Intertech Bilgi İşlem ve Paz. Tic. AŞ.

Sanayi Mah. Teknopark Blv. No:1/3C
Kurtköy – Pendik / İstanbul

It was founded in 1987 as a software company with 43 employees. In 2002, it joined the DenizBank DFHG body. In 2006, it joined the DEXIA group. In 2012, it joined the SBERBANK group. In 2019, it joined the Emirates NBD group.

Intertech's vision is "To be number 1 in its region in financial technology products and services," and its mission is "To provide indispensable experiences." With over 35 years of experience and more than 1600 expert employees, its 100+ products are used by more than 50 financial institutions in 12 countries, primarily in Turkey, as well as in the Europe and MEA regions.

Its parent organization is Denizbank.

Current Facilities:
Intertech Ar-Ge Merkezi, Intertech Yerinde Ar-Ge Şubesi, Intertech Vadi İstanbul Şubesi, Intertech Bilkent CYBERPARK Şubesi, Intertech İzmir Şubesi, Intertech Viyana Şubesi

Number of employees: 1600+

Technologies I Used During My Internship

Hardware

During the internship, I used a high-performance laptop allocated by the company to develop the project, participate in meetings, and conduct research. This computer possessed sufficient RAM and processing power to run multiple development tools, containers, and test servers simultaneously without issue.

Software

In the project development process, I used various software, technologies, and platforms to build the backend infrastructure for a banking chatbot.

1. Programming Languages and Frameworks

    Python: This was the main programming language for the project. All backend services, business logic, and APIs were written using Python.

FastAPI: A high-performance, asynchronous web framework used to develop the MCP (Model Context Protocol) server and the RESTful APIs provided to the frontend team. It was chosen for its modern structure and automatic documentation (Swagger) features.

Pydantic: Used integrally with FastAPI to define API request and response data models and to guarantee the validation of this data.

2. Database

SQLite: A simple, serverless, file-based relational database management system used to store the project's conversation logs and user feedback. It was chosen for its rapid setup and simplicity.

3. Containerization and Deployment

Docker: Used for "imagization" packaging the developed backend application (the "MCP Server") along with all its dependencies into a standard and isolated environment. This eliminated the "it worked on my machine" problem and ensured the project could be easily moved to different environments (e.g., testing, production).

4. Protocols and Specifications

Model Context Protocol (MCP): This was less a piece of software and more a foundational design specification for the project. It defined the data structure the backend would use to communicate with the AI model and how it would manage context and external tools. The server I developed was the main component that implemented this protocol.

5. Development Tools and Utilities

Git: Used as the project's version control system. It was the primary tool for tracking code changes, working on branches, and merging code with the teams (AI, Frontend).

Visual Studio Code (VS Code): My main code editor (IDE) for writing code, debugging, and performing terminal operations.

Postman: Actively used to test the API endpoints I developed, send requests for various scenarios, and verify the correctness of the responses.

Pytest: Used to write unit tests to confirm that the business logic functions I wrote in the service layer (e.g., data masking, tool calling) worked as expected.

Swagger (OpenAPI): Through this interface provided by FastAPI, I interactively documented the APIs I developed, allowing the frontend team to use this documentation live.

6. Communication and Project Management

Jira: Used to track my tasks within two-week sprints, monitor project progress, and report on completed work as part of the Agile methodology.

Microsoft Teams: The corporate communication platform used for daily stand-up meetings, instant messaging, and screen sharing with the Frontend, AI, BI teams.

During my 6-week internship, I was fully integrated into the AI department and assigned to a single, high-priority project: the Banking Chatbot Project.

Project Description

The primary goal of this project was to build the complete backend infrastructure for a new, intelligent banking chatbot. This backend server acted as the central "brain," responsible for:

1. Receiving messages from bank customers via a RESTful API.
2. Managing the conversation's state and security.
3. Communicating with the core AI model using a custom-built "Model Context Protocol" (MCP).
4. Connecting to other (simulated) bank services to fetch live data.

Assigned Tasks and Responsibilities

As a Backend Developer Intern, my responsibilities covered the entire development lifecycle of this backend server. My specific given tasks included:

MCP Server Skeleton Setup: Building the foundational structure of the server, including the project layout, configurations, and API routers using FastAPI.

Frontend API Development: I was responsible for designing, implementing, and documenting (with Swagger) the RESTful API endpoints that the frontend and client-server teams would use to send messages and receive responses.

Business Logic Implementation: Writing the core logic functions within the service layer. This involved processing incoming requests, managing user sessions, and converting data into the required MCP format for the AI model.

