

# Quill App Migration Status and Changes 🤩

---

Updated 01/30/2022



[Doc with more detail](#)

# Overview

1. About Quill
  - Features
  - Help needed
  - Monetization ideas
2. Architecture overview and how data moves
  - Overview
    - UML
3. Back-end
  - Rails ➔ Django
  - PostgreSQL database
  - Migration status and future work
4. Front-end
  - Overview
    - UML
  - React
  - MobX state management
    - Benefits
    - Responsibilities
    - Our stores
    - Example
  - Migration status and future work
5. Going forward

# About Quill

# Features

# Why Make Quill To-Do?

Problems with other todo apps

- Only have due dates and maybe sub-tasks
- Usually only show a list view, no calendar
- Adding a task takes too long and becomes not worth it
- Very few are open-source
- Hard to visualize if your work is distributed well or not

When one might have one or two features, no product exists with all

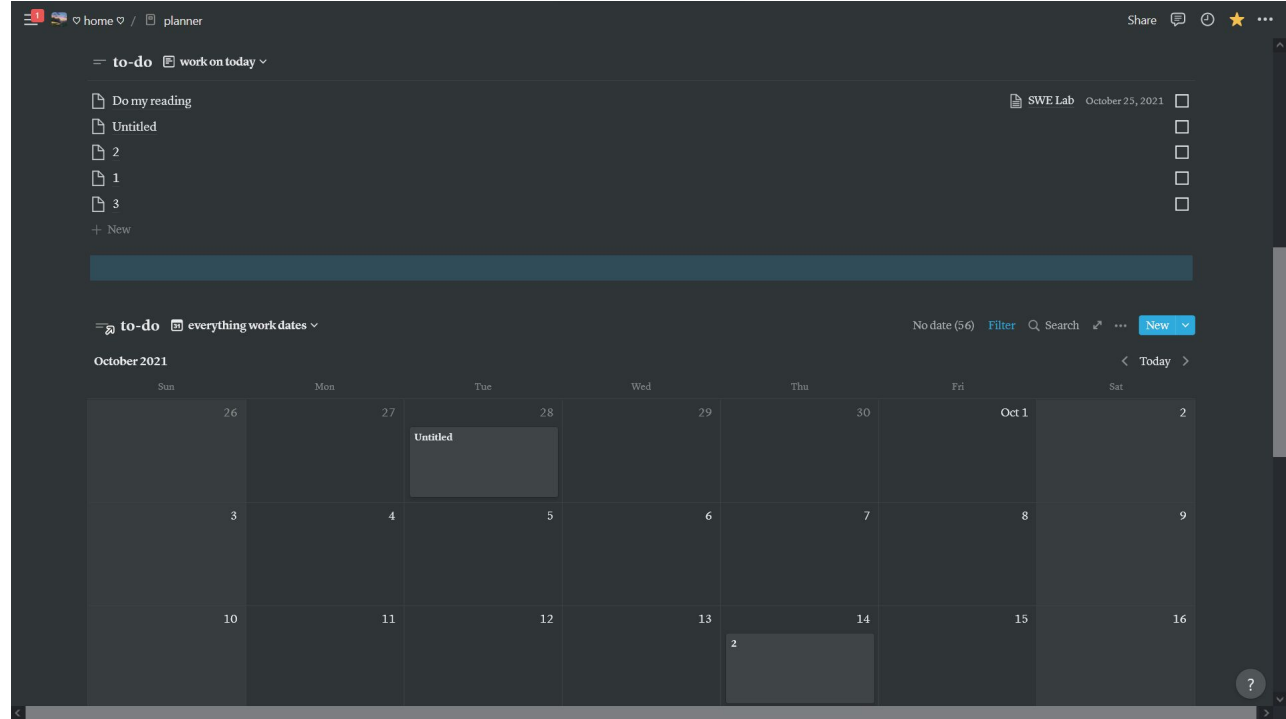
# Ex 1: Notion



- Lots of detail possible
- Calendar and list views



- Maintaining it becomes its own chore
- Takes too long to add a task
- Requires network connection
- Closed-source



