TED UNIVERSITY

Computer Engineering

2022-2023
Spring Semester

Low-Level Design Report
for
EasySearch

Dinçer İnce
Halil Mert Güler
Süleyman Samed Çalışkan

Table of Contents

1 Introduction

When we say files and documents, the things that come to mind are to categorize them in the right folder, store them and access them whenever we want. Everyone can have hundreds or even 1000s of files on their computer or phone. Although it is the best method to categorize these files and store them accordingly, sometimes you cannot reach the files you are looking for when you want. The files are lost, you can't find them, or even if it's in front of you, you can't know which one is among the hundreds of files. If you want to do a detailed search, the current performance of the computers may be insufficient, or the files may be lost in the depths of the system. EasySearch is a software developed for all these problems and will solve them all. EasySearch, as the name suggests, will now be very easy to search. Thanks to EasySearch, all documents can be stored, edited, categorized, and easily accessed anytime, anywhere.

Users can easily integrate EasySearch into their own pages or other pages and use them as they wish. Not only the searches, even EasySearch itself is very easy to use. EasySearch was developed and presented as an answer to all the following questions: "Where are my files?", "Where did this text disappear to?", "Can I store my documents more easily?", "How do I categorize so many documents?".

1.1 Object Design Trade-Offs

1.1.1 Usability vs Functionality

Since ES tries to acquire every audience, it can, from children to adults, the system interface should be easy to understand and the application features should be made very easily.

1.1.2 Availability vs Maintainability

Since EasySearch provides a search and storage service for its users, it should be accessed by users whenever they want. This requires keeping all servers up all the time. Therefore, when there will be maintenance or any work, it will require us to fix or repair the servers that need to be shut down without developing them remotely in a different way.

1.1.3 Security vs Cost

In order for people to trust us and upload their personal files, documents, our security system must be extremely strong and unbreakable because we do not want to be hacked. Increasing the security elements will automatically increase the cost.

1.2 Interface Documentation Guidelines

Class	Description of the class.
Attributes	Attribute type and attribute name.
Methods	Method name, parameters and return value.

1.3 Engineering Standards

In the reports, UML [1] design principles are used in the description of class interfaces, diagrams, scenarios and use cases, subsystem compositions, and hardware-software components depiction. UML is a commonly used standard that allows simpler description of the components of a software project. With standard UML models we were able to represent the system structure, software components, and functionalities.

1.4 Definitions, Acronyms, and Abbreviations

ES: EasySearch

UML: Short for Unified Modeling Language

API: Application Programming Interface

2 Packages

Our solution Consists of two applications which are the main application which is API, and the web application which is the manager of the API. The API consists of two main packages which are the request management, and data processing. Whereas the web application consists of three main packages which are the model, view, presenter.

2.1 Client (Web Application)

The main use case of the web application is the way to manage access, and settings of the features of the API, so it is required to use the web application to utilize the system. However, the second main use case of the web application is to use it in a way that makes the system a document management solution in a way it becomes a cloud stored, document searching, analyzing, and editing application, to easily manage and access your data from anywhere.

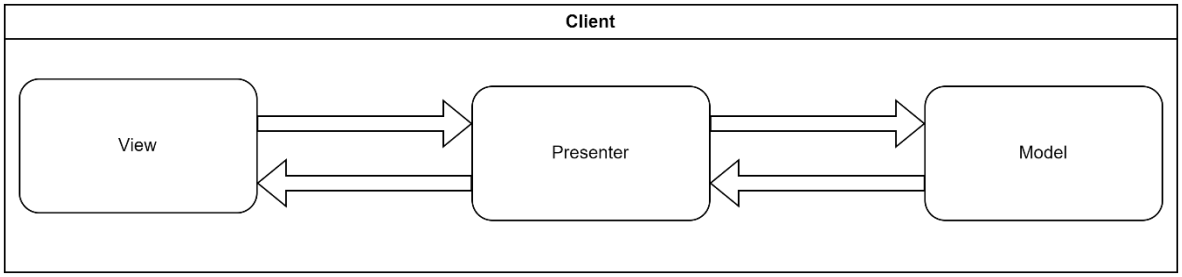

Figure 1 Client Package

2.1.1 View

View Pages have one to one correspondence with the classes in presenter, where every view has its own presenter class. View consists of the following pages that the user can interact with,

