Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Модели данных и системы управления базами данных

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему
Day Tracker

БГУИР КП 1-40 04 01 011 ПЗ

Выполнила студентка:
гр. 753501 Самойлик Е.Н.

Руководитель:
Ассистент
кафедры информатики
Удовин И.А.

Минск, 2020

Содержание

ВВЕДЕНИЕ

На данный момент все большую популярность завоевывают облачные сервисы, которые не требуют от пользователя дорогого и мощного оборудования и установки чего-либо кроме веб-браузера. Примером могут служить Google Stadia, платформа облачного гейминга, для использования которой требуется только высокоскоростное интернет-соединение, другим примером может служить большое количество онлайн платформ для просмотра фильмов, прослушивания музыки и много другого. Идея этого веб-приложения заключается в предоставлении пользователю возможности следить за своим временем и делами, иными словами, это трекер задач с которым можно работать с помощью веб-браузера, что позволяет использовать его с компьютера, телефона.

1. Анализ предметной области

1.1 Аналоги

Все рассмотренные списки дел (Wunderlist, Todoist, Any.Do) обладают схожим набором функций, таких как создание задачи, добавление подзадач, назначение времени напоминаний, сортировка задач по важности. Более подробно будет рассмотрен Wunderlist.

- Wunderlist

  1. Простой мобильный список дел.

  2. Создание задач для одного человека, приоритеты, прикрепление файлов, создание подзадач, с назначением дополнительного времени для каждой подзадачи.

  3. Интеллектуальное определение времени задач, например, создание задачи "сходить завтра вечером в кино" автоматически установит время задачи на следующий день на 19:00).

  4. Возможность распечатать список дел.

  5. Нет возможности создания шаблонов.

  6. Нет возможности показать календарь с запланированными событиями.

- Google календарь

  1. Календарь с возможностью добавления задач на конкретный день/время.

  2. Обладает набором функций схожими с более простыми списками дел, но календарь тесно взаимодействует с другими сервисами.

  3. Возможность добавления календаря другого человека/календаря праздников.

  4. Возможность создания повторяющихся событий.

- Яндекс календарь

  1. Набор функций схож с функциями Google Календаря.

  2. Возможность уведомления с помощью смс.

- Яндекс трекер

  1. Мощный трекер задач для команд разработчиков.

  2. Возможность создавать задачи для нескольких человек.

  3. Возможность сортировать задачи по группам, создавать очереди задач.

  4. Возможность быть наблюдателем задачи.

  5. Возможность комментировать задачи.

- YouTrack

  1. Использует базу данных по принципу ключ-значение.

2. Действия с группами задач.
3. Возможность импортировать задачи, созданные в других системах отслеживания ошибок.
4. Интеграция с системами управления версиями.

2. Используемые технологии

Для начала рассмотрим набор языков и технологий, которые будут использоваться в разработке.

2.1 Node.js

Node (или более формально Node.js) — это кроссплатформенная среда выполнения с открытым исходным кодом, которая позволяет разработчикам создавать всевозможные серверные инструменты и приложения на JavaScript. Среда выполнения предназначена для использования вне контекста браузера (т.е. работает непосредственно на компьютере или серверной ОС). Таким образом, среда исключает специфичные для браузера API JavaScript и добавляет поддержку более традиционных API ОС, включая HTTP и библиотеки файловой системы.

С точки зрения разработки веб-сервера Node имеет ряд преимуществ:

- Отличная производительность. Node был разработан для оптимизации пропускной способности и масштабируемости в веб-приложениях и является хорошим решением многих распространенных проблем веб-разработки (например, веб-приложений в реальном времени).
- Код написан на «простом старом JavaScript», что означает, что меньше времени тратится на «сдвиг контекста» между языками, когда вы пишете как клиентский, так и серверный код.
- JavaScript — это относительно новый язык программирования, который выигрывает от улучшений в дизайне языка по сравнению с другими традиционными языками веб-серверов (например, Python, PHP и т. Д.). Многие другие новые и популярные языки компилируются / конвертируются в JavaScript, поэтому вы также можете использовать TypeScript, CoffeeScript, ClojureScript, Scala, LiveScript и т. Д.
- Диспетчер пакетов узлов (NPM) обеспечивает доступ к сотням тысяч повторно используемых пакетов. Он также имеет лучшее в своем классе разрешение зависимостей и может использоваться для автоматизации большей части инструментальной цепочки сборки.
- Node.js портативен. Он доступен в Microsoft Windows, macOS, Linux, Solaris, FreeBSD, OpenBSD, WebOS и NonStop OS. Кроме того, он хорошо поддерживается многими провайдерами веб-хостинга, которые часто предоставляют определенную инфраструктуру и документацию для размещения сайтов Node.

2.2 Express

Express — это самая популярная веб-платформа Node и базовая библиотека для ряда других популярных веб-платформ Node. Он предоставляет механизмы для:

- Написания обработчиков запросов с разными HTTP-командами на разных URL-путях (маршрутах).
- Интегрирования с механизмами визуализации "view", чтобы генерировать запросы путем вставки данных в шаблоны.
- Установки общих параметров веб-приложения, такие как порт, который будет использоваться для подключения, и расположение шаблонов, которые используются для отображения запросов.

Хотя сам Express довольно минималистичен, разработчики создали совместимые пакеты промежуточного программного обеспечения для решения практически любых проблем веб-разработки. Существуют библиотеки для работы с файлами cookie, сеансами, логинами пользователей, параметрами URL, данными POST, заголовками безопасности и многим другим.

2.3 Java Script

Изначально JavaScript был создан, чтобы «сделать веб-страницы живыми». Программы на этом языке называются скриптами. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы. Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска.

Сегодня JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся «движком» JavaScript. У браузера есть собственный движок, который иногда называют «виртуальная машина JavaScript». Разные движки имеют разные «кодовые имена». Например:

- V8 – в Chrome и Opera.
- SpiderMonkey – в Firefox.
- Ещё есть «Trident» и «Chakra» для разных версий IE, «ChakraCore» для Microsoft Edge, «Nitro» и «SquirrelFish» для Safari и т.д.

Движки сложны. Но основы понять легко.

- Движок (встроенный, если это браузер) читает («парсит») текст скрипта.
- Затем он преобразует («компилирует») скрипт в машинный язык.
- После этого машинный код запускается и работает достаточно быстро.

- Движок применяет оптимизации на каждом этапе. Он даже просматривает скомпилированный скрипт во время его работы, анализируя проходящие через него данные, и применяет оптимизации к машинному коду, полагаясь на полученные знания. В результате скрипты работают очень быстро.

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. Например, в браузере JavaScript может:

- Добавлять новый HTML-код на страницу, изменять существующее содержимое, модифицировать стили.
- Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш.
- Отправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы (технологии AJAX и COMET).
- Получать и устанавливать куки, задавать вопросы посетителю, показывать сообщения.
- Запоминать данные на стороне клиента («local storage»).

Однако возможности JavaScript в браузере ограничены ради безопасности пользователя. Цель заключается в предотвращении доступа недобросовестной веб-страницы к личной информации или нанесения ущерба данным пользователя. Примеры таких ограничений включают в себя:

- JavaScript на веб-странице не может читать/записывать произвольные файлы на жёстком диске, копировать их или запускать программы. Он не имеет прямого доступа к системным функциям ОС.
- Различные окна/вкладки не знают друг о друге. Иногда одно окно, используя JavaScript, открывает другое окно. Но даже в этом случае JavaScript с одной страницы не имеет доступа к другой, если они пришли с разных сайтов (с другого домена, протокола или порта).
- JavaScript может легко взаимодействовать с сервером, с которого пришла текущая страница. Но его способность получать данные с

других сайтов/доменов ограничена. Хотя это возможно в принципе, для чего требуется явное согласие (выраженное в заголовках HTTP) с удалённой стороной. Опять же, это ограничение безопасности.

На JS планируется реализация всего приложения, включая сервер.

2.4 HTML

HTML (HyperText Markup Language — «язык гипертекстовой разметки») — самый базовый строительный блок Веба. Он определяет содержание и структуру веб-контента. Другие технологии, помимо HTML, обычно используются для описания внешнего вида/представления (CSS) или функциональности/поведения (JavaScript) веб-страницы.

Под гипертекстом ("hypertext") понимаются ссылки, которые соединяют веб-страницы друг с другом либо в пределах одного веб-сайта, либо между веб-сайтами. Ссылки являются фундаментальным аспектом Веба. Загружая контент в Интернет и связывая его со страницами, созданными другими людьми, вы становитесь активным участником Всемирной паутины.

HTML использует разметку ("markup") для отображения текста, изображений и другого контента в веб-браузере. HTML-разметка включает в себя специальные "элементы". HTML-элемент выделяется из прочего текста в документе с помощью "тегов", которые состоят из имени элемента окруженного "<" и ">". Имя элемента внутри тега не чувствительно к регистру. То есть, оно может быть написано в верхнем или нижнем регистре, или смешано. Например, тег <title> может быть записан как <Title>, <TITLE>, или любым другим способом.

2.5 CSS

CSS (Cascading Style Sheets) — язык таблиц стилей, который позволяет прикреплять стиль (например, шрифты и цвет) к структурированным документам (например, документам HTML и приложениям XML). Обычно CSS-стили используются для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML и XHTML, но также могут быть применены к любому виду XML-документа, в том числе XML, SVG и XUL. Отделяя стиль представления документов от содержимого документов, CSS упрощает создание веб-страниц и обслуживание сайтов.

