Potential System Architecture

Sustainable Work through Women-in-tech Application for Older Women in Malaysia and Thailand: Integrating Action Research and Design Science Approach

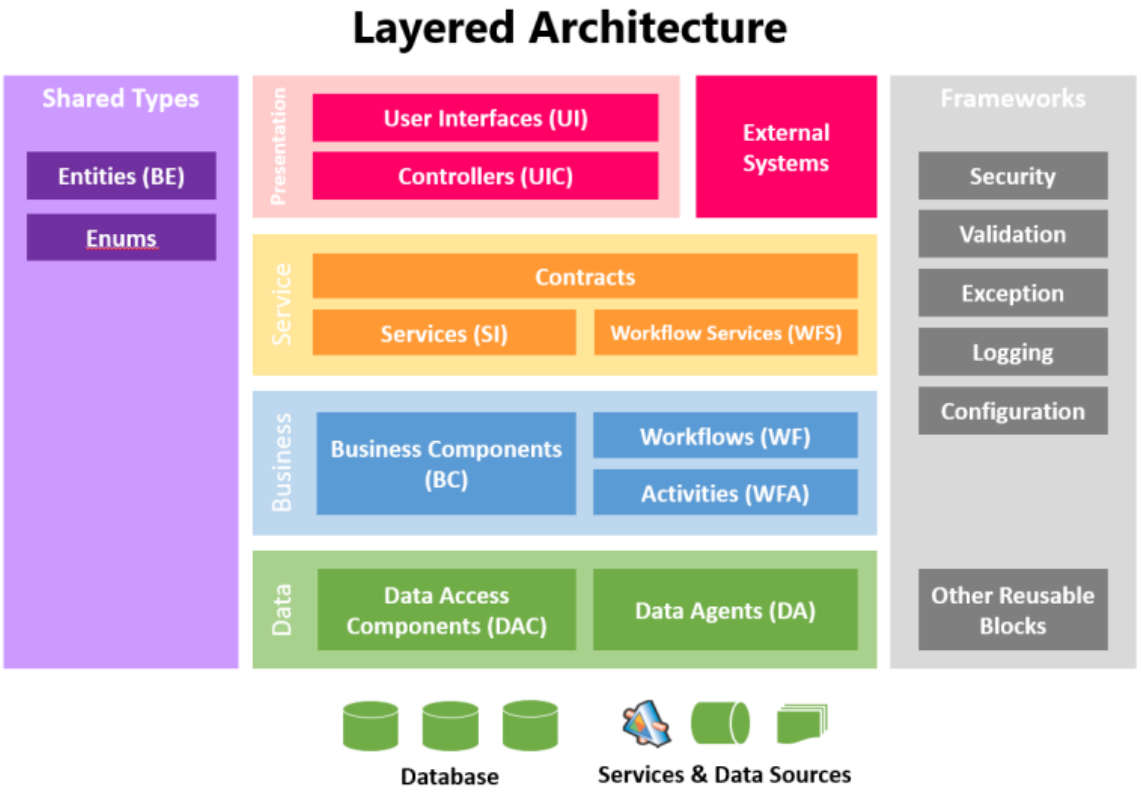
Introduction

The software architecture patterns that are taken into consideration are layered architecture and microservices architecture. The layered architecture, which is also known as the n-tier architecture is our first choice, since it's main benefit is that it may be a good starting point when it is hard to determine which pattern suits the application features and functionalities the best.The basic idea behind this architecture is that it's components are classified into horizontal layers where each layer has their own role and responsibility. The layers are closed, which means that they are isolated from random access and thus they can be accessed only from the layer which is right above or right below the chosen layer. The application built following the layered architecture may be monolithic and hard to deploy, as well as it may have low performance and low agility. However, after the brainstorming, it was concluded that all of these risks are not critical issues for our project since they are easily resolvable considering the scale of our app.

Next, our second option is microservices architecture which is a method of developing software systems that focus on building single functionality modules with well-defined interfaces and operations. Each component in this architecture is deployed as separate units in order for easier deployment through an effective and streamlined delivery pipeline, increased scalability, and a high degree of application and component decoupling within the application. Furthermore, it is a distributed architecture in which all the components within it are fully decoupled from one other and accessed through some sort of remote access protocol. It also provides the ability to do real-time production deployments, thereby significantly reducing the need for the traditional monthly deployments.

