Система управления задачами

Мой проект “Система управления задачами” реализует возможность:

1. Создавать задачи и сотрудников
2. Назначать сотрудникам задачи
3. Назначать задачам сотрудников
4. Просматривать, добавлять, редактировать, удалять сотрудников и задачи из списка
5. Переназначать задачи другим сотрудникам
6. Обновлять статусы задач в зависимости от наличия назначенного сотрудника этой задаче и статуса задачи
7. Менять позицию сотрудника и переназначать его задачи другим
8. Логировать уведомления о действиях в системе и ошибки, возникающие в процессе выполнения процессов
9. Разграничивать права доступа и предоставлять возможность управлять задачами только администратору
10. Фильтровать задачи по типу, приоритету, статусу, сроку выполнения и назначенному сотруднику (можно фильтровать как по одному параметру, так и по нескольким сразу)
11. Сортировать задачи по дате создания, срокам выполнения и приоритету. Также реализована возможность выбрать один из двух алгоритмов сортировки.

Дополнительно реализованы unit-тесты и использованы паттерны проектирования *Фасад(Facade)*, *Наблюдатель(Observer)* и *Одиночка(Singleton)*.

Структура проекта:

Структура проекта отображена в диаграмме.

Проект состоит из следующих классов:

class Employee

Это класс, который содержит информацию о сотруднике и позволяет создавать сотрудника с указанием имени, фамилии, должности. Например,

const employee = new Employee(‘John’, ‘Doe’, ‘engineer’);

В этом классе есть такие поля:

- *id* - идентификатор сотрудника (генерируется автоматически при добавлении нового сотрудника)
- *first_name* - имя сотрудника
- *last_name* - фамилия сотрудника
- *position* - должность
- *tasks* - задачи, назначенные этому сотруднику.

Основные методы класса - геттеры. Они позволяют получить информацию из каждого поля.

Методы *tasks* и *position* - сеттеры, с помощью которых сотруднику можно присвоить новую должность и добавить задачу.

Метод *assignTask* - метод, который вызывается при назначении задачи пользователю менеджером (EmployeeManager) и сразу же назначает текущей задаче текущего пользователя.

class Task

Это класс, который содержит информацию о задачах и позволяет создать задачу с указанием информации о ней. Например:

const task = new Task('Task 1', 'This is task number one', TaskTypesEnum.STORY, TaskPrioritiesEnum.HIGH, TaskStatusesEnum.FINISHED, 'week');

В этом классе есть такие поля:

- *id* - идентификатор задачи (генерируется автоматически при добавлении новой задачи)
- *name* - название задачи
- *description* - описание задачи
- *type* - тип задачи (реализован с помощью enum *TaskTypesEnum*, который предоставляет варианты - bug, story, task)
- *priority* - приоритет задачи (реализован с помощью enum TaskPrioritiesEnum, который предоставляет варианты - high, medium, low)
- *status* - статус задачи (реализован с помощью enum *TaskStatusesEnum*, который предоставляет варианты - new, in_progress, delayed, finished)
- *employee* - содержит информацию о сотруднике, назначенном к этой задаче.

Основные методы класса - геттеры. Они позволяют получить информацию из каждого поля.

Методы *status* и *setEmployee* - сеттеры, с помощью которых задаче можно присвоить новый статус и добавить сотрудника.

Метод *assignEmployee* - метод, который вызывается при назначении пользователя задаче менеджером (TaskManager) и сразу же назначает текущему пользователю текущую задачу.

class EmployeeList

Это класс, который хранит в себе список сотрудников в виде массива.
Одноименные геттер и сеттер *employeeList* позволяют получить список сотрудников и добавить в список нового сотрудника.

class TaskList

Этот класс, хранит в себе список задач в виде массива. Одноименные геттер и сеттер *taskList* позволяют получить список задач и добавить в список новую задачу.

class EmployeeManager

Это класс, который предназначен для управления сотрудниками. Он содержит поле *employees*, через которое реализует доступ к массиву с сотрудниками (EmployeeList).

