1. **Introduction**

Welcome to the Software Design Document (SDD) for the Todo List Web Application. This document serves as a comprehensive guide to the design and architecture of the todo list application, outlining its purpose and objectives.

Throughout this document, we'll explore the various components of the To-Do List application, including its architecture, features, data model, and deployment strategy. By the end, you'll have a clear understanding of how the application works and how it fulfills the needs of its users.

* 1. **Purpose**

The purpose of the todo list web application is to provide users with a convenient and efficient tool for managing their tasks and organizing their daily activities. By offering features such as task creation, editing and gathering related todo lists together in a folders and sharing them , the application aims to easy task management processes and enhance productivity.

* 1. **Objectives**

The main objectives of the todo list web application include:

* + - Providing users with a centralized platform for organizing and managing their tasks.
    - Offering a range of features to facilitatethe usage .

Tasks Managemen**t**: Create, edit, and delete tasks, enabling users to stay organized and prioritize their activities effectively.

Folder Management**:** Create, edit, and delete folders to categorize tasks.

Task Organization: Add tasks to todo lists and organize todo lists within folders, providing users with flexibility in structuring their tasks and projects.

* + - Enhancing user productivity through arranging his tasks in one place.
    - Ensuring the security and integrity of user data through robust authentication and data protection measures.
    - allowing user to share todo list and folder .

By achieving these objectives, the todo list web application aims to become an efficient tool for individuals seeking to effectively manage their tasks and maximize their productivity.

1. **Architectural Overview**
   1. **System Architecture**

The To-Do List application follows a scalable and modular architecture designed to support its key functionalities efficiently. At a high level, the system architecture comprises three main components: the frontend, backend, and database.

* + - **Frontend**: The frontend component is responsible for rendering the user interface and handling user interactions. It consists of a web-based interface built using modern web technologies:
      * HTML
      * CSS
      * JavaScript.
      * ajax
    - **Backend**: The backend component serves as the core logic as it handles user authentication, task management operations, and business logic implementation. It interacts with the database to store and retrieve task data and data processing layer of the application. It is implemented using
      * Spring Boot
* **Database**:
  + PostgreSQL: An open-source relational database management system for storing structured data. Known for its reliability, extensibility, and support for complex queries and transactions.
* **Design patterns:**:
  + We usedThe architecture of the todo list web application follows the Model-View-Controller (MVC) design pattern, which is essential for building scalable and maintainable web applications. The MVC pattern divides the application into three interconnected components, each with its own distinct responsibility.
* **security**:
  + (JWT)
  + Authguard

1. **System Interface**

# API Interfaces

Description:The To-Do List application exposes a set of APIs to facilitate communication between the frontend and backend components, allowing for the retrieval and manipulation of task data

* + - ​Health Check Endpoint:
      * Purpose: Endpoint to check the health/status of the application services.
      * Method: GET
      * URL: /health
      * Functionality:
        + Returns a message indicating that the service is alive.
    - Share Endpoint:
      * Purpose: Shares a folder, todo, or task.
      * Method: POST
      * URL: /{resourceType} (e.g., /folder, /todo, or /task)
      * Request Body: JSON object representing the share request (ShareDto, ShareSettings, or any relevant DTO).
      * Functionality:
        + Shares the specified resource based on the provided share request.
        + Returns true if sharing is successful, false otherwise.
    - Retrieve View Settings Endpoint:
      * Purpose: Retrieves the view settings for a folder, todo, or task.
      * Method: GET
      * URL: /{resourceType}/view (e.g., /folder/view, /todo/view, or

/task/view)

* + - * Parameters:
        + request: DTO object containing relevant settings (e.g., ShareDto).
      * Functionality:
        + Maps the relevant settings from the request to the appropriate DTO object.
        + Retrieves and returns the view settings for the specified resource.
    - Delete Resource Endpoint:
      * Purpose: Deletes a folder, todo, or task.
      * Method: DELETE
      * URL: /{resourceType}/{resourceId} (e.g., /folder/{folderId},

/todo/{todoId}, or /task/{taskId})

* + - * Parameters:
        + resourceId: Path parameter specifying the ID of the resource to be deleted.
      * Functionality:
        + Deletes the resource specified by the ID.
    - Create Resource Endpoint:
      * Purpose: Creates a new folder, todo, or task.
      * Method: POST
      * URL: /folders/{folderId}, /todos/{todoId}, or /tasks/{taskId}
      * Request Body: JSON object representing the resource request (FolderRequest, TodoRequest, TaskRequest, or any relevant DTO).
      * Functionality:
        + Creates a new resource based on the provided request.
        + Returns the created resource.
    - ​Update Resource Endpoint:
      * Purpose: Updates an existing folder, todo, or task.
      * Method: PUT
      * URL: /folders/{folderId}/{resourceId}, /todos/{todoId}/{resourceId}, or

/tasks/{taskId}/{resourceId}

* + - * Parameters:
        + resourceId: Path parameter specifying the ID of the resource to be updated.
      * Request Body: JSON object representing the updated resource request (FolderRequest, TodoRequest, TaskRequest, or any relevant DTO).
      * Functionality:
        + Updates the resource specified by the ID with the data provided in the request body.
        + Returns the updated resource.

# External System Interfaces

Description: The To-Do List application interacts with external systems, including the PostgreSQL database for storing task data.

