SOFT4902
Graduation Design Project

FEYZİYE SCHOOLS FOUNDATION
IŞIK UNIVERSITY

Project Title
PET ADOPTION WEBSITE

By SELİN KAROL/20SOFT1023,
TAYLAN TANYERİ/19SOFT1010

SYSTEM DESIGN DOCUMENT

SYSTEM DESIGN DOCUMENT

1. Introduction

This part of the document aims at providing an overview of design goals and software architecture, by taking into consideration similar projects, the requirement analysis document that has previously been written and references to other documents, providing a solid foundation for understanding the architectural choices and objectives when it comes to Pet Adoption Website. The requirements being usability, reliability, performance, supportability, external interface; there needs to be some design goals such as but not limited to understandability, reuse, robustness, by ensuring modular composability, debugging and testing, identifying risks, continuity and low coupling with high cohesion to reduce dependencies [16]. Moreover, the system architecture is explained in detail in forthcoming sections for managing the interactions of components, maintaining changes in demands, distributing technologies necessary for different modules. In order to ensure what the workloads are and how they are processed, this document covers the design goals, current software architecture and the proposed one, including assigned subsystems, data storage and management, access control and safety measures, implementing the global software and boundary conditions.

1.1.Purpose of the System

This SDD is intended to provide a software system design that will satisfy functional and nonfunctional requirements stated in the RAD Document of pet adoption project. The purpose of this document is to serve as a guideline throughout the development phase of the project for developers. It draws an outline when it comes to system architecture and technical design of the software. The primary objective of the document is to have an understanding of the architecture of the system, components, interactions and technologies used in the process of developing the platform. The system is divided into subsystems for further development and for each, a service is provided. To maintain these services and manage the interactions of this complex system, each subsystem must be implemented based on the determined architecture with proper tools, which will be a crucial step for

management of changes and deployment & bridging the gap between the problem and the existing system via system design [6].

1.2.Design Goals

To access design goals for our project, we first need to identify design goals. This will help us model the new system design by dividing it to sets of subsystems. In order to identify design goals, non-functional requirements and trade-offs must be highlighted. In the RAD, there are statements about non-functional and functional requirements, object models and dynamic models. These analysis models have an influence on system design, helping the definition of design objectives and subsystem breakdown. Acknowledging that stakeholders have different design goals, some of the design goals [6] this project aims at completing can be listed below (some goals may intersect with different groups):

For client side:

- Low cost: System should be developed in a low-cost, preventing further unnecessary spendings that are above the determined budget.
- Traceability of requirements: Each requirement should be able to be traced to its corresponding system functions and vice versa [10].
- Rapid development: The platform must be open to continuous integration and rapid development.
- Completeness: All functions required by customers should be included in the project [1].
- Scalability: System should be flexible. It needs to handle load increases without having the need to decrease the performance & handle the rapid increase of the load [1].

For end users:

- Safety and security: The system must not grant unauthorized access from third parties and must be safe in terms of protection of personal information against cyber-attacks. There needs to be secure connection, authentication and restrictions [1].
- User-friendliness: The platform must be easy to use, understand and learn to ensure a user-friendly atmosphere.
- Usability: Easiness of use. Users should be able to navigate site easily [10].
- Accessibility: Platform should be accessible from as many people from different locations as possible, without assistance including people with disabilities [9].
- Visibility: Functionalities in a system must be visible to prevent problems. Users should be able to tell the states of a functionality and alternatives by just looking at the system [9].
- Reliability: The system should be reliable, preferably in an environment that has less failures. It needs the ability to continue to operate under predefined conditions [1].

- Feedback: The system should send information back to users about a process and what has been done. For example, the system should inform users that they logged in, or they need to login for some actions [9].

- Constraints: Restriction of actions to prevent the user from selecting unwanted/incorrect options. For example, if a user is not verified, the system will not allow him/her to start adoption process [9].

For developers:

- Minimum errors: The developed system must have minimum number of errors when operating.

- Reusability: Components should be reusable in other components of the system with small changes. The navigation bar applying to every page may be an example of reusing a code without copying and pasting [1].

- Maintainability: The system should support changes, such as new business requirements etc. [1].