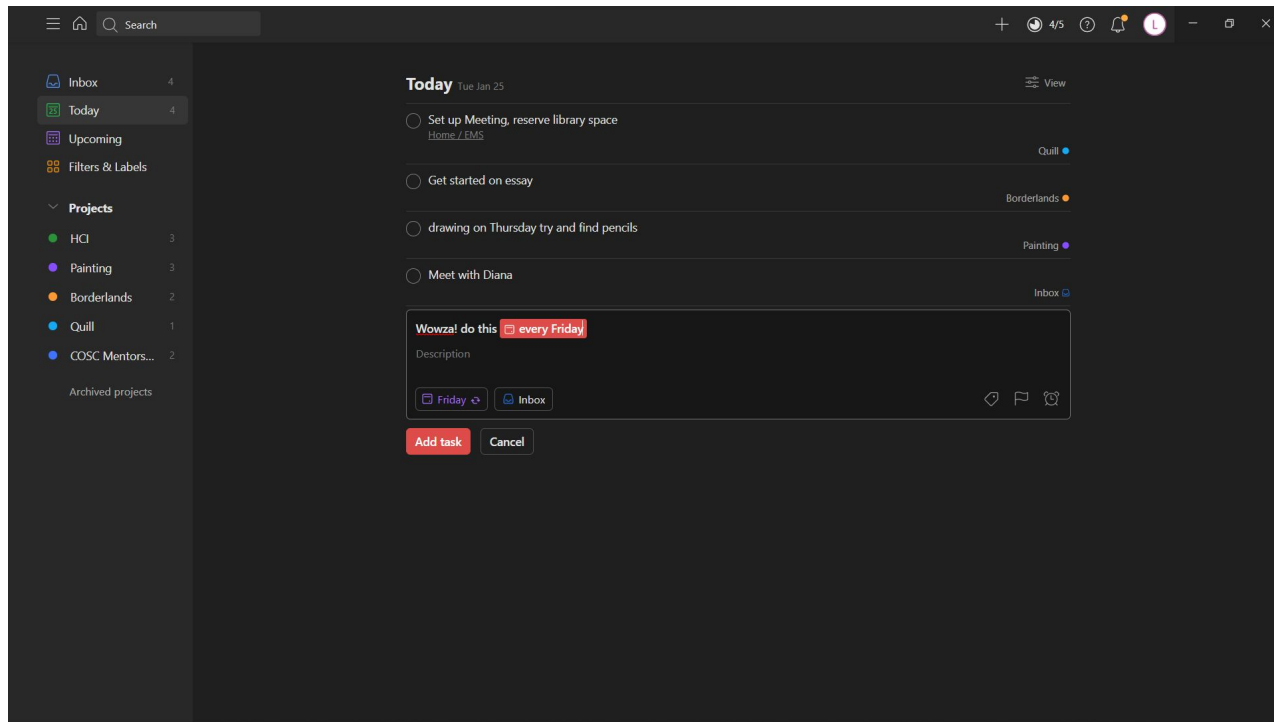
# Ex 2: Todoist



- Super fast and nice task addition using natural language processing



- Only supports due dates, tasks tend to sneak up on you
- No calendar view
- Closed-source



# The Core of Quill

- Recreating my ideal planner setup
- List and calendar view
- Add any number of different “work” dates/subtasks for a task and a final due date
  - One-click addition of subtasks
- Markdown task description
- Fast creation of tasks using NLP
- Drag to reschedule tasks, within calendar or list or between them
- A system to encourage the user to re-distribute work evenly
- Open source
- Would like to make desktop app with optional disconnection from network





**Monetization**

# Ideas

## **Priority: Making this accessible for people who can't afford to pay and remaining open-source**

The idea of a sliding payment scale is very appealing.

- Limited number of tasks or groups, pay to permanently or increase this number or pay monthly
- Have other features in addition to a task list that could be paywalled
  - Ex: Habit tracker
- Make collaboration paywalled
  - I'm not as into this idea because Todoist does this and it's annoying and would give us a competitive advantage to not
- Pay to be a supporter and get access to beta and development insight
- Need to look into how open-source projects are monetized

**Help Needed!**

# Why Develop for Quill To-Do?




- As big or little of a commitment as you want
  - Don't need to commit to weekly meetings, completing a certain number of issues, pick anything up that looks interesting to you and I'll be glad to help you get started
  - I'll try to keep issues up-to-date, with a good description and labeled so you know what's expected to be involved
  - If you pick up an issue and realize you can't complete it, I only ask you commit to explaining your progress and decision making to the next person picking it up