CSS поддерживает таблицы стилей для конкретных носителей, поэтому авторы могут адаптировать представление своих документов к визуальным

браузерам, слуховым устройствам, принтерам, брайлевским устройствам, карманным устройствам и т.д.

Каскадные таблицы стилей описывают правила форматирования элементов с помощью свойств и допустимых значений этих свойств. Для каждого элемента можно использовать ограниченный набор свойств, остальные свойства не будут оказывать на него никакого влияния.

Объявление стиля состоит из двух частей: селектора и объявления. В HTML имена элементов нечувствительны к регистру, поэтому «h1» работает так же, как и «H1». Объявление состоит из двух частей: имя свойства (например, color) и значение свойства (grey). Селектор сообщает браузеру, какой именно элемент форматировать, а в блоке объявления (код в фигурных скобках) перечисляются форматирующие команды — свойства и их значения.

2.6 jQuery

jQuery — библиотека JavaScript, содержащая в себе готовые функции языка JavaScript, все операции jQuery выполняются из кода JavaScript.

Библиотека jQuery производит манипуляции с html-элементами, управляя их поведением и используя DOM для изменения структуры веб-страницы. При этом исходные файлы HTML и CSS не меняются, изменения вносятся лишь в отображение страницы для пользователя.

Для выбора элементов используются селекторы CSS. Выбор осуществляется с помощью функции $(). При вызове функция $() возвращает новый экземпляр объекта JQuery, который оборачивает ноль или более элементов DOM и позволяет взаимодействовать с ними различными способами.

2.7 MySQL

MySQL — это реляционная система управления базами данных с открытым исходным кодом. В настоящее время эта СУБД одна из наиболее популярных в веб-приложениях — подавляющее большинство CMS использует именно MySQL (часто только её, без альтернатив), а почти все веб-фреймворки поддерживают MySQL уже на уровне базовой конфигурации (без дополнительных модулей).

Из преимуществ СУБД MySQL стоит отметить простоту использования, гибкость, низкую стоимость владения (относительно платных СУБД), а также масштабируемость и производительность.

MySQL позволяет хранить целочисленные значения со знаком и беззнаковые, длиной в 1, 2, 3, 4 и 8 байтов, работает со строковыми и текстовыми данными фиксированной и переменной длины, позволяет осуществлять SQL-команды SELECT, DELETE, INSERT, REPLACE и UPDATE, обеспечивает полную поддержку операторов и функций в SELECT- и WHERE- частях запросов, работает с GROUP BY и ORDER BY, поддерживает групповые функции COUNT(), AVG(), STD(), SUM(), MAX() и MIN(), позволяет использовать JOIN в запросах, в т.ч. LEFT OUTER JOIN и RIGHT OUTER JOIN, поддерживает репликацию, транзакции, работу с внешними ключами и каскадные изменения на их основе, а также обеспечивает многие другие функциональные возможности.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Есть и другие типы таблиц, разработанные сообществом.

СУБД MySQL появилась в 1995. Написана на C и C++, протестирована на множестве различных компиляторов и работает на различных платформах. С 2010 года разработку и поддержку MySQL осуществляет корпорация Oracle. Продукт распространяется как под GNU GPL, так и под собственной коммерческой лицензией. Однако по условиям GPL, если какая-либо программа включает исходные коды MySQL, то и эта программа тоже должна распространяться по лицензии GPL. Для нежелающих открывать исходные тексты своих программ как раз предусмотрена коммерческая лицензия, которая, в дополнение к возможности разработки под «закрытой» лицензией, обеспечивает качественную сервисную поддержку. Сообществом разработчиков MySQL созданы различные ответвления — Drizzle, OurDelta, Percona Server и MariaDB, все эти ответвления уже существовали на момент получения прав на MySQL корпорацией Oracle.

Сейчас MySQL вместе с форком MariaDB занимают почётное первое место, а следом за ними идёт PostgreSQL. Остальные СУБД в веб-проектах используются значительно реже.

Для работы с сервером MySQL в Node.js можно использовать ряд драйверов. Один из наиболее популярных – mysql, именно его я буду использовать в данном проекте.

3. Программная реализация

Сервис будет представлять собой несколько модулей, которые будут взаимодействовать с помощью вызовов функций API, основной модуль – сервер не будет ничего знать о взаимодействии с пользователем, а только представлять собой бизнес-логику. Для взаимодействия с пользователем используется логика веб-приложения.

3.1 Домашняя страница приложения

При успешном запуске приложения неавторизованный пользователь попадает на домашнюю страницу приложения, представленную на рисунке 1:

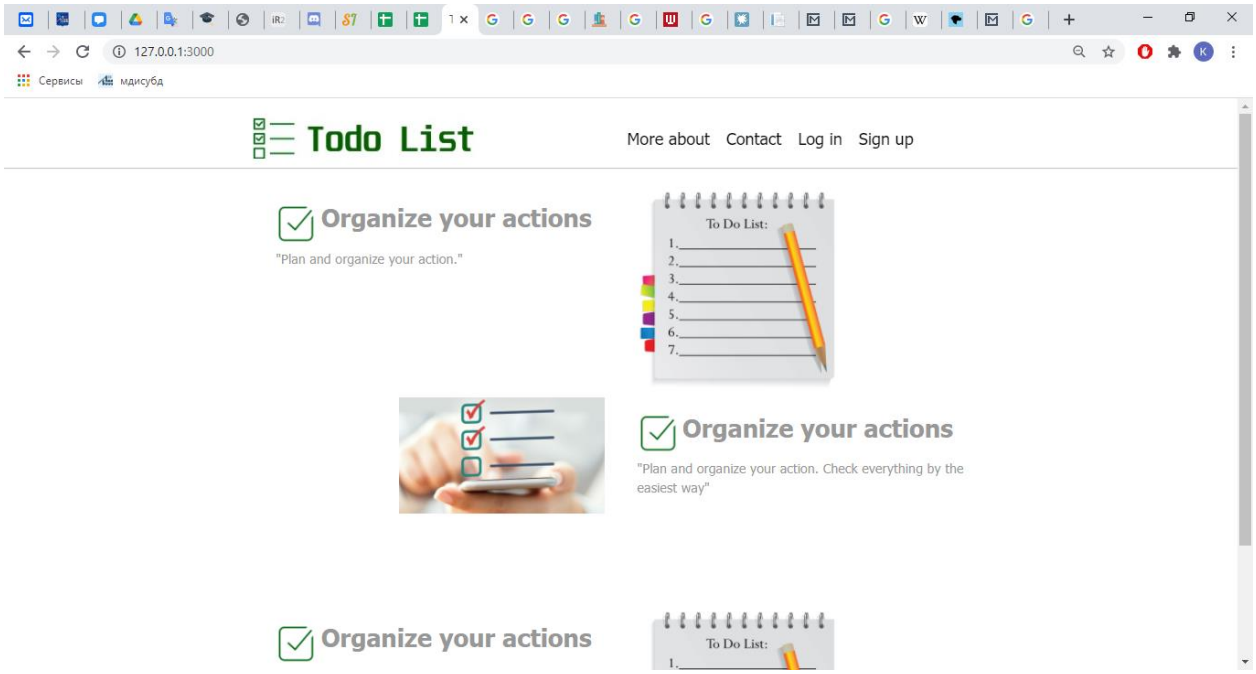


Рис. 1: Домашняя страница.

С этой страницы пользователь может перейти на страницу с подробной информацией о сервисе, просмотреть контакты разработчика, авторизоваться или зарегистрироваться.

3.2 Авторизация и регистрация

Для регистрации в приложении необходимо ввести e-mail, который будет являться логином, пароль и свое имя. Все поля являются обязательными, для успешной регистрации каждое поле должно пройти соответствующую валидацию. В противном случае, пользователю высветится сообщение об ошибке с подсказкой о корректных данных. При регистрации проверяется наличие записи в базе данных в таблице с пользователями с аналогичным e-mail. Регистрация происходит только в случае отсутствия таких записей.

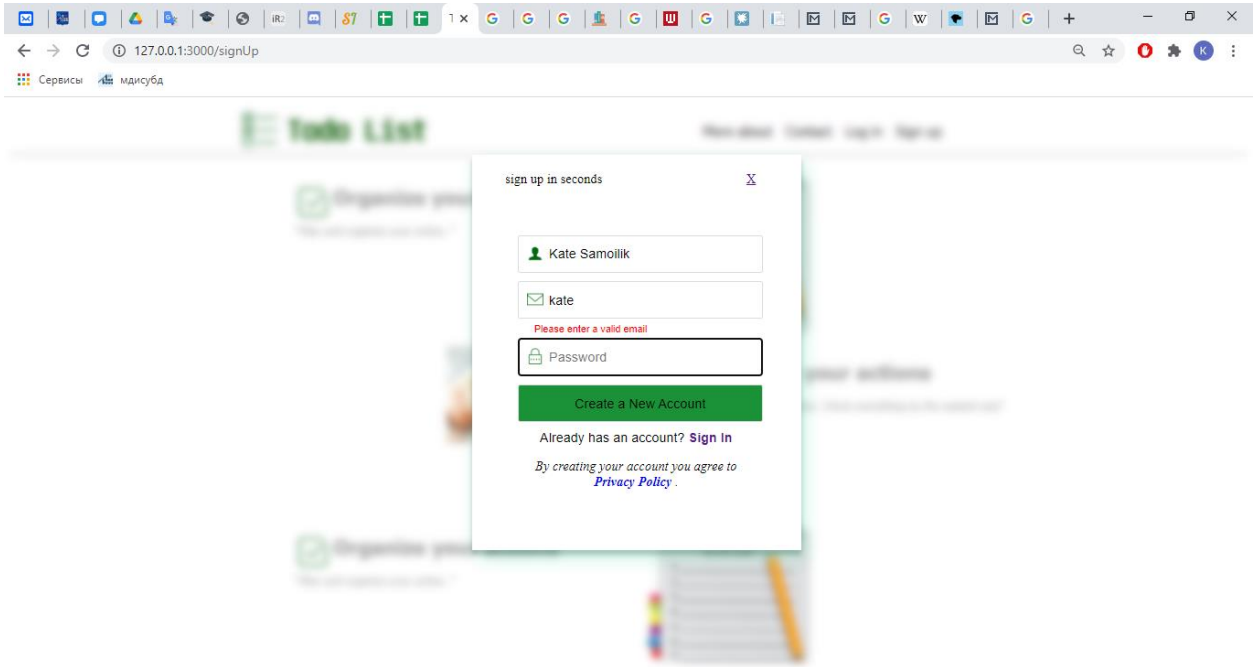


Рис. 2: Валидация поля e-mail.

