Lecture 13

Data Storage and Interface Design

BITP 2223 Software Requirements and Design

Introduction to Data Storage Design

Where are we in the requirement engineering process

Data Storage Design

Trends

In today's data-driven world, organizations rely heavily on the ability to store and access information effectively.

Position

Data serves as the foundation for critical operations, decision-making, and innovation.

Role

Therefore, data storage design plays a pivotal role in establishing this critical foundation.

Assurance

Data storage design is pivotal to ensure accessibility, reliability, scalability, and security of the software.

Accessible Data can be retrieved and modified efficiently when needed by the application.

Reliable Data integrity is maintained, meaning it's accurate and hasn't been corrupted.

Scalable The storage solution can accommodate growth in data volume and complexity over time.

Secure Data is protected from unauthorized access, modification, or deletion.

Data Storage Design Framework

Key Elements of Data Storage Design

Key Elements of Data Storage Design Framework

1. Data Analysis

2. Storage Selection

3. Data Modelling

4. Security Measures

5. Performance Optimization

6. Documentation

1st Key Element of Data Storage Design Framework

Data Analysis[1]

Data analysis forms the foundation of effective data storage design.

It involves a thorough analysis of the data characteristics the application will manage.

The analysis helps determine the most suitable storage solution and optimize its performance.

1. Identify Data Types

Identify the types of data the application will store – structured, unstructured or semi-structured.

a) Structured Data

A structured data has a well-defined format and organisation.

It often fits neatly into tables with rows and columns, where each column represents a specific attribute

Example: customer names, product IDs, purchase dates in an e-commerce database

b) Unstructured Data

An unstructured data lacks a predefined format

Example: documents, emails, images, audio, and video files.

c) Semi-structured Data

A semi-structured data has some internal structure but doesn't conform to a strict tabular format.

Examples: JSON (JavaScript Object Notation) and XML (Extensible Markup Language) files.

1st Key Element of Data Storage Design Framework

Data Analysis[2]

2. Analyze Data Volume and Growth

Initial Estimation

Estimate the initial amount of data the application will store.

Consider factors like user base, data generated per user, and frequency of data collection.

Growth Projection

Project future growth trends to ensure the chosen storage solution is scalable.

Ask critical questions to project the growth. *Will the data volume double in a year? Will it increase tenfold over five years?*

3. Analyze Data Access Patterns

Access Pattern

Data access pattern refers to how often the application read, write and update the data.

Will it be primarily read-intensive *(e.g., displaying product information)* or frequent writes and updates *(e.g., processing customer transactions)*?

Identify access patterns for specific data types.

Are there frequently accessed tables in a database, or are image files retrieved more often than documents?

2nd Key Element of Data Storage Design Framework

Storage Selection[1]

Data storage selection refers to choosing the most suitable storage device or service for the software's data needs.

There are five factors to be considered when selecting the data storage device.

a. Capacity	How much data the software need to store? This factor determines the size of the storage device or service required.
b. Performance	How quickly does the software need to access your data? Some storage options offer faster read and write speeds than others.
c. Cost	Storage solutions can vary significantly in price. Consider upfront costs, ongoing maintenance fees, and scalability options.
d. Security	How sensitive is the software data? Choose a storage solution with appropriate security features, such as encryption and access controls.
e. Durability	How long does the application need to store the data? Some storage media degrades over time, while others offer long-term archival capabilities.

2nd Key Element of Data Storage Design Framework

Storage Selection[2]

Some of the common choices for data storage selections.

Hard Disk Drives (HDDs)

HDDs are affordable storage with high capacity. However, the access speed became slower as data grows.

Solid State Drives (SSDs)

Data access using SSD is much faster than HDDs. Typically, an SDD is more expensive and has a lower capacity than an HDD.

Cloud Storage

Cloud storage offers better scalability than HDD and SDD. It also offers remote access and potential cost-effectiveness but relies on internet connectivity.

Tape Storage

Tape storage is Ideal for long-term archival of infrequently accessed data. However, the data retrieval can be slow.

3rd Key Element of Data Storage Design Framework

Data Modelling

Data modelling is the process of defining how data is structured, organised, and accessed.

