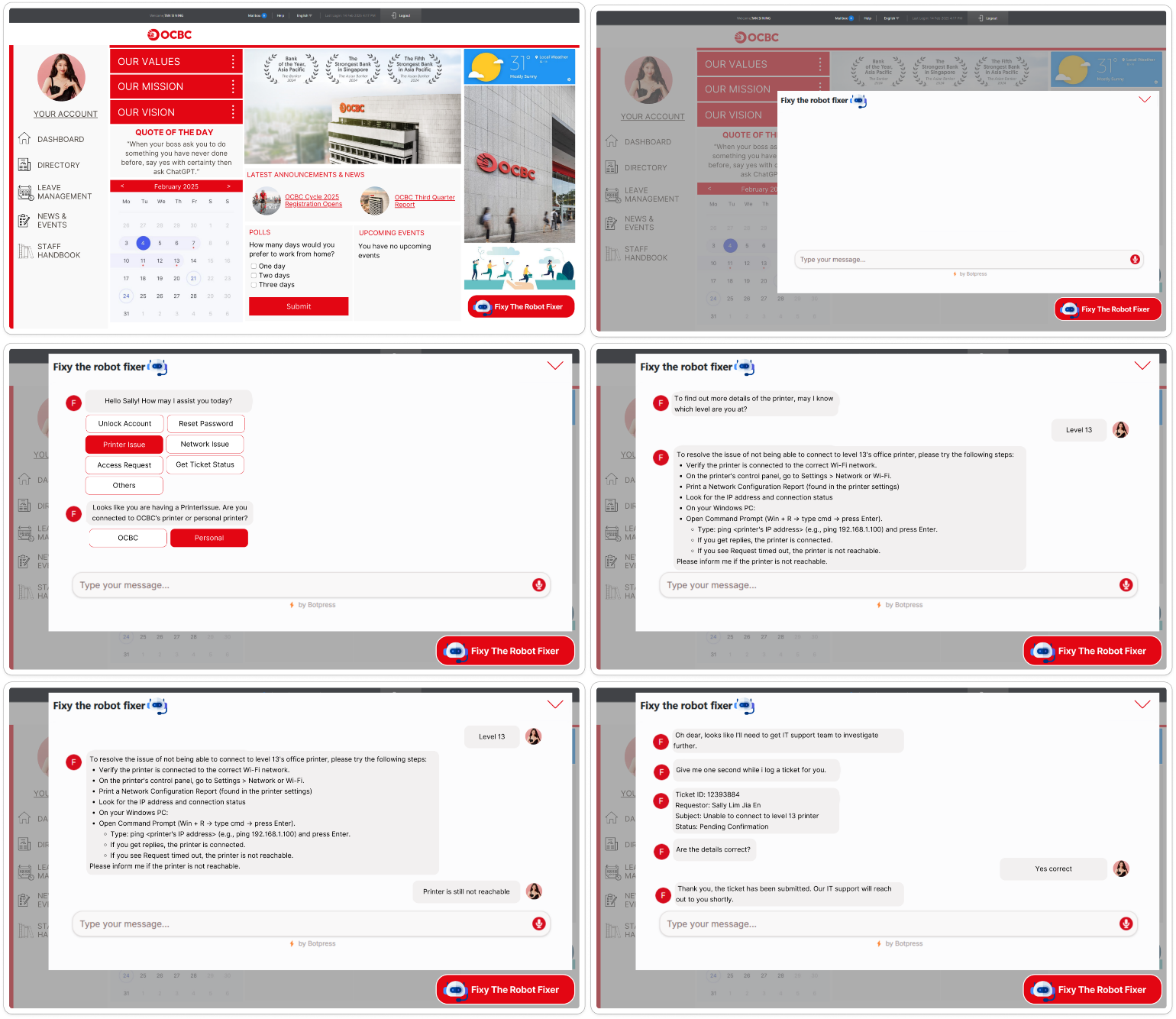
AI-generated content may be incorrect.