Potential Architectures

Potential Architecture 1: Layered/n-Tier Architectural Pattern



As it was mentioned in the introduction the main idea behind the layered architecture is that it's components are organized into horizontal layers, each layer performing a specific role within the application. The number of layers may vary depending on the scope of the application, however there are five default levels that may be highlighted:

- **Presentation layer:** responsible for interaction with users through screens, forms, menus, etc and browser communication logic. It is the most visible layer of the application, since it defines the application interface
- **Functionality layer:** presents the functions, methods, and procedures of the system based on the business rules layer.
- **Business rules layer:** This layer contains rules that determine the behavior of the whole application, since it executes specific business rules associated with each request.
- **Application core layer:** contains the main programs, code definitions, and basic functions of the application. Programmers work in this layer most of the time.
- **Database layer:** This layer contains the tables, indexes, and data managed by the application. Searches and insert/delete/update operations are executed here.

The whole process is based on one level accepting the certain request which is sent from another level and responding to it in a proper way. Since the scope of each layer is very clear it makes it easier for the developer to figure out the role and responsibilities of each layer and understand the connections between them due to the code's traceability. Another important advantage of the n-Tier architectural pattern is that it makes the application easier to test, because the code is more modular and defining independent units is simple. Moreover, even though modification and extension of the program requires a very deep analysis, the autonomous layers make the program easier to extend, since the group of changes in one layer does not affect the others.

Properties and their pros and cons

Property	Description	Advantage	Disadvantage
Separation of concerns among components	Components within a specific layer deal only with logic that pertains to that layer	- easy to build effective roles and responsibility models into your architecture - makes it easy to develop, test, govern, and maintain applications - well-defined component interfaces and limited component scope.	-
Closed layers/Layers of isolation	As a request moves from layer to layer, it must go through the layer right below it to get to the next layer below that one.	- changes made in one layer of the architecture do not impact or affect components in other layers: the change is isolated to the components within that layer, and possibly another associated layer. - each layer is independent of the other layers, thereby having little or no knowledge of the inner workings of other layers in the architecture -reducing coupling and interdependencies between components	-lower speed and therefore lower performance

Open layers	If the layer is marked as open, meaning requests are allowed to bypass this open layer and go directly to the layer below it.	- Helps define the relationship between architecture layers and request flows and -Provides designers and developers with the necessary information to understand the various layer access restrictions within the architecture	-if not communicated properly which layers are open and which are closed and why may result in tightly coupled and brittle architectures that are very difficult to test, maintain, and deploy due to the lack of layer isolation