The approach to data modelling differs depending on the type of data the application needs.

Structured Data Modelling

Structured data is highly organized and follows a predefined format, typically stored in relational databases.

Entity-relationship diagrams (ERDs) visually represent the entities in software systems and the relationships between them.

The relational model organises data into tables with rows and columns.

Structured data excels in situations requiring efficient querying and data integrity.

Unstructured Data Modelling

Unstructured data has no predefined format.

Example: text documents, images, or videos

Metadata (data about the data) is crucial for understanding and utilizing unstructured data.

Tags, keywords, and classifications can help categorize and search this data.

Unstructured data is valuable for capturing rich information and future analysis.

Semi-Structured Data Modelling

Semi-structured data has some organization but lacks a strict schema.

Semi-structured data uses languages like JSON Schema or XML Schema to define the expected structure and data types.

These databases store data in documents with key-value pairs, allowing flexibility for data with varying structures.

Semi-structured data offers a balance between flexibility and organization.

4th Key Element of Data Storage Design Framework

Security Measures[1]

1. Access Control

An access control dictates who can access what information and software systems.

The access control restricts system and data access to authorized users.

This prevents unauthorized individuals from viewing, modifying, or deleting sensitive information.

Methods

Methods commonly implemented as security measures include password management, user permissions and access logs.

Password Management

Enforce strong, unique passwords with multi-factor authentication (MFA) for added security.

MFA requires a second verification step, like a code from your phone, to log in.

User Permissions

Assign permissions based on a user's role.

Example: Someone in accounting may need access to financial data but not marketing materials.

Access Logs

Monitor who is accessing what data and when. This helps identify suspicious activity.

4th Key Element of Data Storage Design Framework

Security Measures[2]

2. Encryption

Encryption scrambles data using algorithms, making it unreadable to anyone without the decryption key.

This protects data at rest (stored on a device) and in transit (transmitted over a network).

Commonly Used Encryption Method

Full Disk Encryption

This method encrypts the entire storage drive of a computer or device.

File Encryption

This method encrypts specific files or folders.

Data Encryption in Transit

This method encrypts data while it's being sent over a network, such as using HTTPS for secure websites.

4th Key Element of Data Storage Design Framework

Security Measures[3]

3. Backup Plan

A backup plan ensures data can be recovered in case of a software failure, cyberattack, or accidental deletion.

Commonly Used Backup Plan Method

Regular Backups

Create copies of the software data at scheduled intervals and store them securely, ideally offsite, in case of a physical disaster.

Backup Rotation

Implement a backup rotation scheme to ensure the software has multiple versions of your data in case the most recent backup is corrupted.

Disaster Recovery Plan

Develop a plan to restore the software and data in the event of a major outage, including the method for accessing the backup and getting the systems back online quickly.

5th Key Element of Data Storage Design Framework

Performance Optimization

Introduction

In software development, performance optimization, which concerns data design, refers to techniques that make the program handle data as efficiently as possible.

This translates to faster processing times, quicker data retrieval, and overall responsiveness of your software.

Common Strategy

Choose appropriate storage media (HDD, SSD) based on access speed requirements.

Design database queries to efficiently retrieve data.

Consider caching strategies to improve access times for frequently used data.

6th Key Element of Data Storage Design Framework

Documentation

Introduction

Data storage design documentation is a crucial part of the system design process.

The documentation is a collection of materials that explain all key elements of the data storage design framework.

The content of the document includes:-

a. Data requirements
b. Storage selection
c. Data modelling
d. Security measures
e. Performance optimization
f. Scalability considerations

Importance

The documentation for data storage design acts as a roadmap, ensuring the software functions as intended, can be effectively maintained, and is understood by the stakeholders.

Interface Design

Managing Component Communication

Types of Software Interfaces[1]

	Inter-connection Software Interfaces	Intra-connection Software Interfaces
Type of Communication	Focus on communication between separate applications or systems	Focus on communication within a single application or system.
Benefit	Modular design Allows for independent development and maintenance of applications.	Modular design Allows for better code organization and maintainability.
	Reusability Enables applications to leverage existing functionalities offered by other applications.	Reduced code duplication Promotes code reuse within the application.
	Integration Facilitates the creation of complex systems by combining functionalities from different applications.	Improved testability Enables testing of individual modules independently.

