Министерство образования, культуры и исследований

Республики Молдова

Технический Университет Молдовы

Департамент Программной Инженерии и Автоматики

**Курсовая работа**

**По дисциплине**

Tehnici și Mecanizme de Proectare a Software

**Тема:** Создание веб-приложения To-do List

Выполнил ст. гр. TI-196: Герца Николай

подпись, дата

Научный руководитель: Поштару Андрей

подпись, дата

Булдумак Андрей

подпись, дата

*Кишинёв 2022*

**Оглавление**

[Введение 3](#_Toc73740495)

[1. Сравнительное исследование системы 4](#_Toc73740496)

[1.1. Анализ исследуемой области 4](#_Toc73740497)

[1.2. Сравнение с существующими аналогами 6](#_Toc73740498)

[2. Анализ системы 9](#_Toc73740499)

[3. Проектирование системы 10](#_Toc73740500)

[4. Разработка приложения 13](#_Toc73740501)

[4.1. Описание используемых инструментов разработки 13](#_Toc73740502)

[4.2. Описание используемых паттернов проектирования 16](#_Toc73740503)

[Заключение 24](#_Toc73740504)

[Библиография 25](#_Toc73740505)

[Приложения 26](#_Toc73740506)

# Введение

В современном мире существует громадное количество языков программирования. Часть из этих языков появились еще десятки лет назад, часть из них появились несколько лет назад. Некоторые языки давно забросили и больше не поддерживают, некоторые получает обновления несколько раз за год, постоянно становясь лучше и эффективней. Действительно же выбрать первый язык программирования, с которого начнется путь человека в сферу IT – крайне тяжело. Мало того, что для каждой конкретной сферы и проблемы существуют от двух или трех до нескольких десятков различных языков, так еще и существует куча различных фрэймворков, которые помогает реализовать ту или иную задачу. Так же, если у человека есть желание стать программистом, в первую очередь, он должен быть коммуникабельным, потому что ни один проект и ни одна программа серьезного уровня не была написана и реализована в одиночку, прежде всего – это командная работа.

Каждый существующий язык хорош по-своему для своей области. Для программирования микроконтроллеров, хорошо подойдет С++ или С; для программирования десктопных приложений подойдут Java, Python, C#, Ruby; для разработки веб приложений подойдут PHP, JavaScript, Java, Python; для разработки мобильных приложений подойдут С, Java, C#, Swift и так далее. Так же, многие из них можно комбинировать для большей эффективности. К сожалению, выучить все языки программирования ни у одного человека никогда не получится, так как их тысячи. Куда лучше выбрать один, изучить его досконально для понимания концепции и архитектуры данной сферы. И в будущем, при желании карьерного роста, можно выбрать еще несколько и так же их постепенно изучать и практиковать. Для выбора того, на чем учиться, для начала, стоить изучить ранок, изучить спрос, узнать, какие сферы сейчас более популярные, какие менее, выбрать что-то подходящее человеку по интересу и уже потом выбирать язык.

По такому пути пошел я и остановился на разработке frontend приложений, используя HTML, CSS, JavaScript. В интернете достаточно много документаций, связанный с данными технологиями, что позволит без труда находить нужную информацию.

Я поставил перед собой задачу в данной курсовой работе: проанализировать, спроектировать и реализовать приложение to-do list для записи и сохранения задач, которые необходимо сделать.

Для моделирования системы был выбран язык UML.

# 1. Сравнительное исследование системы

## 1.1. Анализ исследуемой области

Веб-разработка - процесс создания веб-сайта или веб-приложения. Основными этапами процесса являются веб-дизайн, вёрстка страниц, программирование на стороне клиента и сервера, а также конфигурирование веб-сервера.

Основные этапы разработки WEB-сайта:

- Создание технического задания. Работа с заказчиком начинается с заполнения брифа, в котором заказчик излагает свои пожелания относительно визуального представления и структуры сайта, указывает на ошибки в старой версии сайта, приводит примеры сайтов конкурентов. Исходя из брифа, программист составляет техническое задание, учитывая возможности программных и дизайнерских средств. Этап заканчивается после утверждения технического задания заказчиком. Важно сразу отметить, что этапы проектирования веб-сайтов зависят от многих факторов, таких как объём сайта, функциональность, задачи, которые должен выполнять будущий ресурс и многое другое.

- Дизайн основной и типовых страниц сайта. Начинается работа с создания дизайна, обычно в графическом редакторе. Дизайнер создаёт один или несколько вариантов дизайна, в соответствии с техническим заданием. При этом отдельно создаётся дизайн главной страницы, и дизайны типовых страниц (например: статьи, новости, каталог продукции). Собственно, «дизайн страницы» представляет собой графический файл, слоеный рисунок, состоящий из наиболее мелких картинок-слоев элементов общего рисунка.

- HTML-верстка. Утверждённый дизайн передаётся HTML-верстальщику, который «нарезает» графическую картинку на отдельные рисунки, из которых впоследствии складывает HTML-страницу с использованием CSS. В результате создаётся код, который можно просматривать с помощью браузера. А типовые страницы впоследствии будут использоваться как шаблоны.

- Программирование. Далее готовые HTML-файлы передают программисту. Программирование сайта может осуществляться как «с нуля» на основе существующих фрэймворков, например, ReactJS (который и будет использоваться в последствии), так и на основе CMS — системы управления сайтом.