- Consistency: Interfaces must have similar operations and elements. Page formats should be matching, the buttons must be in a single format etc [9].

1.3.Definitions, Acronyms, and Abbreviations

- API: A collection of specified guidelines that allow different apps to communicate with each other [12].

- Boundaries: Objects that communicate with actors in system, such as UIs etc. [18].

- Cohesion: Refers to the "belong together ratio" of the elements inside a module of a system. We need to change the elements of a module together, not separately if it has high cohesion. This is the opposite of coupling [17].

- Component: Software systems that are developed once and frequently used within that project's system. Examples can be given as applications, databases, a network, security products etc. [14].

- Controllers: Objects that direct the execution of commands, mediating between boundary and entity objects [18].

- Coupling: It can be briefly summarized as follows; shows how much we need to apply changes to other modules when we want to change some module in the system [17].

- Debugging: The process of fixing errors and bugs and also detecting, handling them via tracers, profilers, interpreters etc. [20].

- Deployment: Making software available for system use by users. Acts like a checklist in software development process [8].

- Entities: Objects thar represent system data. In this case, users are considered as entities [18].
- Interaction: An interaction between instances of components defines how these instances' behavior should change [5].
- Layer: Functional division of a software [24].
- Microservices: An architecture type that develops independent services to ensure independent deployment, communicating via APIs. Each microservice has its own database and operates a specific business logic [25].
- Module: A component for easy replacement or a section of code for reusability [26].
- Monolith: An architecture type where all functionality in a system is deployed together. It has 3 types; the single-process monolith, the modular monolith and the distributed monolith [3].
- MVC: An architectural style having 3 components; model-view-controller, that separates presentation and interaction from data [11].
- RAD: Aka requirements analysis document, a document indicating the requirements gathering and analysis, containing scope, objectives, requirements, use cases, system models and more, necessary for having an understanding of a project [13].
- Software Architecture: Represents the design decisions of our overall system and its behavior. It contains titles like implementation details & design, technologies that we used, system design (monolith or microservices), and infrastructure of the project [7].
- Testing: Operations to verify and observe that a software does what it is intended to do. Aims of the testing can include; increase in performance, reduce costs and prevent bugs [22].
- Tier: A layer that is running on infrastructure separate from other divisions [24].

2. Current Software Architecture

The architecture of the previous system is the modular monolith. It consists of separate modules to work independently but combined together for deployment. The work can be carried out in a parallel way since having a simple deployment topology. The data is stored and managed in one database [3]. It has a presentation layer featuring the user interface created by HTML and CSS, an application layer using languages like Java, C# and Python, and a database layer, in which the data are stored. If an update is needed, the entire application must be deployed, resulting in slow changes for large applications [19].

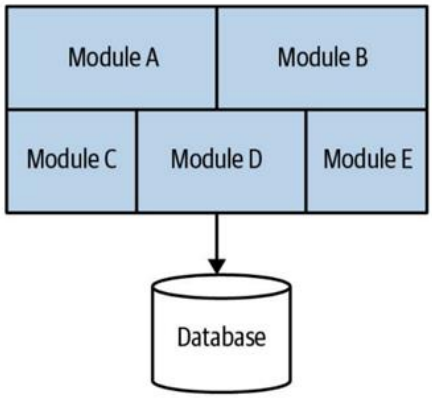

Figure 1. Modular Monolith from [3]

The web browser is the client-side frontend component, interacting with users, taking inputs and managing user interactions within the system. The web server, aka backend component, handles the business logic on the server side. It processes the requests coming from users and manages the operations. The last tier, database server provides the data to be used within the operations. Data-related tasks are handled on the dB server. So, the communication between client and server is managed by the web server. Therefore, this architecture prevents users from directly accessing the data by enabling another layer to do it. It is important for security and data integrity. The centralized architecture simplifies version control and managing the changes, as every component is within a single codebase [25].

