PAWDOPT

Architectural Documentation

Table of contents

Architectural design strategy

Component-based Decomposition- we divided the required system described in the tender document supplied by the industry mentor, into subsystems for our team to implement as component modules which, together, provide the functionality of the entire system. This allowed us to identify which architectural patterns would be suitable for each subsystem and which pattern would allow optimal integration for all the components in the larger system.

Architectural quality requirements

Q1: Reliability

The System allows for people to adopt dogs and ultimately impact their lives, therefore reliability is extremely important in the sense that information needs to be kept confidential and users should be held accountable in the event of false information.

- Users can report other users putting false information on the system to hold them accountable.
- It is expected that the system should be available 99.5% of the time. This ensures that the system be available for any anomalies that it detects and rightly informs the user, whilst also allowing a down time for developers to maintain and perform routine checks on the system.

Q2: Usability

The System deals with the livelihood of unowned Dogs and as a result a User needs to be able to use the full functionality of the system in order to prevent ambiguity and confusion in the process of an adoption.

- All unit and integration tests are complete to ensure each function performs as intended, as well as test users being able to perform any task they need to on the system easily.

Q3: Performance

Since users will interact with the system in many different ways that will require the backend of the system to return information, the backend of the app will have to be quick in loading and returning this requested information at all times.

- Users with a stable internet connection should not have to wait more than 30 seconds for information to be retrieved.

- The ML model should determine the breed within a space of 30 seconds.

Q4: Scalability

Since the limit for adopting dogs is infinite and dogs need to be adopted all over the world, the system must be able to handle an increasingly large amount of data and users.

- There will always be storage available in the event of a new user. No user should be unable to use the system due to storage constraints.
- The accuracy of predictions within the system should improve over time with the addition of data to the system through use to allow for different breeds around the world.

Q5: Security

Since users will be interacting with many different parts of the system and putting personal information on the system, the system must protect them and their information at all times.

- There should never be an instance whereby a person gets incorrect information about a dog.
- All possibly sensitive information requires the user's permission to make it visible. Sensitive information storage abides by POPIA.
- Confidential information will be encrypted and the site must resist 100% of attacks with malicious intent.

Q6: Testability

- Unit tests and E2E tests need to test that all app functions work as intended.
- Tests need to use abstraction and virtualisation such that tests can be run as transactions which can be rolled back at the end of testing. Virtualisation facilitates this through creating test environments with modifiable parameters to test various cases wherein the environment is entirely under the tester's control.
- Functions and tests should have detailed error logging and reporting as to assist in debugging and error tracing
- Structural complexity should be limited as to improve testability

Architectural styles

Layered Pattern

Explanation:

The layered pattern is an n-tiered pattern where the components are organised in horizontal layers and separated in terms of processing. All processes are interconnected but they are not dependent on each other.

Quality Requirements Addressed:

- Security: No single layer is dependent on another layer which allows for an instance in a layer to only have access to what they should.
- Scalability: This layered pattern is an **n-tiered** pattern and thus multiple components can be organised as many times needed by the development team and individual layers can be upgraded as needed.
- Testability: Will allows us to test the different layers independently.

How it fits in the system:

The system is built with the following layers:

- Layer 1 : User interface Layer - frontend pages and views
- Layer 2 : Business Logic Layer - System API, controllers and services and the repository
- Layer 3 : Persistence Layer - Database and ML model

Motivate the choice:

Separation of the database and ML model from the other components allows for increased security and privacy. The isolation of the UI from the application layer allows users to only have access to parts of the system they should have access to. This Pattern also allows for partitioning of concerns and reliability in the system as no layer depends on another.

Model View Controller (MVC) Pattern

Explanation:

MVC is known as an architectural pattern, which embodies three parts Model, View and Controller, or to be more exact it divides the application into three logical parts: the model part, the view and the controller.