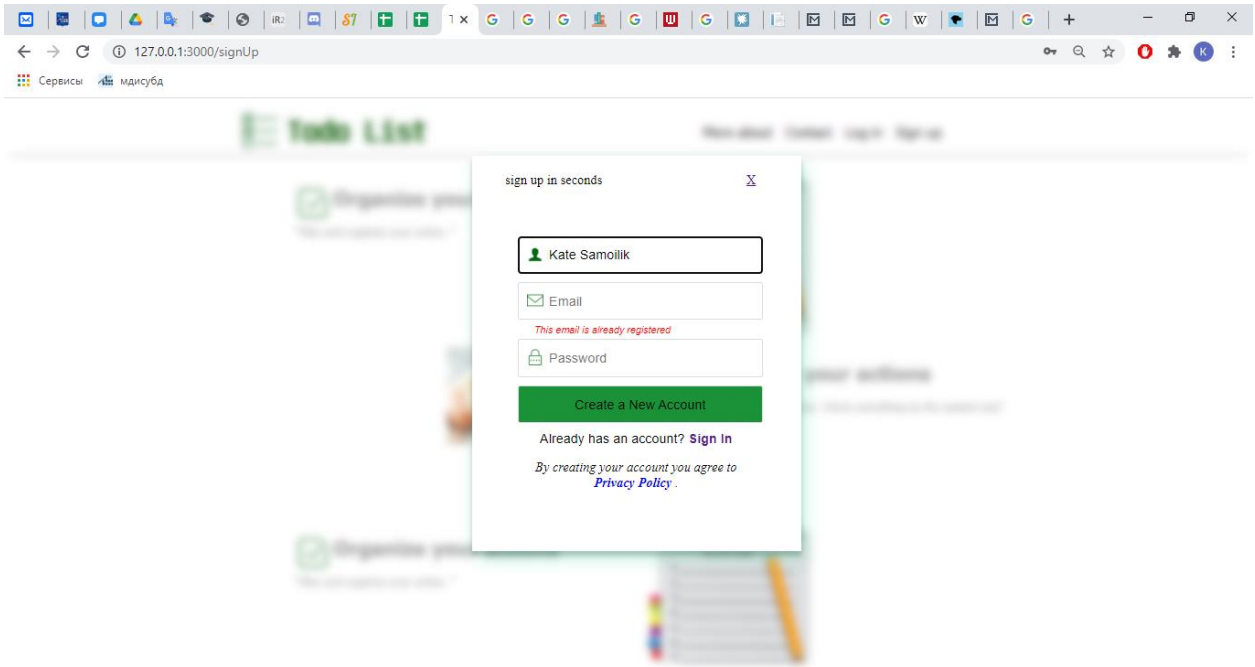


Рис. 3: Попытка регистрации с уже существующим e-mail.

В случае успешной регистрации автоматически происходит авторизация пользователя.

При авторизации пользователя проверяется наличие записи в базе в таблице пользователей, с совпадающими логином и паролем. В случае отсутствия такой записи отображается соответствующее сообщение об ошибке. В противном случае – происходит авторизация пользователя, и он переходит на свою личную страницу.

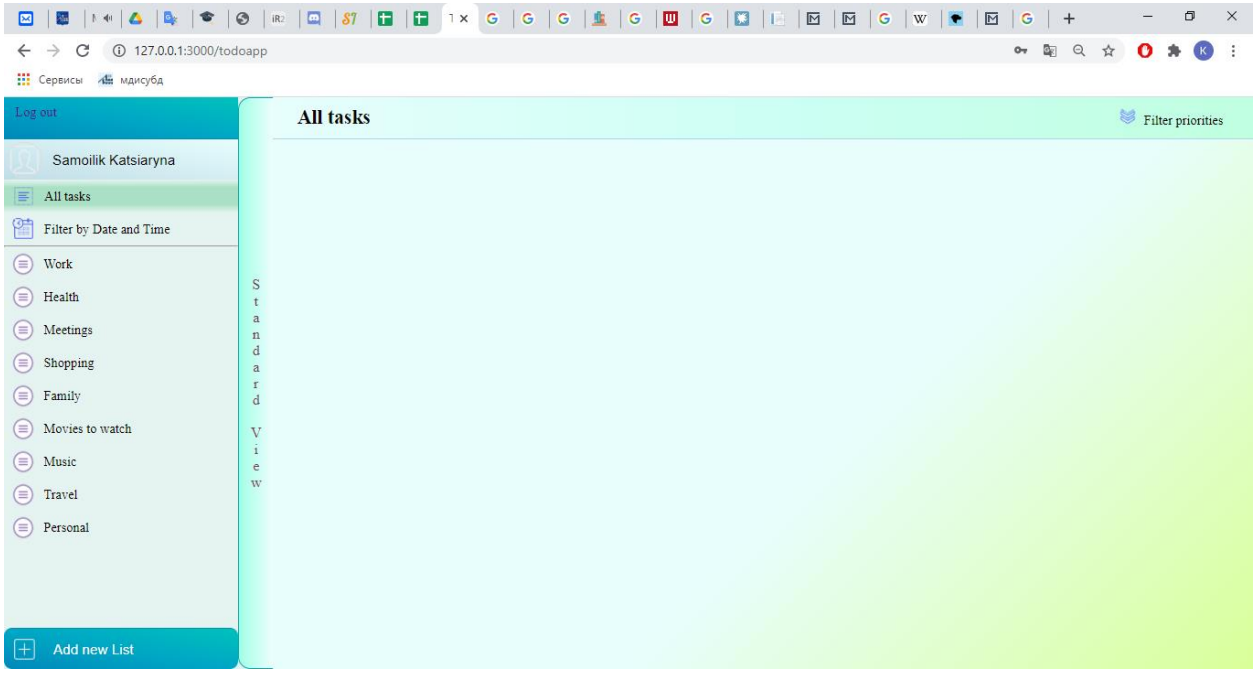


Рис. 4: Личный кабинет пользователя.

3.3 Личный кабинет пользователя

По умолчанию, после регистрации у пользователя создается несколько списков задач. Пользователь может отредактировать существующий список, удалить его или создать свой.

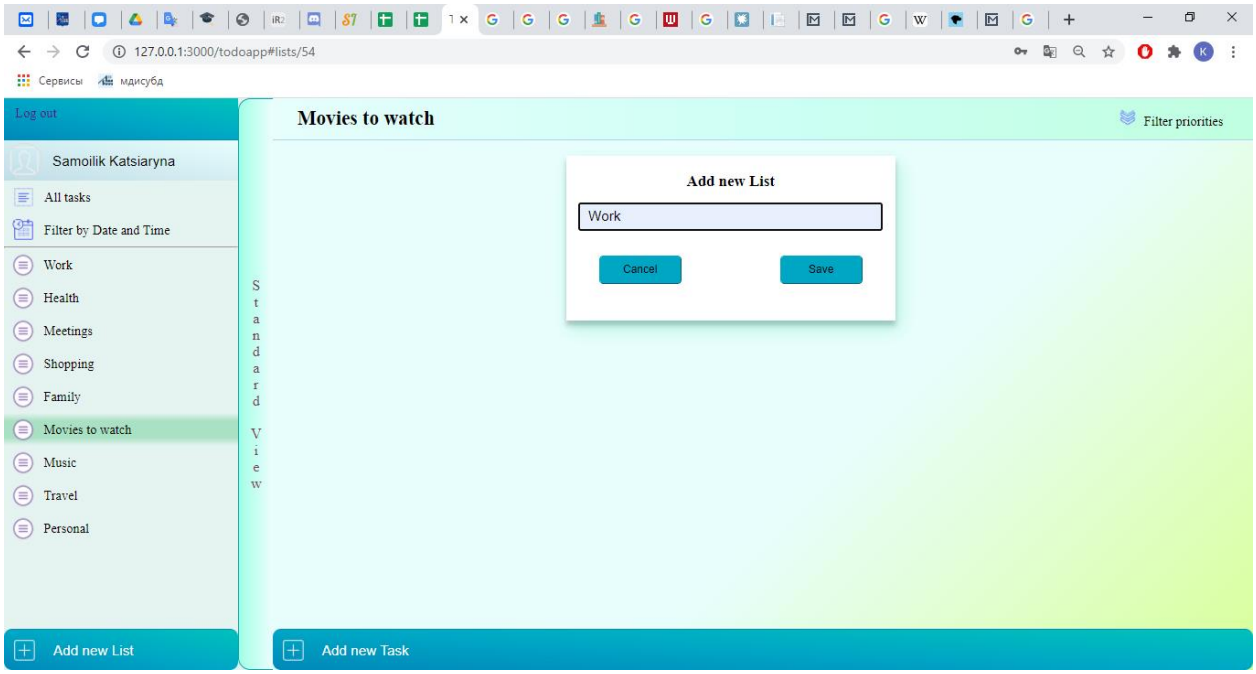


Рис. 5: Добавление списка задач.

