Solution Direction Document: *TOP DEAL AUTO WEBSITE*

Company: Top Deal Auto Melbourne, Australia

Software: Car Selling Website

Team name: Prestige K/DA

Team Members:

Name	**ID**	**Roles**
Pham Duc Linh	103792371	Product Owner - BA
Pham Anh Vu	103806447	Solutions, IT Architect
Nguyen Thanh Dat	103804881	Project Manager - Scrum Master
Tran Tuan Nam	103792643	Lead Developer (BE)
Phung Xuan Tung	103792054	Lead Developer (FE)

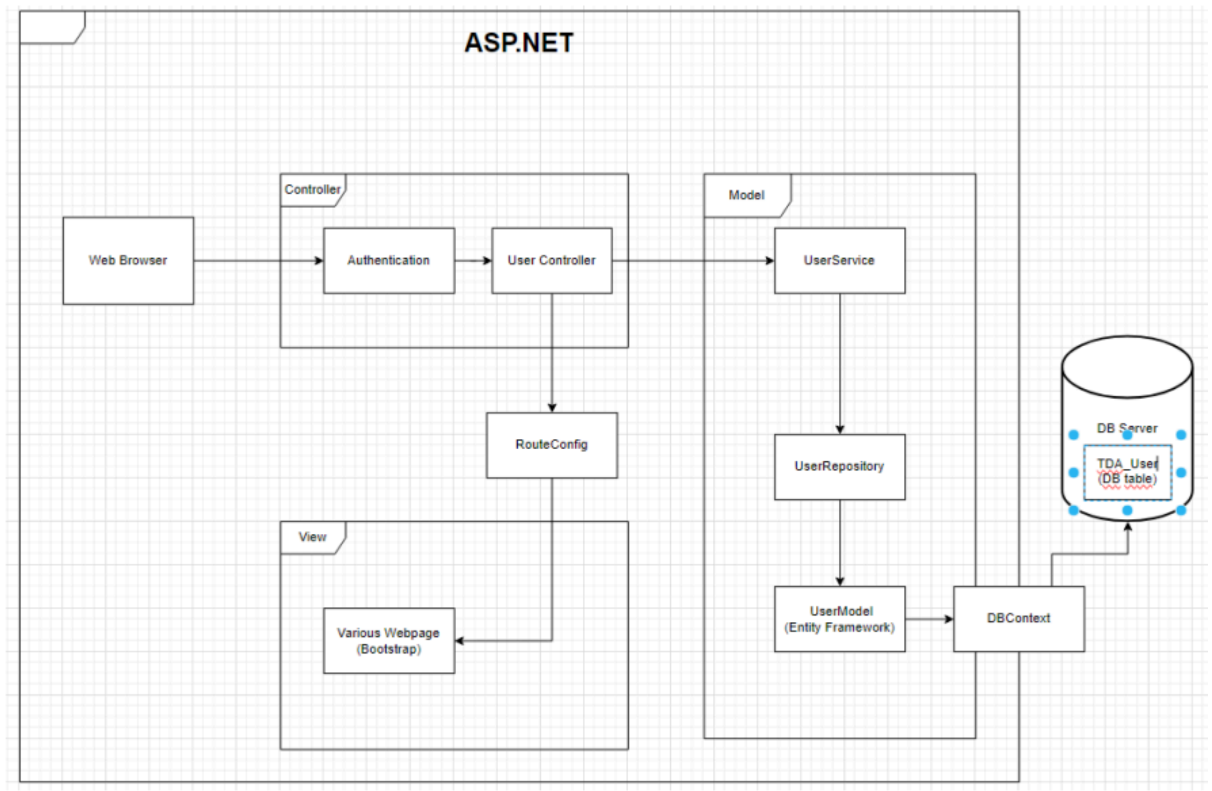
Tutorial class: Fri 1:00 PM DT7.2

Tutor's Name: Dr. Pham Thi Kim Dzung

Design Solutions

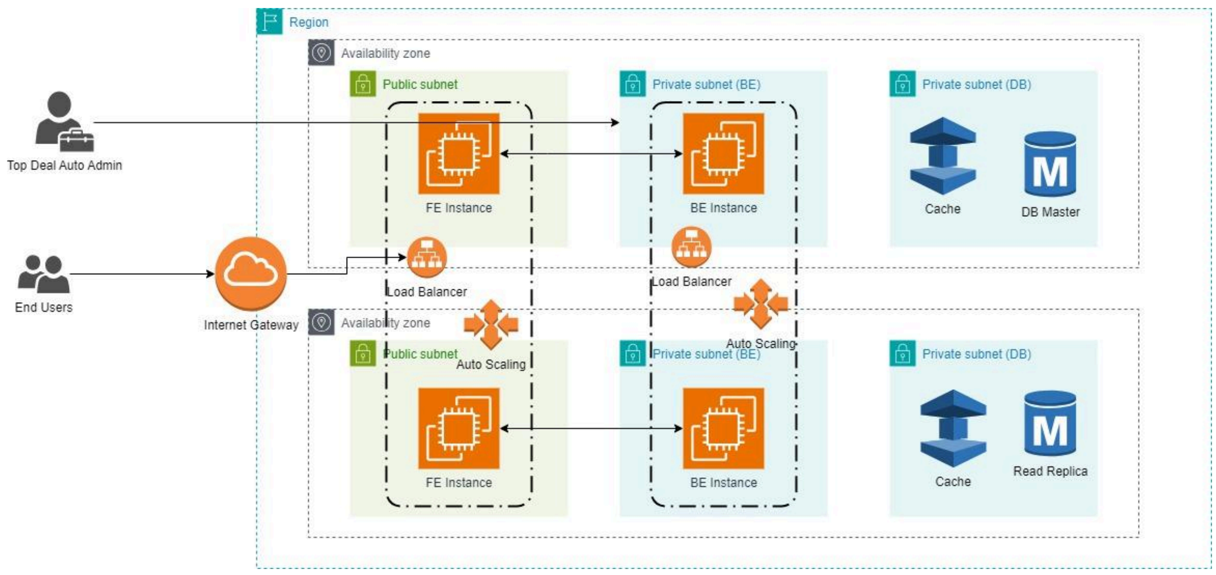
By the end of the meeting, the team had proposed 4 solution designs, out of which 2 were chosen for consideration.

MVC-based approach with .NET



The diagram illustrates the high-level design of the web application following the Model-View-Controller (MVC) pattern. When a user initiates a request to view the user webpage, the process begins with the user interacting with the View component. The View, responsible for the presentation layer, receives the user input and communicates with the corresponding Controller. The Controller, acting as an intermediary, interprets the user's request and interacts with the Model component to retrieve or manipulate data as necessary. The Model, representing the application's data and business logic, processes the request and provides the required information to the Controller. The Controller, in turn, updates the View with the relevant data, ensuring a separation of concerns between the user interface and the underlying business logic. The updated View is then presented to the user, completing the cycle of the MVC architecture. This design promotes modularity, maintainability, and scalability by clearly defining the roles and responsibilities of each MVC component in handling user interactions and data flow

Cloud-based dynamic website



Above is the overall high-level architecture that Vu proposed. Here is the rough explanation of the components.

- **Internet Gateway:** The Gateway that allows communication between the application and the Internet.
- **Load Balancer**: Balance the Request load between the multiple instances
- **Auto Scaling:** Automatically increase/decrease the number of instances based on the load.
- **FE Instance:** The instance that hosts the FE (UI) of the website.
- **BE Instance:** The instance that hosts the BE for logic handling.