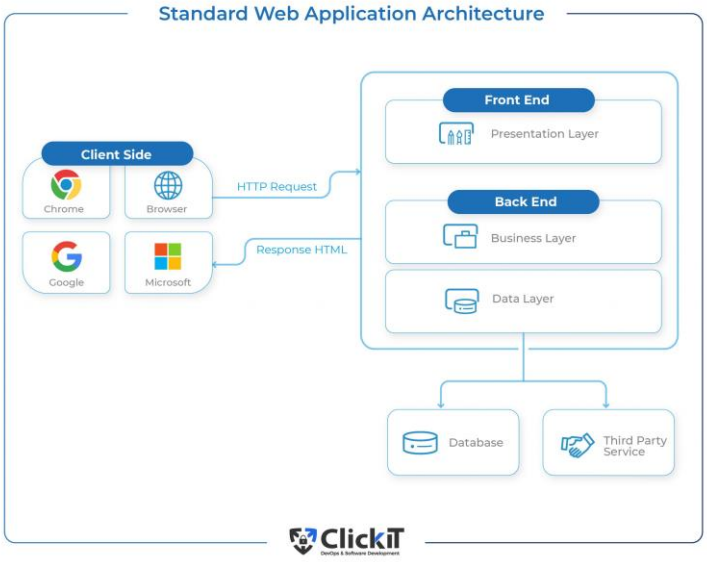

Figure 2. 3-Tier Monolith From [25]

When it comes to current architecture, it is seen that the functionalities are coupled. Though it offers simplicity and easy development, the scalability is harder. Because there is no independent deployment. A change will affect another, as a result. Moreover, the data is stored in a shared database, which will cause slower response time and as the system grows, it would be harder to maintain the database which will contain larger data, as new pets are added frequently, and new users are registered day by day. The entire monolith may be affected by some part of the system, and this will cause delays in feedback and

status of the pets and potential adopters, causing miscommunication [4].

To summarize, even though monoliths are suitable for code-reuse, easy development and simplicity in smaller applications, they have one single database that handles the data, resulting in complications when the database grows. This makes it harder to handle changes, as they affect one another. Therefore, when size and complexity of the codebase reaches a certain limit, this type of architecture creates problems of response time, applying changes, clean code problems [15].

3. Proposed Software Architecture

This project uses MVC (model-view-controller) architectural design pattern. This architecture organizes application's logic into different layers, which have distinct tasks & also interact with each other to ensure functionality of the application is coordinated [26]. This type of architecture is used widely among UI and web applications, as it provides a base for building interfaces and divides the responsibilities into layers for the developers [2], decoupling data access and data representation [11].

As mentioned, MVC has 3 components that separate presentation and interaction from system data. Model component manages data and the state. It processes data to and from data store, in this case, database. It is independent of the other two, making it eligible for testing independently. After it receives directives from controllers, it updates data.

View component is related to operations performed on data. It can also be called a “presentation”. It displays data to the user by using an interface based on information the controller provides.

Controller component is the one that manages user interaction. Controller objects direct these interactions to View’s boundary objects and Model’s entity objects. It acts as an intermediary between the model and the view. The figure below summarizes the architectural pattern:

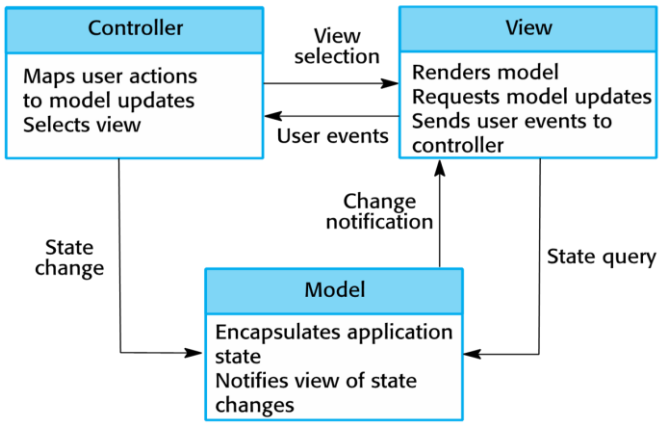

Figure 3. MVC Pattern from [11]

This architectural pattern is eligible for this platform, since it separates the data from presentation, making it easier to make changes in one and test it. By dividing the whole system into subsystems, it is easier to manage each subsystem as the dependency decreases while interaction between subsystems increases. That’s why it is suitable for interactive systems [2] [11].