**Progress Dashboard**

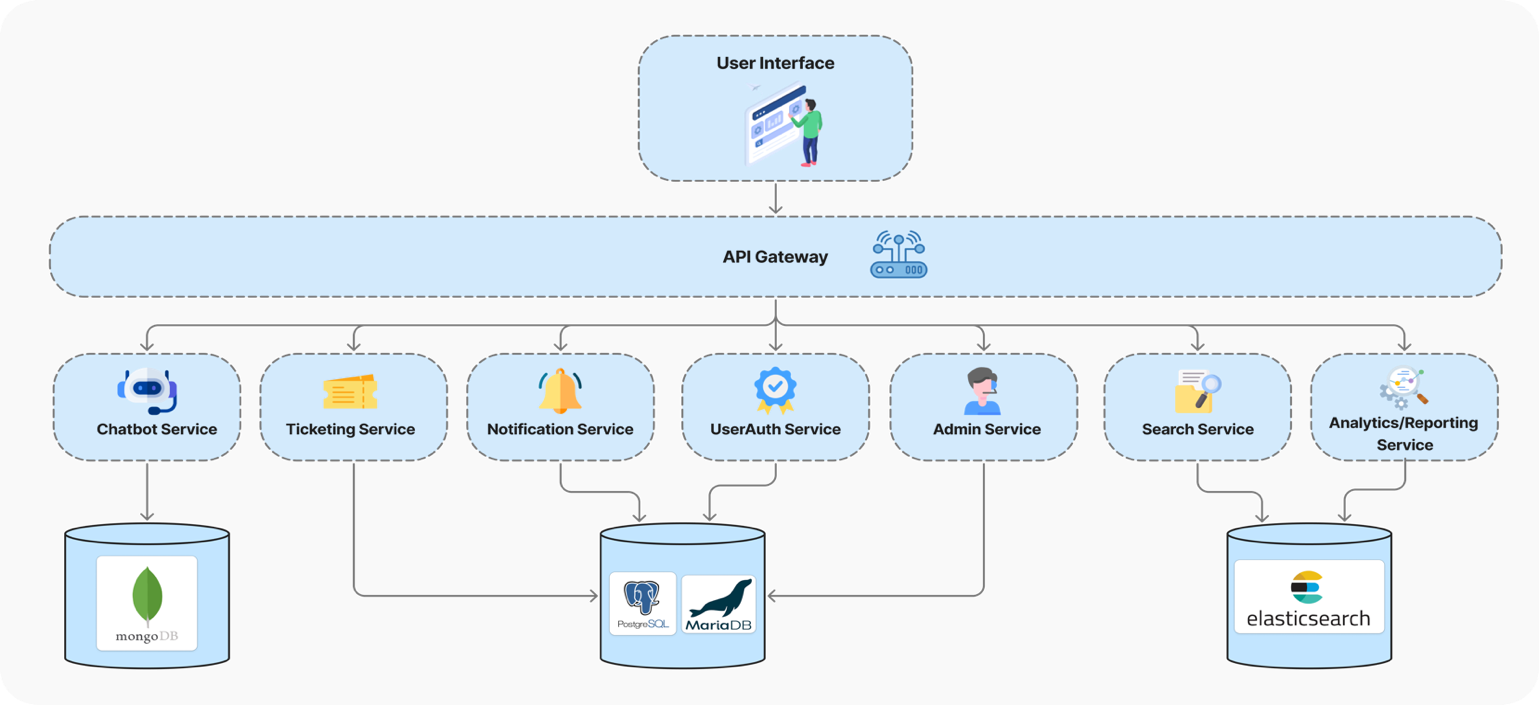
A screenshot of a computer

AI-generated content may be incorrect.