# Database Interface:

* Purpose: Stores and retrieves task-related data
* Technology: PostgreSQL
* Interaction:

- The backend communicates with the PostgreSQL database using SQL queries.

# 4. Component:

1)TODO:

1.1) Task Management:

-creation of tasks within a todo list -updating tasks within a todo list -retrieval of tasks by their ID -retrieval of all tasks within a specified folder -deletion of tasks

1.2)Sharing:

-sharing todolis and add shortcuts to todo lists.

1.3)Folder Management:

-creation of folders to organize todo lists -retrieval of todo lists within a specified folder -delete folders and associated todo lists

1.4)Access Control:

-validation of user permissions for creating and updating todo lists -check for read and write permissions on folders and todo lists

1.5)utility class:

-contains utility methods used for building query parameters, URLs, and other shared functionalities

1.6)Integration with external service:

-integration with external services for sharing todo lists -usage of RESTful APIs for communication with other services

1.7)error handling:

-handling of access control errors and null pointer exceptions

2)Folder:

2.1)Folder Management:

• Creation of folders

• Updating folder details

• Retrieval of folders by their ID

• Retrieval of folders within a specified parent folder Retrieval of folders owned by a specific user Deletion of folders by their ID

2.2) Sharing Module:

• Sharing folders with other users

• Generating shareable links for folders

2.3) Access Control

• Validation of user permissions for editing and reading folders

• Checking if a user is the owner of a folder

2.4) Utility Methods

• Building query parameters for RESTful API requests and Building URLs for RESTful API requests

2.5) Integration with External Services:

• Integration with external services for sharing folders

• Usage of RESTful APIs for communication with other services

2.6) Error Handling:

• Handling of access control error

• Handling of runtime exceptions for folder operations

3) Share:

1)Share Management

• Creation or update of share settings

• Retrieval of file view

• Checking if a file is editable Checking if a file can be read by a user Checking if a file can be read and written by a user

2. User Management

• Saving users to either a TodoFile or a FolderFile

• Removing users from either a TodoFile or a FolderFile

3. Utility Methods:

• Checking if a user is found for a specific file

• Checking if a user is found for a TodoFile

• Checking if a user is found for a FolderFile

• Saving TodoFile users

• Saving FolderFile users

• Removing TodoFile users

• Removing FolderFile users

4. Data Repositories:

• ShareRepository for managing share settings.

• FolderUserRepository for managing folder-user relationships.

• TodoUserRepository for managing todo-user relationships.

5)Task

1. Model Components:

• Task: Represents a model for a task.

• Todo: Represents a model for a to-do list.

2. Data Access Interface Components (Repositories):

• TaskRepository: Used to interact with task data.

• TodoRepository: Used to interact with to-do list data.

3. Service Components:

• TaskService: Provides functions related to tasks, such as updating tasks, creating a new task, deleting a task, etc.

4. Utility Components:

• UtilityClass: Contains helper functions such as building query parameters and checking whether a user has permission to edit a specific task.

5. Other Libraries:

• Optional from java.util: Used to avoid calling methods on null objects.

• MultiValueMap from org.springframework.util: Used to represent maps that contain multiple keys mapping to multiple values.

• Objects from java.util: Used for checking for null objects.

• URI from java.net: Used to represent a Uniform Resource Identifier.

• Map from java.util: Used to represent key-value mappings.

2)Data Design:

User Table:

• Name: Contains the user's name.

• Username: Holds the username used for logging in.

• Password: Stores the user's password.

• Role: Specifies the user's role in the system, such as regular user or administrator.

• Organization\_ID: Foreign key referencing the Organization table, linking the user to the organization they belong to.

Organization Table:

• Name: Stores the name of the organization.

TodoList Table:

• Name: Records the name of the task list.

• Created\_Time: Indicates the date and time when the task list was created.

• Updated\_Time: Shows the date and time of the last update to the task list.

• Owner\_User\_ID: Foreign key linking the task list to the user who owns it.

• Permission\_ID: Foreign key associating the task list with its permissions.

• View: Specifies the visibility level of the task list, such as public or private.

Task Table:

• Name: Holds the name of the task.

• TodoList\_ID: Foreign key connecting the task to the task list it belongs to.

• Status: Indicates the status of the task, such as "completed" or "in progress".

• Editor\_User\_ID: Foreign key linking the user who edited the task.

Folder Table:

• Name: Contains the name of the folder.

• Owner\_User\_ID: Foreign key associating the folder with the user who owns it.

• Permission\_ID: Foreign key connecting the folder to its permissions.

• View: Specifies the visibility level of the folder, such as public or private.

Permission Table:

• Type: Specifies the type of permission, such as "read-only" or "public".

Todo\_Folder Table:

• TodoList\_ID: Foreign key linking the task list to the folders it belongs to.

• Folder\_ID: Foreign key connecting the folder to the task list it belongs to.

Todo\_Users Table:

• TodoList\_ID: Foreign key associating the task list with the users linked to it.

• User\_ID: Foreign key connecting the users to the task lists they belong to.

Permission\_Entity Table:

• Entity\_Type: Specifies the type of entity, such as "user" or "folder".

• Entity\_ID: Identifies the entity.

• User\_ID: Foreign key linking the entity to the user.

• Permission\_ID: Foreign key connecting the entity to its permissions.