3.1.System Overview

Presentation Layer includes GUI serves to bring the UI to our users. From here users can navigate various actions. Some of which are handling user inputs and managing interfaces. Our Business Layer includes the use case operations and how they are connected to the other systems. In Business Layer our users have 3 subclasses. They are different variations of the user with different authority. There is also an Inheritance relationship between those 3 and each one adds different actions to the parent. Aside from that, we have Admin and its use case operations, and we have the pets with their status. Overall diagrams also show where we should access whenever a request is processed in our database. Whenever a visitor visits the site, the Presentation Layer communicates with business. In the case of registration, adoption etc., the Business Layer accesses Data Layer to store information. Some operations in Business may access data layer and some may not. It is decided if storage is required, and can be seen below:

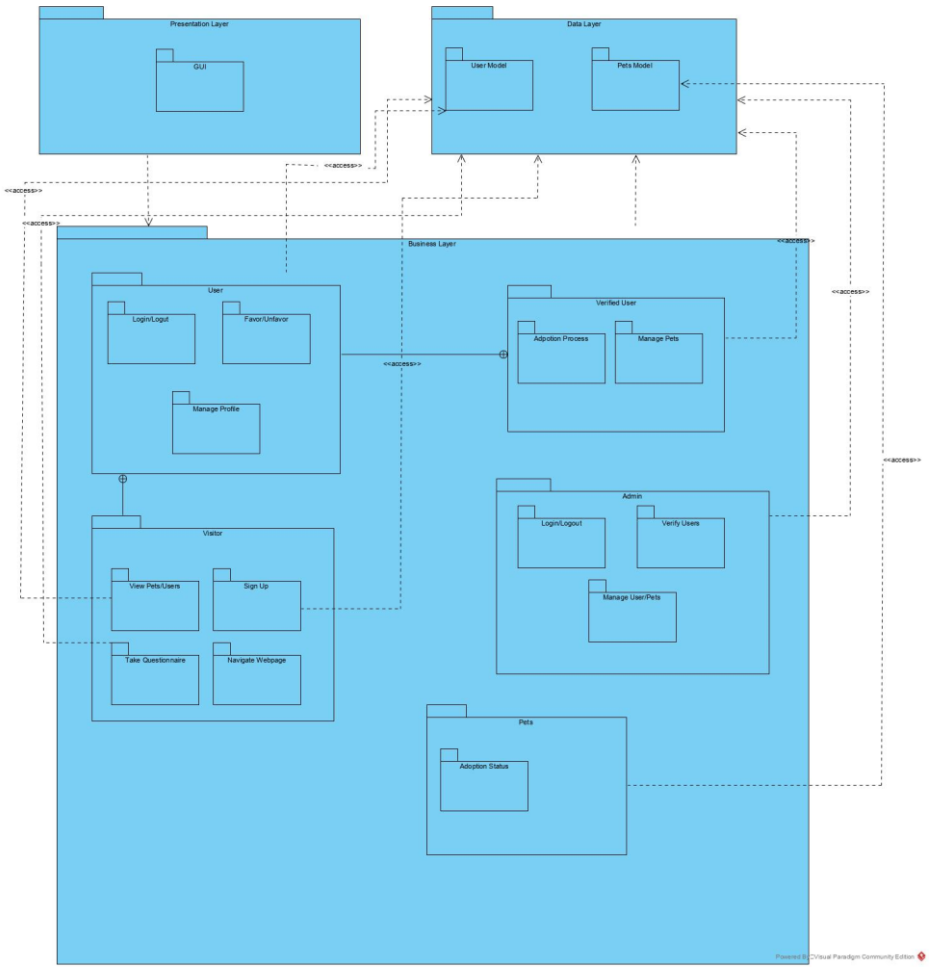

Figure 4. Package Diagram of the Platform

3.2.System Decomposition

The system is decomposed into subsystems to maintain changes without affecting the other components. There is a User Interface Subsystem, which presents the information to users, handling interactions of users within system and database and inputs. There is also an Admin Interface Subsystem beside the user, which handles user interactions from admin's perspective. Each offer

operations to be performed by user actions. The application subsystem has the operations that the user performs. For example, taking a questionnaire, favoring pets and finding pets by using the search filter operation. Starting the adoption process is the next step of these operations, allowing users to choose pets and proceed with the adoption, which is the main goal. For this purpose, the adoption status of the selected pet is updated, accessing the database. This subsystem is data management, which manages, stores and retrieves data, and handles database interactions, as well as storing pet information, caretakers, user information. Data management subsystem also proceeds with table relationships and entities.