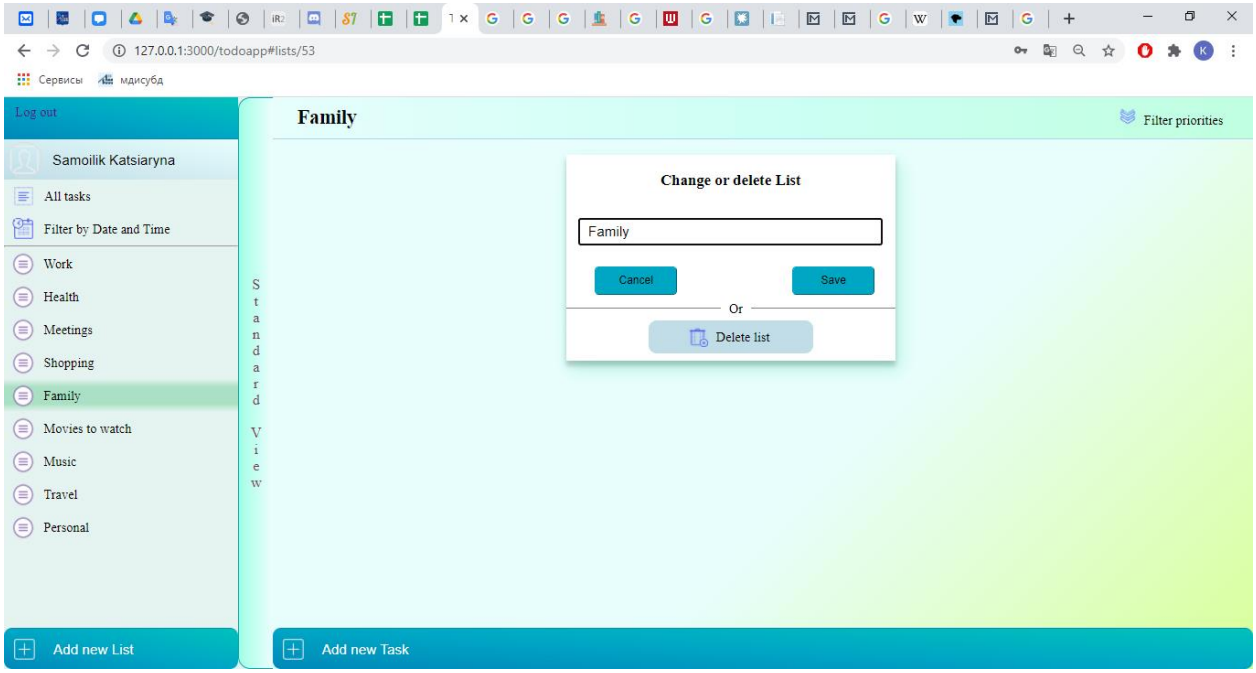


Рис. 6: Редактирование/удаление списка задач.

Выбрав список задач, пользователь может создать задачу в нем. При создании задачи пользователь должен указать ее название. Он так же может выбрать дату для данной задачи (события), указать должна ли данная задача повторяться и выставить для нее приоритет.

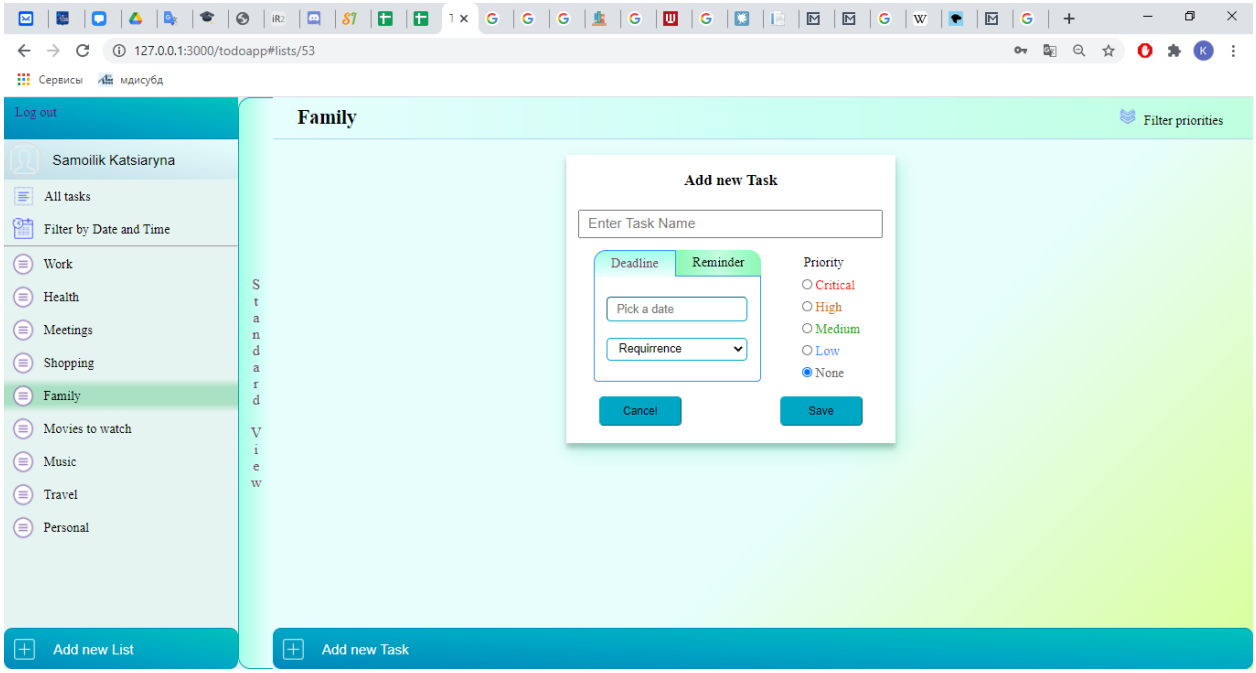


Рис.7: Добавление задачи.

Пользователь имеет возможность посмотреть задачи, принадлежащие каждому списку или все сразу. В каждом списке задач, в зависимости от приоритета каждая задача подсвечивается соответствующим цветом.

Существует возможность отметить выполненные задачи. Просроченные задачи подсвечиваются красным цветом.

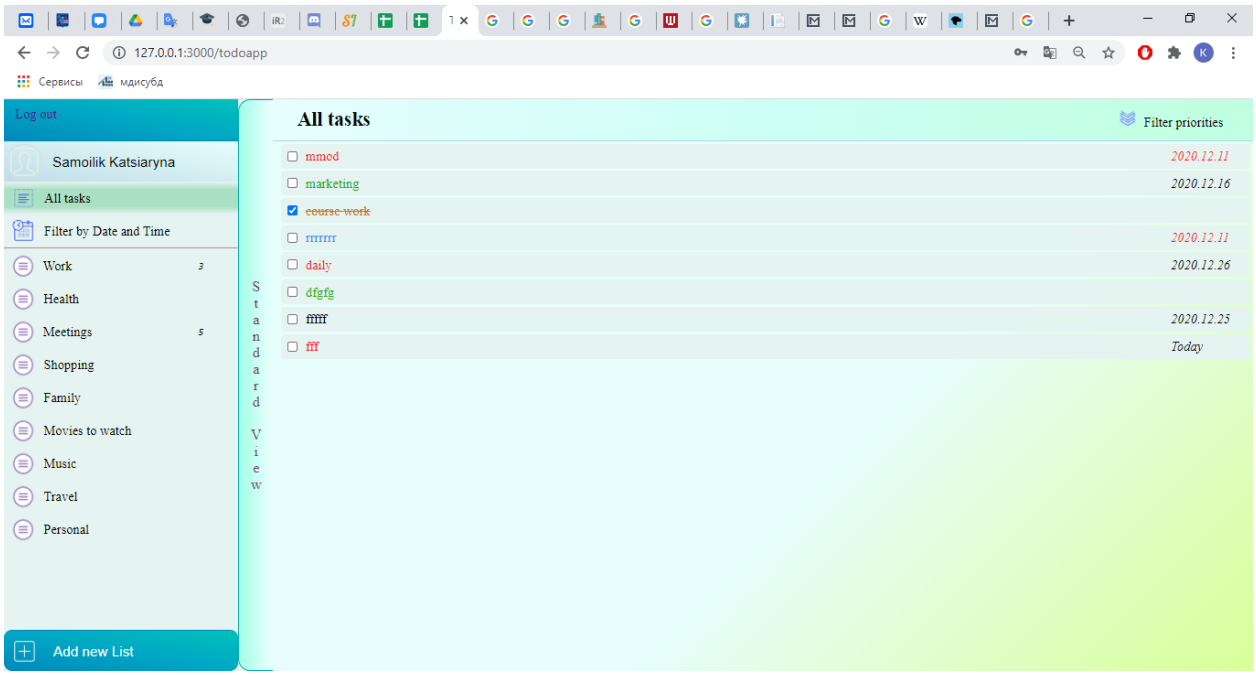


Рис. 8: Список задач пользователя.

Также у пользователя существует отфильтровать любой список задач по приоритету, либо по дате выполнения.

