API DESIGN PRINCIPLES PRACTICE TEST (SAMPLE)

STUDY GUIDE



SAMPLE STUDY GUIDE WITH LIMITED QUESTIONS

VISIT US ONLINE FOR THE FULL VERSION STUDY GUIDE

https://apidesignprinciples.examzify.com

Copyright © 2026 by Examzify - A Kaluba Technologies Inc. product.

ALL RIGHTS RESERVED.

No part of this book may be reproduced or transferred in any form or by any means, graphic, electronic, or mechanical, including photocopying, recording, web distribution, taping, or by any information storage retrieval system, without the written permission of the author.

Notice: Examzify makes every reasonable effort to obtain accurate, complete, and timely information about this product from reliable sources.

SAMPLE

Table of Contents

Copyright 1

Table of Contents 2

Introduction 3

How to Use This Guide 4

Questions 5

Answers 8

Explanations 10

Next Steps 16

SAMPLE

Introduction

Preparing for a certification exam can feel overwhelming, but with the right tools, it becomes an opportunity to build confidence, sharpen your skills, and move one step closer to your goals. At Examzify, we believe that effective exam preparation isn't just about memorization, it's about understanding the material, identifying knowledge gaps, and building the test-taking strategies that lead to success.

This guide was designed to help you do exactly that.

Whether you're preparing for a licensing exam, professional certification, or entry-level qualification, this book offers structured practice to reinforce key concepts. You'll find a wide range of multiple-choice questions, each followed by clear explanations to help you understand not just the right answer, but why it's correct.

The content in this guide is based on real-world exam objectives and aligned with the types of questions and topics commonly found on official tests. It's ideal for learners who want to:

- Practice answering questions under realistic conditions,
- Improve accuracy and speed,
- Review explanations to strengthen weak areas, and
- Approach the exam with greater confidence.

We recommend using this book not as a stand-alone study tool, but alongside other resources like flashcards, textbooks, or hands-on training. For best results, we recommend working through each question, reflecting on the explanation provided, and revisiting the topics that challenge you most.

Remember: successful test preparation isn't about getting every question right the first time, it's about learning from your mistakes and improving over time. Stay focused, trust the process, and know that every page you turn brings you closer to success.

Let's begin.

SAMPLE

How to Use This Guide

This guide is designed to help you study more effectively and approach your exam with confidence. Whether you're reviewing for the first time or doing a final refresh, here's how to get the most out of your Examzify study guide:

1. Start with a Diagnostic Review

Skim through the questions to get a sense of what you know and what you need to focus on. Your goal is to identify knowledge gaps early.

2. Study in Short, Focused Sessions

Break your study time into manageable blocks (e.g. 30 – 45 minutes). Review a handful of questions, reflect on the explanations.

3. Learn from the Explanations

After answering a question, always read the explanation, even if you got it right. It reinforces key points, corrects misunderstandings, and teaches subtle distinctions between similar answers.

4. Track Your Progress

Use bookmarks or notes (if reading digitally) to mark difficult questions. Revisit these regularly and track improvements over time.

5. Simulate the Real Exam

Once you're comfortable, try taking a full set of questions without pausing. Set a timer and simulate test-day conditions to build confidence and time management skills.

6. Repeat and Review

Don't just study once, repetition builds retention. Re-attempt questions after a few days and revisit explanations to reinforce learning. Pair this guide with other Examzify tools like flashcards, and digital practice tests to strengthen your preparation across formats.

There's no single right way to study, but consistent, thoughtful effort always wins. Use this guide flexibly, adapt the tips above to fit your pace and learning style. You've got this!

SAMPLE

Questions

SAMPLE

1. **Is the wrong order of design considered to be "implement functionality -> design API"?**

   A. True

   B. False

   C. It can be correct

   D. No established order

2. **What does OAuth stand for?**

   A. Open Access for Users

   B. Open Authentication

   C. Open Authorization

   D. Optional Authentication

3. **What distinguishes synchronous from asynchronous API calls?**

   A. Synchronous calls do not require user interaction, but asynchronous calls do

   B. Synchronous calls wait for a response, while asynchronous calls do not

   C. Synchronous calls are faster than asynchronous calls

   D. Synchronous calls can only be made one at a time

4. **What is the single responsibility principle in API design?**

   A. Each API should have a clear, defined purpose and should not handle multiple unrelated responsibilities.

   B. Each API should be used for a single request type.

   C. Each API should limit its response size to one entity.

   D. Each API should maintain a single version at all times.

5. **What are hypermedia links in REST?**

   A. Links provided in the API responses that allow users to navigate related resources.

   B. Links used to link API endpoints for security purposes.

   C. Links embedded in data for compression.

   D. Links connecting APIs to external databases.

SAMPLE

6. Is it often the case that API users are less concerned with how the API works and more focused on what it accomplishes?