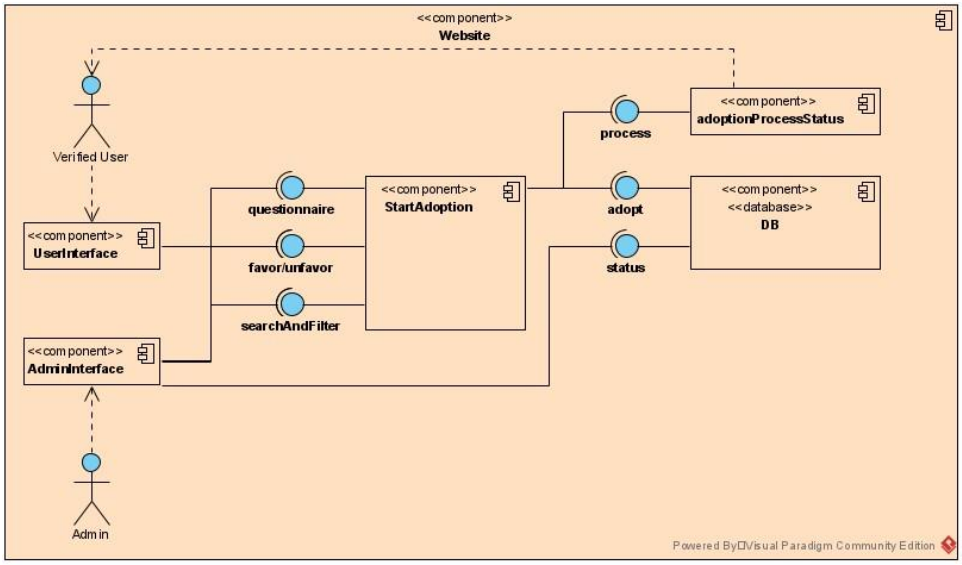

Figure 5. Component Diagram

3.3.Hardware/Software Mapping

The computer, application tier, and database service are the three tiers of the architecture displayed in the deployment diagram. Users communicate with the Application Tier over HTTP, including users mentioned on the diagram. Tier is made up of HTML and CSS for generating the User Interface and Backend includes the Flask. For the backend data store SQLite is used by the Database Service module. The HTTP and TCP/IP protocols enable communication, allowing data to move throughout the system.

There may be coordination and communication issues between nodes, scalability challenges as there will be multiple nodes and loads, maintenance problems when it comes to dB updates etc. There may also be security concerns regarding the protection of data in SQLite. Therefore, while dividing the system into subsystems and components, there will be potential challenges when it comes to preservation of the individual nodes [23].

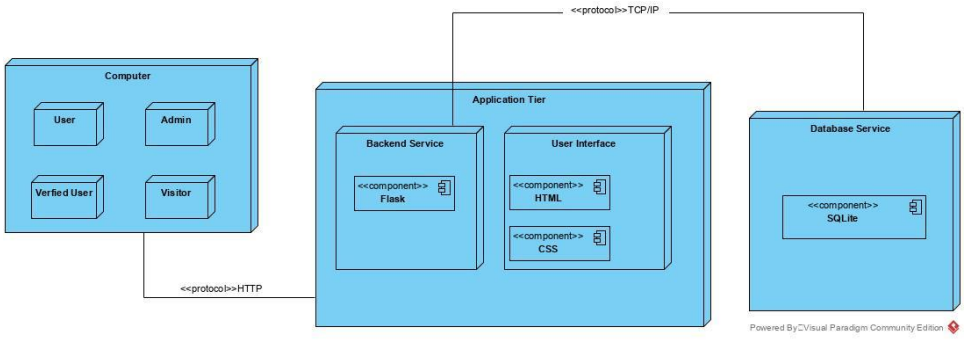

Figure 6. Deployment diagram

3.4.Persistent Data Management

Our project hosts 2 different tables when it comes to databases. The first one is “User” and the other one is “Pet”. Their attributes are shown below, in the entity-relationship diagram:

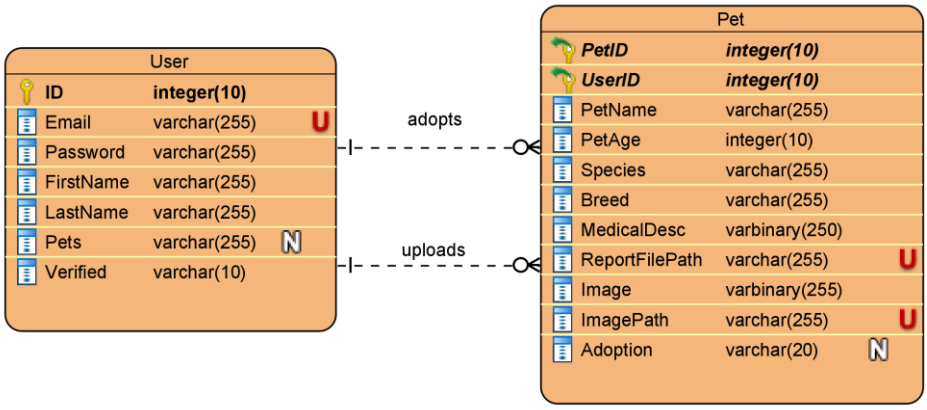

Figure 7. ER Diagram

The user id is going to be the key to determine the user. The email is also supposed to be unique for safety and accessibility concerns. Everything except user pets has to have an input, meaning they cannot be NULL. If a user does not enter any pets, user pets may be NULL. On the other table, the file path that has the medical description report and the image path that stores the image of pets must be unique. They need to be saved separately so that none of them are saved on top of each other, causing the system to lose its data. If adoption is NULL, that means the pet is not adopted. A user may upload and adopt multiple pets; however, a pet may only have one adopter or one caretaker, meaning there is a 1-to-many relationship between them, which will be handled according to pet and user IDs.

To perform these actions and storage, we decided to use a database called SQLite, which is a lightweight opensource database engine. It is eligible for medium traffic websites, which is around 100K hits/day according to its original website. It is a relational database, since it has a table structure that stores data and provides relationships between them. SQLite is used for analyzing large datasets and caching data from RDBMS, by reducing latency and load. When it comes to data encapsulation, the database is protected from unauthenticated users. Only admins are eligible for accessing data. To ensure that, some attributes will be having private labels to prevent direct access. There are get-methods and set methods in order to limit access. By design, SQLite databases are stored as files on disk, and access to these files is controlled by the underlying operating system's file permissions.

However, the usage of private labels for specific properties becomes essential [23]. Here is an example of tables pet and user:

Table 1. User Table

id	email	password	first_name	last_name	pets	verified
1	User1@gmail.com	Hashed_password_1	Selin	Karol	NULL	0
2	User2@gmail.com	Hashed_password_2	Taylan	Tanyeri	Pet1, Pet2	1

Table 2. Pet Table

id	User ID	name	age	species	breed	Medical_ desc	file_path	image	Image_path	Adoption
1	2	Fadik	6	Cat	Tekir	File1	file/file1	img1	img/img1	1
2	2	Pakize	2	Chicken	Grey	File2	file/file2	img2	img/img2	0

3.5.Access Control and Security

For the sake of safety, privacy and data protection laws must be followed. It is necessary to provide pet rights and copyrights. All parties must agree to the terms and conditions of a contract protecting the rights of adopters and adoptees. Liabilities need to be clarified. For these reasons, the platform uses a password authentication protocol PAP that requires an email and a password for registration, putting some constraints on password length for password strength. Also, the tables below describe data access & user operations constraints briefly:

Table 3. Data Access per User Type

Data Access	Visitor	User	Verified User	Admin
View Public Content	Yes	Yes	Yes	Yes
Access User Profile	No	Yes	Yes	Yes
Access Detailed Pet Info	No	No	Yes	Yes

Table 4. Operations per User Type

Operations	Visitor	User	Verified User	Admin
Register Account	Yes	No	No	No
Log In	No	Yes	Yes	Yes
Edit Profile	No	Yes	Yes	Yes
Delete Account	No	Yes	Yes	Yes
Admin Operations	No	No	No	Yes
Add/Delete Pets	No	No	Yes	Yes

