CS4227 GROUP B-a11 REFLECTION REPORT By Konan Heney (20242603) & Jai (20045247)

In software development, selecting appropriate architectural patterns for system building is foundational for robustness, scalability, and maintainability. Architecture patterns provide time-tested resolutions to repeated design challenges that help developers better organize their application's structure and behaviour. The Google Maps API is explored along with the MVC architecture pattern in this document.

The modern software development world especially on web and mobile applications has seen the MVC architecture pattern become a mainstay over time since its inception in the late 1970s. An MVC application consists of three interconnected components namely a Model that represents data and business logic; a View which presents the UI to the user; and a Controller that handles user input and connects the Model with the View. It encourages the separation of functionality within an application by making use of a modular programming approach thereby promoting code reusability and testability hence potential changes to codes can be isolated.

Simultaneously, the Google Maps API has developed into a useful tool used to integrate mapping as well as location-based services across different platforms. Google Maps API is a programming interface that provides developers with important tools to design interesting and informative location apps. The user experience of an app can be improved by adding Google Maps functionality, which also enables geospatial analysis as well as other location-related applications.

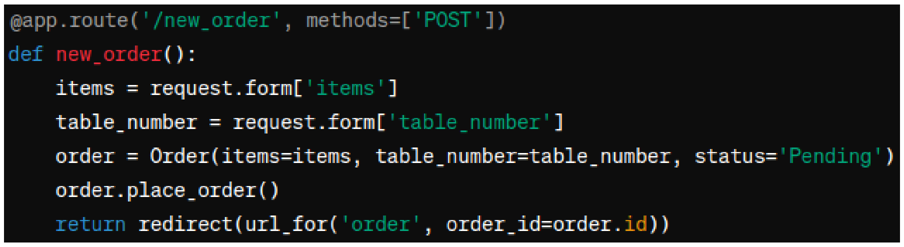
This paper looks into the nitty-gritty of fusing MVC architecture, Google Maps API, HTML with Jinja2 and HTTP; this involves exploring best practices, design considerations and implementation strategies. The application's components structure in line with the MVC pattern will be discussed to ensure maintainability and extensibility through a clear separation of concerns. Furthermore, we shall discuss integration techniques into the View layer such as embedding interactive maps, displaying markers or handling user interactions like in an application called "Fuel Price Comparison".

We will have code examples and practical recommendations for developers. This document will highlight differences between the two projects envisaged; different development strategies and general outcomes of both applications.

What architectures or project management methods/techniques did you use in your FYP? How was your experience? (Jai)

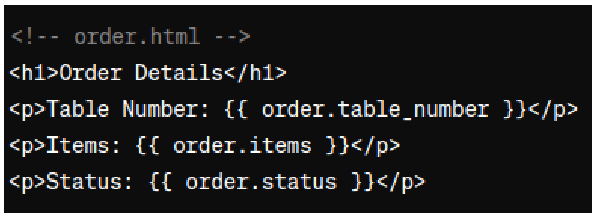
The Model-View-Controller (MVC) design pattern can be described as software architecture which is used in a designing process for the development of application components that communicate among themselves.
This isolation gives the flexibility to deal with alleged missions, saves configurable code and is a crucial tool for cultivating many components in parallel. With that, I will show you how the MVC approach can be applied to Flask, a micro web framework written in Python, to develop a restaurant management system.
Components of MVC

Model: Serves as the link between data and the application as to the logic layer of the application. The restaurant management scheme could be outlined as a grasp of data structure for the menus, orders, reservations, and accounts of users. It talks to the data repository to get, save, and change information by the commands of the controller.



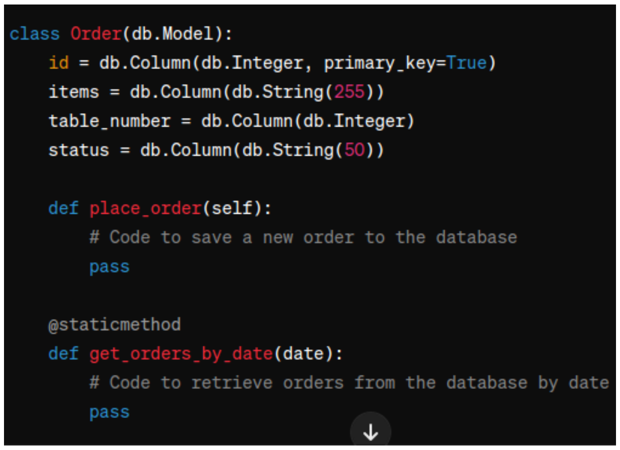
```python
@app.route('/new_order', methods=['POST'])
def new_order():
    items = request.form['items']
    table_number = request.form['table_number']
    order = Order(items=items, table_number=table_number, status='Pending')
    order.place_order()
    return redirect(url_for('order', order_id=order.id))
```