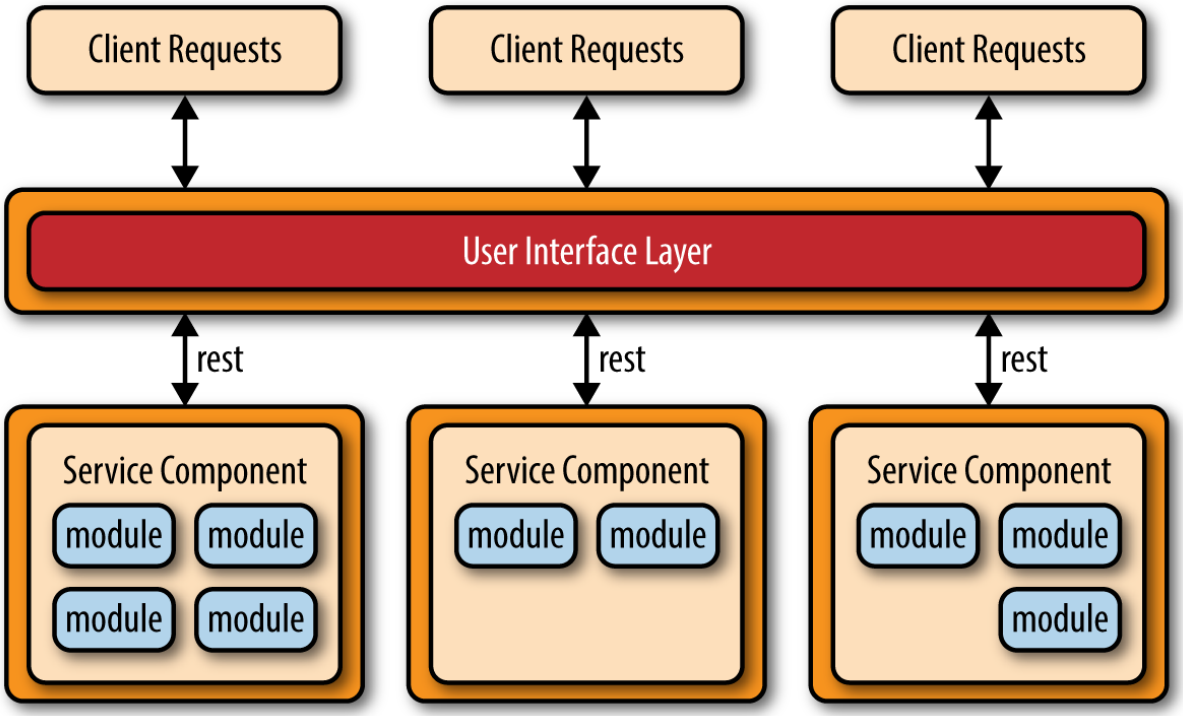
Risks

1. **Risk event 1**: *Tends to lend itself toward monolithic applications*: A monolithic application is built as a single unit. It does pose some potential issues in terms of deployment, general robustness and reliability, performance, and scalability.
2. **Risk event 2:** *The architecture sinkhole anti-pattern:* the situation where requests flow through multiple layers of the architecture as simple pass-through processing with little or no logic performed within each layer. Every layered architecture will have at least some scenarios that fall into the architecture sinkhole anti-pattern.
3. **Risk event 3:** *Low agility (ability to respond quickly to a constantly changing environment:* difficult and time consuming to make changes in this architecture pattern because of the monolithic nature of most implementations as well as the tight coupling of components.
4. **Risk event 4:** *Low ease of deployment:* Usually becomes an issue for large applications. Results in the situation when one small change to a component can require a redeployment of the entire application (or a large portion of the application).
5. **Risk event 5:** *Low performance:* The inefficiencies of having to go through multiple layers, since each layer in the upper levels must connect to those in the lower levels for each operation in the systeem, may worsen the performance.

Risk Event	Probability	Impact	Risk Response
Risk event 1	80%	High	A monolithic application means a single-tiered software app in which the UI and data access code are combined into a single program from a single platform. The following risk may be not a problem, since the application that we are developing is self-contained and

			independent from other computing applications, which makes the fact that it's monolithic unproblematic
Risk event 2	20%<	Low	Make some of the architecture layers open
Risk event 3	40%	Medium	Each change requires deep analysis
Risk event 4	10%	Low	Since our application is not considered a large one, it's highly possible that the following problem will not occur.
Risk event 5	20%	Low	The deterioration of the performance should be invisible for the end user

Potential Architecture 2: Microservices Architecture



Microservices architecture is an architecture pattern that emphasises on a distributed architecture by having a system made up of separately deployed units. The client communicates through a user interface layer which in turn communicates with the required service components which contain modules. As a result, the components within the architecture will be fully decoupled which allows the individual modules to be able to be developed and deployed separately.

This architecture pattern closely resembles our app structure, with well-defined modules within the application which can be worked on independently. Development of modules such as the Chatbot and the forum becomes easier due to the isolated and smaller scope. They also can be worked on and deployed independently and do not rely on each other as the user only interacts with the user interface layer. Besides, since there is no coupling between modules, the project can scale easily without breaking the existing code in the event where more features are required to be added to the application.

Properties and their pros and cons

Property	Description	Advantage	Disadvantage

Usage of abstract user interface layer	Client never communicates with the service components directly, but through an abstract user interface layer.	- Service components which have issues/need to be redeployed do not cause the whole system to be down - User can just be redirected to an error or waiting page	-
Distributed nature of services	Service components within the architecture are fully decoupled from one another and are accessed through a remote protocol.	- Allows for continuous deployment - Provides ease of deployment and development as modules can be worked on separately - Scalability of system is high - Solves common issues found in monolithic applications - Ease of testing	- Not a high-performance architecture - Difficult to maintain a single transactional unit of work across service components - Shares certain complex issues found in event-driven architecture

Risks

1. **Risk event 1**: *Service components are too interdependent on each other*. This introduces many dependencies, losing the benefits of the architecture.
2. **Risk event 2**: *Scope of service components are too large*. This violates the Single Responsibility Principle.
3. **Risk event 3**: *Scope of service components are too small*. The whole system becomes very complex and has a higher number of dependencies between them.
4. **Risk event 4**: *Certain tasks cannot be split into independent units*. Some components cannot be broken down into smaller pieces which makes the component become too large.