3.6.Global Software Control

In our project first we need to run the main.py to execute the database and the application. Requests are usually started in a Flask application via HTTP endpoints. These endpoints process incoming HTTP requests (GET, POST, etc.) and are defined using routes. Connection pooling is a common technique

used by Flask to manage SQLite database connections [23]. Also, we use SQLAlchemy to abstract away some of the complexity of direct SQL queries and provide higher-level abstractions for database operations. Flask Blueprints allow you to organize your application into reusable components. File-level locking is typically used with SQLite. To prevent any problems, there is a need for exercise when allowing concurrent access to the database in a multi-threaded context (like a Flask application). By ensuring such operations, we can manage concurrent access to dB, manage connections efficiently, do optimizations and reduce the likelihood of conflicts. Error handling, monitoring and testing will ensure a reliable software control system by handling user requests efficiently and provide synchronization between layers. Otherwise, users may have problems regarding pet status', inconsistent information due to edition/deletion pets and user profiles, which will reduce the reliability.

3.7.Boundary Conditions

The system should run the necessary components and modules in the background when we start the project. For example, it should initiate the main package and also create the database within. After creating the database, the connection must be created, and UI should be generated for user actions. The system also checks the authentication. If the user is already signed up or not, verified or not, and if logged in as administrator. The system should produce easy-to-understand error messages when there are errors that affect the user interface or actions. Messages need to recommend some actions in addition to explaining what went wrong to users. Password length, incorrect password or verification for adoption may be such examples when it comes to directing the user and implementing boundary conditions.

4. Subsystem Services

User Interface Subsystem has services such as presenting data to users, displaying UI elements in an organized way, rendering web pages. Capturing user inputs, handling submissions and button clicks are other examples of this service. Application Subsystem implements business rules, process data and execute operations. Such operations can be classified as filtering, registration, taking questionnaires, deletion and addition operations etc. Manipulating and processing data based on business requirements are the responsibilities for this subsystem. Lastly, Data Management Subsystem stores and retrieves data from our database. It executes updating operations and accessing data objects. After a visitor is registered, the data of the registration is saved under the user table in the database. If a user has pets that need adoption, the database also stores them in pet table, including their relationships with the user table. For more detailed information, please check the package and component diagrams.

5. References

1. Arköse, Tuğberk. "Software Architecture SOFT3205." Week 2 – 12 October 2022. SOFT3205 Lecture, 12 Oct. 2022, Istanbul, Turkey.
2. Arköse, Tuğberk. "Software Architecture SOFT3205." Week 9 – 14 December 2022. SOFT3205 Lecture, 14 Dec. 2022, Istanbul, Turkey.
3. Arköse, Tuğberk. "Software Architecture SOFT3205." Week 10 - 21 December 2022. SOFT3205 Lecture, 21 Dec. 2022, Istanbul, Turkey.
4. Avisari. "Architectural Showdown: Microservices vs. Event-Driven Architecture in Healthcare

Data Management." Medium, 9 June 2023, avisari.medium.com/architectural-showdown-microservices-vs-event-driven-architecture-in-healthcare-data-management-d654c9e9e0df#:~:text=Microservice%20Architecture%3A%20Granular%20control%20over,flexibility%20of%20the%20microservice%20architecture. Accessed 7 Nov. 2023.

