CSC4311: Lecture Note 2

Application Programming Interface (API)

An API, or application programming interface, is a set of rules or protocols that let software applications communicate with each other to exchange data, features and functionality.

APIs simplify application development by allowing developers to integrate data, services and capabilities from other applications, instead of developing them from scratch. APIs also give application owners a simple, secure way to make their application data and functionality available to internal departments within their organizations. Application owners can also share or market that data and functionality to business partners or third parties.

How an API works

A simple way to understand how APIs work is to look at a common example—third-party payment processing. When a user purchases a product on an ecommerce site, they may be prompted to "Pay with Card" or another type of third-party system. This function relies on APIs to make the connection.

- When the buyer clicks the payment button, an API calls to retrieve information—also known as a request. This request is processed from an application to the web server through the API's Uniform Resource Identifier (URI) and includes a request verb, headers, and sometimes, a request body.
- After receiving a valid request from the product webpage, the API makes a call to the external program or web server, in this case, the third-party payment system.
- The server sends a response to the API with the requested information.
- The API transfers the data to the initial requesting application, here the product website.

While the data transfer will differ depending on the web service being used, the requests and responses all happen through an API. There is no visibility on the user interface, meaning APIs exchange data within the computer or application, and appear to the user as a seamless connection.

API benefits

APIs simplify design and development of new applications and services, and integration and management of existing ones. But they offer other significant benefits to developers and organizations at large.

Improved collaboration

APIs enable integration between applications. Without APIs, many enterprises would lack connectivity, causing information silos that compromise productivity and performance.

Accelerated innovation

APIs offer flexibility, allowing companies to make connections with new business partners and offer new services to their existing market. This flexibility enables companies to, ultimately, access new markets that can generate massive returns and drive digital transformation.

Data monetization

Many companies choose to offer APIs for free, at least initially, so that they can build an audience of developers around their brand and forge relationships with potential business partners. If the API grants access to valuable digital assets, the businesses monetize it by selling access. This practice is referred to as the API economy. For Instance, when AccuWeather launched its self-service developer portal to sell a wide range of API packages, it took just 10 months to attract 24,000 developers, selling 11,000 API keys. This move helped to build a thriving community in the process.

System security

APIs separate the requesting application from the infrastructure of the responding service, and offer layers of security between the two as they communicate. For example, API calls typically require authentication credentials. HTTP headers, cookies or query strings can provide additional security during data exchange. And an API gateway can control access to further minimize security threats.

API protocols

As the use of web APIs has increased, it has lead to the development of certain protocols. These protocols provide users with a set of defined rules, or API specifications that create accepted data types commands and syntax. In effect, these API protocols facilitate standardized information exchange.

- **Simple Object Access Protocol (SOAP):** Built with XML, SOAP enables endpoints to send and receive data through SMTP and HTTP. SOAP APIs make it easier to share information between apps or software components that are running in different environments or written in different languages.
- **XML-Remote Procedure Call (XML-RPC):** The XML-RPC protocol relies on a specific XML format to transfer data. XML-RPC is older than SOAP, but much simpler, and relatively lightweight in that it uses minimum bandwidth.
- **JSON-RPC:** Like XML-RPC, JSON-RPC is a remote procedure call, but JSON (JavaScript Object Notation) is used instead of XML to transfer the data.
- **Representational State Transfer (REST):** REST is a set of web API architecture principles. REST APIs—also known as a RESTful API)—are APIs that adhere to certain REST architectural constraints. It’s possible to build RESTful APIs with SOAP protocols, but the two standards are usually viewed as competing specifications.

Software Development Environment

A software development environment, often referred to as Integrated Development Environment (IDE), is a comprehensive set of tools, processes, and frameworks that developers use to create software applications. It encompasses everything needed for software development, from writing and testing code to debugging and deploying applications.

Key components of a software development environment typically include:

- **Code Editor:** A text editor with features tailored for writing code, such as syntax highlighting, code completion, and code formatting.
- **Compiler/Interpreter**: Software that translates source code into machine code or bytecode that can be executed by a computer.
- **Debugger:** Tools for identifying and fixing errors (bugs) in the code by allowing developers to step through code execution, inspect variables, and analyze program behavior.
- **Version Control System (VCS):** Software for tracking changes to code over time, enabling collaboration among developers, and facilitating code management.
- **Build Automation Tools:** Utilities for automating the process of compiling, testing, and deploying software, streamlining development workflows.
- **Testing Frameworks**: Frameworks and libraries for writing and executing automated tests to ensure the reliability and quality of software.
- **Integrated Documentation:** Tools for generating and managing documentation directly within the development environment, ensuring that codebases remain well-documented.

Several IDEs are in use today and include Eclipse, IntelliJ Idea, BLUEJ, JCreator, J Developer, Android Studio, and NetBeans.

Class Browsers

Most modern IDEs for object-oriented implementation has a Class Browser as one of their features. The *Class Browser* provides a comprehensive look at all the classes defined to or used by your application. It also lets you browse the methods and properties of external class objects and those of system and user class objects.

Software Requirement Engineering

Software Requirement Engineering is the systematic process of gathering, documenting, analyzing, and managing requirements for software systems. It is a crucial phase in the software development lifecycle as it lays the foundation for building a software product that meets the needs and expectations of its stakeholders.

The main objectives of software requirement engineering include:

Understanding Stakeholder Needs: Identifying and comprehending the requirements of all stakeholders involved in the software development process, including end-users, customers, and managers.

Eliciting Requirements: Employing various techniques such as interviews, surveys, workshops, and observations to extract requirements from stakeholders and understand their needs, preferences, and constraints.

Analyzing Requirements: Examining and prioritizing requirements to ensure clarity, consistency, completeness, and feasibility. This involves resolving conflicts and ambiguities and defining the scope of the software project.

Documenting Requirements: Creating comprehensive documentation, such as requirement specifications, use cases, user stories, and diagrams, to capture and communicate the requirements effectively to all stakeholders.

Validating Requirements: Verifying that the documented requirements accurately reflect the stakeholders' needs and are aligned with the overall objectives of the software project

Managing Requirements: Establishing a systematic approach for managing changes to requirements throughout the software development lifecycle. This includes tracking changes, maintaining traceability, and ensuring that requirements remain consistent and up-to-date.

Software Tools

Software tools refer to programs or applications designed to assist users in performing specific tasks related to software development. Software tools include the following:

- Requirements Analysis and Design tools such as UML, IBM Rational Software Architect, and Data modelling tools.
- Testing tools: Unit Testing tools, Stress/Load Testing tool, API Testing tools, and Security testing tools

Tool integration mechanism

Tool integration mechanisms refer to the methods and approaches used to connect and synchronize different software tools within a software development environment. Integrating tools enables seamless data flow, collaboration, and automation across various stages of the software development lifecycle. Some common mechanisms for tool integration include: APIs Webhook, Plug-ins and Extensions