- Завершающим этапом разработки сайта является тестирование. Процесс тестирования может включать в себя самые разнообразные проверки: вид страницы с увеличенными шрифтами, при разных размерах окна браузера, при отсутствии флэш-плеера и многие другие. Также — юзабилити-тестирование.

Теперь более подробно про то, что такое todo list и зачем он вообще нужен в повседневной жизни человека.

Грамотно расставить приоритеты помогает техника todo List, или по-простому – список задач. Правильно составленный, этот список помогает завершать действительно важные дела, отслеживать «хвосты» и чувствовать себя полноценным и продуктивным.

Todo List – один из очень важных инструментов планирование времени. Представляет из себя список задач или каких-либо запланированных дел. Это может быть как список рабочих задач, так и ежедневных задач, которые нужно выполнить человеку. Так же там могут быть задачи, которые он откладывает или которые нужно выполнить в ближайшем будущем, чтобы не забыть, они записываются в todo list.

Очень полезно иметь такой список для часто повторяющихся дел. Например, сбор вещей в отпуск, командировку, вещей ребенку в школу. Имея под рукой такой список сборы пройдут быстрее и меньше шансов что-либо забыть.

## 1.2. Сравнение с существующими аналогами

Существует множество аналогов данного приложения от разных компаний. Самые популярный из них: TickTick; Any.do; Trello; Google Keep; Microsoft To-Do и так далее.

Планировщик TickTick поможет разобраться с делами и установить напоминания, добавляя к задачам подпункты, теги, файлы и документы. Дела группируются в тематические папки и стеки. В TickTick есть поддержка голоса, поэтому задачи можно ставить не только письменно. А для улучшения концентрации при работе создатели добавили классический и удобный таймер в стиле pomodoro. TickTick есть в виде приложений для Android и iOS и как расширение браузеров Google Chrome и Firefox.

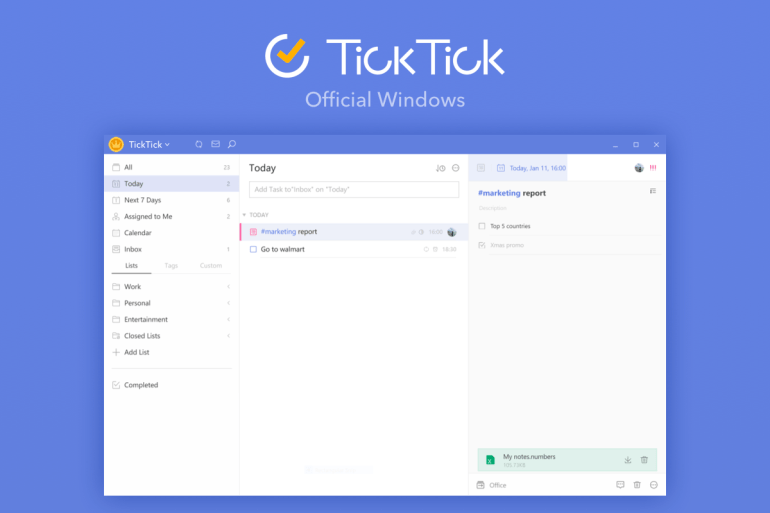


Рисунок 1. - Планировщик TickTick

Any.do —самый минималистичный менеджер задач из списка популярных планировщиков. При этом у приложения много возможностей, включая добавление тегов к делам, напоминания, привязанные к месту или времени, вложения и структурированные подзадачи. С Any.do удобно ходить за покупками, записывая нужные продукты или товары во встроенном заметочнике, а для детальной планировки дел тут есть календарь. Составленными задачами, списками покупок и поездками удобно делиться с членами семьи, коллегами и друзьями. Any.do поддерживает управление голосом (Alexa и Siri) и доступен на Android и iOS.

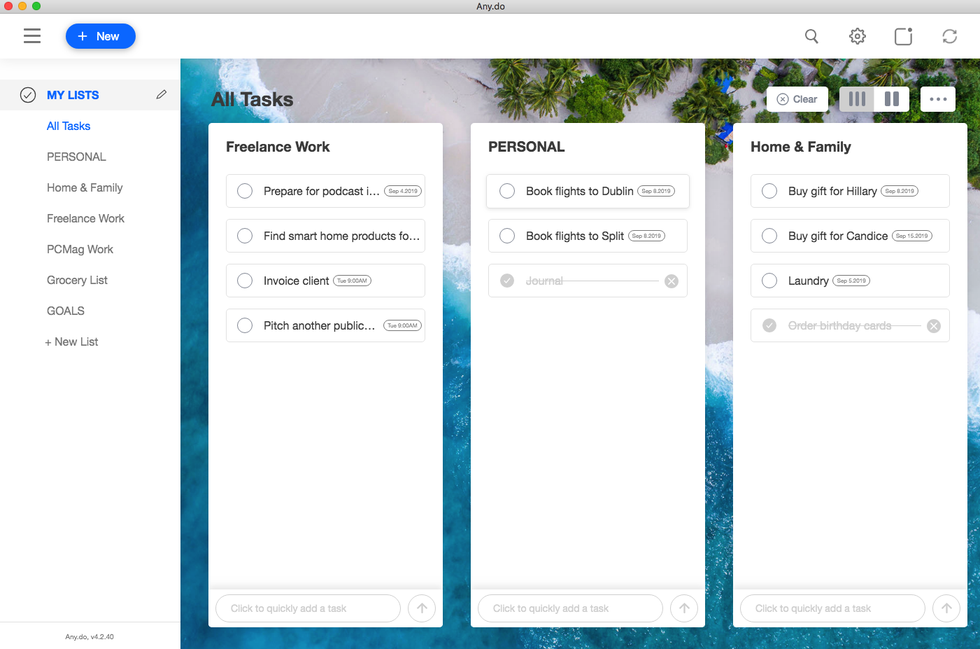
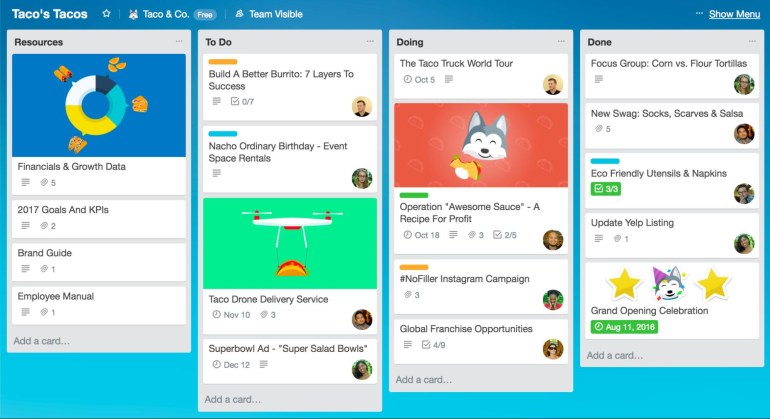


