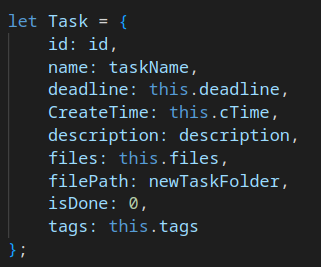
**~~CRINGELINER~~ DEADLINER API**

Работа со списком

Сам по себе задачник расположен в src/todo.js. В нём как раз таки и есть его логика, но работать надо с preload, он просто есть, импортировать не надо

Формат каждого таска:



Id – сам создаётся при добавлении таска

Name – название

Deadline – время окончания

createTime – время создания задания

description – описание

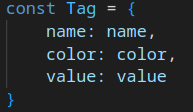
files – массив файлов

filePath – путь к папке с файлами задания

isDone — выполнено ли задание

tags — массив с тегами

формат тегов:



# **функции для работы**

# **С TODO**

**Функции доступны через чвойство “todo.///”**

# **ADD**

Func add (task, deadline, description, files) => void

Добавляет таск в список. Ничего не возвращает

Task – название задания

Deadline – ну, собственно дедлайн задания

*# время лучше передавай в формате (dd.mmyyyyThh:mm), с этим тут беда, #ну не только с этим, но сейчас я про время говорю*

Description(опц.) – это описание

Files(опц.) – закреплённые файлы. Передавать надо не сам файл, а путь к нему

# **getTask**

Func getTask (id) => [Task]

Принимает id задания. Возвращает массив содержащий задание

Id – id задания

# **Delete**

Func delete (id) => Boolean

Удаляет задание по id. Если удалено возвращает true, иначе false

# **Get**

Func get() => [Tasks]

Возвращает массив со всеми заданиями

# **Edit**

Func edit (id, name)=> void

Принимает id задания и новое название

#я что то забыл про эту функцию…..пока только имя меняет, потом доделаю

# **removeFile**

func removeFile (name) => void

удаляет файл указанный в name. сам файл ищет в папке для файлов задания

# **openFile**

openFile(name)=>void

открывает файл name. сам файл ищет в папке для файлов задания

# С ТЕГАМИ

**getAllTegs**

getAllTegs() => array

возвращает массив со ВСЕМИ СУЩЕСТВУЮЩИМИ тегами

**getTeg**

getTeg(name) => object

возвращает тег по имени

**createTeg**

createTeg(name, color?, value?) => object

создаёт и возвращает объект хранящий новы тег. Обязательный параметр только название, омтальные будут пустой строкой

**createTegs**

createTeg(arr) => object

создаёт и возвращает объект хранящий новы массив с новыми тегами. Остаются только уникальные теги. То есть если в массиве есть повьторения с уже сущестующими тегами, они не добавятся. Передавать надо массив в котором объекты в форме тегов(см. Формат тегов)

**addTegsToTask**

addTegToTask(id, tegs[teg]) => void

добавляет теги к заданию. Id — это id задания, а tegs — это массив с объектами тегов, можно смовмещять с функцией createTegs

**addTegToTask**

addTegToTask(id, teg) => void

добавляет тег к заданию. Id — это id задания, а teg — это тег, можно использовать в связке с createTeg

**removeTegFromTask**

removeTegFromTask(id, name) => boolean

удаляет тег из таска. Id — id таска, name — название тега

**removeTeg**

removeTeg(tegName) => void

полностью удаляет тег

**getCompTask**

getComplitedTask() => array

возвращает массив с выполненными тасками

**getUnCompTask**

getUnComplitedTask() => array

возвращает массив с невыполненными тасками

**toCompTask**

toCompliteTask(id) => void

переводит задание указанное по id в состояние «выполненно»

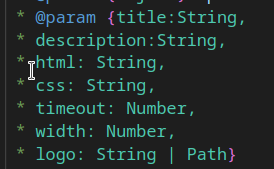
# **Уведомление**

Доступно в свойстве «notification.»

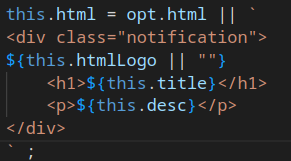
**createNotification**

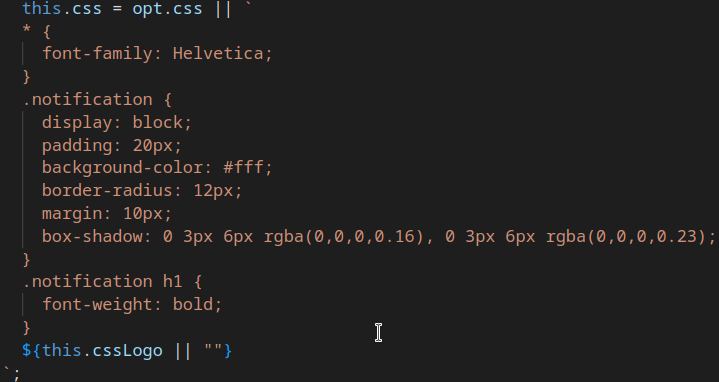
createNotification(opt) => void

в opt передаётся объект со значениями. opt состоит из:



по умолчанию html и css выглядит так:





В **description** (desc) передаётся описание

в **title** заголовок

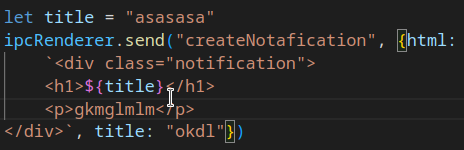
в **width** длина уведомления

в **timeout** то, сколько уведомление будет отображаться (в миллисекундах)

в **logo** (на рис. cssLogo) передаётся путь указывающий на картинку(png)

в **html** надо передать строку содержащую html разметку. В разметке обязательно должен быть основной блок с классом “notification” (прим. <div class="notification">…...</div>). внутри может быть что угодно

всё что указано на картинках это работает по умолчанию и параметры завязаны на них(desckription, logo, title). Если ты решишь сделать свои стили то использовать эти параметры уже не получится, добаляешь, свои (щас пример будет)



в **css** передаётся в виде строки: css, как удивительно.

# **Обработчики**

Все обработчики (за некоторым исключением) принимают в себя callback который будет вызываться при наступлении события

**ErrorHadler**

errorHadler

обработчик ошибок. Все происходящие в программе ошибки будут прилетать сюда в виде массива в котором содержится сообщение ошибки

**eventHandler**

eventHandler

обработчик различных событий. Пока что ни для чего не используется. Он просто есть

**exitHandler**

exitHandler

обработчик который срабатывает, когда пользователь нажимает на кнопку выхода в трее

**exit**

exit()

ничего не принимает. При вызове закрывает приложение

**addTaskTrayHandler**

addTaskTrayHandler

обработчик, который срабатыет, когда пользователь в нажимает “добавить задание” в трее

**DOOMDAYEvent**

DOOMDAYEvent

обработчик, который срабатывает при пропуске таска

**DOOMDAY**

DOOMDAY()

функция вызывает удаление тасков и т.д DOOMDAY короче