- **DB Master:** The master Database to store data
- **Read Replica:** A replica of the master DB for READ operations only.
- **Cache:** A temporary storage for frequently accessed data

This architecture (in my opinion) checks all the boxes that our solution needs to achieve.

- **Performance:** The separation between the Frontend and Backend allows for lack of dependency, thus improving user experiences. The internet Gateway provided by AWS automatically assigns users to the nearest edge location, allowing for low latency. The cache also allows for faster searching of frequently accessed data.
- **Ease of maintenance:** AWS allows ease of maintenance for the application, with minimal downtime for each update. The architecture will also grow/shrink along with the use case required, guaranteeing optimum efficiency.
- **Security:** The data, administrative process are all put in private subnets, which are only accessible by the administrators themselves. This ensures security and privacy of crucial data and business processes.

- **Resilience:** AWS provides automatic backup features, which allow for high resilience and rollback capabilities in case of system/security failures. The recovery processes, depending on the service only take anywhere from a couple of minutes to maximum 1 to 2 days. This allows the application to be resilient to failure, ensuring that the business process is minimally affected by these failures.

Gap Analysis

Architecture and Design Pattern

MVC-based approach with .NET

- Utilizes the Model-View-Controller pattern.
- Separates the user interface (View), business logic (Model), and user input handling (Controller), enhancing modularity and maintainability.
- Focused more on the internal structure of the application.

Cloud-based dynamic website

- Architecturally focused on cloud infrastructure components like Internet Gateway, Load Balancer, Auto Scaling, Frontend and Backend Instances, Databases, Read Replicas, and Cache.
- Designed for scalability, performance, and resilience.
- Does not explicitly mention a design pattern like MVC for the application's internal architecture, focusing more on the deployment and scalability aspects.

Scalability and Performance

MVC-based approach with .NET

- Scalability is dependent on how the MVC application is hosted. It could be scaled manually or automatically based on the hosting environment.
- Performance optimization is primarily at the application level.