Quality Requirements Addressed:

- Usability: The backend is a depiction of the controller and model, hence the quality of these attributes affect the view in which the user will use the system.
- Performance: The speed of access of the backend is directly affected by the Controller in this pattern hence performance is affected
- Security: The MVC pattern assists in assertion of security standards through separation of concerns. The pattern allows for the delivery of information related only to the requested view. This reduces the risk of unauthorised access to information that is not intended for the specific user.
- Reliability: This pattern directly interacts with the storage of data as well as the reporting of users, therefore this pattern can allow for checks to be done in the instance of unreliable use of the system.

How it fits in the system:

It will be used to design our mobile application's graphical user interface. We will have an interface whereby a connection with a Model (database and ML) will be made and a controller will manage data transfer between the two components.

Motivate the choice:

The MVC pattern provides a granular precision for what is presented to the user and allows for user interaction with the solution to be curated in a way that provides apt information that conforms to their needs.

Component Based Development Pattern

Explain:

Component-based development focuses on the decomposition of the design into individual functional or logical components that represent well-defined communication interfaces containing methods, events, and properties. It provides a higher level of abstraction and divides the problem into sub-problems, each associated with component partitions.

Quality Requirements Addressed:

- Testability: Since this pattern provides a high level abstraction and divides major problems into sub-problems, each component that has an issue to be addressed can be tested individually before the application and its logical components can be functional.
- Performance: Using a component based pattern provides improved performance through separation of concerns by allowing each component to fulfil its purpose without affecting the running of other components.

- Scalability: The separation of features into components allow features to be updated and implemented separately, this allows new features to easily be added or for horizontal scalability of the system through adding more components which perform the intended functionality.

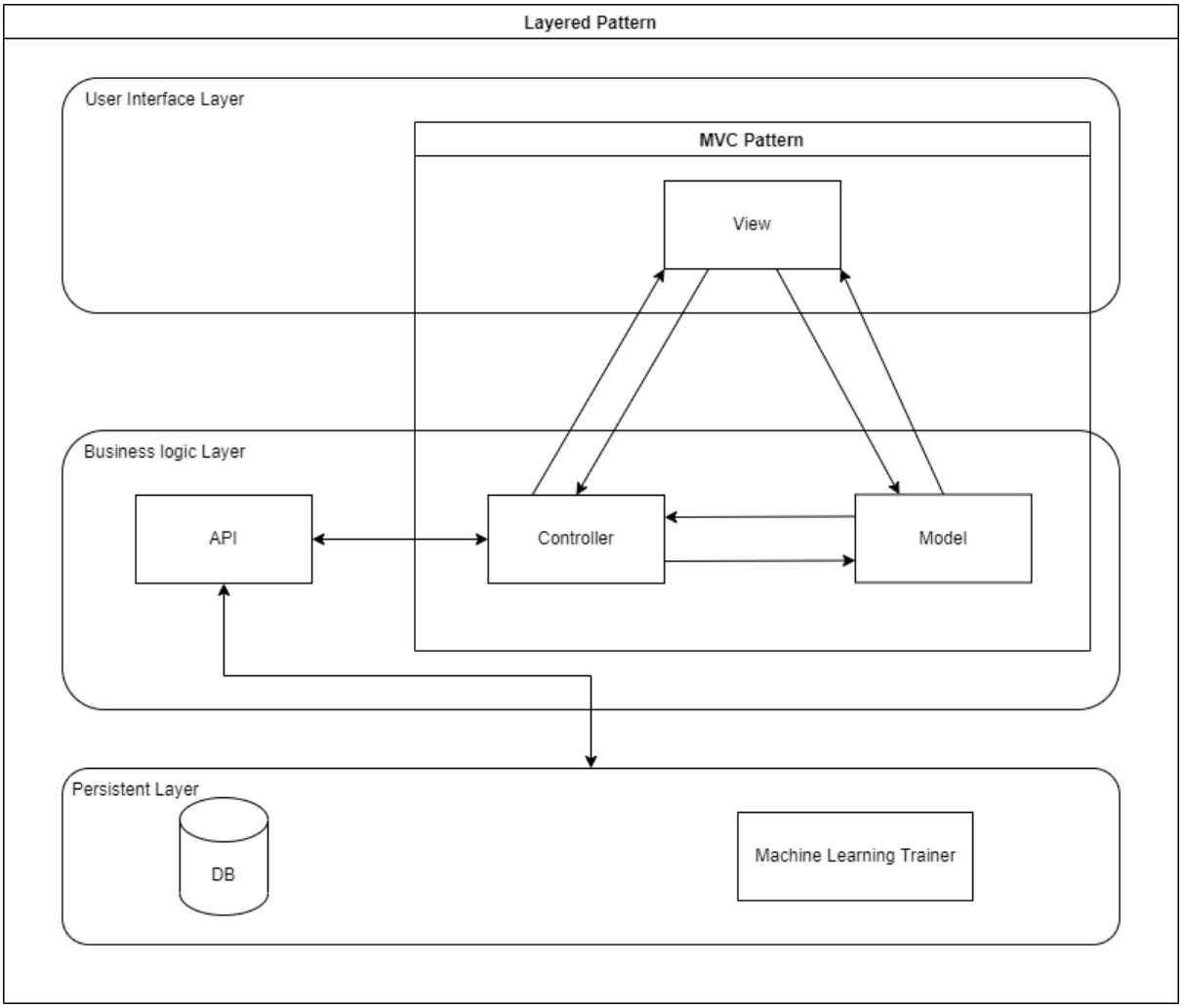
How it fits in the system:

Each of the subsystems has been designed as a component, which includes the database, user interface, ML model, repository and system API.

Motivate the choice:

This pattern introduces separation of concerns with respect to functional availability through dividing the system into separate parts which each provide their own prescribed service. This, in turn, protects the system should a single component fail or need to be updated in that it will not stop other components from functioning.

Architectural design & pattern



Architectural constraints

The Development team was constrained with using Ionic for the frontend of the system. Besides the previously stated constraint the team is not constrained to any particular hosting service or framework. The following System Architecture constraints include:

- All code and technologies used must be open source.
- The system must store many images with quick access.
- No user should be able to access a layer that they are not supposed to or a component they are not allowed to.

Technology choices

Ionic

Overview:

Ionic is a powerful HTML5 SDK that helps you build native-feeling mobile apps using web technologies like HTML, CSS, and Javascript. Ionic is focused mainly on the look and feel, and UI interaction of your app.

How it Fits the Architecture:

Using Ionic for the front end allows us to create a cross platform app that will function on Android and iOS with a single code base. We have decided to use an Angular implementation as we have recently worked on a project which uses this framework and we have used it extensively within the project.

Pros

- It fills the gap between Angularjs and hybrid mobile applications
- Allows you to use native UI component that are compatible with any operating system
- Allows you to transfer your ionic application to a desktop application

Cons

- It lacks suitability when with performance heavy applications, it will slow down

Angular

Overview:

Angular is a TypeScript-based free and open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations.

How it Fits the Architecture:

Angular was built with Model-View-Controller architecture and the framework synchronised the model and the view. This fits into our architectural design through and thus integration with routing between pages using the typescript environment with angular makes dynamic functions and pages for our application. This allows for view generation for users and facilitates the MVC implementation.

Pros:

- Angular provides a great selection of third-party integrations that can be added to the framework with ease (in our case the swiping cards, etc.)
- Angular compiles HTML and TypeScript into JavaScript during the development, which means all of the code is compiled before the browser even loads your web app making it more efficient to code.

Cons:

- Angular is a versatile environment and thus there is always more than one way to complete any task. This will produce some form of confusion among us software developers.
- Angular framework is quite difficult to learn with such a complex web of modules, coding languages, integrations and customising capabilities, understanding Angular definitely takes some time.

NestJS

Overview:

NestJS is a framework used for writing scalable, testable and loosely coupled applications. This allows better integration for the component-based nature of the project.