View: The view has the role of the presentation layer, which deals with both the users and the system simultaneously. It displays the data (you can see it in the Model) to the user and the user commands (e.g. button click, form submit) are sent to the controller. In Flask, templates (often called "themes" or designs composed of HTML with Jinja2 templating language) can serve as a view showing the menu of restaurants, reservation forms, or order details in a user-friendly way.



```html
<!-- order.html -->
<h1>Order Details</h1>
<p>Table Number: {{ order.table_number }}</p>
<p>Items: {{ order.items }}</p>
<p>Status: {{ order.status }}</p>
```

Controller: Plays the role of an agent that is responsible for the interaction between a model and a view. It handles all the requests by the users, searches through the view, eliminates the unwanted ones, and then returns the output once again to the view. In Flask, the route functions are commonly used as the controller, which responds on specified endpoints to HTTP requests.



```python
class Order(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    items = db.Column(db.String(255))
    table_number = db.Column(db.Integer)
    status = db.Column(db.String(50))

    def place_order(self):
        # Code to save a new order to the database
        pass

    @staticmethod
    def get_orders_by_date(date):
        # Code to retrieve orders from the database by date
        pass
```

What architectures or project management methods/techniques could have helped you in your FYP? What did you do in contrast to these, where would you have applied them in hindsight? (Jai)

In my project, I utilized the Model-View-Controller (MVC) architecture, which was crucial for keeping the application's data handling, user interface, and business logic separate and manageable. This framework made it easier to update parts of the application without affecting others, enhancing manageability and development efficiency. Reflections on MVC:
Organization: MVC helped me maintain clarity and separation within the project.
Flexibility: It allowed for independent updates to the UI or business logic.

Considering Microservices:
Though MVC served me well, in hindsight, integrating microservices could have offered additional benefits, particularly in scalability and flexibility. Microservices break down an app into smaller, focused services, each with a specific function, enhancing the project in several ways:
Scalability: They allow for the independent scaling of application components, optimizing resource use.
Agility: Microservices enable quicker updates and deployments for each service.
Technology Fit: This approach allows choosing the best technology stack for each service's needs.

MVC provided a solid foundation for my project, but exploring microservices could have elevated its scalability and development agility, offering a more nuanced approach to project architecture.

Did you use/experience any of the architectures or project management techniques during your time in a corporation? In which background? What are your experiences? (Jai)

In the corporate environment of LM Ericsson, the adoption of Agile methodologies was a strategic choice that significantly impacted project management and execution. Agile's core principles of flexibility, iterative development, and stakeholder collaboration were especially pertinent in the fast-evolving telecom industry. This approach enabled teams to work in sprints, allowing for frequent reassessment of project goals and adjustments based on real-time feedback and changing market demands. The experience with Agile at Ericsson underscored its advantages in fostering a dynamic and responsive work culture. Teams could rapidly prototype, test, and refine products, aligning development closely with customer needs and technological advancements. This iterative cycle not only enhanced product quality but also improved team morale by offering a clear sense of progress and achievement. However, implementing Agile also brought to light the importance of maintaining clear communication and a unified vision among team members and stakeholders. It demanded a shift from traditional hierarchical structures to more fluid and cross-functional team dynamics, which could be challenging but ultimately rewarding. Reflecting on the use of Agile within LM Ericsson, it's clear that this methodology offers significant benefits in terms of adaptability, efficiency, and customer alignment. Compared to more linear and segmented project management approaches, Agile facilitated a more engaged and flexible development process, crucial for staying competitive in the telecommunications sector.

What architectures or project management methods/techniques did you use in your FYP? How was your experience? (Konan)

Architecture patterns play a crucial role in shaping the structure and behaviour of software systems, guiding how components should be organized and interact to fulfil the application's requirements. In the case of the "Fuel Price Comparison" application, which aims to assist users in finding the nearest fuel station with the cheapest available fuel, the selection of appropriate architecture patterns is pivotal to ensure scalability, reliability, and maintainability. This essay explores various architecture patterns suitable for different aspects of the project, including website hosting, Google Maps API integration, location database management, and the mobile application interface.

Website Hosting:
For hosting the web application, a common architectural pattern is the Client-Server model. In this pattern, the client (user's web browser) interacts with the server hosting the application. The server processes user requests, queries the database, and serves the requested content back to the client. To ensure scalability and reliability, a distributed architecture can be employed, where multiple server instances are deployed across different geographical locations. This approach facilitates load balancing, fault tolerance, and improved responsiveness for users accessing the application from various regions.