**Appendix F – High Fidelity Mock-Up Screens**



**Appendix G-1 – High-Level Architecture**



**Appendix G-2 - Architecture Details**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Service/**  **Component** | **Purpose** | **Key Technologies** | **Key Responsibilities** | **Data Storage** | **Security Considerations** |
| **User Interface (UI)** | Front-end layer where users interact with the system (web/mobile). | React, Angular, or Vue; HTML/CSS/JS | Render pages and handle user input; Communicate with API Gateway; Enforce basic security (HTTPS) | N/A (Primarily front-end) | **Confidentiality:** Enforce strong authentication & authorization (MFA, RBAC) to prevent unauthorized access  **Integrity:** Input validation & sanitization to prevent injection attacks (XSS, SQL Injection) that can alter data  **Availability:** Secure session management (timeouts, secure cookies) and DDoS protection to ensure UI remains accessible |
| **API Gateway** | Entry point for all requests; handles routing, load balancing, and security. | Kong, NGINX, AWS API Gateway; JWT/OAuth2 | Route requests to microservices; Enforce authentication/authorization; Rate limiting, logging, and monitoring | N/A (Stateless, but logs to a database) | **Confidentiality:** Encrypt data in transit using TLS/SSL and secure API authentication via OAuth2, JWT)  **Integrity:** Implement digital signatures for API requests to prevent data tampering  **Availability:** Rate limiting & throttling prevent abuse and DDoS protection keeps APIs operational |
| **Chatbot Service** | Processes user queries via NLU, dialog management, and response generation. | Rasa, spaCy, or Hugging Face; Python/Node.js | Interpret user intent; Maintain conversation flow/context; Generate appropriate responses | MongoDB (for conversation data, KB) | **Confidentiality:** Access control ensures chat logs with banking details are restricted  **Integrity:** Input sanitization prevents chatbot exploitation through script injection  **Availability:** Implement load balancing and failover mechanisms to keep the chatbot responsive |
| **Ticketing Service** | Manages support tickets (creation, updates, status tracking). | Node.js/Java/Python; RESTful APIs | Create/update/close tickets; Assign tickets to agents; Track ticket lifecycle | PostgreSQL (relational ticket data) | **Confidentiality:** Encryption of tickets at rest and in transit ensures only authorized personnel can access them  **Integrity:** Audit Trails track modifications to tickets, ensuring changes are legitimate  **Availability:** Redundant storage and failover strategies ensure tickets are always accessible |
| **Notification Service** | Sends alerts (email, SMS, push) triggered by various events. | Twilio, SendGrid, Firebase; Node.js/Java | Format and dispatch notifications; Manage user preferences; Integrate with message queues for async handling | MySQL (templates, logs, preferences) | **Confidentiality:** Avoid exposing sensitive data to notifications (e.g. masking customer details in emails)  **Integrity:** Ensure message authenticity through cryptographic signatures  **Availability:** Implement rate limiting to prevent spam and queue-based processing to handle peak loads |
| **UserAuth Service** | Central authentication/authorization; issues and validates tokens. | OAuth2/OpenID Connect; JWT-based auth | Handle user registration/login; Manage roles and permissions; Securely store credentials | Could use PostgreSQL or MySQL | **Confidentiality**: MFA, password hashing (bcrypt), and encrypted authentication tokens such as JWT or OAuth2 to secure user credentials  **Integrity:** Account lockout policies & anomaly detection prevent brute force attacks  **Availability:** Implement distributed authentication mechanism to prevent downtime |
| **Admin Service** | Provides administrative functions (user management, system config, monitoring). | Node.js/Java/Python; Admin dashboard UI | Manage user roles/permissions; Adjust service configurations; Monitor system status | MariaDB (admin logs, config) | **Confidentiality:** RBAC & least privilege principle ensure admin accounts are highly secure  **Integrity:** Logging & auditing track admin changes to prevent unauthorized modifications  **Availability:** Ensure redundant admin access paths with failover authenticationmechanisms |
| **Search Service** | Enables fast text search and indexing across data sources. | Elasticsearch | Index data for quick lookup; Provide relevancy-based search; Serve queries from Chatbot/Ticketing | Elasticsearch | **Confidentiality: RBAC & masking sensitive search results prevent unauthorized access**  **Integrity: Secure queries to prevent search injection attacks**  **Availability:** Index replication ensures search functions remain available during failures |
| **Analytics/Reporting** | Collects data for insights, metrics, and reporting dashboards. | Elasticsearch or data warehouse; Kibana/Grafana for visualization | Aggregate logs and metrics; Generate performance and usage reports; Dashboard visualization | Elasticsearch or data warehouse | **Confidentiality:** Data anonymization ensures reports do not expose sensitive user data  **Integrity:** Hashing & digital signatures verify report authenticity  **Availability:** Load balancing & caching mechanisms ensure timely access to analytics |