How it Fits the Architecture:

NestJS is a versatile backend based on Javascript. It will integrate well with Ionic as we will be using the Angular framework, which uses Typescript. This means that the front and backend of the project will be written in compatible languages. This linguistic compatibility will allow for easier communication between backend and frontend and will in turn allow for a better experience with less risks of frontend and backend incompatibility. NestJS incorporates the component-based pattern and MVC pattern by design philosophy and thereby allows projects using those patterns to flourish.

Pros:

- Allows components to be updated and developed separately
- Manages integration of all components
- Easily facilitates communication between frontend and backend

Cons:

- Complicated to understand if one has no experience in NestJS.
- Can be difficult to debug

Nx

Overview:

Nx is a build system that enables simplified project management with a monorepo approach. It allows for unified resource management and a single point of contact to create libraries and add components to the system.

How it Fits the Architecture:

By using Nx, it allows easier management of component creation to reinforce the component-based aspect of our architecture choice. Nx integrates well with the other technologies as most use a common underlying language (Javascript). This allows us to make sure that the MVC structure is held by using Nx commands to enforce it as well as the dependencies.

Pros:

- Nx allows for separation of concerns through dividing the project into apps and libraries. This allows for code reuse through importing components where required.
- Dividing the project into apps allows reinforcement of the layered design pattern

- Nx follows a monorepo philosophy so it maintains a single package list to ensure dependencies are all upgraded together and reused where applicable.

Cons:

- Only supports Javascript so components written in other languages need to be manually managed within the project.

Keras - (Tensorflow)

Overview:

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

How it Fits the Architecture:

Keras as a ML library has abundant documentation and, while Keras has pre-trained models related to dog breed identification, it is also an easy to implement library which can utilise the Stanford datasets from the requirements to train our own model specifically for the app. Keras can function in a standalone manner so it can be deployed independently from the NodeJS server. Keras is written in Python, but python natively accepts JSON objects. As such, interaction with the database and backend will not be impacted.

Pros

- Accepts JSON objects
- Easy to train
- Can function independently from the server

Cons

- Can be slow to run
- Can be difficult to integrate with Angular

MongoDB

Overview

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.

How it Fits the Architecture:

MongoDB is a NoSQL database that will allow for photos and other media to be stored together in a single repository. MongoDB uses a JSON format for storing records, which integrates well with both NodeJS and Angular. This data fits in the persistence layer and is also its own component.

Pros:

- Allows for intuitive data storage of any type of data.
- Works easily with GraphQL and angular with simple setup.
- Has built in encryption and security features.

Cons:

- Does not have a set structure and can allow for some ambiguity.
- Does have a maximum size of 16mb per record.

GraphQL

Overview

GraphQL is a query language for your API, and a server-side runtime for executing queries using a type system you define for your data. GraphQL isn't tied to any specific database or storage engine and is instead backed by your existing code and data.

How it Fits the Architecture:

Allows us to easily test all of the queries we need for the API and fits in the controller and model of the MVC model

Pros

- Fast execution
- Made for complex systems like angular

Cons

- Can be complex to implement.

Yarn

Overview

Yarn is a javascript package manager to automate the installation of dependencies.

How it Fits the Architecture:

Yarn is used to install packages and automate processes related to project setup. This allows the project to maintain portability and improves scalability as setting it up in different environments can be simplified to a few yarn commands wherein the project environment setup is automated.

Pros

- Automates project setup

Cons

- Is less popular than NPM, so would require an additional install before the project can be configured.

Docker

Overview:

Docker is a container manager which allows for parts of the project to be containerized and run in individual environments. This allows for maximum portability and separation of concern between components.

How it Fits the Architecture:

Docker's ability to separate concerns through the use of containers reinforces the component-based and layered patterns used in that it creates isolated environments which need to be explicitly connected to to access. This increases security, and because containers are preconfigured, allows for maximum portability of the application as the components within the docker containers are self-contained and can run in any environment that supports containers.

Pros:

- Maximises portability
- Minimises configuration
- Increases security

Cons:

- Virtualises aspects of the project and thereby limits performance and hardware accessibility in specific cases