# Cont.

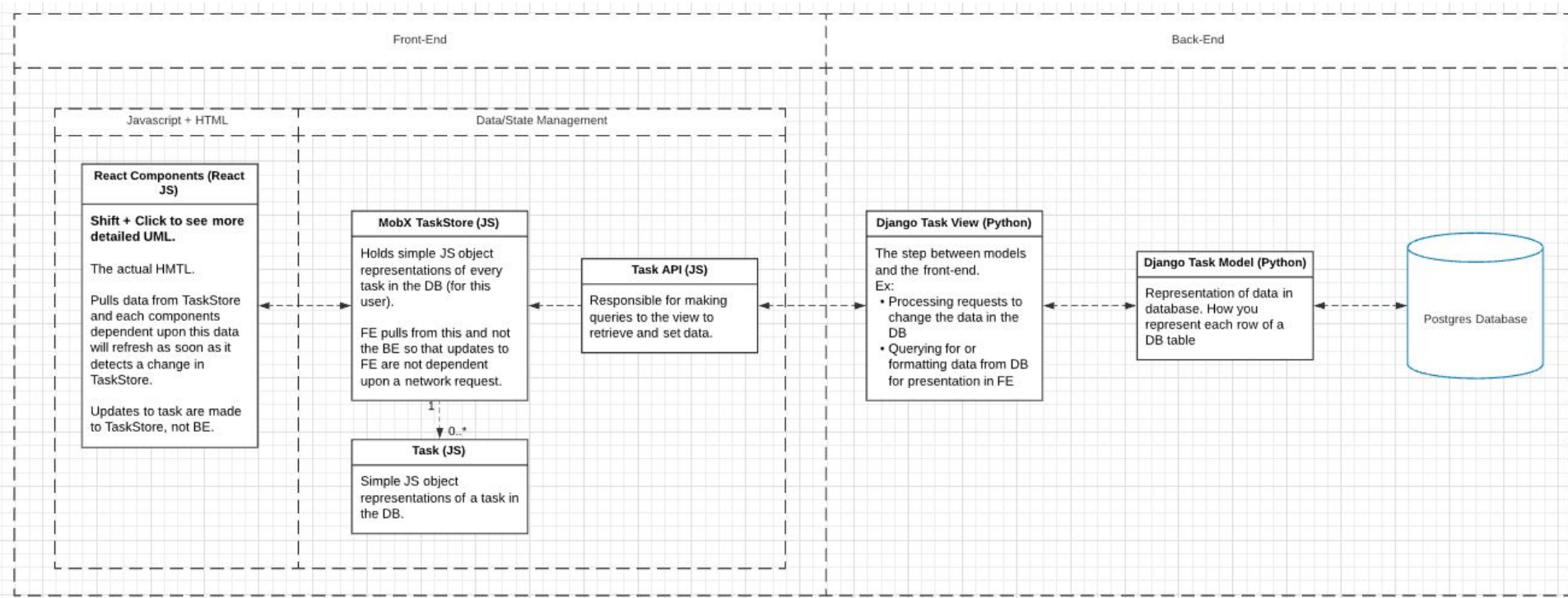
- Learn tech + libraries not taught at Colgate
  - More advanced Javascript
  - ★ React
  - State management
  - Django
  - Full-stack web app development
- Get experience on a development team
  - Trying to follow industry standards and best practices for collaborative dev work
- Get experience working with an existing code base
  - Except it's better than working with a big scary legacy codebase because the people who made it are still here and are really excited about helping and answering questions :')
- Get in early!
  - Lol but also maybe for realsies
  - If this does take off and you've contributed
    - i. It would look cool on a resume
    - ii. It could turn into a job
- I'll love u 🥰

# **Architecture Overview**

# Very High-Level Changes

- Architecture changes:
  - Rails →  Django back-end (BE)
  - Templates/Views →  React front-end (FE)
- BE is doing a lot less work, a lot more FE heavy now
- Moved repos, now in  Quill-ToDo
  - Rails app and NLP task parser repos are also there

# Architecture UML - How Data Moves





**Back-End**

**Rails → Django**

# Rails → Django (Terminology)

- MVC (model-controller-view) → MVT (model-view-template)

