Having a clear set of best practices when building APIs provides guidelines and standards that promote consistency in API design and development, ensure reliability, and make it much easier for developers to work with an API. Without best practices for a team to follow, developers may write code in their preferred style and create inconsistent code with a higher probability of errors.

Our technical teams at Xome® make it a point to set and follow best practices for building APIs to guarantee scalability of code and a positive developer experience.

## **Best practices for building real estate APIs**

Xome’s API best practices revolve around the idea of implementing clean core architecture, which promotes separation of concerns and maintainability in software development.

An API project structure should align well with the principles of [domain-driven design (DDD)](https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/best-practice-an-introduction-to-domain-driven-design) where the domain models are at the center of the architecture and encapsulate the business rules and behaviors. Workflows should be kept simple by using branching strategies like GitHub Flow that allow for continuous delivery and integration.

There are four important architectural principles and practices in software development that we follow:

* **Independence of frameworks:** The core business logic and domain models are independent of any specific framework or external dependencies. This allows for easier testing and flexibility to switch frameworks or technologies in the future.
* **Separation of concerns:** The architecture is organized into three different layers — the presentation layer (UI), application layer (business logic), and infrastructure layer (data access and external services). Each layer has a specific role and boundaries, and dependencies flow inward towards the core.
* **Dependency rule:** Dependencies between layers follow the dependency rule, which states that inner layers should not depend on outer layers. In other words, the core should not have direct knowledge of specific frameworks, databases, or external services. Instead, abstractions and interfaces are used to define boundaries between layers.
* **Testability**: By decoupling the core business logic from external dependencies, it becomes easier to write unit tests for the application. Tests can focus on validating the behavior and rules of the domain models without a complex setup or reliance on external resources.

## **Common coding standards for building APIs**

Following best practices also includes adopting common coding standards. Our development teams at Xome set these, which define rules to follow when writing code for building APIs. This includes consistent naming conventions, clear and concise code, and using HTTP verbs and RESTful principles.

**Other common coding standards followed include:**

* Proper error handling
* Logging and monitoring
* Unit testing
* Documentation

To understand how these best practices and standards come together in a real-world example, Xome’s Property Insights API provides an ideal use case.

## **Building the Xome Property Insights API**

Property Insights is Xome’s data offering as an API that provides developers with access to a wide range of real estate data and functionalities. It offers insights on properties, such as their [Xome Value®](https://www.xome.com/pages/disclosures), basic property data, tax and price history, and comparable sales.

The API is designed in a way that keeps the domain models at the base and derives the user models from it. Following [Xome’s cloud migration](https://www.xome.com/blog/migrating-digital-assets-to-the-cloud-a-xome-technology-case-study/), Microsoft Azure is used for storage and monitoring and statistics are collected through the API gateway, a component of Microsoft Azure API Management (APIM). APIM helps us to maintain property insights effectively and takes care the API management, authentication, versioning and different environments under the same roof.

We also make sure our APIs are containerized. Having APIs containerized allows for isolation of resources. Resource intensive services will not deprive other APIs of their needed resources allowing for consistent performance. Containerized APIs allow them to evolve independently, provide fast deployments and startup, and provide resource flexibility as they only consume what is needed. Containerized APIs also allow for fine-grained permissions and limited access.

After migrating to cloud services, our teams’ QA processes follow unit testing standards through [extensive load testing](https://www.xome.com/blog/microsoft-azure-load-testing/)using Microsoft’s Apache JMeter-based Azure Load Testing tool. This allows to test scaling during peak traffic consumption, which ranges anywhere between 100,000 requests an hour to up to 1 million a day.

## **Building high-quality digital products with well-designed APIs**

Following a set of best practices as a part of any [software development lifecycle](https://www.xome.com/blog/how-the-machine-learning-lifecycle-helps-improve-avms/) is crucial to building high-quality digital solutions, including APIs. The more thought-out an API design is, the more streamlined the integration process is for developers — plus, code is much easier to maintain.

Cloud services help to simplify meeting these standards so developers can focus more on building robust APIs. When APIs are developed with intention and long-term goals in mind, that shines through in the finished product.

See our digital products in action while searching [properties for auction](https://www.xome.com/auctions) or [for sale](https://www.xome.com/homes-for-sale)!