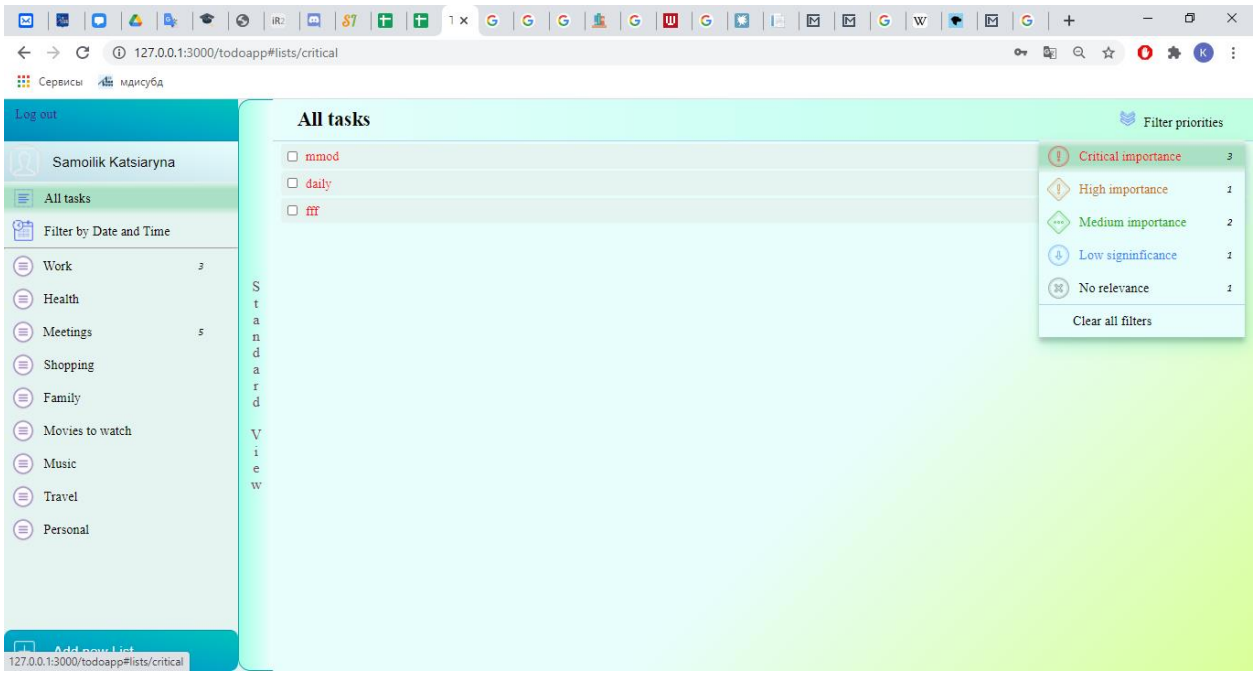


Рис. 9: Выставлен фильтр – критические задачи.

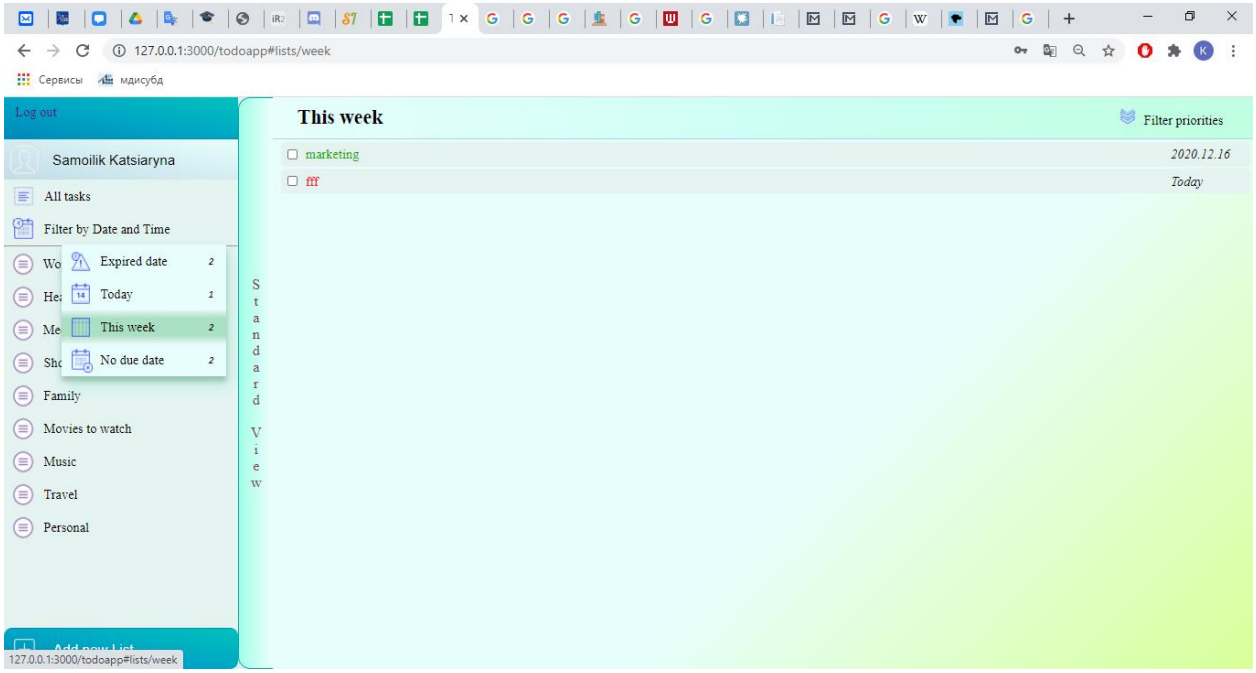


Рис. 10: Выставлен фильтр – задачи на данной неделе.

3.4 База данных приложения

На рисунке ниже представлена схема базы данных данного приложения – todolist. В базе содержатся три таблицы: таблица с пользователями, таблица со списками задач и таблица с самими задачами.

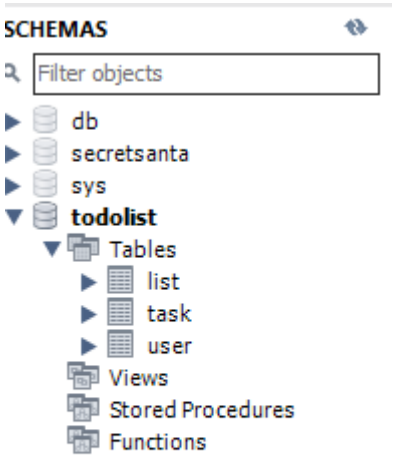
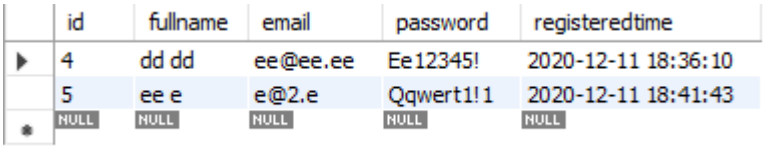


Рис. 11: Схема базы данных.

Таблица пользователей user содержит следующие поля:

- id – автогенерируемый уникальный идентификатор
- fullname – заполняется пользователем при регистрации

- email – заполняется пользователем при регистрации
- password – заполняется пользователем при регистрации
- registeredtime – время регистрации пользователя



id	fullname	email	password	registeredtime
4	dd dd	ee@ee.ee	Ee12345!	2020-12-11 18:36:10
5	ee e	e@2.e	Qqwert1!1	2020-12-11 18:41:43
NULL	NULL	NULL	NULL	NULL

Рис. 12: Таблица user.

Таблица списков задач list содержит следующие поля:
- id – автогенерируемый уникальный идентификатор
- userId – PK на пользователя-владельца
- listname – название списка задач

id	userId	listname
11	2	Work
12	2	Health
13	2	Meetings
14	2	Shopping
15	2	Family
16	2	Movies to watch
17	2	Music
18	2	Travel
19	2	Personal
20	3	Work
21	3	Health
22	3	Meetings

Рис. 13: Таблица list.

Таблица задач task содержит следующие поля:
- id – автогенерируемый уникальный идентификатор
- taskname – заполняется пользователем при создании
- listId – PK на родительский список задач
- userId – PK на пользователя-владельца
- reminderdate – время напоминания
- deadline – время окончания задачи
- requirrence – повторяемость
- priority – приоритет
- finished – завершенность

	id	fullname	email	password	registeredtime
▶	4	dd dd	ee@ee.ee	Ee12345!	2020-12-11 18:36:10
	5	ee e	e@2.e	Qqwert1!1	2020-12-11 18:41:43
*	NULL	NULL	NULL	NULL	NULL

Рис. 14: Таблица task.

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта был реализован Day Tracker задач, в процессе было рассмотрено большое количество технологий и методов разработки, особенно подробно были рассмотрены Node.js и Java Script, в результате был запущен сервис.

Я считаю, что у приложения есть большой потенциал для развития, например, можно добавить большое количество уведомлений, как более простых, например telegram бот, так и более сложные такие как смс-сообщения или электронная почта. Больше особенностей добавления задач, например приложение файлов или картинок. Также сделать отображение в виде календаря с задачами разнесенными по дням.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бибо Бер , Кац Иегуда jQuery. Подробное руководство по продвинутому JavaScript; Символ-плюс - М., 2017. - 624 с..

2. Изучаем Node.js; Питер - М., 2015. - 400 с.

3. Макфарланд Дэвид JavaScript и jQuery. Исчерпывающее руководство (+ DVD-ROM); Эксмо - М., 2015. - 688 с.

4. Фримен Адам jQuery для профессионалов; Вильямс - М., 2015. - 960 с.

ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД

Ниже представлены основные модули.

dbHandler.js

```
//var User = require('../models/user');
//var List = require('../models/list');
let Task = require('../models/task');
let constants = require('../database/constants')

let mysql = require('mysql'),
  async = require('async');

const con = mysql.createPool({
  connectionLimit: 10,
  host: "localhost",
  user: "root",
  password: "student",
  database: "todolist"
});

function addUser(user) {
  return new Promise(function(resolve, reject) {
    con.getConnection(function(err, tempCont) {
      if (err) {
        tempCont.release();
        throw err;
      } else {
        let sql = "INSERT INTO " + constants.user.USERS_TABLE +
           " (" + constants.user.USERS_NAME + ", " +
           constants.user.USERS_EMAIL + ", " +
           constants.user.USERS_PASSWORD + ", " +
           constants.user.USERS_REGISTER + ") VALUES ('" +
           user.getName + "', '" + user.getEmail + "', '" +
           user.getPassword + "', " + "now()" + ")";

        tempCont.query(sql, function(err, res) {
          if (err) {
            return reject(err);
          } else {
            resolve(res);
          }
        });
      }
      tempCont.release();
    });
  });
};
```

```
 function findUser(user) {
  return new Promise(function(resolve, reject) {
    con.getConnection(function(err, tempCont) {
      if (err) {
        tempCont.release();
        throw err;
      } else {
        let sql = "SELECT * FROM " +
          constants.user.USERS_TABLE +
          " WHERE " + constants.user.USERS_EMAIL + " =? ";

        tempCont.query(sql, [user.getEmail], function(err, res) {
          if (err) {
            return reject(err);
          } else {
            resolve(res);
          }
        });
      }
      tempCont.release();
    });
  });
};


function addList(list) {
  return new Promise(function(resolve, reject) {
    con.getConnection(function(err, tempCont) {
      if (err) {
        tempCont.release();
        throw err;
      } else {
        let sql = "INSERT INTO " +
          constants.list.LIST_TABLE + " (" +
          constants.list.LIST_USERID + ", " +
          constants.list.LIST_NAME + ") VALUES ?"

        tempCont.query(sql, [list], function(err, res) {
          if (err) {
            return reject(err);
          } else {
            resolve(res);

          }
        });
      }
      tempCont.release();
    });
```

```
  });
};

function getList(userID) {
  return new Promise(function(resolve, reject) {
    con.getConnection(function(err, tempCont) {
      if (err) {
        tempCont.release();
        throw err;
      } else {
        let sql = "SELECT * FROM " +
          constants.list.LIST_TABLE + " WHERE " +
          constants.list.LIST_USERID + " =? ";

        tempCont.query(sql, [userID], function(err, res) {
          if (err) {
            return reject(err);
          } else {
            resolve(res);
          }
        });
      }
      tempCont.release();
    });
  });
};

function updateList(updateObj) {
  return new Promise(function(resolve, reject) {
    con.getConnection(function(err, tempCont) {
      if (err) {
        tempCont.release();
        throw err;
      } else {
        let sql = `UPDATE ${constants.list.LIST_TABLE} SET ${constants.list.LIST_
NAME} = "${updateObj.listName}" WHERE ${constants.list.LIST_ID} = ${updateObj.lis
tId}`;
        tempCont.query(sql,function(err, res) {
          if (err) {
            return reject(err);
          } else {
            resolve(res);
          }
        });
      }
      tempCont.release();
    });
  });
};
```

```
function deleteList(listID) {
  return new Promise(function(resolve, reject) {
    con.getConnection(function(err, tempCont) {
      if (err) {
        tempCont.release();
        throw err;
      } else {
        let sql = `DELETE FROM ${constants.list.LIST_TABLE} WHERE ${constants.lis
t.LIST_ID} = ${listID}`;
        tempCont.query(sql,function(err, res) {
          if (err) {
            return reject(err);
          } else {
            resolve(res);
          }
        });
      }
      tempCont.release();
    });
  });
};

 function getTask(userID) {
  return new Promise(function(resolve, reject) {
    con.getConnection(function(err, tempCont) {
      if (err) {
        tempCont.release();
        throw err;
      } else {
        let sql = "SELECT * FROM " +
          constants.task.TASK_TABLE + " WHERE " +
          constants.task.TASK_USERID + " =? ";
        tempCont.query(sql, [userID], function(err, res) {
          if (err) {
            return reject(err);
          } else {
            resolve(res);
          }
        });
      }
      tempCont.release();
    });
  });
};

 function addTask(task) {
  return new Promise(function(resolve, reject) {
    con.getConnection(function(err, tempCont) {
```

```
      if (err) {
        tempCont.release();
        throw err;
      } else {
        let sql = "INSERT INTO " +
          constants.task.TASK_TABLE + " (" +
          constants.task.TASK_NAME + ", " +
          constants.task.TASK_LISTID + ", " +
          constants.task.TASK_USERID + ", " +
          constants.task.TASK_REMINDER + ", " +
          constants.task.TASK_DEADLINE + ", " +
          constants.task.TASK_REQUIRRENCE + ", " +
          constants.task.TASK_PRIORITY + ", " +
          constants.task.TASK_FINISHED + ") VALUES ?"
        tempCont.query(sql,[task], function(err, res) {
          if (err) {
            return reject(err);
          } else {
            resolve(res);
          }
        });
      }
      tempCont.release();
    });
  });
};

module.exports={
  addUser,
  findUser,
  addList,
  getList,
  updateList,
  deleteList,
  addTask,
  getTask
}
```

app.js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var expressValidator = require('express-validator');
var expressSession = require('express-session');


var indexRouter = require('./routes/index');
```

```
var usersRouter = require('./routes/users');
var listsRouter = require('./routes/lists');
var tasksRouter = require('./routes/tasks');

var app = express();

//паарметры шаблонов для express
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(expressValidator());
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
app.use(expressSession({secret: 'max', saveUninitialized: false, resave: false}))
;


app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/lists', listsRouter);
app.use('/tasks', tasksRouter);

app.use(function(req, res, next) {
  next(createError(404));
});


app.use(function(err, req, res, next) {
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};
  res.locals.error = err;

  res.status(err.status || 500);
  console.log(err);
  res.render('error',{
    message:err.message,
    error:err
  });
});

module.exports = app;
```

index.js

```
let express = require('express');
let router = express.Router();
let User= require('../models/user');
```

```
let Task = require('../models/task');
let validateSignUp = require('../actions/validateSignUp');
let validateLogin = require('../actions/validateLogin');
let newUser = require('../actions/newUser');
let loginUser = require('../actions/loginUser');
let addTask = require('../actions/addTask');
let changeList = require('../actions/changeList');


router.get('/', function(req, res, next) {
  res.render('index');
});

router.get('/about', function(req, res, next) {
  res.render('about');
});

router.get('/contact', function(req, res, next) {
  res.render('contact');
});


router.get('/todoapp', function(req, res, next) {
  res.render('todoapp',{user: req.session.user, list:req.session.list, task:req.s
ession.task});
});

router.get('/signUp', function(req, res, next) {
  const user = {name: "", email: "", psw: ""};
  const errResp = {name: "", email: "", psw: ""};

  if(!req.session.user){
    req.session.user=user;
    req.session.resp=errResp;
  }
  res.render('signUp', {user: req.session.user, resp: req.session.resp });
  req.session.destroy();
});


router.get('/login', function(req, res, next) {
  const errResp = {email:"", pwd:""};
  if(!req.session.resp){
    req.session.resp=errResp;
  }
  res.render('login',  {resp:req.session.resp });
});


 module.exports = router;
```

task.js

```
'use strict'

class Task {
  constructor(id, taskName, listID, userID, planedTime, deadLineTime, requirrence
, priority, finished) {
    this.id = id;
    this.taskname = taskName;
    this.listId = listID;
    this.usersId = userID;
    this.reminderDate = planedTime;
    this.deadline = deadLineTime;
    this.requirrence = requirrence;
    this.priority = priority;
    this.finished = finished;

    this.setPlanedTime = function(newTime) {
      this.planedTime = newTime;
    };

    this.setDeadLineTime = function(newTime) {
      this.deadLineTime = newTime;
    };

    this.setRequrrence = function(req) {
      this.requrrence = req;
    };

    this.setPriority = function(newPriority) {
      this.priority = newPriority;
    };

    this.setFinished = function(finish) {
      this.finished = finished;
    };

  }

  get getID() {
    return this.id;
  }

  get getTask() {
    return this.task;
  }

  get getListID() {
    return this.listID;
```

```
  }

  get getUserID() {
    return this.userID;
  }

  get getPlanedTime() {
    return this.planedTime;
  }

  get getDeadLineTime() {
    return this.deadLineTime;
  }

  get getRequrrence() {
    return this.requrrence;
  }

  get getPriority() {
    return this.priority;
  }

  get getFinished(){
    return this.finished;
  }

}

module.exports = Task;
```

todoapp.js

```
$(document).ready(function() {

  function getListID(string) {
    const pos = string.lastIndexOf("/");
    return parseInt(string.substring(pos + 1));
  }

  function getYesterday() {
    return (new Date().setHours(0, 0, 0, 0) - (86400 * 1000));
  }

  function getTomorrow() {
    return (new Date().setHours(0, 0, 0, 0) + (86400 * 1000));
  }

  function getWeekAhead() {
    return (new Date().setHours(0, 0, 0, 0) + (86400 * 1000 * 7));
  }
```

