Osman Ghani Granday

FA21-BSE-024 |

Software Design and Architecture

Assigmnment # 1

**A Detailed Information About MVC**

* MVC is an architectural pattern used extensively in software development and is Model-View-Controller. It improves the level of efficiency of the developers' code, also giving them the ability to manage the developed applications properly, and hence they become well-maintainable and scalable.  
    
  History:  
    
  The Model–View–Controller pattern was invented first by Trygve Reenskaug in 1979 when he worked for Xerox PARC on his Smalltalk project. The objective was to design a structure of architecture that is divided into three different components namely controller, model and view so that it becomes easy to understandize and manage the codes and would reduce the errors also.  
    
    
  **Pros of MVC architecture:**  
    
  1**. Separation of concerns:** MVC is based on the MVC that the implementation logic of the application is decomposed into three independent components; this facilitates improved management and maintenance of the production code in addition to optimizing the app performance.  
    
  2. **Reusability:** The use of MVC in modular design means that developers will be able to reuse components and therefore save time and effort without having to develop anything new all over again.  
    
  3. **Easier testing**: This isolated nature is giving us the opportunity to verify the components without interfering with the rest of the things.  
    
  4. **Improved collaboration**: MVC enables several programmers to be involved with the same codebase and work along on different parts at the same time to increase the development's overall speed.  
    
  5. **Scalability:** With modular design, applications can start simple and expand to more complex ones as they grow in complexity with MVC.  
   **Cons of MVC architecture:**  
    
  1. **Increased complexity**: MVC entails additional complexity than other architectural patterns which can be challenging, particularly for smaller projects with tight deadlines or a greenfield approach.  
    
  2**. Overhead:** MVC offers the advantage of splitting concerns into separate components, which could make a mess of the system with the addition of additional tasks of files and folders creation and management.  
    
  3. **Learning curve**: Introduction to MVC by developers new to this pattern may seem difficulty to understand and implement in a proper way.

1. Components of MVC:
   * Model: The Model represents the application's data and business logic. It encapsulates the data, handles data manipulation, and interacts with the database or other data sources.
   * View: The View is responsible for the presentation layer of the application. It defines how the data should be displayed to the user and handles user interface elements.
   * Controller: The Controller acts as an intermediary between the Model and the View. It processes user inputs, manipulates data from the Model, and updates the View accordingly.
2. Workflow in MVC:
   * The user interacts with the application through the View (e.g., clicking a button or submitting a form).
   * The Controller receives the user input, processes it, and communicates with the Model to perform the necessary data manipulation.
   * The Model updates the data and sends it back to the Controller.
   * The Controller then updates the View with the new data, and the user sees the changes in the user interface.
3. Variations of MVC:
   * There are several variations of the MVC pattern, such as Model-View-Presenter (MVP), Model-View-ViewModel (MVVM), and Model-View-Adapter (MVA). These variations aim to address specific concerns or improve upon the original MVC pattern.
4. MVC in modern web development:
   * MVC is widely used in web development, particularly in frameworks like Ruby on Rails, Django, ASP.NET MVC, and Laravel. These frameworks provide built-in tools and conventions to help developers implement the MVC pattern efficiently.

**Examples of MVC usage:**

MVC is widely used in various software applications, particularly in web development. Some popular web frameworks that implement MVC include:

1. **Ruby on Rails**: A popular web application framework for the Ruby programming language that follows the MVC pattern.
2. **Django**: A high-level Python web framework that encourages rapid development and clean, pragmatic design, using the MVC pattern.
3. **ASP.NET MVC:** A Microsoft web framework that implements the MVC pattern for building dynamic, data-driven web applications using the .NET platform.
4. **Laravel:** A popular PHP web framework that follows the MVC pattern and is designed for building robust web applications.

**REFRENCES**

1. History of MVC:
2. "A Brief History of MVC" by Pete LePage: <https://www.scholarhat.com/tutorial/mvc/a-brief-history-of-aspnet-mvc-framework>
3. Pros and Cons of MVC:
   * "MVC Architecture: Advantages and Disadvantages" by educative.io:  <https://www.educative.io/answers/what-are-the-pros-and-cons-of-different-software-architectures>
4. Examples of MVC usage in web frameworks:
   * Ruby on Rails: <https://rubyonrails.org/>
   * Django: <https://www.djangoproject.com/>
   * ASP.NET MVC: <https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions>
   * Laravel: <https://laravel.com/>
5. Variations of MVC:
   * "MVC, MVP, MVVM, and MVA: What's the Difference?" by GeeksforGeeks: <https://www.geeksforgeeks.org/difference-between-mvc-mvp-and-mvvm-architecture-pattern-in-android/>
6. MVC in modern web development:
   * Ruby on Rails: <https://rubyonrails.org/>
   * Django: <https://www.djangoproject.com/>
   * ASP.NET MVC: <https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions>
   * Laravel: <https://laravel.com/>

**Data-centred Architectural Style**

**Failed Example:** The United States' **HealthCare.gov** launch (initial phase)

HealthCare.gov was launched in 2013 as a part of the Affordable Care Act (ACA) to provide a platform for Americans to purchase health insurance. The initial launch of the website was marred by numerous issues, many of which were related to its data-centric architecture.