A. True

B. False

C. Only for developers

D. It's dependent on API complexity

7. In API design, what outcome is expected when subclassing is executed correctly?

A. Increased functional redundancy

B. Notable performance reduction

C. Clearer class hierarchies

D. Complicated code interactions

8. What is one guideline to favor in API design regarding exceptions?

A. Favor checked exceptions

B. Favor unchecked exceptions

C. Throw exceptions only when necessary

D. Avoid exceptions altogether

9. What is the recommended API style for users who must configure many GUI settings?

A. Property-based API with setters

B. Method-based API with callbacks

C. Event-driven API

D. Functional API

10. What is meant by a RESTful API's self-descriptiveness?

A. It limits user access based on permissions

B. It provides enough context in its responses to facilitate understanding without additional documentation

C. It requires extensive external documentation

D. It is designed exclusively for internal use

SAMPLE

Answers

SAMPLE

1. A
2. C
3. B
4. A
5. A
6. A
7. C
8. B
9. A
10. B

SAMPLE

Explanations

SAMPLE

1. Is the wrong order of design considered to be "implement functionality -> design API"?

A. True

B. False

C. It can be correct

D. No established order

The assertion that the wrong order of design is considered to be "implement functionality -> design API" is accurate. In API design, the process should ideally start with the design of the API before implementing any functionality. This allows developers to clearly define how the API will be used, including the endpoints, request and response formats, and any potential error handling. Starting with the design helps ensure that the API meets the needs of its users, maintains consistent principles, and can be integrated smoothly into the larger system architecture. Conversely, implementing functionality first can lead to various challenges, such as misaligned user expectations or an API that is cumbersome to use and integrate, ultimately resulting in a product that may require significant rework. In terms of the other options, while there could be situations where an iterative development process occurs, the foundational best practice in API design emphasizes the importance of planning and designing the API ahead of implementation. Hence, it is crucial for the integrity and usability of the API to establish a clear design framework before diving into implementation.

2. What does OAuth stand for?

A. Open Access for Users

B. Open Authentication

C. Open Authorization

D. Optional Authentication

OAuth stands for "Open Authorization." This framework is designed to allow third-party applications to obtain limited access to an HTTP service, either on behalf of a resource owner or by allowing the third-party application to obtain access on its own behalf without sharing the user's credentials. The term "authorization" in OAuth indicates its primary purpose, which is to grant permissions to applications to access specific user data while keeping user credentials secure. By using OAuth, users can authorize applications to interact with their resources hosted on other servers without exposing sensitive information like passwords. The other options, while they may sound plausible, do not capture the correct purpose of OAuth. "Open Access for Users," "Open Authentication," and "Optional Authentication" suggest varying meanings but fail to accurately describe the main functionality and focus of the OAuth framework, which is about authorization rather than simply access or authentication processes.

SAMPLE

3. What distinguishes synchronous from asynchronous API calls?

A. Synchronous calls do not require user interaction, but asynchronous calls do

B. Synchronous calls wait for a response, while asynchronous calls do not

C. Synchronous calls are faster than asynchronous calls

D. Synchronous calls can only be made one at a time

The correct response highlights a fundamental characteristic of synchronous and asynchronous API calls. In synchronous calls, the client sends a request and then waits idly for the response from the server before proceeding with further execution. This means that the execution flow is blocked until the response is received, ensuring that the client receives the requested data promptly before continuing. In contrast, asynchronous calls allow the client to send a request and then continue processing without waiting for the response. The client can handle other tasks while the server processes the request and responds at a later time, which can lead to more efficient use of resources and improved application responsiveness. The other options may seem relevant but do not accurately capture the key difference between synchronous and asynchronous calls. User interaction isn't a defining factor in distinguishing between these two types of calls, speed cannot universally be declared without considering the context of the operation, and the ability to make calls simultaneously does not solely define synchronous calls either. Thus, the unique trait of blocking behavior for synchronous calls and non-blocking behavior for asynchronous calls is the essential concept conveyed by the correct answer.