```
function getMonthAhead() {
  const today = new Date();
  const nextMonth = new Date(today.getFullYear() + "-" +
    (today.getMonth() + 2) + "-" +
    today.getDate());
  return nextMonth.setHours(0, 0, 0, 0);
}

function applyFilter(filterName) {
  switch (filterName.trim()) {
    case "All tasks":
      {
        filterAllTask();
        break;
      }
    case "Today":
      {
        filterDate(0);
        break;
      }
    case "Tomorrow":
      {
        filterDate(1);
        break;
      }
    case "This week":
      {
        filterBeforeDate("week");
        break;
      }
    case "This month":
      {
        filterBeforeDate("month");
        break;
      }
    case "Critical importance":
      {
        filterPriority(4);
        break;
      }
    case "High importance":
      {
        filterPriority(3);
        break;
      }
    case "Medium importance":
      {
        filterPriority(2);
```

```
        break;
      }
    case "Low signinficance":
      {
        filterPriority(1);
        break;
      }
    case "No relevance":
      {
        filterPriority(0);
        break;
      }
    case "Clear all filters":
      {
        clearPriorityFilter();
        break;
      }
    case "Yesterday":
      {
        filterDate(-1);
        break;
      }
    case "Expired date":
      {
        filterBeforeDate("expired");
        break;
      }
    case "No due date":
      {
        filterBeforeDate("nodate");
        break;
      }
  }
}

function deleteList(listID) {
  let updateObj = {
    expired: 0,
    yesterday: 0,
    today: 0,
    tomorrow: 0,
    week: 0,
    month: 0,
    dateless: 0,
    crit: 0,
    high: 0,
    med: 0,
    low: 0,
    none: 0
```

```
  }
  $("#myList-ul").children().each(function() {
    if (getListID($(this).children().attr("href")) == listID) {
      $(this).remove();
      return true;
    }
  });
  $("#myTask-ul").children().each(function() {
    const taskListID=$(this).children().find(".taskListId").val();
    if (taskListID == listID) {
      const priority = $(this).find(".priority").val();
      const date = $(this).find(".task-datetime-holder").val();
      getTaskFilterObj(updateObj, date, priority);
      $(this).remove();
    }
  });
  for(var key in updateObj){
    console.log(key+":"+updateObj[key]);
    if(updateObj[key]>0){
      updateFilterCount(key, updateObj[key]);
    }
  }

  console.log(updateObj);
}


function filterAllTask() {
  $("#myTask-ul").children().each(function() {
    $(this).show();
  })
}

function filterDate(extraday) {
  $("#myTask-ul").children().each(function() {
    const date = $(this).find(".task-datetime-holder").val();
    $(this).hide();
    if (date != "0") {
      const today = new Date().setHours(0, 0, 0, 0) / 1000;
      const filteredDay = today + (extraday * 86400);
      (date == filteredDay) ? $(this).show(): $(this).hide();
    }
  });
}

function filterBeforeDate(filter) {
  $("#myTask-ul").children().each(function() {
    const date = $(this).find(".task-datetime-holder").val();
    const today = new Date().setHours(0, 0, 0, 0) / 1000;
    let nextDate;
```

```
      if (date != "0") {
        if (date >= today) {
          if (filter === "week") {
            nextDate = getWeekAhead() / 1000;
          } else if (filter === "month") {
            nextDate = getMonthAhead() / 1000;
          }
          (date <= nextDate) ? $(this).show(): $(this).hide();
        } else {
          (filter === "expired") ? $(this).show(): $(this).hide();
        }
      } else {
        (filter === "nodate") ? $(this).show(): $(this).hide();
      }
    });
  }

  function filterPriority(priorityNumber) {
    $("#myTask-ul").children().each(function() {
      const checkPriority = $(this).find(".priority").val();
      (priorityNumber == checkPriority) ? $(this).removeClass("filterDisplay-
none"): $(this).addClass("filterDisplay-none");
    });
  }

  function clearPriorityFilter() {
    $("#myTask-ul").children().each(function() {
      $(this).removeClass("filterDisplay-none");
    });
  }

  function getListsTaskCount() {
    let listObjCount = {};
    $("#myTask-ul").children().each(function() {
      const id = $(this).find(".taskListId").val();
      (listObjCount.hasOwnProperty(id)) ? (listObjCount[id]++) : (listObjCount[id
] = 1);
    });

    $("#myList-ul").children().each(function() {
      Object.keys(listObjCount).some((key) => {
        if (getListID($(this).children().attr("href")) == key) {
          $(this).find(".list-taskcount").text(listObjCount[key]);
          delete listObjCount[key];
          return true;
        }
      });
    });
  }
```

```
  function getTaskFilterObj(obj, date, priority) {
    switch (priority) {
      case "4": {  obj.crit++;  break;}
      case "3": {  obj.high++;  break;}
      case "2": {  obj.med++;   break;}
      case "1": {  obj.low++;   break;}
      case "0": {  obj.none++;  break;  }
      default : {  console.log("Problem with creating priority of checkFiltersSta
rt");
        }
    }

    if (date > 0) {
      const yesterday = getYesterday() / 1000;
      const today = new Date().setHours(0, 0, 0, 0) / 1000;
      const tomorrow = getTomorrow() / 1000;
      const thisWeek = getWeekAhead() / 1000;
      const thisMonth = getMonthAhead() / 1000;

      if (date >= today) {
        if (date == today) {
          obj.today++;
        } else if (date == tomorrow) {
          obj.tomorrow++;
        }
        if (date < thisWeek) {
          obj.week++;
        }
        if (date < thisMonth) {
          obj.month++;
        }
      } else {
        obj.expired++;
        if (date == yesterday) {
          obj.yesterday++;
        }
      }
    } else {
      obj.dateless++;
    }
  }


  function checkFiltersStart() {
    let filterObj = {
      expired: 0,
      yesterday: 0,
      today: 0,
      tomorrow: 0,
```

```
    week: 0,
    month: 0,
    dateless: 0,
    crit: 0,
    high: 0,
    med: 0,
    low: 0,
    none: 0
  }

  $("#myTask-ul").children().each(function() {
    const taskCheckBox = $(this).find(".taskCheckBox").is(":checked");
    const priority = $(this).find(".priority").val();
    const date = $(this).find(".task-datetime-holder").val();
    const today = new Date().setHours(0, 0, 0, 0) / 1000;
    getTaskFilterObj(filterObj, date, priority);
    const returnDate= showDate(date);

    $(this).find(".task-datetime").text(returnDate);

    if( date<today){
      $(this).find(".task-datetime").addClass("pastDateTimeTask");
    }
    if (taskCheckBox) {
      $(this).find(".task-name").addClass("taskFinished");
    }
  });
  return filterObj;
}

function listFocusRemove() {
  $("#date-filters-ul").find("a").removeClass("focusList");
  $("#myList-ul").find("a").removeClass("focusList");
  $("#list-filter-ul").find("a").removeClass("focusList");
  $("#allTaskFilter").removeClass("focusList");
}

function showDate(date) {
      const today = new Date().setHours(0, 0, 0, 0) / 1000;
  const tomorrow = getTomorrow() / 1000;
  const yesterday = getYesterday() / 1000;

  if (date > 0) {
    if (date == today) {
      dateFormat = "Today";
    } else if (date == tomorrow) {
      dateFormat = "Tomorrow";
    } else if (date == yesterday) {
      dateFormat = "Yesterday";
```

```
    } else {
      dateFormat = timeFormat(date)
    };
  } else {
    dateFormat = "";
  }
  return dateFormat;
}


function timeFormat(time) {
  const datetime = new Date(time * 1000);
  let date = datetime.getDate();
  let month = datetime.getMonth() + 1;
  if (date < 10) {
    date = "0" + date;
  }
  if (month < 10) {
    month = "0" + month;
  }
  return ("" + datetime.getFullYear() + "." + month + "." + date);
}


function showOnlyActiveFilters(obj) {
  Object.keys(obj).forEach((key) => {
    if (obj[key] > 0) {
      const filterID = key + `TaskFilter`;
      $("#" + filterID).parent().show();
      $("#" + filterID).children(".list-taskcount").text(obj[key]);
    }
  })
}

function incrementCount(countString) {
  let count;
  if (countString !== "") {
    count = parseInt(countString) + 1;
  } else {
    count = 1;
  }
  return count;
}

function updateFilterCount(key, value) {
  const filterID = key + `TaskFilter`;
  //$("#" + filterID).parent().show();
  const taskcount = parseInt($("#" + filterID).find(".list-taskcount").text());
  const updatedCount = taskcount-value;
  if (updatedCount>0){
```

```
    $("#" + filterID).find(".list-taskcount").text(updatedCount);
  }else if(updatedCount==0){
    $("#" + filterID).find(".list-taskcount").text("");
    $("#" + filterID).parent().hide();
  }else{
    console.log("Problem with updating counts after deleteList")
  }
}

function incrFilterCount(key) {
  const filterID = key + `TaskFilter`;
  $("#" + filterID).parent().show();
  const taskcount = $("#" + filterID).find(".list-taskcount").text();
  const updatedCount = incrementCount(taskcount)
  $("#" + filterID).find(".list-taskcount").text(updatedCount);
}

function updateListbyID(listID, listname) {
  $("#myList-ul").children().each(function() {
    if (getListID($(this).children().attr("href")) == listID) {
      $(this).children().find(".list-name").text(listname);
      return true;
    }
  });
}

function incrListCount(listID) {
  $("#myList-ul").children().each(function() {
    if (getListID($(this).children().attr("href")) == listID) {
      const taskcount = $(this).find(".list-taskcount").text();
      const updatedCount = incrementCount(taskcount)
      $(this).find(".list-taskcount").text(updatedCount);
      return true;
    }
  });
}

function updateFilterListCountNewTask(obj) {
  //let showDateFormat=showDate(obj.deadline,obj.reminderDate);
  let date;
  (obj.deadline > 0) ? (date = obj.deadline) : (date = obj.reminderDate);
  if (date > 0) {
    const today = new Date().setHours(0, 0, 0, 0) / 1000;
    const tomorrow = getTomorrow() / 1000;
    const thisWeek = getWeekAhead() / 1000;
    const thisMonth = getMonthAhead() / 1000;
    if (date == today) {
      incrFilterCount("today");
    } else if (date == tomorrow) {
```