Рисунок 2. – Менеджер задач Any.do

Trello — одно из самых популярных приложение с канбан-досками. Это удобные карточки с делами, которые рассортировываются по тематическим столбикам. Каждому заданию присваивается название, прикрепляются документы, файлы, добавляются участники и сроки выполнения, расписываются подробности дела и так далее. Чаще всего Trello используется не только в личных целях, но и для работы. Приложение помогает маленьким или большим компаниям, корпорациям и СМИ. В Trello организована детальная визуальная настройка: можно установить задний фон, прикрепить к карточке картинку, загрузить видео или покрасить в другой цвет. Trello есть в виде веб-версии с расширениями для Chrome и Firefox, отдельной программой для Windows и MacOS или в виде приложения на Android и iOS.

Рисунок 3. – доска Trello

Эти приложения помогут организовать личную жизнь и работу, повысят количество выполненных дел, не дадут забыть о важных вещах и освободят голову от постоянного «шума».

# 2. Анализ системы

Проектирование системы будет происходит в Visual Studio Code. VSCode это бесплатный редактор кода от Microsoft. Он имеет открытый исходный код и легковесен при установке. VSCode отлично подходит для начинающих JavaScript-разработчиков, поскольку имеет хороший набор функционала «из коробки», без необходимости устанавливать дополнительные плагины. Но этот редактор популярен не только среди начинающих программистов. Он может стать идеальным выбором для более продвинутых пользователей, которым нужно просто быстро приступить к работе, не тратя лишнее время на настройки.

Уникальная особенность VSCode — возможность использовать его в браузере. Таким образом вы можете пользоваться редактором на планшете и при этом иметь такую же среду, к какой привыкли в десктопной версии. Чтобы этот функционал заработал, нужно еще настроить code-server в сети, к которой вы имеете доступ, но возиться с настройками придется только единожды, и оно того стоит.

Git здесь встроен в редактор, но интеграция не такая надежная, как в некоторых других редакторах. Например, пользователи WebStorm предпочитают применять push/merge, а не то, что предлагает VSCode.

Проект будет написан на основе фрэймворка (библиотеки) ReactJS. React может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость, простоту и масштабируемость. В качестве библиотеки для разработки пользовательских интерфейсов React часто используется с другими библиотеками, такими как MobX, Redux и GraphQL.

# 3. Проектирование системы

В качестве языка для проектирования системы был выбрал язык UML. UML – унифицированный язык моделирования (Unified Modeling Language) – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем.

Первая диаграмма – это диаграмма Use Case. Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