Types of Software Interfaces[2]

	Inter-connection Software Interfaces	Intra-connection Software Interfaces
Examples	APIs (Application Programming Interfaces) These are the most common type, allowing applications to exchange data and functionalities regardless of programming language.	Function Calls Used within a program to invoke functionalities defined in other parts of the same program.
	Web Services Standardized protocols for exchanging information between applications over the internet (e.g., SOAP, REST).	Method Calls Similar to function calls, but specific to object-oriented programming languages, where methods are associated with objects.
	Messaging Systems Enable applications to send asynchronous messages to each other (e.g., RabbitMQ,Kafka).	Module Interfaces Define how different modules within a large application interact with each other.

This course will be focusing on API design.

Introduction to API

Abbreviation

API stands for Application Programming Interface

Definition

An API is a well-defined collection of protocols and rules that act as an intermediary between software applications.

Purpose of API

This intermediary function facilitates communication, enabling applications to exchange data, functionalities, and services.

APIs essentially codify a contract or agreement, outlining the permissible methods of interaction and data exchange between applications.

Why do Developers Use API?

Enables communication between applications

APIs provide a standardized way for different software applications to communicate, regardless of the programming language in which they are written.

This allows developers to easily integrate data and functionality from other applications into their own software.

Promotes code reusability

Developers can leverage existing functionality instead of having to write everything from scratch by using the APIs,

This saves time and effort and can also help improve the quality of code.

Facilitates data exchange

APIs allow applications to share data with each other.

This will create powerful new applications and services that combine data from multiple sources.

Enhances user experience

APIs can be used to create more integrated and user-friendly applications.

For example, a weather app might use an API to retrieve weather data from a weather service.

Other Terms Associated with API

Library

Language: C/C++

Package

Language: Java

Module

Language: Python

API
Building Blocks

1. Class
2. Object
3. Method/Functions
4. Properties
5. Documentation

API Design Process

Key Phases and Task to Design the API

Stages of API Design Process

Stage 1

Define the Purpose of the API

Stage 2

Craft the API Contract

Stage 3

Validate the Assumptions

Stage 4

Document the API

Stage 1 of API Design Process

Define the Purpose of the API

Introduction

Defining the purpose of the API is the first stage in the API design process.

This stage focuses on clearly outlining the API's intended functionality.

Outlining the Purpose

Outlining the purpose involves gathering input from stakeholders to establish the API's business use case.

The business use case can be obtained from the software requirement specification (SRS).

Example

An API responsible for an authentication workflow will have different requirements than an API that allows a user to browse a product catalog.

Therefore, it is important to align on the use case before making any other decisions.

Stage 2 of API Design Process

Craft the API Contract

Once the purpose is defined, the API's contract is meticulously crafted in the second stage of the API design process.

Essentially, this stage defines how developers will interact with the API.

An API contract specifies the following:-

1	2	3	4
The resources the API will manage	The data format and structure	The relationships between these resources	The methods available for interaction

Design Balance

It's also important to determine the desired levels of abstraction and encapsulation in the API.

This will help you strike a balance between reusability and legibility.

Validate the Assumptions[1]

This stage involves validating initial assumptions through prototypes and testing to ensure the API aligns with software needs.

Assumptions

Assumptions are beliefs or expectations about the API's users, functionality, and usage patterns.

Users

Assumptions about users' desired tasks, technical expertise, and device types.

Functionality

The API's features and functionalities, data format, and support for specific requests are assumed.

Usage Patterns

Assumptions may include the frequency of API calls, the data volume it will handle, and the integration complexity with different applications.

Validate the Assumptions[2]

Validation

Mock-ups and simulations can be created to gather feedback and identify potential issues before full-fledged development begins.

Mock-ups refer to simulated environments created to that mimic the behaviour of a real API.

Once an API definition has been completed, it can be used to generate mock servers.

Mock Servers

Mock servers play a pivotal role in the API design process.

The servers return sample data in response to requests, confirming that the API will work as required.