Методы класса:

- *employeeList* - геттер, реализует доступ к списку сотрудников
- *employeeWithTasks* - геттер, реализует доступ к списку сотрудников, у которых есть назначенные задачи
- *addEmployeeToList* - реализует добавление нового сотрудника в список сотрудников
- *editEmployeeData* - реализует изменение данных сотрудника (кроме поля id)
- *deleteEmployee* - реализует удаление сотрудника из списка сотрудников
- *setTaskToEmployee* - реализует назначение задачи сотруднику
- *editPosition* - реализует изменение должности сотрудника и переназначение незавершенных задач другим сотрудникам
- *resetTask* - приватный метод, который используется внутри метода *editPosition* и реализует переназначение задач другим сотрудникам

Внутри метода *deleteEmployee* используется паттерн *Фасад(Facade)*. В процессе проверки информации перед удалением вызывается метод *findAndProcessEmployees*, внутри которого реализована проверка статусов задач, которые уже назначены текущему пользователю. Поскольку у менеджера *EmployeeManager* отсутствует доступ к задачам, то он вызывает метод фасада, который предоставляет ему нужную информацию.

Также в классе подключен *logger*, который позволяет записывать действия об изменениях данных сотрудников в отдельный массив внутри класса *Logger*. Он реализован с помощью паттерна проектирования *Одиночка(Singleton)*. Поэтому внутри класса *EmployeeManager* не создается инстанс логгера, а сразу вызывается его метод, который создает инстанс внутри класса Logger.

Кроме того, в классе *EmployeeManager* реализован паттерн *Наблюдатель(Observer)*, который позволяет логгеру подписываться на события класса и реагировать на них (записывать логи к себе в массив). Наблюдатель реализован с помощью метода *update*.

class TaskManager

Это класс, который предназначен для управления задачами. Он содержит поле *tasks*, через которое реализует доступ к массиву с задачами (*TaskList*).

Методы класса:

- *taskList* - геттер, реализует доступ к просмотру списка задач
- *taskWithEmployee* - геттер, реализует доступ к просмотру списка задач, которым назначен сотрудник
- *role* - геттер и сеттер, позволяет посмотреть права доступа менеджера и изменить их
- *createNewTask* - реализует добавление задачи в список задач
- *setEmployeeToTask* - реализует назначение сотрудника задаче
- *editTask* - реализует изменение задачи
- *deleteTask* - реализует удаление задачи
- *renewTaskStatus* - реализует смену статуса задачи
- *filterTasks* - реализует фильтрацию задач по указанному условию (одному или нескольким)
- *bubbleSortTasks* - реализует сортировку задач по заданному полю ('date_created', 'term', 'priority') с использованием алгоритма “Сортировка пузырьком)
- *choiceSortTasks* - реализует сортировку задач по заданному полю ('date_created', 'term', 'priority') с использованием алгоритма “Сортировка выбором)
- *canManageTasks* - приватный метод, который проверяет права доступа менеджера

В этом классе также используется логгер в виде паттерна проектирования *Одиночка(Singleton)* в сочетании с паттерном *Наблюдатель(Observer)*, который представлен в виде метода *update*.

class TaskManagementSystem

Это главный класс, который находится в корне проекта - index.ts. Он предназначен для осуществления всех ранее описанных функций системы управления задачами, путем обращения к методам двух основных менеджеров - *TaskManager* и *EmployeeManager*.

Он не содержит в себе реализацию функционала системы управления задачами, но предоставляет “интерфейс” для того, чтобы можно было осуществить взаимодействие сотрудников и задач.

В этом классе есть 2 поля:

1. *taskManager* - инстанс класса TaskManager
2. *employeeManager* - инстанс класса EmployeeManager

Методы класса:

- *tasks* - геттер, просмотр списка задач
- employees - геттер, просмотр списка сотрудников
- *tasksWithAssignedEmployees* - геттер, просмотр списка задач, которым назначены сотрудники
- *employeesWithAssignedTasks* - геттер, просмотр списка сотрудников, которым назначены задачи

- *addTask* - добавление задачи
- *addEmployee* - добавление сотрудника
- *editTask* - редактирование задачи
- *editEmployee* - редактирование информации о сотруднике
- *editTaskManagerRole* - изменение прав доступа
- *deleteTask* - удаление задачи
- *deleteEmployee* - удаление сотрудника
- *sortTasks* - сортировка задач с выбором алгоритма ('bubble', 'choice')
- *filterTasks* - фильтрация задач по заданному полю
- *renewTaskStatus* - обновление статуса задачи
- *assignTaskToEmployee* - назначение задаче сотрудника
- *assignEmployeeToTask* - назначение задачи сотруднику
- *editEmployeePosition* - изменение должности сотрудника

class TaskFilter

Это вспомогательный класс, который используется для указания опциональных параметров в функции фильтрации. Инстансу класса необходимо присвоить те поля, которые будут использоваться при фильтрации задач и передать его в качестве параметра методу *filterTasks.*

Выводы: Система управления задачами реализована таким образом, что ее функционал можно дополнять и расширять, не нарушая работу уже существующих методов.