Cloud-based dynamic website

- Explicit focus on scalability through Auto Scaling and Load Balancing.
- Designed for high performance with features like caching, read replicas, and an Internet Gateway to reduce latency.

Ease of Maintenance

MVC-based approach with .NET

- Maintenance is more focused on the application's codebase. Updates and modifications are primarily at the code level.

- Hosting environment can impact the ease of maintenance.

Cloud-based dynamic website

- Cloud environment (e.g., AWS) offers tools for easier maintenance and updates with minimal downtime.
- Auto-scaling ensures optimal resource utilization without manual intervention.

Resilience and Reliability

MVC-based approach with .NET:

- Depends on how the application is deployed and managed.
- Resilience strategies need to be implemented at both application and hosting levels.

Cloud-based dynamic website:

- High resilience with features like automatic backups and quick recovery processes.
- The cloud infrastructure inherently offers better reliability with distributed resources.

In conclusion, the MVC-based approach with .NET is more focused on the internal structure of the application, ensuring modularity and maintainability at the code level. In contrast, the Cloud-based dynamic website approach emphasizes scalability, performance, and resilience through its cloud infrastructure, with a strong focus on deployment and operational aspects. The choice between these two depends on specific project requirements, such as the need for scalability, performance, security, ease of maintenance, and the existing technology stack.

Team Discussion

Name	Rationale
Nguyen Thanh Dat	I believe that the cloud-based solution offers significant advantages, especially in scalability and performance. From my perspective, its auto-scaling feature is a game changer, as it automatically adjusts resources in response to changing user demands. This adaptability is crucial in today's fast-paced digital environment, where traffic can be unpredictable. Additionally, I appreciate the enhanced performance through features like low-latency access and efficient data handling, ensuring a consistently fast and reliable user experience. Regarding maintenance and security, I find the cloud-based approach superior. The comprehensive infrastructure provided by cloud services simplifies tasks like system updates and backups, making management more straightforward for me. Security-wise, the isolation of sensitive data and administrative processes in private subnets gives me peace of mind. Moreover, the resilience of this system, with its automatic backups and rapid recovery

	capabilities, minimizes disruptions due to system failures. This blend of ease of maintenance, robust security, and resilience makes the cloud-based solution an attractive option for me, particularly when looking for a flexible, reliable, and efficient digital infrastructure.
Pham Anh Vu	Personally, I think the Cloud-based solution is the best fit for the gathered requirement of a secured, highly functional and scalable e-commerce website. While the MVC .NET solution allows for a cheaper, easier to deploy solution, it falls short of the scalability, dynamic capability and security requirements provided by the stakeholders, who are willing to sacrifice some time and capital in exchange for better quality, scalability and security.
Pham Duc Linh	After discussing and comparing the different utilities of the models in the first meeting, I still oppose my bias toward the MVC methodology. The team has previous experiences in similar projects using MVC model; as a result, less effort will be needed to execute tasks by this solution.
Tran Tuan Nam	From the critical discussion in our daily meeting, I believe that the Cloud-based solution better fits the requirements. The problem with scalability is crucial for a business like Top Deal Auto. The Cloud-base solution efficiently tackles this problem. This is key to my approval for this solution direction.
Phung Xuan Tung	While MVC .NET has some advantages, such as easy debugging and deployment, it also has some drawbacks, such as being outdated, platform-dependent, and monolithic. On the other hand, cloud-based solutions are more modern, scalable, and flexible, as they allow web applications to run on any operating system, use microservices architecture, and pay only for the resources they consume. Therefore from our , cloud-based solutions should be chosen for the project, as they offer better performance, reliability, and cost-effectiveness than MVC .Net
Final decision	After thorough discussion and evaluation, our team unanimously agrees that the cloud-based solution is the optimal choice. We collectively recognize its superior scalability, performance, and robust security measures as key factors in meeting our project's needs. The consensus is that this approach will not only streamline our maintenance efforts but also ensure resilience and flexibility for our evolving digital requirements.

Appendix A: Meeting Summary

First Daily Meeting Summary
Vu's design:
3 tier design (Front-End, Back-End, Database)
Front-end: public subnet (for users) (UI,display)
Back-end: private subnet (for admin) (Business logic, transaction handling)
Database: 2nd private subnet
Auto scaling
Load balancer for both FE and BE
Add more microservices

Need more info about the customer's needs ?
Admin functionalities ?

Dat's design:

Microservices
User -> FE service -> authentication
React
Admin -> Admin UI (FE)

Linh's design:
MVC: faster dev time, lower cost to other directions

Familiar with team members

Scalability problems
Deliver the product, no maintenance.

Nam's design:

MVC ASP.Net framework (C#)
Lack of Admin functionalities
Scalability (Azure)