Mocks can also be used alongside API tests, run manually, on a schedule, or automatically within development and integration pipelines.

Stage 4 of API Design Process

Document the API

A well-documented API is crucial for smooth adoption by developers.

1. Content of Documentation

This stage involves creating comprehensive documentation that clearly explains every aspect of the API, including resources, methods, parameters, path, response codes, and error handling.

2. Examples of Use

Documentation might also include examples of API requests and responses, which give consumers crucial insight into how a particular API supports common business needs.

3. Purpose of Documentation

The documentation helps validate the design and ensure that consumers can start using the API as quickly as possible.

4. Automate Documentation

The development teams do not have to worry about their documentation becoming outdated because there are tools that automatically generate documentation from an API definition.

Examples of API

Some examples as reference

Example 1

Java HashMap

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/HashMap.html

Module java.base

Package java.util

Class HashMap<K,V>

java.lang.Object
    java.util.AbstractMap<K,V>
        java.util.HashMap<K,V>

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

All Implemented Interfaces:

Serializable, Cloneable, Map<K,V>

Direct Known Subclasses:

LinkedHashMap, PrinterStateReasons

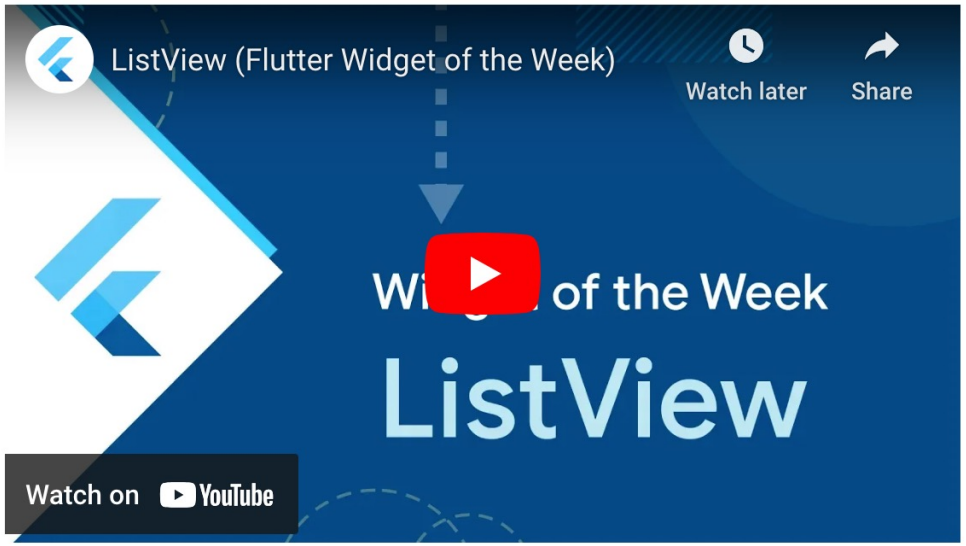
Example 2

Flutter ListView

https://api.flutter.dev/flutter/widgets/ListView-class.html

ListView class

A scrollable list of widgets arranged linearly.



ListView is the most commonly used scrolling widget. It displays its children one after another in the scroll direction. In the cross axis, the children are required to fill the ListView.

If non-null, the itemExtent forces the children to have the given extent in the scroll direction.

CONSTRUCTORS

ListView
builder
custom
separated

PROPERTIES

anchor
cacheExtent
center
childrenDelegate
clipBehavior
controller
dragStartBehavior
hashCode
itemExtent
itemExtentBuilder
key
keyboardDismissBe...

Example 3

GoogleMap FindPlace

https://developers.google.com/maps/documentation/places/web-service/search-find-place

Find Place requests

A Find Place request is an HTTP URL of the following form:

```
https://maps.googleapis.com/maps/api/place/findplacefromtext/output?parameters
```

where `output` may be either of the following values:

- `json` (recommended) indicates output in JavaScript Object Notation (JSON)
- `xml` indicates output as XML

Certain parameters are required to initiate a Find Place request. As is standard in URLs, all parameters are separated using the ampersand (`&`) character.

End of Lecture 13

Next lecture – Lecture 14 Structural and Behavioral Design of Components