5. **Risk event 5:** *Low performance.* Due to the distributed nature of the architecture, this pattern naturally does not allow high-performance applications.

Risk Event	Probability	Impact	Risk Response
Risk event 1	20%	Medium	Combine similar service components into a larger service component.
Risk event 2	60%	Medium	Service components can be refactored and broken down into more finely divided parts.
Risk event 3	30%	Low	Service components can be refactored and combined together to form larger service components.
Risk event 4	80%	Medium	Conduct deep analysis to try and break down components into smaller pieces.
Risk event 5	10%	Low	Deterioration of performance should not be visible to the end user as it is not a high-performance application.

Potential Languages and Frameworks

Potential Full-Stack Framework

Full-Stack Framework	Advantages	Disadvantages
Python		

DJango	• Batteries-included framework
	  - Consist of readymade packages
	  - Saves time from writing code from scratch
	• Highly secure
	  - Has protection against security attacks like XSS, SQL injections, Clickjacking, etc
	  - Hides website's source code
	  - Always encrypts passwords and essential information with a security key during data transmission
	• Python libraries
	  - REST for building application programming interfaces
	  - CMS for managing website content
	• Object Relational Mapper
	  - Library that automatically transfers data stored in databases into objects commonly used in application code
		• Multiple requests issue
		  - Unable to handle multiple requests at the same time
		  - Slow development
		• Monolithic framework
		  - Hard to use due to limited dependencies
		  - Focus on providing code-oriented programming instead of learning packages and tools

	- Developers don't necessarily have to know the language used for database communication - Allows to create tables using classes and insert data with the help of objects of the class	
Web2py	● Batteries-included - Provide package with SQL database, web-based interface and fast-multithreaded web server ● Low running time - Able to run Python compiled code	● Does not support unit testing ● Poor IDE support - Unable to use standard Python development tools without modifications ● Unable to see error instantly - Need to navigate to ticket to check error when exception occured
Pyramid	● Simplicity - Easy to use even if no background at all ● Flexibility - Get to choose templating language, libraries, or database layers.	● Hard to maintain - Treats the whole web application as single file
Javascript		
Node.js	● Fast-processing - Use V8 engine compiles JavaScript into native machine code directly results in higher speed and more efficient code execution ● Asynchronous I/O - Able to perform multiple operations concurrently	● Immaturity of tooling - Many tools are not properly documented or tested ● Callback issueRely too much on callbacks ● Nested callbacks within other callbacks might impact code quality ● Difficult to understand and maintain

	- popular choice for chats that requires constantly updated data ● Supports REST API	
Sails.js	● Object Relational Mapping tool(Waterline) - Supports multiple databases - Easy to query and manipulate data without writing vendor-specific integration code - Able to store data in anywhere or any databases ● Fast REST API development - Blueprints allows Auto-generate APIs ● Supports websocket without additional code - translates incoming socket messages - automatically compatible with every route in app - Able to get real time interaction - WebSockets provide a persistent connection between a client and server that both parties can use to start sending data at any time	● Slow framework

Potential Frontend Framework

Frontend Framework	Advantages	Disadvantages
Javascript		
ReactJS	• Reusability of components makes it easy to collaborate and reuse them in other parts of the application • Consistent and seamless performance with the use of virtual Document Object Model(DOM) • Advanced and useful tools	• Requires external libraries • Mixing HTML into JS (JSx) • Hard to make proper documentation, which may be confusing for beginner developers
Vue.js	• Extensive and detailed documentation • Simple syntax for developers with JS background • Flexibility to design the app structure • Advanced and useful tools	• Lack of information about it due to its massive use in non-english speaking countries • Lack of support for large-scale projects • Limited resources compared to other frameworks
jQuery	• Simplicity • Lightweight, at only 30 kB minified and zipped • Cross-browser compatible with Chrome, Edge, Firefox, Internet Explorer, Safari, Android, and iOS • CSS3 compliant	• Slow performance • APIs of the document model are obsolete • Open source - vulnerable to any modifications • Difficult to debug if developer does not have a strong js foundation

Ember.js	• Well-organised • Fastest framework • Two-way data binding • Proper documentation	• Heavy framework for small applications • Very hard to learn for non-professional developers • Complex syntax and small updates
Angular	• Prominent features like two-way data binding are provided by default, which reduces amount of code • Reduces coupling and dependencies • Dependency injection makes the components reusable and easy to manage	• Very hard to learn for non-professional developers • Not the highest performance: may be laggy and inconvenient due to its size