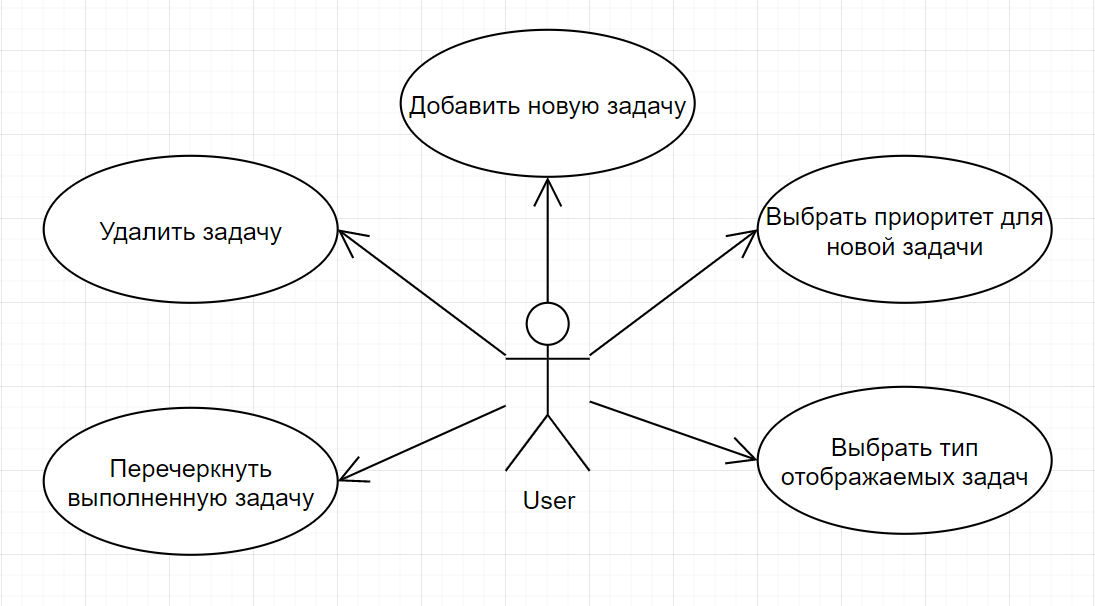


Рисунок 4. – Диаграмма Use Case. Общий функционал системы

На данной диаграмме вариантов использования показано, что может делать пользователь на сайте, а именно: добавить новую задачу, выбрать приоритет для новой задачи, выбрать тип отображаемых задач, перечеркнуть выполненную задачу, удалить задачу.

Следующий вид диаграммы – диаграмма последовательности. Для моделирования взаимодействия объектов в языке UML используются диаграммы последовательности. Говоря об этих диаграммах, имеют в виду два аспекта взаимодействия. Взаимодействия объектов можно рассматривать во времени, и тогда для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности.

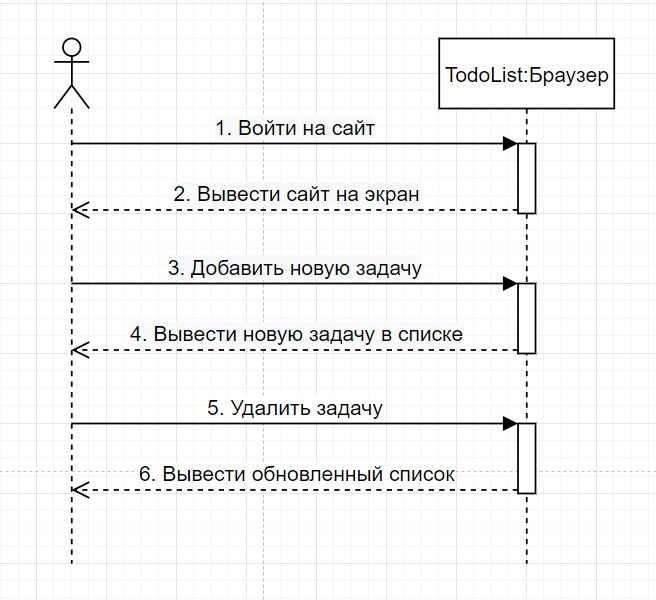


Рисунок 5. – Диаграмма последовательности. Работа с сайтом

На данной диаграмме последовательности показано, как пользователь может взаимодействовать с сайтом, а именно с приложением todo list. Он входит на сайт, добавляет новую задачу, и когда это необходимо – удаляет ее.

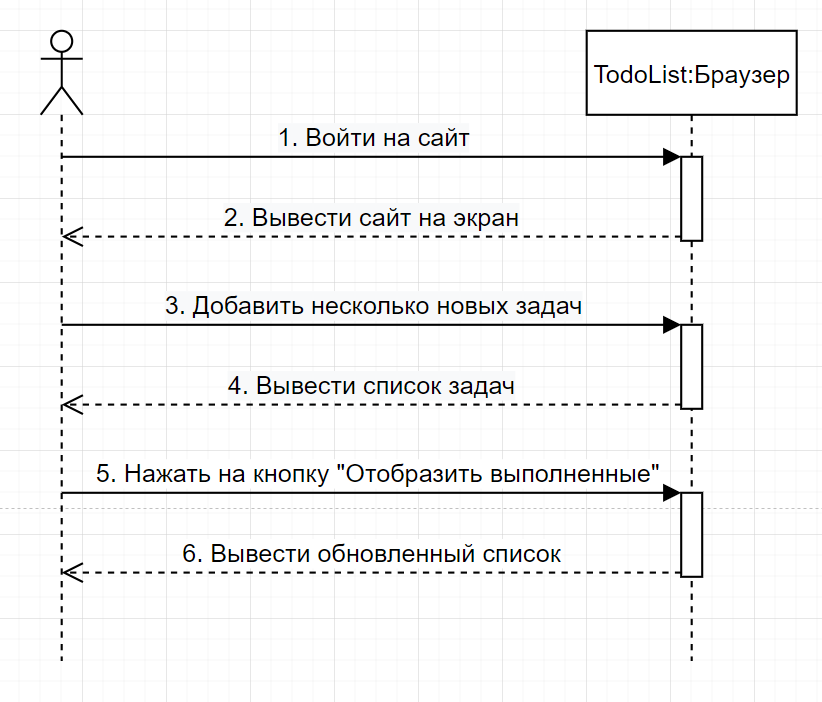


Рисунок 6. – Диаграмма последовательности. Отображенные выполненных задач

На данной диаграмме последовательности показано, как пользователь может выводить необходимые ему задачи, то есть: все, выполненные, невыполненные. В данном примере он выводит только выполненные задачи.

# 4. Разработка приложения

## 4.1. Описание используемых инструментов разработки

Современный JavaScript – это «безопасный» язык программирования. Он не предоставляет низкоуровневый доступ к памяти или процессору, потому что изначально был создан для браузеров, не требующих этого.

Возможности JavaScript сильно зависят от окружения, в котором он работает. Например, Node.JS поддерживает функции чтения/записи произвольных файлов, выполнения сетевых запросов и т.д.

В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером.

