News Agency Client-Server

Analysis and Design Document

Mureșian Dan-Viorel
30433

Table of Contents

1. Requirements Analysis

1.1. Assignment Specification

Use Java/C# API to design and implement a client-server application for a news agency. The application has three types of users: the readers, the writers and an administrator. The readers can view a list of articles, read an article and do not need to login in order the use the application.

1.2. Functional Requirements

- The writers need to authenticate in order to create, update or delete articles;
- The admin is the only one who can create writer accounts, but cannot create new admin accounts;
- The admin accounts are preset by the application developer and cannot be altered;

An article has the following components:

- Title;
- Abstract;
- Body;
- List of related articles.

Application Constraints and Technical Requirements:

- Use a Client-Server architectural pattern;
- Use the Observer design pattern to update the list of articles in real-time;
- In order to send data from client to server, use JSON serialization;
- When writing an article, show a list that supports multi-select for choosing the related articles.

2. Use-Case Model

Use Case: View an article
Level: client level
Primary Actor: client
Main success scenario:

- Start application;
- Click the View Articles button;
- Select an article from the list;
- Click the Read Article button.

Extensions:

- There are no articles in the list.

Use Case: Write an article
Level: writer level
Primary Actor: writer
Main success scenario:

- Start application;
- Click the Login button;
- Login with a writer account;
- Click the Write button;
- Enter the data for the article;
- Click the Publish button.

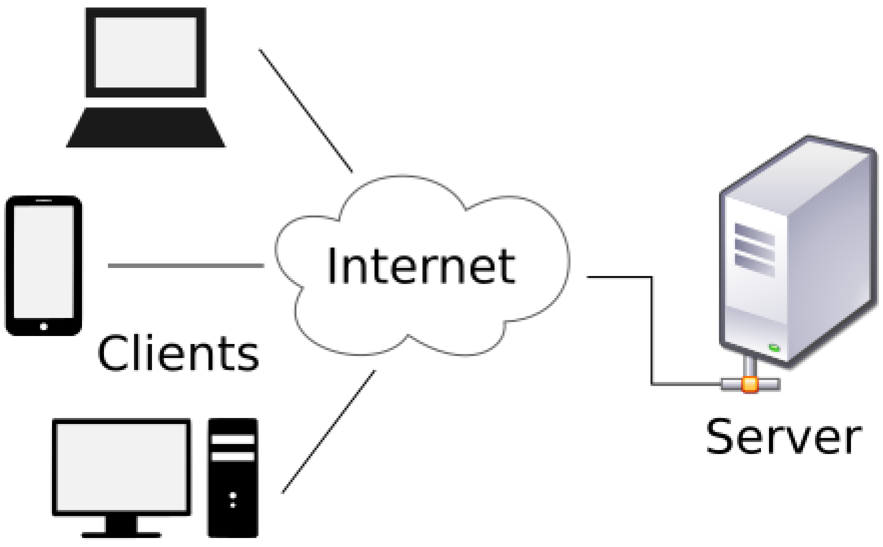
Extensions:

- Entering null data will generate an error;
- You cannot create an article without a writer account;
- Invalid Login credentials;

3. System Architectural Design

3.1. Architectural Pattern Description

The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web.



The client-server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. Servers are classified by the services they provide. For example, a web server serves web pages and a file server serves computer files. A shared resource may be any of the server computer's software and electronic components, from programs and data to processors and storage devices. The sharing of resources of a server constitutes a service.

Whether a computer is a client, a server, or both, is determined by the nature of the application that requires the service functions. For example, a single computer can run web server and file server software at the same time to serve different data to clients making different kinds of requests. Client software can also communicate with server software within the same computer.[2] Communication between servers, such as to synchronize data, is sometimes called inter-server or server-to-server communication.

In general, a service is an abstraction of computer resources and a client does not have to be concerned with how the server performs while fulfilling the request and delivering the response. The client only has to understand the response based on the well-known application protocol, i.e. the content and the formatting of the data for the requested service.

Clients and servers exchange messages in a request–response messaging pattern. The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication. To communicate, the computers must have a common language, and they must follow rules so that both the client and the server know what to expect. The language and rules of communication are defined in a communications protocol. All client-server protocols operate in the application layer. The application layer protocol defines the basic patterns of the dialogue. To formalize the data exchange even further, the server may implement an application programming interface (API).[3] The API is an abstraction layer for accessing a service. By restricting communication to a specific content format, it facilitates parsing. By abstracting access, it facilitates cross-platform data exchange.[4]

A server may receive requests from many distinct clients in a short period of time. A computer can only perform a limited number of tasks at any moment, and relies on a scheduling system to prioritize incoming requests from clients to accommodate them. To prevent abuse and maximize availability, server software may limit the availability to clients. Denial of service attacks are designed to exploit a server's obligation to process requests by overloading it with excessive request rates.