Google Maps API Integration:
To integrate mapping functionality into the application, the Adapter pattern can be employed. The Google Maps API serves as an external service that the application needs to communicate with. By using the Adapter pattern, the application can encapsulate the complexities of interacting with the Google Maps API behind an interface that is tailored to the application's specific needs. This abstraction layer simplifies integration, promotes modularity, and allows for easier maintenance and future updates.

Database for Locations:
Managing the database for storing fuel station locations and pricing data requires a robust architecture that ensures data integrity, scalability, and efficient querying. The Repository pattern is well-suited for this purpose. In this pattern, a repository acts as an intermediary between the application's business logic and the underlying data storage. By abstracting away the details of data access and manipulation, the repository enables the application to remain agnostic to the specific database technology being used. This facilitates easier migration to different database systems and promotes testability by enabling the use of mock repositories in unit tests. The database will also be used to store the values of the prices of Petrol or Diesel to make them readily viewable to the user.

Mobile Application Interface:
For the mobile application interface, the Model-View-Controller (MVC) pattern can provide a structured approach to organizing the codebase. In MVC, the Model represents the data and business logic, the View handles the presentation layer, and the Controller orchestrates the interaction between the Model and the View. By separating concerns in this manner, the application becomes easier to maintain, as changes to one component do not necessitate modifications to others. Additionally, the MVC pattern facilitates code reuse and promotes a clean architecture that enhances readability and scalability.
The "Fuel Price Comparison" application can benefit from the adoption of various architecture patterns to address different aspects of its design and implementation. By leveraging patterns such as Client-Server, Adapter, Repository, and MVC, the application can achieve its objectives of providing users with accurate and timely information on fuel prices and locations while maintaining scalability, reliability, and maintainability. Effective architecture patterns not only contribute to the success of the current project but also lay the foundation for future enhancements and expansions of the application.

What architectures or project management methods/techniques could have helped you in your FYP? What did you do in contrast to these, where would you have applied them in hindsight? (Konan)

For the “Fuel Price Comparison”, I used the Model-View-Controller (MVC) architecture, which was important for keeping the application's data handling, user interface, and business logic separate. Reflections on MVC for my project:
Organization: The MVC helped me separate my project.
Flexibility: It allows for independent updates to the UI or business logic.

In the "Fuel Price Comparison" application, caching could have been implemented to store commonly accessed data, such as fuel station locations, prices, and mapping information, thereby reducing the need to repeatedly fetch this data from external sources.

As fuel prices may not change frequently, you can cache the fetched fuel price data locally in the application's database or in-memory cache.
Set an appropriate expiration time for the cached data to ensure that it remains up-to-date. You can periodically refresh the cache based on the frequency of price updates from your data sources.
Implement cache invalidation mechanisms to handle cases where fuel prices are updated or changed unexpectedly, ensuring that users always have access to the most accurate and up-to-date data.

Did you use/experience any of the architectures or project management techniques during your time in a corporation? In which background? What are your experiences? (Konan)

During my internship, I used multiple systems for team collaborations, project development and communications. To communicate with fellow team members, we used Mattermost. This service included the features of team chats, known as ‘channels, and direct messages between team members. This was a useful service as it was always enabled and automatically synchronised between your devices. When the team needed to have a video call, Microsoft Teams was utilised. The team were constantly pushing changes, so in this case, GitHub was used thoroughly. Using these software services combined, allowed for a seamless flow of information, and therefore, everybody was always informed of any changes or updates made to a project.

Comparison between the 2 Projects (Konan and Jai)

MVC: A Shared Foundation

Both of them however were advocates of Model-View-Controller (MVC) architecture, appreciating its ability to separate data, logic and presentation. This commonality denotes their understanding of the importance of MVC in making a project less complex, more maintainable and modular. The difference between them is this moment when they think about it and ask themselves what would have been different if they chose otherwise.

Road to Microservices and Beyond

His reflection on microservices indicates that he was thinking about scalability and agility in terms of development. It can be inferred from this comment that he wanted an architecture that enables independent scaling up or down quickly as well as updating each component without affecting all other components. Conversely, Konan's research went further into specific architectural patterns for a Fuel Price Comparison app pointing out the wider exploration for scalability, reliability and maintainability solutions.

Corporate Experiences: Agile vs Collaboration Tools

In the corporate sector, Jai has written about how LM Ericsson benefitted from adopting agile methodologies. In highlighting his journey with agile techniques, Jai shows the flexibility that is needed in the fast-paced telecoms sector. On the other hand, Konan focused on recounting how practical tools supported team collaboration and project management during his internship. This emphasis on tools like Mattermost and GitHub illustrates a pragmatic approach to ensuring efficient teamwork and project continuity.