Возможности JavaScript в браузере ограничены ради безопасности пользователя. Цель заключается в предотвращении доступа недобросовестной веб-страницы к личной информации или нанесения ущерба данным пользователя.

Три сильные стороны JavaScript: полная интеграция с HTML/CSS; простые вещи делаются просто; поддерживается всеми основными браузерами и включён по умолчанию. JavaScript – это единственная браузерная технология, сочетающая в себе все эти три вещи. Вот что делает JavaScript особенным. Вот почему это самый распространённый инструмент для создания интерфейсов в браузере. Хотя, JavaScript позволяет делать приложения не только в браузерах, но и на сервере, на мобильных устройствах и т.п.

Реализованный проект был построен на основе ReactJS. React — это инструмент для создания пользовательских интерфейсов. Его главная задача — обеспечение вывода на экран того, что можно видеть на веб-страницах. React значительно облегчает создание интерфейсов благодаря разбиению каждой страницы на небольшие фрагменты. Называются эти фрагменты компонентами. Вот пример разбивки страницы на компоненты:



Рисунок 7. – Компоненты страницы

Компонент React — это участок кода, который представляет часть веб-страницы. Каждый компонент — это JavaScript-функция, которая возвращает кусок кода, представляющего фрагмент страницы. Для формирования страницы мы вызываем эти функции в определённом порядке, собираем вместе результаты вызовов и показываем их пользователю.

Основные элементы, которые нужны для полноценной работы с React:

- компоненты React;

- рендеринг ReactDOM;

- классы компонентов и функциональных компоненты;

- JSX;

- состояние (state);

- обработка событий;

- асинхронный метод setState;

- свойства (props);

- ссылки (refs).

React используется сотнями крупных компаний по всему миру, включая Netflix, Airbnb, American Express, Facebook, WhatsApp, eBay и Instagram. Это доказательство того, что у инструмента есть ряд преимуществ, с которыми невозможно конкурировать.

Вот некоторые из причин, чтобы использовать его:

- простота в Использовании;

- поддерживает Java-компонент Многократного Использования;

- простое Написание Компонентов;

- лучшая Производительность с Virtual DOM;

- SEO Friendly.

При создании React приложения, появляются много лишних файлов, после их удаления и добавления необходимых компонентов и картинок получается следующую структуру:

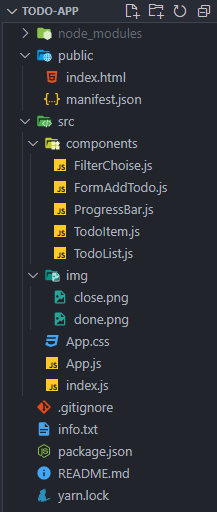


Рисунок 8. - Структура проекта

Демонстрация приложения показана в: Приложении А, Приложении Б, Приложении В, Приложении Г, Приложении Д, Приложении Е.

## 4.2. Описание используемых паттернов проектирования

Паттерн проектирования — это часто встречающееся решение определённой проблемы при проектировании архитектуры программ.

В отличие от готовых функций или библиотек, паттерн нельзя просто взять и скопировать в программу. Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет ещё подстроить под нужды вашей программы.

Паттерны часто путают с алгоритмами, ведь оба понятия описывают типовые решения каких-то известных проблем. Но если алгоритм — это чёткий набор действий, то паттерн — это высокоуровневое описание решения, реализация которого может отличаться в двух разных программах.

Паттерны необходимо знать, так как они дают данные преимущества в процессе разработки:

- Проверенные решения. Вы тратите меньше времени, используя готовые решения, вместо повторного изобретения велосипеда. До некоторых решений вы смогли бы додуматься и сами, но многие могут быть для вас открытием;

- Стандартизация кода. Вы делаете меньше просчётов при проектировании, используя типовые унифицированные решения, так как все скрытые проблемы в них уже давно найдены;

- Общий программистский словарь. Вы произносите название паттерна, вместо того, чтобы час объяснять другим программистам, какой крутой дизайн вы придумали и какие классы для этого нужны.

Существуют три основных вида паттернов:

- Порождающие паттерны беспокоятся о гибком создании объектов без внесения в программу лишних зависимостей;

- Структурные паттерны показывают различные способы построения связей между объектами;

- Поведенческие паттерны заботятся об эффективной коммуникации между объектами.

В данном проекте были реализованы следующие паттерны проектирования: одиночка, фасад, шаблон, наблюдатель. Так же косвенно тут используется паттерны: итератор, посредник, состояния и стратегия.

**Signleton (Одиночка. Порождающий паттерн)**

Singleton - это паттерн проектирования который ограничивает создание некого класса, позволяя создавать только один экземпляр. Обычно он используется для управления неким глобальным состояние приложения. Singleton должен быть неизменяемым с точки зрения его использования, поэтому не должно быть никаких рисков при попытке создать несколько экземпляров подобного объекта.

Это подход, когда класс может иметь только один экземпляр и есть какая-то точка доступа к этому экземпляру. Так как я использовал функциональное программирование и тут нет классов, то это можно перефразировать таким образом: объект может иметь только один экземпляр и есть какая-то точка доступа к этому экземпляру.