Role	Rails	Django
Representation of data in database. How you interact with the tables	Model	Model
<p>The step between models and the front-end. Ex:</p> <ul style="list-style-type: none"><li>● Processing requests to change the data in the DB</li><li>● Querying for or formatting data from DB for presentation in FE</li></ul>	Controller	View
HTML templating: sticking data passed from DB intermediary into HTML	View	Template

# Django Admin Site

- Django comes with a nice admin site to edit database data
- Access by going to <port number (like 127.0.0.1:8000)>/admin in a browser)

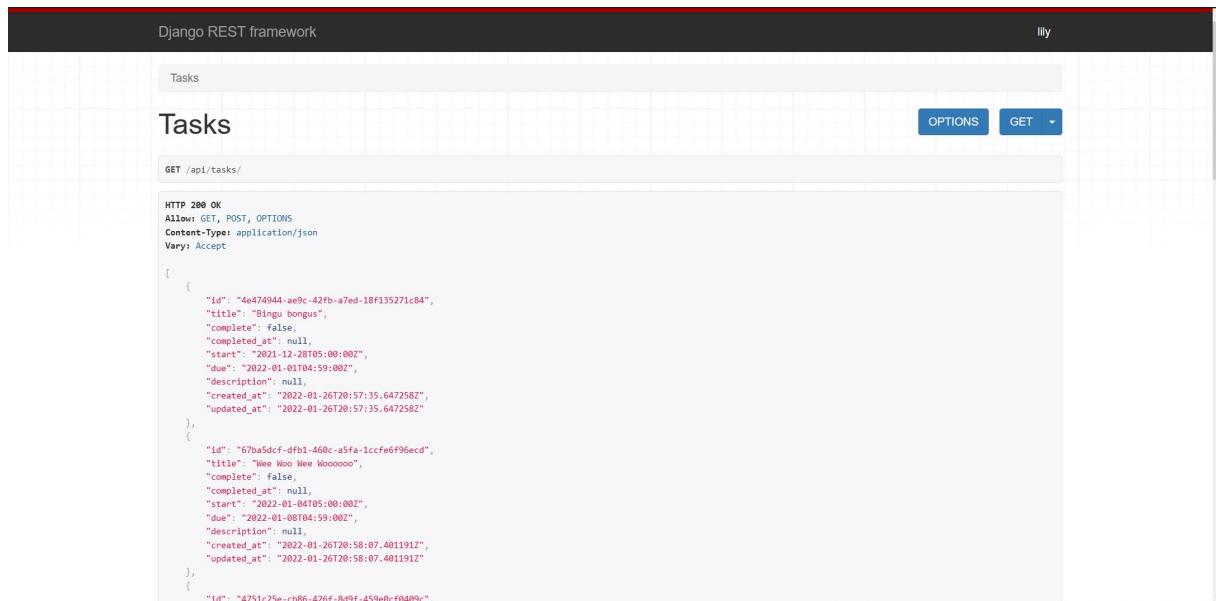
The screenshot displays the Django Admin Site interface. At the top, the header reads "Django administration" on the left and "WELCOME, LILY VIEW SITE / CHANGE PASSWORD / LOG OUT" on the right. Below the header, a breadcrumb trail shows "Home > Tasks > Tasks". The left sidebar contains a search bar and a list of apps: "AUTHENTICATION AND AUTHORIZATION" (with sub-items "Groups" and "Users"), and "TASKS" (with sub-item "Tasks"). The "Tasks" app is selected and highlighted. The main content area is titled "Select task to change" and includes an "ADD TASK +" button. Below this, there is a table with 9 tasks. The table has columns for "COMPLETE", "TITLE", "START", and "DUE". The "COMPLETE" column contains checkboxes and status icons (red 'x' for incomplete, green checkmark for complete). The "TITLE" column lists task names. The "START" column shows start times, and the "DUE" column shows due times.