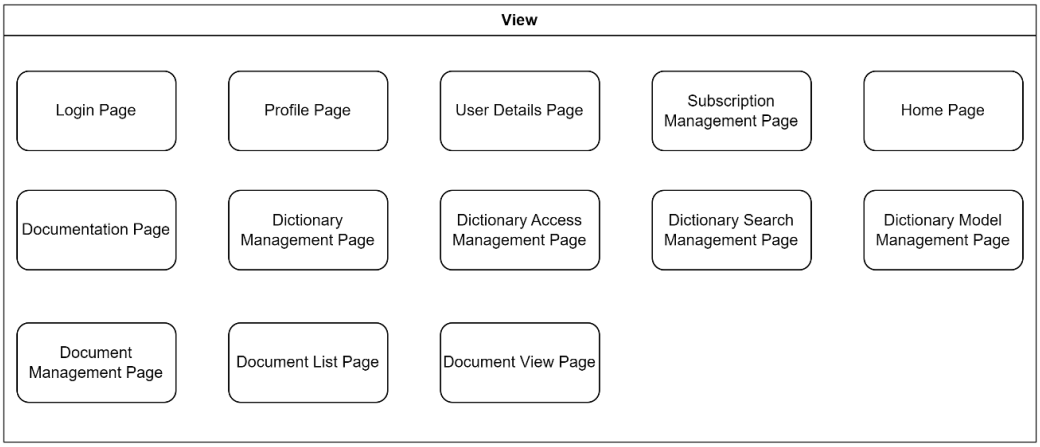

Figure 2 View Package

These pages follow the following routes.

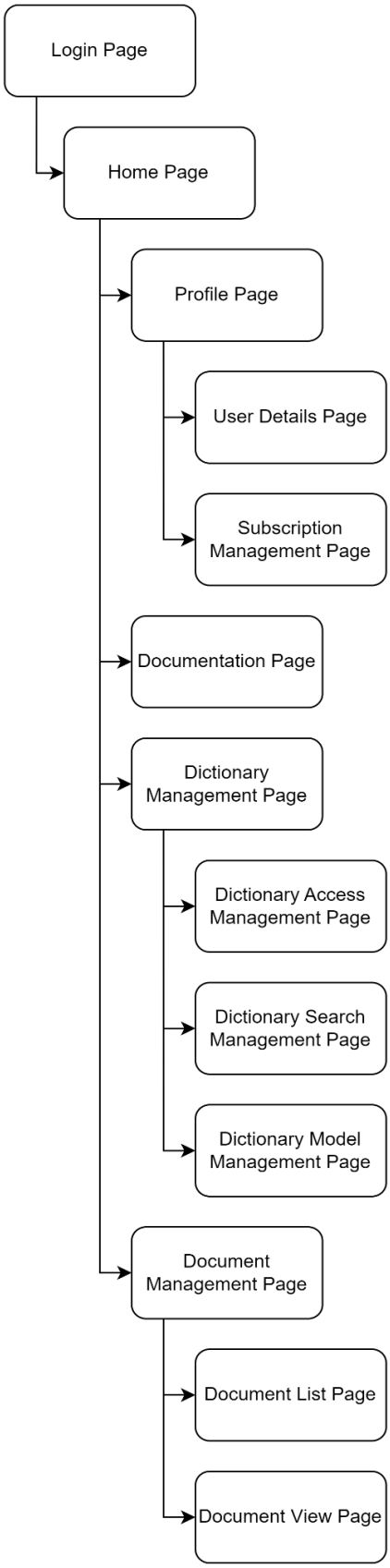

Figure 3 View Routes

2.1.2 Presenter

Presenter classes contain the logic of its corresponding view page, each presenter follows the events happening on the view and acts accordingly, whether it's making a call to the model package, or changing the route and etc.