5. Blay-Fornarino, Mireille, et al. "Software interactions." *The Journal of Object Technology*, vol. 3, no. 10, 2004, p. 161, https://doi.org/10.5381/jot.2004.3.10.a4. Accessed 9 Nov. 2023.
6. Bruegge, Bernd. "Design Goals & System Decomposition." Software Engineering I Lecture 7. https://ase.in.tum.de/lehrstuhl_1/files/teaching/ws0607/Software%20Engineering%20I/L07_DesignGoalsSubsystemDecomposition.pdf. Accessed 10 Nov. 2023.
7. Cocca. "The Software Architecture Handbook." *freeCodeCamp.Org*, 4 Aug. 2022, www.freecodecamp.org/news/an-introduction-to-software-architecture-patterns/#what-is-software-architecture. Accessed 8 Nov. 2023.
8. "Deploying Software." *IBM*, 5 Apr. 2023, www.ibm.com/docs/en/zos/3.1.0?topic=task-deploying-software. Accessed 9 Nov. 2023.
9. Ekin, Emine. "Human Computer Interaction – Week 2". COMP2502, Spring 2022, Istanbul, Turkey.
10. Eroğlu, Günet. "Requirements Engineering." Principles of Software Engineering Week 4.2. SOFT2101, Fall 2021, Istanbul, Turkey.
11. Eroğlu, Günet. "Software Architecture." Principles of Software Engineering Week 6.2. SOFT2101, Fall 2021, Istanbul, Turkey.
12. Frye, Ma-Keba. SEO Content Writer. "What Is an API? (Application Programming Interface)." *MuleSoft*, www.mulesoft.com/resources/api/what-is-an-api. Accessed 9 Nov. 2023.
13. Hu, Prof. Dr. Daning. "Requirement Analysis." University of Zurich. Zurich, Switzerland, https://www.ifi.uzh.ch/dam/jcr:e68f207f-003b-4ef2-887a-86574ecf2f8f/CH1.2.pdf Source University of Zurich. Accessed 8 Nov. 2023.
14. Hubbard, Mark. "Common Components." *Common Components - DfE Architecture*, dfe-digital.github.io/architecture/common-components/#common-components. Accessed 9 Nov. 2023.
15. Kalske, Miika. "Transforming Monolithic Architecture towards Microservice Architecture." *Core, University of Helsinki*, Department of Computer Science, 18 Nov. 2017, https://core.ac.uk/reader/157587910. Accessed 7 Nov. 2023.
16. Kästner, Christian, and Charlie Garrod. "Principles of Software Construction: Objects, Design and Concurrency Design Goals." Carnegie Mellon University. Pittsburgh, Pennsylvania,

https://www.cs.cmu.edu/~charlie/courses/15-214/2014-spring/slides/11a-design%20goals.pdf. Accessed 8 Nov. 2023.

17. Pagade, Ganesh. "Difference Between Cohesion and Coupling." *Baeldung on Computer Science*, 9 Nov. 2022, www.baeldung.com/cs/cohesion-vs-coupling#:~:text=Effectively%2C%20the%20coupling%20is%20about,and%20calling%20each%20other%27s%20methods. Accessed 8 Nov. 2023.
18. Pearce, Jon. "Implementing Use Cases." *San José State University*, www.cs.sjsu.edu/faculty/pearce/modules/patterns/enterprise/ecb/ecb.htm. Accessed 10 Nov. 2023.
19. Saraswathi, Ravi. "Four Architecture Choices for Application Development in the Digital Age." IBM Blog, 6 Jan. 2020, www.ibm.com/blog/four-architecture-choices-for-application-development/. Accessed 7 Nov. 2023.
20. "Software Engineering: Debugging." *GeeksforGeeks*, 20 Apr. 2023, www.geeksforgeeks.org/software-engineering-debugging/. Accessed 9 Nov. 2023.
21. "Web Application Architecture - Detailed Explanation." *InterviewBit*, 17 June 2022, www.interviewbit.com/blog/web-application-architecture/. Accessed 7 Nov. 2023.
22. "What Is Software Testing and How Does It Work?" *IBM*, www.ibm.com/topics/software-testing. Accessed 9 Nov. 2023.
23. "What Is SQLite?" *SQLite*, www.sqlite.org/index.html. Accessed 12 Nov. 2023.
24. "What Is Three-Tier Architecture." *IBM*, www.ibm.com/topics/three-tier-architecture#:~:text=A%20'layer'%20refers%20to%20a,separate%20from%20the%20other%20divisions. Accessed 9 Nov. 2023.
25. William. "Web Application Architecture: The Latest Guide 2022." ClickIT, DevOps, 27 Oct. 2023, www.clickittech.com/devops/web-application-architecture/. Accessed 7 Nov. 2023.
26. Wright, Gavin. "What Is a Module in Software, Hardware and Programming?" *WhatIs.Com*, TechTarget, 2 June 2022, www.techtarget.com/whatis/definition/module#:~:text=In%20computer%20hardware%2C%20a%20module,is%20designed%20for%20easy%20reusability. Accessed 9 Nov. 2023.