COMPLETE	TITLE	START	DUE
<input type="checkbox"/>	Bingu bongus	Dec. 28, 2021, 5 a.m.	Jan. 1, 2022, 4:59 a.m.
<input type="checkbox"/>	Wee Woo Wee Woosooo	Jan. 4, 2022, 5 a.m.	Jan. 8, 2022, 4:59 a.m.
<input checked="" type="checkbox"/>	Test add	-	Jan. 20, 2022, 5:59 a.m.
<input checked="" type="checkbox"/>	Get ready for stuff	-	Jan. 20, 2022, 5:59 a.m.
<input type="checkbox"/>	Blah	-	Jan. 21, 2022, 4:59 a.m.
<input checked="" type="checkbox"/>	Work on quill	Jan. 19, 2022, 6 a.m.	Jan. 22, 2022, 5:59 a.m.
<input type="checkbox"/>	another one	Jan. 18, 2022, 5 a.m.	Jan. 23, 2022, 4:59 a.m.
<input type="checkbox"/>	omg broslf	Jan. 1, 2022, 5 a.m.	Jan. 24, 2022, 4:59 a.m.
<input checked="" type="checkbox"/>	Add first day classes	-	Jan. 24, 2022, 5:59 a.m.

9 tasks

# Web Browsable API

- Lets you easily interact with the back-end API without a client like Postman or using curl, provided by Django Rest Framework
- Access by going to <port num>/api/tasks/



# PostgreSQL Database

# PostgreSQL Database

- Django recommends using the same DB for development and production, so we are using a heavier duty DB called Postgres for dev and prod
- Requires a little additional setup, covered in contributing.md.
  - Install Postgres
  - Initialize a database in a certain way - we have a script that does this for us
- Our tasks will not be synced between us in dev



**BE Status**

# Done

- Basic task model without validations
- Basic serializer using Django Rest Framework (helps to access and define end-points) which is built using task model
- In task view:
  - Serving FE landing page after FE is built (`$ npm run build`)
  - Getting all tasks
  - Getting a task (id)
  - Patching (updating) a task (id)
  - Deleting a task (id)
  - Posting a task (model validations are needed here)
- Postgres DB set up

# Needed for Migration

- Model validations (making sure we only save data that's formatted correctly, etc.)
  - This is needed because the task serializer for our rest endpoints uses model validations to make sure the request is valid
- Change some queries to only allow requests to access/modify certain allowed fields
  - Ex: Shouldn't be able to change task ID
- User model and related associations to tasks
- Back-end testing
  - Not using a running server, basically just unit tests

 [See all BE migration issues](#)

# Future Work

- Figuring out a more efficient way to leverage the BE
  - RN passing all data to FE and using  $O(n)$  searching/filtering in JS
- Filling out a nice API for launch with documentation
  - Adding more endpoints that may be useful for users
  - Will require a lot of work for security, validations, etc. to make sure users can't break things if we expose our API

 See all future BE issues

# Front-End

# High-Level Changes


- Front-end is a lot more complicated
- A lot of conditional rendering logic and event handling is done in FE instead of BE
  - Not just templating (inserting data into HTML)
- FE has a state management system now

It can get a little confusing, don't hesitate to refer to the FE UML, I'll try to keep it updated

 [FE UML Diagram](#)

**React**

# React Basics

- React works as a Javascript framework (technically a library I think)
  - Much more ideal for a single-page app than our Rails solution
  - If one tiny part of the page changes within a bigger component, only the part that is actually changed is re-rendered in the DOM - much faster
  - Using  Create React App toolchain - very easy to get projects up and running but provides less control
- React is built to use composition structure (bigger pieces/components are made up of smaller components)
  - Mostly a top-down hierarchy where data is passed from the bigger/higher components to the smaller components which compose them, but the task store can be accessed at any level
- Components can have their own methods and state and return templated HTML
  - If something in its state (or the task store) changes, the component is re-rendered to show the updated changes (and ONLY the changes are actually re-rendered)



# **MobX State Management**

# Basics



- MobX provides state management beyond what React can
- **Stores:** Collections of data that are monitored for changes by using functions provided by MobX
  - The store itself is a vanilla JS object with some special MobX methods, the data it stores can be anything, in our case another object
- **Observing components:** Components wrapped with the “observer” MobX function
  - If something in the store is changed this component is re-rendered to reflect possible changes
- Our task store is basically a replica of the data we have in the database that the components pull from instead of the DB directly