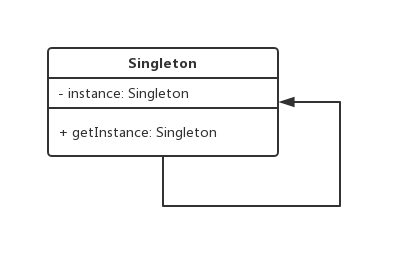


Рисунок 10. - Структура паттерна Singleton

Реализация в коде:

const [todoList, setTodoList] = useState([]);

const [filteredList, setFilteredList] = useState([]);

…

useEffect(() => {

    getLocalStorage();

  }, []);

  useEffect(() => {

    filterHandler();

    createLocalStorage();

  }, [todoList, filt]);

  const createLocalStorage = () => {

    localStorage.setItem("todoItem", JSON.stringify(todoList));

  };

  const getLocalStorage = () => {

    if (localStorage.getItem("todoItem" === null)) {

      localStorage.setItem("todoItem", JSON.stringify([]));

    } else {

      setTodoList(JSON.parse(localStorage.getItem("todoItem")));

    }

  };

Мы вызываем переменную todoList каждый раз, когда на странице что-то произошло. И только в первый раз, когда мы загружаем страницу, эта переменная создается и далее, с помощью хука useState() только обновляется значение в ней, а не пересоздается. То есть, если todoList у нас уже есть, то возвращается этот же todoList. Так же тут реализован local storage, который позволяет сохранить данные, которые были уже введены. В случае, если данных нет, создается новый item в local storage, если же там есть данные, то он не создает новый item, а загружает тот, который уже есть.

**Facade (Фасад. Структурный паттерн)**

Facade - это структурный паттерн проектирования, который предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку. Его задача скрыть сложную логику за простым фасадом, то есть собрать различные сложные структуры, объединить их, и выдать простой способ манипуляции.



Рисунок 11. - Структура паттерна Facade

У меня есть в компоненте (отдельный файл) FormAddTodo шаблон для создания новой задачи. В нем есть метод addToDo, который в свою очередь вызывает функцию createNewTask, в которую передает название задачи, приоритет ее и дату создания, а после этого вызывает setValue, который обнуляет input, то есть объединение сразу двух методов, которые срабатывают одновременно. createNewTask ничего не знает о реализации функции, в которую она передает три параметра. Уже функция newTask, которая принимает в файле App.js эти параметры, имеет логику, которая создает новую задачу.

Так же, у меня есть хук useEffect, которые объединяет два метода, которые вызываются, если todoList был изменен или была изменена опция отображения списка, то есть все, выполненные или невыполненные.

Реализация в коде:

function addToDo(*event*) {

*event*.preventDefault();

    if (value.trim()) {

      createNewTask(value, prio, today);

      setValue("");

    }

  }

…

  useEffect(() => {

    filterHandler();

    createLocalStorage();

  }, [todoList, filt]);

**Template Method (Шаблонный метод. Поведенческий паттерн)**

Template Method - это поведенческий паттерн проектирования, который определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы. Паттерн позволяет подклассам переопределять шаги алгоритма, не меняя его общей структуры.

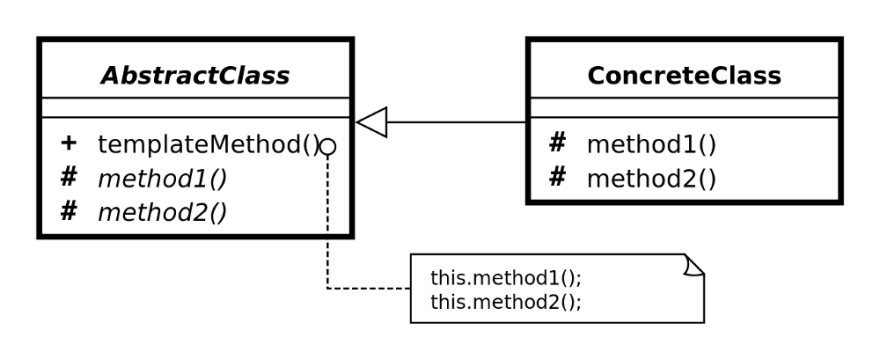


Рисунок 12. - Структура паттерна Template Method

У меня есть компонент TodoList, который в тэге ul создает маркированный список. В этом списке метод map() проходится по массиву и для каждого элемента создает свой компонент TodoItem, то есть для каждого элемента массива создается свой шаблон. Для каждого элемента стили, кнопки и ключи являются одинаковым, но разные значения ключей объекта.

Реализация в коде:

 return (

    <ul>

      {*todoList*.map((*item*) => {

        console.log(*item*);

        return <TodoItem *item*={*item*} *key*={*item*.id} *changeCompleted*={*changeCompleted*} *deleteTask*={*deleteTask*} />;

      })}

    </ul>

  );

**Observer (Наблюдатель. Поведенческий паттерн)**

Observer - это поведенческий паттерн проектирования, который создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

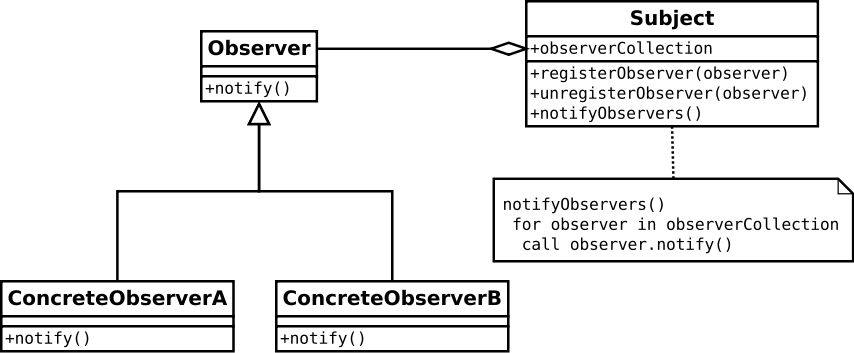


Рисунок 13. - Структура паттерна Observer

В компоненте TodoItem у меня есть кнопка, которая в случае завершения задачи, перечеркивает текст в задаче. В зависимости от того, какое значение находится в свойстве ключа completed, будет добавляться или удаляться стиль, который перечеркивает стиль. То есть, переменная todoList подписывается на изменения, которые происходят по нажатию на кнопку.