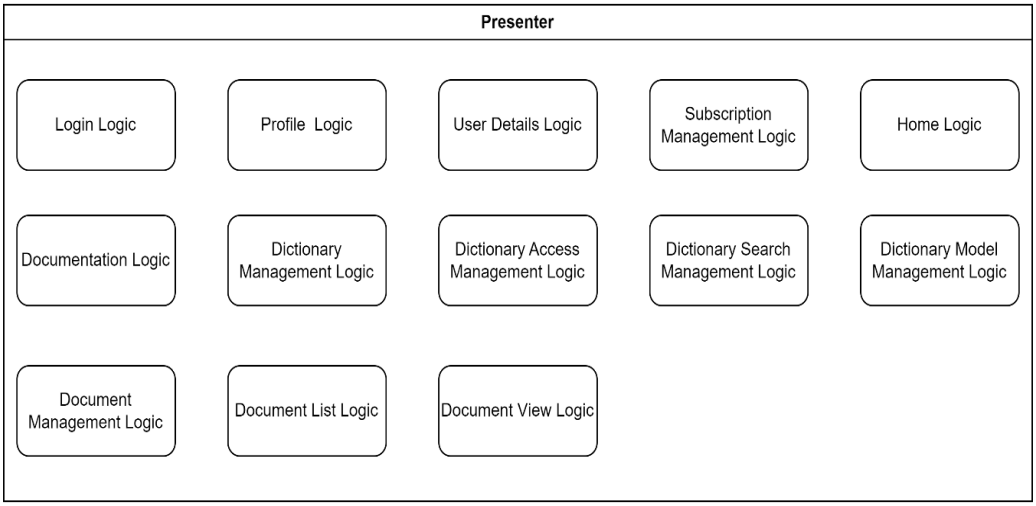

Figure 4 Presenter Logic

These classes are categorized according to the routes defined in the view routes, as it is a single page application, routes are managed by the presenter classes itself.

2.1.3 Model

Model Package stores the current state of the application and provides services which directly interacts with the API itself. According to the request sent from the presenter package, model changes the state of the application and/or sends the corresponding request to the API, then notifies the presenter of the result of the request.

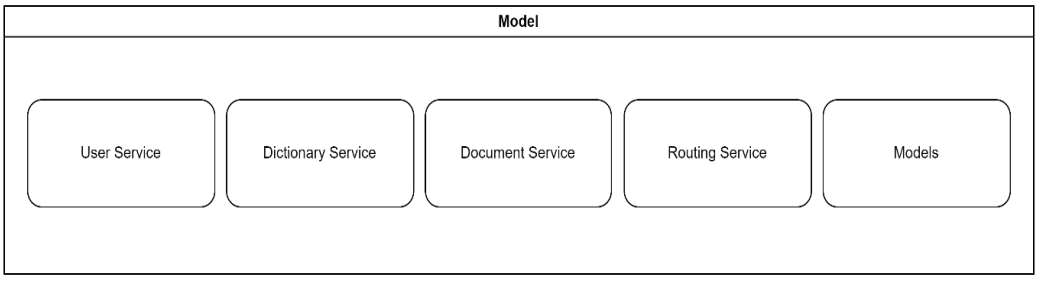

Figure 5 Model package

2.1.3.1 User Service

User service contains the state of the user (whether a user is logged in,), and its token. Additionally, it also contains all the requests that the API features about users, from handling the subscription to editing the details of the user.

2.1.3.2 Dictionary Service

Dictionary Service contains the methods to send requests to the API about the dictionary. Methods contains changing the model of the dictionary, changing the access tokens of the dictionary, creating new tokens with specific permissions, and changing the search behavior of the dictionary (search results bringing the documents containing the exact query, similar query, semantically similar query, etc.)

2.1.3.3 Document Service

Document service stores the documents already requested from the API in order to readily serve to the user. It also contains the methods to send requests to the API about the document, from simple get requests to editing and analyzing the document.

2.1.3.4 Routing Service

Routing service stores the current route of the application as well as the routes permitted to be routed according to the state of the application. For the view to change page it sends a request to its presenter which receives confirmation from the routing service. With the service users are unable to navigate to pages they don't have access to.

2.1.3.5 Models

Models contains the interfaces of the objects that are either sent or received by the API.

2.2 Server (API)

The server is the heart of our system as it is not just a server that communicates with our web application but also communicates to each of the users who have created an API key to access it. The server consists of two packages which communicates with each other.