# Responsibilities of a Store


- Make and handle all network requests involving its data
  - In our case, this is the only place the FE API used to access the BE is used
- Keep the server data up-to-date
  - As soon as a task is changed in the FE, update the DB
- Hold accurate data
  - Populate the store based on the DB data on init
- Make any computations that can be made, immediately
  - As soon as it has the data to make computations, it does so behind the scenes
  - When we make a request to the store if it has already computed that data the result is almost instantaneous

 More details

# Benefits

- Faster
  - Updates can be made from the FE to the task store instead of the back-end
  - The change will then percolate to the rest of the component tree and re-render any affected components immediately
  - At the same time, behind the scenes the actual network call to update the DB which takes a comparatively long time is made
  - If this request fails for any reason, we catch the failure response, notify the user of a failure, and revert the change in the store and FE
- Simple
  - Using the store in a component requires one method call and  wrapping the component in an observer call
- Global
  - We don't have to use a complex series of callbacks to access state way higher up in the component tree, we can access it at any level using React  context

# Our Stores

- Task Store
  - Holds a list of tasks and methods to change and interact with them
  - Each task is a Javascript object holding data relevant to the task and some methods to handle changes and interact with the task
  -  This task object is different from the task component - the task object only holds data and has methods to change data, the task component is responsible for rendering this data into HTML
- Alert store
  - We also have a store for alerts but that is less important than the task store. It basically allows us to render an alert from any component or anywhere we want

 [More details](#)

# Example Process

The show task component (view task details) holds a reference to the task object instance in the store its currently pulling data from

- A user completes a task from the show task component
- In the show task component, a event listener fires when the user clicks the check-box
- The event handler accesses the task object and changes its completion status to true
  - `task.toggleComplete()`
- The MobX task object “hears” this change. It sends a PATCH request to the BE to update this task in the DB
- As that request is going through, almost immediately the list component (visible at the same time as show) “hears” that the task store has changed, and re-renders the task in the list with the new data from the store, rendering a check-mark
- Some time later the BE receives the request. If it was successful, it succeeds silently and the user is none the wiser. If it fails, we listen for and catch the failure response from the BE in the task object. We revert the change that was made in the task object (immediately changing the UI visuals back) and render an alert to let the user know that something went wrong

**FE Status**

# Done

- Basic list view
- Completing and uncompleting tasks
- Task show (view details) on click
- Alert system
  - Less urgent alerts (notice or success types) will scroll out and remove themselves without user interaction, more urgent (failure type) require dismissal
- Improved accessibility a bit which is really helpful for testing and is important in general
- Basic testing for list view (at 95% coverage)
- Task creation popup (PR is drafted, I'm making some last minute changes)
  - Uses time-picker for time with the expectation that clicking on a date in the calendar should fill out that date in the selected creation field
  - Has basic FE validations, still needed in BE



# Needed for Migration

- Finishing task-creation pop-up
- Calendar (in-progress)
- Login page
- Figuring out what's wrong the the tests I had to skip
- At least 1 or two integration tests (making sure we only save data that's formatted correctly, etc.)

 [See all FE migration issues](#)

# Future Work

There are 30 FE issues open RN, feel free to add issues for any ideas you have

Some core features:

- Groups
- Subtasks
- User settings
- View by day for list view
- Dragging within list, from list to calendar to change dates



[See all future FE issues](#)

**Going Forward**

# Immediate Next Steps

- Having people review PRs somewhat quickly is really helpful
- Need to host after migration is done
- Not needed for migration but need people to look into the best way to package the task parser script for use in the app
  - I'm thinking NPM, that way we could call it from FE, but that may only be for JS
  - This also needs more work but I think it's good enough for now that we can begin to figure out how to integrate it into the app
- Design (functionality) questions I'd like to go over with at least someone else to get ideas

Qs:

- Do we want to have weekly meetings?
- Are people getting notified on Slack?

# Getting Started with Contributing

## 1. Poke through issues on Github

- They are labeled by what they are expected to involve
- Some of them have more detailed descriptions than others
- If there are any questions at all, reach out and I'll explain more and can give guidance on where to look for things, how to get started, etc

## 2. If you find an issue you want to take, read CONTRIBUTING.md in the repo for how to get started

- I've tried to make it as detailed as I can but if anything doesn't work or isn't clear, please reach out, I may have forgotten to include something or there may be an OS difference
- *I've made changes that make it easier to get up and running but I need to finish the popup to get the changes into main, before that's merged you can reach out to get the scripts*