MCP Tool Creation: I developed a key "MCP Tool" to give the chatbot external capabilities. This tool allowed the AI model to request real-time data (e.g., current credit interest rates) by triggering a function call on the backend, which would then fetch the data.

Database Implementation: I was tasked with creating a simple database using SQLite and SQLAlchemy. I designed the schema and wrote the CRUD operations to log all conversation histories and store user feedback for future analysis.

Security & Validation: Implementing a critical data masking module to filter and hide sensitive customer data (like IBANs) before it was processed by the AI model.

Docker Containerization: In the final week, I was responsible for writing the Dockerfile for the application, enabling the "imagization" of the server for easy deployment and portability.

Teams and Collaboration

This project required constant communication and collaboration with multiple specialized teams:

Project Team (Backend): I worked under the supervision of a senior backend developer, participating in daily stand-up meetings, code reviews, and sprint planning sessions as part of the core development team.

Frontend Team: I held regular meetings with the frontend developers. In these meetings, I presented the API contract, gathered their feedback, and ensured they had the mock data and documentation needed for integration.

AI Team: I collaborated closely with the AI team. Our meetings were focused on defining and testing the MCP specification, ensuring that the data packets my server constructed were correctly understood by the model, and verifying that the "tools" I built were triggered properly.

During the internship, our team encountered a significant disruption related to a critical project dependency, which required us to adapt quickly and find an effective solution.

The Disruption: Change in Upstream Dependency

Initially, the project's architecture was designed with the expectation that an upstream API would be provided by another team. This API was intended to serve all the necessary data (e.g., product information, user data) that our chatbot backend would require. Based on this plan, our team's backend server and its data access layer were developed to consume this specific, external API.

However, at a late stage in our development sprint, it was communicated that the upstream API would not be delivered as planned. This left our application without a data source and put our project timeline at risk, as a significant portion of our code was now dependent on a non-existent component.

The Solution: Rapid Pivoting and Local Database Implementation

Faced with this challenge, our team had to pivot quickly to unblock our development and meet our deadlines. The immediate solution was to create our own local, persistent data source.

1. Immediate Action: We decided to implement a simple database using SQLite. This choice was made due to its rapid, serverless setup and seamless integration with Python, which allowed us to get a new data source running in a very short amount of time.
2. Refactoring: Our team worked collectively to refactor the application's data access layer, changing the logic to query this new SQLite database instead of making external API calls.
3. A Fortunate Design Choice: This entire transition was made significantly easier due to a good design principle we had implemented from the beginning: a mock_data flag.

Early in development, we had used this boolean flag to switch between the (then-unavailable) upstream API and a local mock data file. This existing structure proved invaluable; we were able to simply adapt this "mock" data path to point to our new SQLite database service, rather than refactoring the entire application.

This experience taught me the importance of anticipating dependency failures and writing decoupled code, as our initial mock_data flag saved us considerable time and allowed us to resolve a critical blocker efficiently.

This internship provided significant practical skills in software engineering. My key professional gains include:

Technical Skills:

Backend Development: Gained proficiency in building a complete, production-style server from scratch using Python and FastAPI.

API Design: Acquired hands-on experience in designing, implementing, and documenting secure RESTful APIs (using Swagger) for frontend integration.

Database Management: Learned to design a database schema and perform CRUD operations using SQLite and the SQLAlchemy ORM.

DevOps: Gained practical experience in containerizing a web application and its dependencies using Docker.

Protocol Implementation: Developed the ability to understand and implement a complex, custom technical specification like the Model Context Protocol (MCP).

Professional Skills:

Agile Methodology: Learned to work effectively within an Agile (Scrum) environment, participating in daily stand-ups, sprints, and task management.

Collaborative Development: Became proficient in using Git for version control in a team setting, including managing branches and participating in code reviews.

Cross-Functional Communication: Developed strong communication skills by collaborating directly with separate Frontend and AI teams to resolve dependencies.

Problem-Solving: Proved my ability to adapt to major, unexpected project changes and implement a robust solution under pressure.

This internship was an invaluable experience that successfully bridged the gap between academic theory and real-world software engineering. I was entrusted with significant, hands-on responsibilities on a challenging and modern project—building a backend server for an AI-powered banking chatbot. The opportunity to develop a complete application from the ground up, including designing APIs (FastAPI), implementing a database (SQLite), and containerizing the service (Docker), was exceptional. Furthermore, collaborating daily with specialized AI and Frontend teams within an Agile framework, and successfully navigating real-world disruptions like the unexpected API failure, has given me practical skills and a level of confidence that classroom learning alone could not provide.