**Appendix G-3 - Database Security Considerations**

|  |  |
| --- | --- |
| **MongoDB (Chatbot and Other Services)** | **Confidentiality:** RBAC & TLS encryption can protect sensitive data.  **Integrity:** Data validation & NoSQL injection prevention ensures data consistency.  **Availability:** Sharding & replication prevent data loss and ensure high availability. |
| **PostgreSQL (Ticketing and UserAuth)** | **Confidentiality:** Data encryption at rest & role-based access controls secure records.  **Integrity:** Data encryption at rest & role-based access controls secure records.  **Availability:** Database replication & backup strategies ensure resilience. |
| **Elasticsearch (Search and Analytics)** | **Confidentiality:** TLS encryption & access control prevent unauthorized data access.  **Integrity:** Data hashing & audit logs ensure search index consistency.  **Availability:** Clustered deployments with failover nodes keep search operational. |

**Appendix H – Backup Strategy with RPO and RTO**

|  |  |  |
| --- | --- | --- |
| **Component** | **RTO (Max Recovery Time)** | **RPO (Max Data Loss)** |
| User Database | 1 hour | 15 minutes |
| Chatbot Service | 2 hours | 30 minutes |
| Ticketing System | 30 minutes | 10 minutes |
| Notification System | 2 hours | 1 hour |
| API Gateway | 5 minutes | Immediate failover |

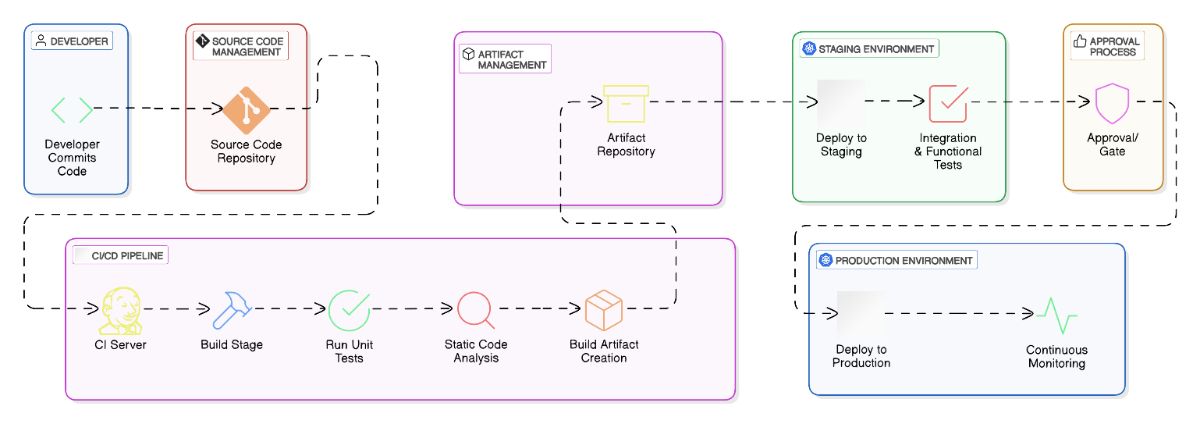
Proposed Backup Strategy:

- Real-time database replication between primary and secondary servers

- Daily full backups with hourly incremental backups

- Offsite backup storage for disaster recovery

**Appendix I – CI/CD Pipeline Diagram**



**CI/CD Pipeline Tools & Their Purpose**

A diagram of software development and deployment workflow

AI-generated content may be incorrect.

**Appendix J – Sprint 1 Burndown Chart**