Реализация в коде:

export default function TodoItem({ *item*, *changeCompleted*, *deleteTask* }) {

  const styles = ["for-item "];

  const forPriority = ["priority "];

  if (*item*.completed) {

    styles.push("done");

    forPriority.push("done");

  }

  if (*item*.priorityTask === "High priority") forPriority.push("red-text");

  if (*item*.priorityTask === "Medium priority") forPriority.push("yellow-text");

  if (*item*.priorityTask === "Low priority") forPriority.push("green-text");

  if (*item*.priorityTask === "None") forPriority.push("none-text");

  return (

    <div *className*="task-item">

      <div *className*={styles.join(" ")}> {*item*.text}</div>

      <div *className*="priority-img">

        <div *className*={forPriority.join(" ")}>{*item*.priorityTask}</div>

        <img

*className*="images-item"

*src*={doneButtonImg}

*alt*="abc"

*height*="20px"

*width*="20px"

*onClick*={() => {

            changeCompleted(*item*.id);

          }}

        />

        <img

*className*="images-item"

*src*={undoneButtonImg}

*alt*="abc"

*height*="19px"

*width*="19px"

*onClick*={() => {

            deleteTask(*item*.id);

          }}

        />

      </div>

    </div>

  );

}

# Заключение

В данной практической работе я несколько месяцев изучал HTML, CSS, JS, ReactJS. В процессе данного изучения я ознакомился с большей частью возможностей данных языков и технологий, выучил много различных принципов работы с React, научился наперед проектировать систему. Основной целью React является упрощение рассуждений об интерфейсе и его состоянии в любой момент времени, разделяя UI на коллекцию компонентов. Так же я конкретно понял, для чего на самом деле нужен React, а именно: отлично подходит для командной разработки, строгое соблюдение UI, и шаблона рабочего процесса; UI код читабельный и прост в сопровождении; Разработка UI на основе отдельных компонентов — это будущее web-разработки.

Todo list помогает организовать себя и свою работу. Особенно полезен, когда вам предстоит сделать много разных мелких дел в сжатые сроки. Этот список позволяет не держать в голове все задачи, а сконцентрироваться на текущей.

# Библиография

1. Паттерны проектирования [Электронный ресурс]: https://refactoring.guru/ru/design-patterns

2. Современный учебник JavaScript [Электронный ресурс]: https://learn.javascript.ru/

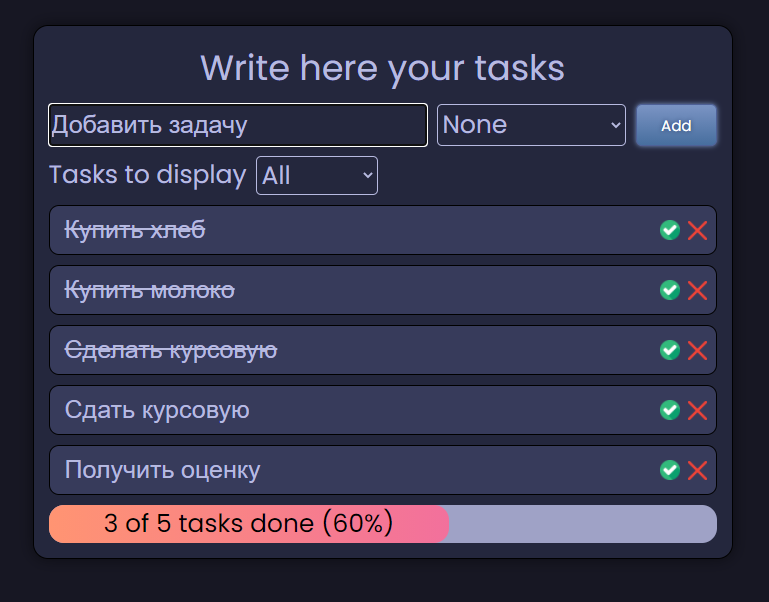
3. MDN Web Docs [Электронный ресурс]: https://developer.mozilla.org/

4. React документация[Электронный ресурс]: https://ru.reactjs.org/

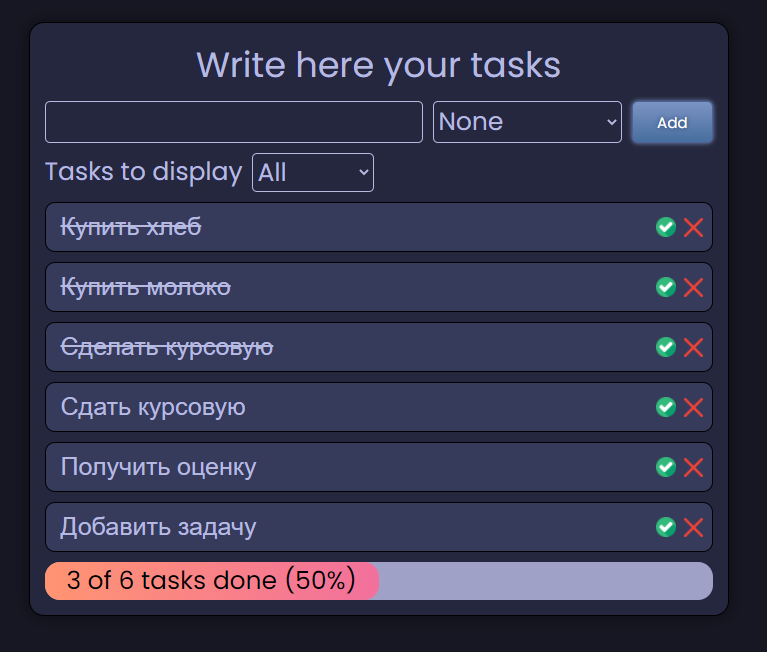
5. Скринкаст по React [Электронный ресурс]: https://learn.javascript.ru/screencast/react#react-components