4. Class Design

4.1. Design Pattern Description

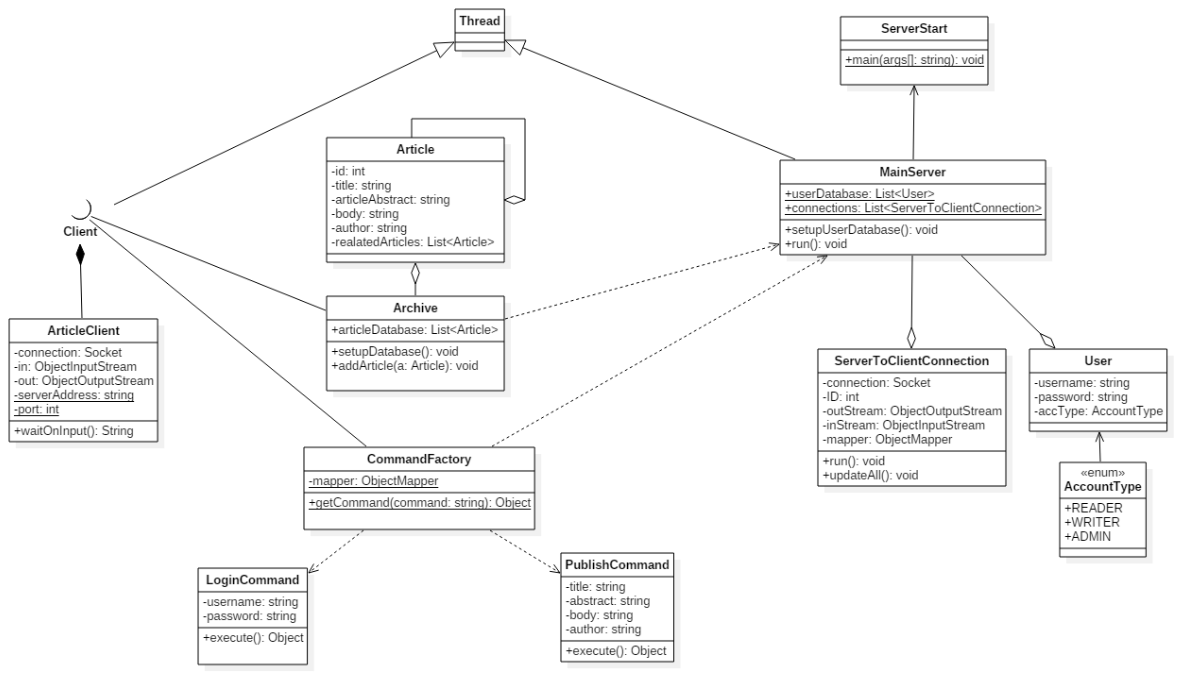
The Observer design pattern is one of the twenty-three well-known "Gang of Four" design patterns that describe how to solve recurring design problems to design flexible and reusable object-oriented software, that is, objects that are easier to implement, change, test, and reuse.

The Observer pattern addresses the following problems:

- A one-to-many dependency between objects should be defined without making the objects tightly coupled.
- It should be ensured that when one object changes state an open-ended number of dependent objects are updated automatically.
- It should be possible that one object can notify an open-ended number of other objects.

Defining a one-to-many dependency between objects by defining one object (subject) that updates the state of dependent objects directly is inflexible because it commits (tightly couples) the subject to particular dependent objects. Tightly coupled objects are hard to implement, change, test, and reuse because they refer to and know about (how to update) many different objects with different interfaces.

4.2. UML Class Diagram



5. System Testing

5.1. White-box Testing

White-box testing uses the control structure of the procedural design to derive test cases that (i) guarantee that all independent paths within a module have been exercised at least once, and that (ii) exercise all logical decisions, all loops and internal data structures. Control structure testing is an example of white-box testing method which includes the following:

a) Condition testing

- Exercises the logical conditions in a program module;
- If a condition is incorrect, then at least one component of the condition is incorrect. Therefore, types of errors in a condition include the following:
  - Boolean operator error (incorrect/missing/extra Boolean operators);
  - Boolean variable error;
  - Boolean parenthesis error;
  - Relational operator error;
  - Arithmetic expression error.

- Strategies:
    - *Branch testing* - for a compound condition C, the true and false branches of C and every simple condition in C need to be executed at least once.
    - *Domain testing* - for E1 <relational-operator> E2, 3 tests are required to make the value of E1 greater than, equal to, or less than that of E2.
    - *Branch and relational testing* - detects branch and relational operator errors in a condition if all Boolean variables and relational operators in the condition occur only once and have no common variables.

a) Loop testing

5.2. Black-box Testing

Black-box testing, also called behavioral testing, focuses on the functional requirements of the software and enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program. Black-box testing is not an alternative to white-box techniques; rather, it is a complementary approach that is likely to uncover a different class of errors than white-box methods. Black-box testing attempts to find errors in the following categories: (1) incorrect or missing functions, (2) interface errors, (3) errors in data structures or external data base access, (4) behavior or performance errors, and (5) initialization and termination errors. Black-box testing includes methods such as the following:

a) Equivalence partitioning

- Divides the input domain of a program into classes of data from which test cases can be derived.
- An equivalence class represents a set of valid or invalid states for input conditions. Typically, an input condition is either a specific numeric value, a range of values, a set of related values, or a Boolean condition. Equivalence classes may be defined according to the following guidelines:
    - If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
    - If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
    - If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
    - If an input condition is Boolean, one valid and one invalid class are defined.

b) Boundary value analysis

- Complements equivalence partitioning by exercising bounding values. Rather than selecting any element of an equivalence class, the boundary value analysis leads to the selection of test cases at the "edges" of the class.

6. Bibliography

1. https://en.wikipedia.org/wiki/Observer_pattern;
2. https://en.wikipedia.org/wiki/Client%E2%80%93server_model;
3. https://www.youtube.com/watch?v=VVUuo9VO2II;
4. http://cs.lmu.edu/~ray/notes/javanetexamples/.