4. What is the single responsibility principle in API design?

A. Each API should have a clear, defined purpose and should not handle multiple unrelated responsibilities.

B. Each API should be used for a single request type.

C. Each API should limit its response size to one entity.

D. Each API should maintain a single version at all times.

The single responsibility principle in API design refers to the idea that each API should be designed with a clear, defined purpose, ensuring it does not take on multiple unrelated responsibilities. This principle fosters better maintainability, scalability, and clarity in the design and usage of APIs. When an API is focused on a single task, it becomes easier for developers to understand, use, and maintain, which can result in reduced likelihood of errors and improved performance. Additionally, this approach promotes the separation of concerns, where different functionalities can be managed and deployed independently, enhancing the overall architectural integrity of the application. The other choices do not accurately capture the essence of the single responsibility principle. While having a single request type, limiting response size, or maintaining a single version may be important considerations in API design, they do not embody the core idea of ensuring that each API handles only one particular functionality or responsibility. The principle is centered around organizational clarity and functionality, rather than technical constraints or operational aspects.

SAMPLE

5. What are hypermedia links in REST?

A. Links provided in the API responses that allow users to navigate related resources.

B. Links used to link API endpoints for security purposes.

C. Links embedded in data for compression.

D. Links connecting APIs to external databases.

Hypermedia links in REST, often referred to as HATEOAS (Hypermedia As The Engine Of Application State), are crucial for enabling clients to navigate an API dynamically. The correct response identifies that these are links included in API responses that direct users to related resources, facilitating a more flexible interaction with the API. In a RESTful architecture, including hypermedia links allows clients not only to discover the available actions they can take on a resource but also to understand the relationships between various resources without needing prior knowledge of the API's structure. For instance, when a client retrieves a specific resource, the server can include links to related entities—such as child resources or actions that can be performed—thus guiding the client on how to proceed next. This dynamic discovery aspect significantly enhances the usability and self-descriptiveness of the API, allowing for better integration with clients that may not have hardcoded knowledge of every endpoint. Overall, hypermedia links support the RESTful principle of statelessness and decoupled client-server interactions by providing a way to navigate an API using standard web protocols.

6. Is it often the case that API users are less concerned with how the API works and more focused on what it accomplishes?

A. True

B. False

C. Only for developers

D. It's dependent on API complexity

The statement that API users are often less concerned with how the API works and more focused on what it accomplishes is indeed accurate. This perspective highlights that the primary interest of API users, whether they are developers, product managers, or other stakeholders, typically revolves around the outcomes and functionality the API provides rather than the intricate details of its internal workings. API users are generally interested in the capabilities and benefits they can derive from the API. For example, they want to know how the API can help them solve a particular problem, enhance the applications they are building, or improve user experience in their services. This focus on results allows them to integrate the API into their systems effectively without needing to delve deeply into its implementation or architecture. This viewpoint is especially prevalent in environments where speed and product development efficiency are prioritized, as users seek to leverage existing solutions to achieve their goals quickly. While some stakeholders, particularly developers who may need technical specifics for implementation and troubleshooting, could intersect with the complexities of how an API operates, the general trend affirms that the end goal remains the focus. In essence, understanding that API users prioritize functionality over implementation details can significantly influence how APIs are designed, documented, and presented, ensuring they meet user needs effectively.