# Приложения

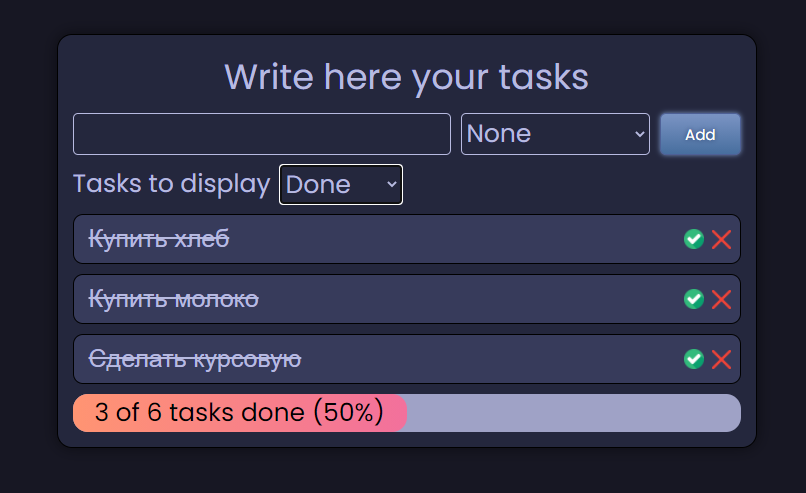
Приложение А



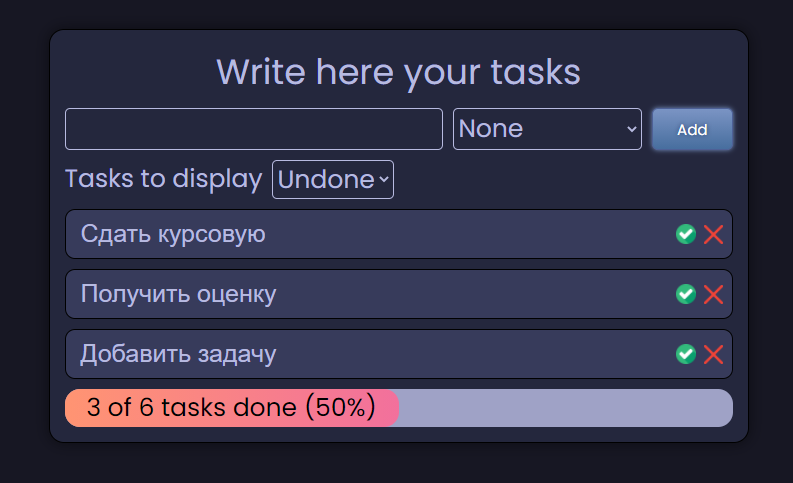
Приложение Б



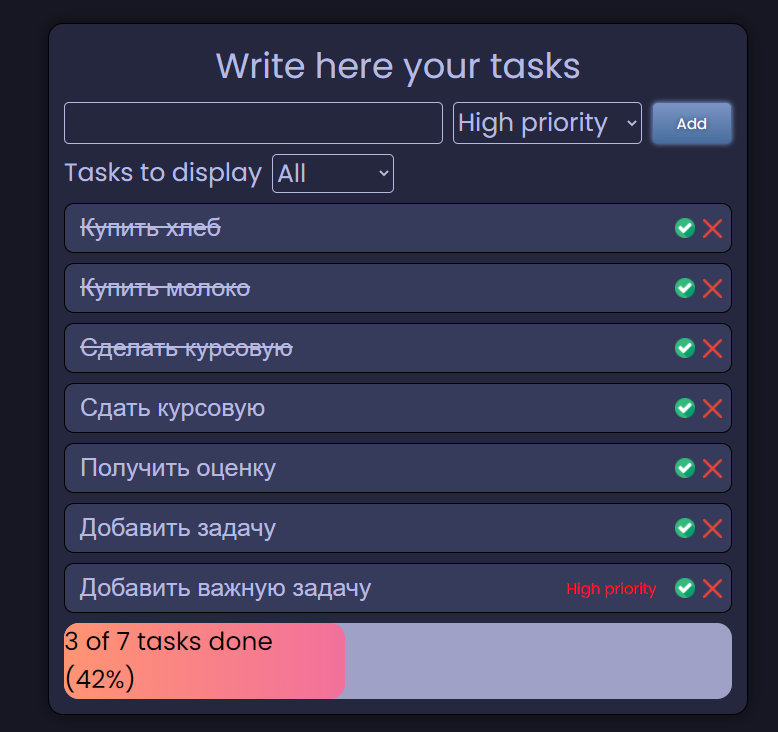
Приложение В



Приложение Г



Приложение Д



Приложение Е