```
      incrFilterCount("tomorrow");
    }
    if (date > today) {
      if (date < thisWeek) {
        incrFilterCount("week");
      }
      if (date < thisMonth) {
        incrFilterCount("month");
      }
    }
  }

  const priority = obj.priority;
  switch (priority) {
    case 4:
      {
        incrFilterCount("crit");
        break;
      }
    case 3:
      {
        incrFilterCount("high");
        break;
      }
    case 2:
      {
        incrFilterCount("med");
        break;
      }
    case 1:
      {
        incrFilterCount("low");
        break;
      }
    case 0:
      {
        incrFilterCount("none");
        break;
      }
    default:
      {
        console.log("Problem with creating priority of checkFiltersStart");
      }
  }
  const listId = obj.listId;
  incrListCount(listId);
}

function startListName() {
```

```
  $("#allTaskFilter").addClass("focusList");
  $("#selectedList-name").text($("#allTaskFilter").text());
  $("#newTask-toolbar").hide();
  filterAllTask();
  let filterTaskObj = checkFiltersStart();
  showOnlyActiveFilters(filterTaskObj);
  getListsTaskCount();
}


startListName();

$("#myTask-ul").on("click", "input", function() {
  if ($(this).is(":checkbox")) {
    if ($(this).is(":checked")) {
      $(this).siblings(".task-name").addClass("taskFinished");
    } else {
      $(this).siblings(".task-name").removeClass("taskFinished");
    }
  }
});


$("#date-filters-ul").on("click", "a", function(event) {
  let focused = document.activeElement;
  if (document.querySelector) {
    focused = document.querySelector(":focus");
  }
  listFocusRemove();
  $(focused).addClass("focusList");
  const filterName = $(focused).find(".list-name").text();
  $("#selectedList-name").text(filterName);
  $("#newTask-toolbar").hide();
  applyFilter(filterName);
});

$("#allTaskFilter").on("click", function(event) {
  listFocusRemove();
  $(this).addClass("focusList");
  $("#selectedList-name").text("All tasks");
  $("#newTask-toolbar").hide();
  applyFilter("All tasks");
})

$("#list-collect").on("click", "a", function(event) {
  let focused = document.activeElement;
  if (document.querySelector) {
    focused = document.querySelector(":focus");
  }
  $("#selectedList-name").text($(focused).text());
```

```
  listFocusRemove();
  $(focused).addClass("focusList");
  $("#newTask-toolbar").show();
  $("#changeList-popup-container").hide();
  const id = getListID($(focused).attr("href"));
  $("#taskListIdForm").val(id);
  $("#myTask-ul").children().each(function() {
    const taskListId = $(this).find(".taskListId").val();
    (taskListId == id) ? $(this).show(): $(this).hide();
  })
  if (event.target.className === "list-options") {
    $("#changeList-popup-container").toggle();
    $("#confirmDeleteList-container").hide();
    $("#changelistId").val(id);
  }
});

$("#filter-priority-ul").on("click", "a", function(event) {
  let focused = document.activeElement;
  if (document.querySelector) {
    focused = document.querySelector(":focus");
  }
  const filterName = $(focused).find(".list-name").text().trim();
  applyFilter(filterName);
});



$("#newList-toolbar").click(function() {
  $("#newTask-popup-container").hide();
  $("#newList-popup-container").toggle();
});

$("#cancel-newlist").click(function() {
  $("#newListName").val('');
  $("#newList-popup-container").hide();
});

$("#cancel-changeList").click(function() {
  $("#editListName").val('');
  $("#changeList-popup-container").hide();
});



$("#newTask-toolbar").click(function() {
  $("#newList-popup-container").hide();
  $("#newTask-popup-container").toggle();
});
```

```
$("#cancel-newTask").click(function() {
  $("#newTask-popup-container").hide();
});


$("#standardViewTab").click(function() {
  $("#calendarView").hide();
  $("#calendarViewTab").removeClass("activeTaskTab");
  $("#standardViewTab").addClass("activeTaskTab");
  $("#standardView").show();
});


$("#calendarViewTab").click(function() {
  $("#standardView").hide();
  $("#standardViewTab").removeClass("activeTaskTab");
  $("#calendarViewTab").addClass("activeTaskTab");
  $("#calendarView").show();
  $("#newTask-popup-container").hide();
});

$("#deadlineViewTab").click(function() {
  $("#task-reminder").hide();
  $("#task-reminder").attr("disabled", "disabled");
  $("#reminderViewTab").removeClass("activeTaskTimeTab");
  $("#deadlineViewTab").addClass("activeTaskTimeTab");
  $("#task-deadline").show();
  $("#task-deadline").removeAttr("disabled");
});

$("#reminderViewTab").click(function() {
  $("#task-reminder").removeAttr("disabled");
  $("#task-reminder").show();
  $("#reminderViewTab").addClass("activeTaskTimeTab");
  $("#deadlineViewTab").removeClass("activeTaskTimeTab");
  $("#task-deadline").hide();
  $("#task-deadline").attr("disabled", "disabled");
});


$("#updateList-form").submit(function(e) {
  e.preventDefault();
  const updatedListName = $('#editListName').val().trim();
  if (updatedListName !== "") {
    const updatedList = $(this).serialize();
    const listID = $('#changelistId').val();
    $.ajax({
      type: 'PUT',
      url: "/lists/update",
      data: updatedList
```

```
      }).done(function() {
        updateListbyID(listID, updatedListName);
        $("#changeList-popup-container").hide();
      }).fail(function(err) {
        console.log("fail..........");
        console.log(err);
      })
    }
  });

  $('#addTask-form').submit(function(e) {
    e.preventDefault();
    const taskName = $('#taskName').val().trim();
    if (taskName !== "") {
      const taskData = $(this).serialize();
      $.post('tasks/create', taskData, function(data) {
        const lastPos = $("#myTask-ul").children().length;
        let date=(data.deadline > 0) ? data.deadline : data.reminderDate;
        const datetime = showDate(date);
        updateFilterListCountNewTask(data);
        $("#myTask-ul").append(
          `<li class="task-items-li">
            <a class="task-
items" tabindex="${lastPos+100} " id="taskItem${lastPos}">
              <input type="checkbox" class="taskCheckBox" id="taskCheckBox${lastP
os}">
              <span class="task-
name priority${data.priority}">${data.taskname}</span>
              <span class="task-datetime">${datetime}</span>
              <span class="task-
options" title="List options" style="visibility:hidden"></span>
              <input type="hidden" class="task-datetime-holder" value="${date}">
              <input type="hidden" class="taskId" value="${data.id}">
              <input type="hidden" class="taskListId" value="${data.listId}">
              <input type="hidden" class="priority" value="${data.priority}">
            </a>
         </li>`
        );
      }).fail(function(error) {
        alert(error)
      });

      $("#newTask-popup-container").hide();
    }
  });


  $('#addList-form').submit(function(e) {
    e.preventDefault();
    const listName = $('#newListName').val().trim();
```

```
    if (listName !== "") {
      const listData = $(this).serialize();
      $.post('lists/create', listData, function(data) {
        const lastPos = $("#myList-ul").children().length;
        $("#myList-ul").append(
          `<li role="menuitem" class="list-items-li">
            <a class="list-
items" href="#lists/${data.id}" tabindex="${lastPos+15}" id="listItem${lastPos+15
}">
              <span class="list-icon-container"><img class="list-
icon" src="/images/listItem.png" /></span>
              <span class="list-name">${data.listname}</span>
              <span class="list-taskcount"></span>
              <span class="list-
options" title="List options" style="visibility:hidden"></span>
            </a>
          </li>`
        )
      }).fail(function(error) {
        alert(error)
      });
      $("#newList-popup-container").hide();
    }
  });


  $("#deleteList").on("click", function() {
    $("#confirmDeleteList-container").show();
  });

  $("#no-confirmDeleteList").on("click", function() {
    $("#confirmDeleteList-container").hide();
  });
  $("#yes-confirmDeleteList").on("click", function() {
    const deleteListID = $("#changelistId").val();
    const deleteObj = JSON.parse(JSON.stringify({
      type: "list",
      id: deleteListID
    }));
    $.ajax({
      type: 'DELETE',
      url: "/lists/delete",
      data: deleteObj
    }).done(function() {
      $("#changeList-popup-container").hide();
      deleteList(deleteListID);
    }).fail(function(err) {
      console.log("fail..........");
      console.log(err);
    })
```

```
  })



  $("#priorityfilter-wrapper").hover(
    function() {
      $("#priorityFilter-container").slideDown('medium');
    },
    function() {
      $("#priorityFilter-container").slideUp('medium');
    });

  $("#datetimefilter-wrapper").on("click", function() {
    $("#datetimefilter-container").slideDown('medium');
  });
  $("#datetimefilter-container").on("mouseleave", function() {
    $("#datetimefilter-container").slideUp('medium');
  });
  $("#datetimefilter-wrapper").on("mouseleave", function() {
    $("#datetimefilter-container").slideUp('medium');
  });


});
```