SAMPLE

7. In API design, what outcome is expected when subclassing is executed correctly?

A. Increased functional redundancy

B. Notable performance reduction

C. Clearer class hierarchies

D. Complicated code interactions

When subclassing is executed correctly in API design, it leads to clearer class hierarchies. The primary purpose of subclassing is to establish a relationship between a base class and derived classes, making it easier to organize and manage code. This clarity allows developers to understand the relationships and functionality of various classes at a glance. A well-designed hierarchy indicates how objects relate to one another and can simplify code maintenance, as developers can easily identify where functionality is inherited and overridden. This organization supports code reuse, as subclasses can extend or modify the behavior of their parent class without unnecessary duplication of code. The other outcomes mentioned do not align with the intention of effective subclassing. Increased functional redundancy would typically suggest poor design practices rather than an effective subclassing strategy. Notable performance reduction can occur in certain scenarios but is not a direct outcome of proper subclassing; it often relates more to how classes are implemented rather than the system of inheritance itself. Lastly, complicated code interactions often arise from improper use of inheritance or too many layers, rather than from the clear hierarchy established by correct subclassing.

8. What is one guideline to favor in API design regarding exceptions?

A. Favor checked exceptions

B. Favor unchecked exceptions

C. Throw exceptions only when necessary

D. Avoid exceptions altogether

In API design, one important guideline regarding exceptions is to favor unchecked exceptions. Unchecked exceptions, which include runtime exceptions, are not required to be declared in the method signatures, making them less cumbersome to handle. This allows for more fluid and concise code, as developers are not forced to catch or declare them unless they specifically want to. Using unchecked exceptions can lead to cleaner APIs, as they enable developers to focus on the normal flow of the program without being bogged down by excessive error handling mechanisms. They should be used to indicate programming errors, such as illegal argument values or access violations, where the caller is usually in a better position to avoid the exception through validation. Additionally, unchecked exceptions can be an effective approach in scenarios where the error conditions are considered unrecoverable or where the calling code typically cannot do anything meaningful to handle the exception. This preference aligns with the principle of simplifying the development experience and encouraging robust software practices by allowing exceptions to propagate without forcing cumbersome handling by the caller.

SAMPLE

9. What is the recommended API style for users who must configure many GUI settings?

A. Property-based API with setters

B. Method-based API with callbacks

C. Event-driven API

D. Functional API

The recommended API style for users who must configure many GUI settings is a property-based API with setters. This approach is intuitive and straightforward, allowing users to set various properties of GUI components directly. By using setter methods, users can easily adjust configuration options without needing to understand intricate method signatures or complex interactions. This streamlined process enhances usability, particularly for users who may not be deeply familiar with programming concepts, as it closely resembles setting properties in more visual programming environments. In many GUI frameworks, properties often reflect the state and appearance of the components, making it natural for users to interact with them in this way. The flexibility of being able to change several properties in a clear and concise manner can significantly improve the overall user experience. Other API styles may be suitable for different contexts, but they may not be as user-friendly when dealing with numerous configuration settings. For instance, method-based APIs with callbacks can introduce complexity and require a deeper understanding of asynchronous programming. Event-driven APIs, while powerful for handling dynamic user interactions, might not lend themselves to straightforward configuration. Functional APIs could also be suitable, but they would generally not provide the same level of clarity and ease of use for property configurations as the property-based API with setters does.

10. What is meant by a RESTful API's self-descriptiveness?

A. It limits user access based on permissions

B. It provides enough context in its responses to facilitate understanding without additional documentation

C. It requires extensive external documentation

D. It is designed exclusively for internal use

A RESTful API's self-descriptiveness refers to its ability to provide sufficient context within its responses, enabling clients to understand the data and how to interact with the API without needing extensive external documentation. This characteristic is pivotal for usability and developer experience, as it allows developers to work with the API more intuitively. When an API is self-descriptive, each response includes information such as data formats, relevant meta-information about the resource, possible actions that can be taken, and links to related resources. This kind of design reduces reliance on separate documentation and helps users quickly grasp how to use the API effectively. The more self-descriptive an API is, the easier it is for developers to integrate and utilize it, ultimately leading to faster development cycles and fewer errors.

SAMPLE

Next Steps

Congratulations on reaching the final section of this guide. You've taken a meaningful step toward passing your certification exam and advancing your career.

As you continue preparing, remember that consistent practice, review, and self-reflection are key to success. Make time to revisit difficult topics, simulate exam conditions, and track your progress along the way.

If you need help, have suggestions, or want to share feedback, we'd love to hear from you. Reach out to our team at hello@examzify.com.

Or visit your dedicated course page for more study tools and resources:

https://apidesignprinciples.examzify.com

We wish you the very best on your exam journey. You've got this!

SAMPLE