Conclusion

To sum up, in the world of software developers like Jai and Konan, the Model-View-Controller (MVC) architecture is a basic building block for constructing scalable, maintainable applications. Their retrospections discuss the way from the theoretical to practical levels how MVC can be used effectively in different project contexts such as Jai's well-oiled management of a restaurant system through to Konan's "Fuel Price Comparison" invention. Furthermore, their corporate life travels explain that there are diverse ways people approach work with Agile being embraced by Jai at LM Ericsson on the one hand and the other hand Konan using collaboration tools for smooth project running. Moreover, this tale demonstrates not only the various applications of MVC and other architectural patterns but also how software development practices have been evolving according to the needs of projects and industry requirements. By sharing their own experiences these two individuals enrich our understanding of structure, technology and teamwork within the software engineering context.

Bibliography

1. Deacon, J., 2009. Model-view-controller (MVC) architecture. *Online][Citado em: 10 de março de 2006.] http://www. jdl. co. uk/briefings/MVC. pdf*, *28*, p.61.
2. Selfa, D.M., Carrillo, M. and Boone, M.D.R., 2006, March. A database and web application based on MVC architecture. In *16th International Conference on Electronics, Communications and Computers (CONIELECOMP'06)* (pp. 48-48). IEEE.
3. Dey, T., 2011. A comparative analysis of modelling and implementation with MVC architecture. *International Journal of Computer Applications*, *1*, pp.44-49.
4. Ping, Y., Kontogiannis, K. and Lau, T.C., 2003, September. We are transforming legacy Web applications to the MVC architecture. In *Eleventh Annual International Workshop on Software Technology and Engineering Practice* (pp. 133-142). IEEE.
5. Sunardi, A., 2019. MVC architecture: A comparative study between Laravel framework and slim framework in freelancer project monitoring system web-based. *Procedia Computer Science*, *157*, pp.134-141.
6. Verma, A., 2014. MVC architecture: A comparative study between ruby on rails and Laravel. *Indian Journal of Computer Science and Engineering (IJCSE)*, *5*(5), pp.196-198.
7. Mcheick, H. and Qi, Y., 2011, May. Dependency of components in MVC distributed architecture. In *2011 24th Canadian Conference on Electrical and Computer Engineering (CCECE)* (pp. 000691-000694). IEEE.
8. La, H.J. and Kim, S.D., 2010, November. Balanced MVC architecture for developing service-based mobile applications. In *2010 IEEE 7th International Conference on E-Business Engineering* (pp. 292-299). IEEE.
9. Tao, Y., 2002, November. Component-vs. application-level MVC architecture. In *32nd Annual Frontiers in Education* (Vol. 1, pp. T2G-T2G). IEEE.
10. Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J. and Babar, M.A., 2021. Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. *Information and software technology*, *131*, p.106449.
11. Ghofrani, J. and Lübke, D., 2018. Challenges of Microservices Architecture: A Survey on the State of the Practice. *ZEUS*, *2018*, pp.1-8.
12. De Lauretis, L., 2019, October. From monolithic architecture to microservices architecture. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (pp. 93-96). IEEE.
13. Salah, T., Zemerly, M.J., Yeun, C.Y., Al-Qutayri, M. and Al-Hammadi, Y., 2016, December. The evolution of distributed systems towards a microservices architecture. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)* (pp. 318-325). IEEE.
14. Jaramillo, D., Nguyen, D.V. and Smart, R., 2016, March. Leveraging microservices architecture by using Docker technology. In *SoutheastCon 2016* (pp. 1-5). IEEE.
15. Surianarayanan, C., Ganapathy, G. and Pethuru, R., 2019. *Essentials of microservices architecture: Paradigms, applications, and techniques*. Taylor & Francis.
16. Amaral, M., Polo, J., Carrera, D., Mohomed, I., Unuvar, M. and Steinder, M., 2015, September. Performance evaluation of microservices architectures using containers. In *2015 14th International Symposium on Network Computing and Applications* (pp. 27-34). IEEE.

17. O'Connor, R.V., Elger, P. and Clarke, P.M., 2017. Continuous software engineering—A microservices architecture perspective. *Journal of Software: Evolution and Process*, *29*(11), p.e1866.
18. Mateus-Coelho, N., Cruz-Cunha, M. and Ferreira, L.G., 2021. Security in microservices architectures. *Procedia Computer Science*, *181*, pp.1225-1236.
19. Munaf, R.M., Ahmed, J., Khakwani, F. and Rana, T., 2019, March. Microservices architecture: Challenges and proposed conceptual design. In *2019 International Conference on Communication Technologies (ComTech)* (pp. 82-87). IEEE.