Potential Backend Framework

Backend Framework	Advantages	Disadvantages
Python		

Flask	• Simple development
	  - Framework is easy to understand
	  - Good for beginners who have a background in Python
	• Flexibility
	  - Most parts of Flask can be changed due to the simplicity and minimalism of the framework
	• High performance
	  - Few levels of abstraction which allows better performance
	• Good documentation
	  - Comprehensive, well-structured documentation full of examples
	• Integrated testing
	  - Flask for web development allows for unit testing through its integrated support
	• Many resources available online
	  - One of the most popular frameworks
	  - Many libraries and plugins for Flask (eg. SQLAlchemy)
	  - Many guides, tutorials and due to large community
		• Code is not standardized
		  - More difficult to write code for large projects
		  - Easy to learn but difficult to adjust to different Flask projects
		  - Inexperienced developers will write worse code because framework is not standardized
		• Does not scale well
		  - Flask has a singular source
		  - Handles requests in turns
		  - If serving multiple requests, may take a longer time
		  - Larger applications do not scale well
		• Cannot handle asynchronous programming
		  - Flask is specifically not designed to not handle async programming

CherryPy	• No external server needed during deployment	• Poor documentation

	- Comes with a production-ready WSGI server ● Simple development - Easy to understand for developers with Python background ● Can help organize code structure - Provides some dispatcher patterns that can help you organize your code	- Lack of comprehensive documentation which can be tough for new users ● Lack of online resources - Not too popular framework - Lesser resources and guides - Reduced response from community experts
TurboGears	● Multi-database support ● Has a transaction manager to help with multi-database deployments ● Modular - Composed of middlewares, simple to put together the way you want	● Poor documentation - Lack of comprehensive documentation which can be tough for new users ● Lack of online resources - Not too popular framework - Lesser resources and guides - Reduced response from community experts
Tornado	● Can provide real-time functionality ● Can support thousands of simultaneous connections - For services such as distributed chat and multiplayer games	● Relatively poor documentation ● Lack of online resources - Not too popular framework - Lesser resources and guides - Reduced response from community experts
Javascript		
ExpressJS	● Simple development - No exotic language structure - Developers who have experience with Javascript will find this easy ● High performance	● Language is not opinionated - Beginners may not know optimal ways to structure their code - May lead to security vulnerabilities ● Event-driven nature

	● Easy scalability
	  - Add additional nodes to the system
	● Good documentation
	  - Comprehensive, well-structured documentation full of examples
	● Many resources available online
	  - Large number of packages available on NPM
	  - Easy integration of third-party services and middleware
	● Great at I/O request handling
		- Developers who have worked with other languages may find it difficult to understand the callback nature

Conclusions

Recommended Architecture Model

Layered architecture is the architecture that was chosen after analysing all the pros and cons and comparing it with other architectures. This architecture is the right choice for developers which are not experienced in developing the web app following certain architecture models, which is our case. Besides that, other architecture models have a higher degree of complexity, which may complicate the development process. In addition, the disadvantages of layered architecture are not too disruptive to the goals of our system. For example, having low performance is one of the cons of this architecture, which may be a serious issue, since the app is expected to be used by the users through the UI and therefore, high performance is vital. However, the extent to which the performance of the app developed following layered architecture is lower than the performance of apps with other architectures is invisible for the end user, which makes this disadvantage not crucial.

Recommended Languages

Group	Task	Language
RedCow	Frontend and backend for forum module	Vanilla JS with Node.js integration
Twenty9	Frontend and backend for recommender system	Vanilla JS with Node.js integration
404	Frontend and backend for chatbot module	Vanilla JS with Node.js integration

Vanilla JS was chosen as the language to be used across all three teams. The teams came to a consensus after agreeing that most team members were more comfortable with vanilla JS as they have had previous experiences with the language. Besides that, another factor to help arrive at this decision was the lack of experience of team members with frameworks. Due to the need of team members having to learn the frameworks suggested, the benefits brought about by using frameworks were balanced out by the time taken to familiarize with the framework to be able to produce a working system.

However, we estimate that this project may scale to a larger size than manageable by vanilla JS alone. If it does arrive at this point, we decided to integrate a framework into our system to ease development. The framework we have chosen is Node.js for the backend as we do not expect the frontend to be unmanageable. On top of that, it is also possible to integrate Node.js with the existing vanilla JS system.