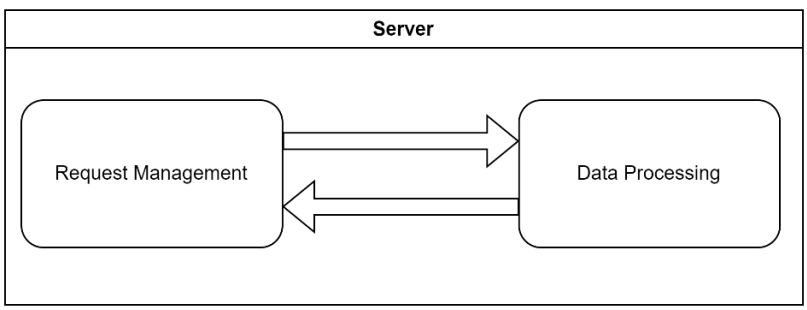

Figure 6 Server Package

2.2.1 Request Management

Request management handles the task of deciding whether the request is acceptable, checking if the request is authenticated, and passing the request to the data processing package's corresponding repository. After that it receives a response from the package and returns the request based on the status of the request

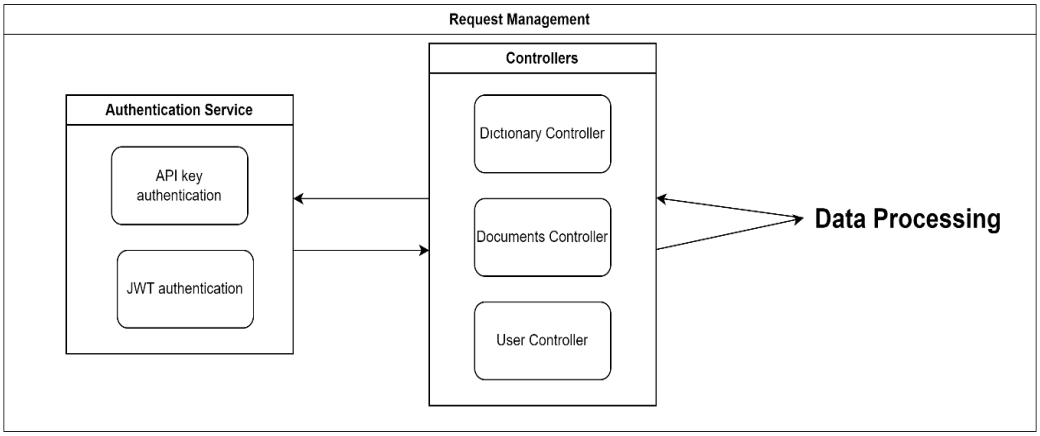

Figure 7 Request Management Package

2.2.1.1 Controllers

Controller classes are the first classes to receive the request, these classes do preliminary checks to see if the request is valid, sends the request to the authentication service to see if its authenticated, then forwards the request to the data processing, and receives back the response to send as a response to the request.

2.2.1.2 Authentication Service

Authentication service confirms the validity of the authentication that is on the request. There are two types of authentications available in the system, which are the API tokens that can only be used for document related operations, and the JWT tokens that the users get when they login to the web application, which can access every part of the system.

2.2.2 Data Processing

Data processing handles the actual logic that is intended in the request it receives, data processing has two parts which are the services and the repositories. After a request is caught on the request management it forwards the request to the data processing for the request to be completed, after that the desired data is then forwarded back to the request management.

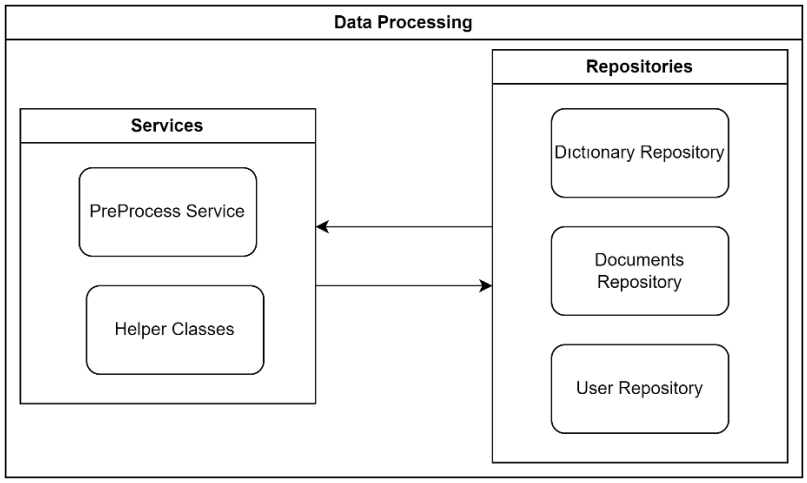

Figure 8 Data Processing

2.2.2.1 Repositories

Repositories are the only classes which have access to the database, it also has a local database to reduce the number of calls being made to the database itself. Repositories contains the CRUD functions as well as the

2.2.2.2 Services

Services contain the helper functions as well as them being a singleton class contains some state, and context required for the repositories to complete their desired action. An example of context would be the machine learning model being loaded to the API so that the actions that require the model would be substantially faster.