**Reasons for failure:**  
  
1. **Inadequate data management:** The site was developed with the ability to handle large volumes of data but was not foolproof to meet data validity, strength and completeness. This in turn led to incorrect submitting, missed enrollments, and other challenges that ended up affecting the users experience.  
  
2. **Complex data integration:** HealthCare.gov faced need for integration with multiple data sources, including the Internal Revenue Service (IRS), Department of Homeland Security (DHS), and different country-level systems. The issue in this level is that multiple systems are involved, hence the data exchange got complicated which in turn caused the delays or errors in the enrolment process.  
  
3. **Insufficient testing:** HealthCare.gov architecture was not properly validated, before going live, until the time it was archived. Furthermore, the frequent problems related to performance, data consistency, and system crashes caused a lot of difficulty for users to easily move and sign up for health insurance plans.  
  
4**. Scalability issues:** HealthCare.gov was expected to take a large number of users and use huge amounts of data as one of its features. In the beginning, the infrastructure of its data was not well-designed to grow bigger. This caused longer reaction times, system crashes, and other performance bottlenecks during the initial deployment period.

REFRENCES

1. A report from the U.S. Department of Health and Human Services (HHS) Office of Inspector General on the challenges faced by HealthCare.gov, including contractor performance and oversight issues: <https://oig.hhs.gov/oei/reports/oei-06-14-00350.asp>
2. An article from The New Yorker discussing the technical challenges faced by HealthCare.gov and the efforts to fix them: <https://www.newyorker.com/tech/annals-of-technology/healthcare-gov-it-could-be-worse>
3. A look back at technical issues with Healthcare.gov by Brookings:

<https://www.brookings.edu/articles/a-look-back-at-technical-issues-with-healthcare-gov/>

1. Real-time outages and **problems** **for** **Healthcare**.**gov**. Is the website down? Can't log in or get insurance? Here you see what is going in. <https://downdetector.com/status/healthcare-gov/>
2. Congress: HealthCare.gov Technical Problems May Impact Enrollment:<https://healthpayerintelligence.com/news/congress-healthcare.gov-technical-problems-may-impact-enrollment>
3. Why healthcare.gov has so many problems

Steven Bellovin, Special to **CNN:** <https://edition.cnn.com/2013/10/14/opinion/bellovin-obamacare-glitches/index.html>

**Successful Example: Netflix**

Netflix is a prime example of a successful software built using a data-centric architecture. Netflix's architecture is designed to collect, process, and analyze massive amounts of data to provide personalized recommendations, stream videos seamlessly, and offer a reliable user experience.

Here's a simple explanation of how they did it:

1. Moving to the cloud: Netflix have moved their organizational model in the cloud, which is a virtual area and belongs to the Amazon Web Services (AWS). Because they can keep the data and do a processing with large amounts of data, this means that their service is always on.
2. Breaking it down into small parts: Netflix refined their large complex into little units called microservices. A microservice has a job of its own – like this one or that one. It did not interrupt its data management process thereby improved the delivery of services.
3. Personalized suggestions: One of the largest data sources for Netflix is the record of what and how users watch on the platform. They tend to use this information to predict the interests of each specific user and recommend movies and shows that each person might enjoy. This is a very essential component of their design and one which helps users keep happy and interested.
4. Storing and processing data: Netflix, would rely on different ways of storing and processing all the data they gather. These instruments see to it that they process a whole bunch of information effectively and aptly handle the fact that many people try to use it at the same time.
5. Delivering videos quickly: Netflix transfers videos using a specific system called Open Connect to send movies to customers. Placing servers next to users helps them to serve videos quickly and without breaking at a hectic time.
6. Making sure it always works: Netflix setup system with sturdiness and reliability in mind. They test it on a regular basis to detect and solve the issues, and so users won’t struggle with having any problems.
7. Using data to make decisions: Netflix pays close attention to the data they collect. They use this information to make smart decisions about things like which movies to buy or produce, and how to improve their service.

**Reasons for success:**

1. Scalability: Netflix's data-centric architecture allows it to handle an enormous amount of data and scale according to the increasing user base and data size.
2. Personalization: By leveraging data, Netflix can provide personalized content recommendations, which significantly improves user engagement and satisfaction.
3. Resilience: Netflix's architecture is designed to be fault-tolerant and highly available, ensuring that users can access the service without interruptions.
4. Data-driven decision making: Netflix relies on data to make informed decisions about content creation, acquisition, and marketing strategies.

**REFRENCES**

1. Netflix's blog post on their microservices architecture: <https://www.techaheadcorp.com/blog/design-of-microservices-architecture-at-netflix/>
2. Netflix's blog post on their recommendation system: <https://recoai.net/netflix-recommendation-system-how-it-works/>
3. how Netflix's data-driven culture and architecture contribute to its success: <https://www.linkedin.com/pulse/3-minute-breakdown-how-netflix-data-driven-culture-them-phil-araujo>
4. Netflix's use of data to personalize its service and improve customer satisfaction: <https://www.econtribune.com/post/how-netflix-uses-big-data-for-customer-satisfaction>
5. how Netflix's data-centric architecture enables it to scale and innovate rapidly: <https://www.linkedin.com/pulse/inside-netflix-deep-dive-its-cutting-edge-system-architecture>

***And more information and how the Netflix works visit the link***

[***https://research.netflix.com/***](https://research.netflix.com/)

**Video Recording :** [**https://youtu.be/8KfT\_pU5isY**](https://youtu.be/8KfT_pU5isY)