3 Class Interfaces

3.1 Server

3.1.1 Models

Class Dictionary
Attributes public int Id public String Name public List<Document> documents public User user public int userId public int NumberOfDocuments public int totalNumberOfWords
Methods Getters and setters

Class Document
Attributes public int Id public String Title public string rawDocument public Dictionay dictionary public int dictionaryId public List<DocumentWord> words public int numberOfWords
Methods Getters and setters

Class DocumentWord
Attributes public int Id public Word word public Guid wordId public Document document public int documentId public int count
Methods Getters and setters

Class User
Attributes public int Id public string userName public string password public string email public List<Dictionary> dictionaries
Methods Getters and setters

Class Word
Attributes public Guid Id public List<DocumentWord> documents
Methods Getters and setters

Class CreateDocumentDTO
Attributes public int DictionaryID public string Title public string RawDocument
Methods Getters and setters

Class DocumentDTO
Attributes public string text public string[] tokens
Methods Getters and setters

3.1.2 Repositories

Class GenericRepository
Attributes protected DataContext _context protected DbSet<T> dbSet
Methods async Task<IEnumerable<T>> GetAll() async Task<T> GetById(int Id) async Task<T> Add(T entity) async Task<bool> Update(T entity) async Task<bool> Delete(int Id)

Class UserRepository
Attributes protected DataContext _context protected DbSet<T> dbSet
Methods Task<IEnumerable<T>> GetAll() Task<T> GetById(int Id) Task<T> Add(T entity) Task<bool> Update(T entity) Task<bool> Delete(int Id)

3.1.3 Controllers

Class DictionariesController
Attributes private readonly DataContext _context
Methods Task<ActionResult<IEnumerable<Dictionary>>> GetDictionaries() Task<ActionResult<Dictionary>> GetDictionary(int id) Task<IActionResult> PutDictionary(int id, Dictionary dictionary) Task<ActionResult<Dictionary>> PostDictionary(int UserID, string Name) Task<IActionResult> DeleteDictionary(int id) bool DictionaryExists(int id)

Class DocumentsController
Attributes private readonly DataContext _context

Methods
Task<ActionResult<IEnumerable<CreateDocumentDTO>>> GetDocuments()
Task<ActionResult<Document>> GetDocument(int id)
Task<IActionResult> PutDocument(int id, Document document)
Task<ActionResult<Document>> PostDocument(CreateDocumentDTO request)
Document createDocument(Document document)
Task<ActionResult<double>> DocumentSimilarity(int Id1,int Id2)
Task<IActionResult> MostSimilarDocuments(int number, int id)
Task<IActionResult> Search(string text)
Task<IActionResult> DeleteDocument(int id)
bool DocumentExists(int id)
Task<IActionResult> GetPosts()
Task<IActionResult> GetPost(int id)

Class UsersController

Attributes
private readonly DataContext _context
private readonly IUserRepository _repository

Methods
Task<ActionResult<IEnumerable<User>>> GetUsers()
Task<ActionResult<User>> GetUser(int id)
Task<IActionResult> PutUser(int id, User user)
Task<ActionResult<User>> PostUser(string UserName, string Password, string Email)
Task<IActionResult> DeleteUser(int id)
bool UserExists(int id)

4 Glossary

4.1 Request

A request includes the URL of the API endpoint and an HTTP request method.

4.2 Dictionary

Object type that created by user holds document and implements selected search algorithms to documents on it.

4.3 Document

Created documents by user, search algorithm will be work for documents and their contents.

4.4 JWT

JSON Web Token is a proposed Internet standard for creating data with optional signature and/or optional encryption whose payload holds JSON that asserts some number of claims.