TASK PLANNING APP VERSION 0.1 SPECIFICATION

1. Summary

Operational lookaheads are used throughout construction and other industries. Although the specific nature of the operations/tasks they describe may change depending on the sector, the purpose of these simple ‘on-the-ground’ planning tools remains the same: to give all the stakeholders in the operational team a **common perspective** on what the next **6-7 days** of the overall construction project entails - most lookaheads are developed as **simple excel spreadsheets**.

The aim of this project is to develop a Single Page Application (SPA) that replicates the functionality of a simple spreadsheet based ‘operational-lookahead’ tool. For now, this will just be a simplified version of the app for personal group use, but future enhancement could be possible.

2. Preferred Tech stack overview

- React.js and Redux frontend
- Node.js / Django REST backend (preferred for ease of managing users, operations cards and other entities)
- Ideally Postgresql as database
- Hosting: AWS EC2 / Heroku(preferred) / Google cloud
- Authentication: JWT token based system with protected routes
- Production WSGI Server: ideally would like uWSGI, Gunicorn or Nginx

3. Overall user journey

The overall user journey for signing up for and using the app is similar to that of trello, todoist or other planning / productivity tools, and would proceed as follows:

3.1. User proceeds to livelookahead.com (root url) and signs up with their email address and some basic personal details, (first name, last name and country).

3.2. Upon verifying their email address, the user then creates a new lookahead instance (this is akin to creating a new board on trello). A user may create multiple lookahead instances, where each one could represent a different building/construction project.

3.3. The user then adds “operations/task cards” to a lookahead instance. These cards have descriptions and time estimates of a particular operation/task. As work progress read-write users update these operations/task cards with actual time durations.

3.4. The user can invite other users into the lookahead instance by adding their email addresses in the settings panel – these additional users can be read-only or read-write users.

3.5. Each lookahead instance has a unique hashed url (e.g. livelookahead.com/q=?h6jkh) which can be used to navigate to that particular lookahead instance. Upon clicking this link, a user should be able to view that particular lookahead instance, providing they are authenticated via a login.

4. Frontend design decisions

Frontend views should be built in React.js with Redux used for state management

4.1. There will be 3 standard views, that should essentially just be replicated versions of bootstrap component examples. Functionality is also standard. These are:

- **Simple static landing page view** (a 'pricing' style bootstrap page as per here: https://getbootstrap.com/docs/4.0/examples/pricing/ that I can edit the content of later, for now just use dummy content but have signup and login links point to the views below)
- **Login view** (simple email address and password).
- **New user registration view** (Simple name, location, email address and password, with confirmation email flow to verify account)
- **Change password + forgot password view** (simple standard functionality)

And 3 specific views, which give the core functionality:

- **Lookaheads view** (showing the lookahead instances which the user has access to)
- **Operations lookahead view** (This is the main view of a specific lookahead instance, showing the operations cards, which represent upcoming operations/tasks.)
- **Lookahead Settings view** (changes settings on a specific lookahead instance, not accessible to read-only users, only to read-write users)

These 3 specific views are now described in detail.

4.2. The **Lookaheads view** should allow the following functions:
4.2.1. Allows the user to view the lookahead instances that they have created or have access to (like viewing your boards on trello)
4.2.2. Allows a new lookahead instance to be created with a custom name and description
4.2.3. Allows a lookahead instance to be deleted
Note: sharing access to a lookahead is not done in the lookahead settings view

4.3. The **Lookahead settings view** should allow the following functions:
4.3.1. Add a new user to share access to this particular lookahead instance. The user is either of "read-only" type or "read-write" type. The new user is added via their email address and should receive a unique signup link emailed to them. This signup email should notify them that they have been invited to access a lookahead, with a link that asks them to set a password before taking them to the actual lookahead instance they have been given access to
4.3.2. Show the list of users which the particular lookahead instance is shared with
4.3.3. Allow the permissions (e.g. read-only or read-write) settings of the users listed above to be changed

4.4. **Operations lookahead** view. This is the main interactive view, built in react to allow for "real-time" updates – changes made here should be instantly reflected if another user is viewing the same page without the need for a page refresh.
4.4.1. It is essentially a table with columns representing the days, which are filled with "operations/task cards" representing an operation/task (inspiration here is the Kanban view used by https://monday.com/product/ and the trello cards layout.)

4.4.2. It should be possible to "click and drag" and "stretch" operations cards to different times and spans, and this should shift all subsequent operations forward (e.g. move an operation/task forward by one hour, all future operations shift forward 1 hour)

4.4.3. A red line should be drawn where the "current time" is, moving as the day progresses

4.4.4. All operations cards **behind this "current time**" line should be grey as they are in the past. All operations in **front of the "current time"** line should be in a different / darker color as they are in the future

4.4.5. For the past operations/tasks, the actual time taken to perform these operations/tasks needs to be entered by the user. To do this, when a past operation/task is clicked, a pop-up should appear **allowing the user to enter the "actual hours" taken** for this operation/task. When entered this should then update the position of the operation/task card in the view, based on the actual time duration entered.

4.4.6. In addition to clicking on the historical operation and entering the actual time, it must be also possible to provide this same input by "clicking and dragging" an edge of the operation/task card to a new time slot. Because this is being done to a past operations/task card, the system should interpret it as an **"actual" time being inputted**, and update all subsequent future operations/tasks accordingly. If an operations/task card in the future (beyond the red current time line) has it's edges dragged to expand its duration, this should just be interpreted as a **general update to the forecasted time** for that specific operation/task, not as an 'actual time taken' being inputted.

4.4.7. When a past operation/task has a tick icon, it means that an "actual" time has been provided. When no "actual" time has been entered, a question mark icon is instead displayed

4.4.8. A hamburger/settings icon should exist in the top right of the operations lookahead view, revealing the lookahead settings view / draw

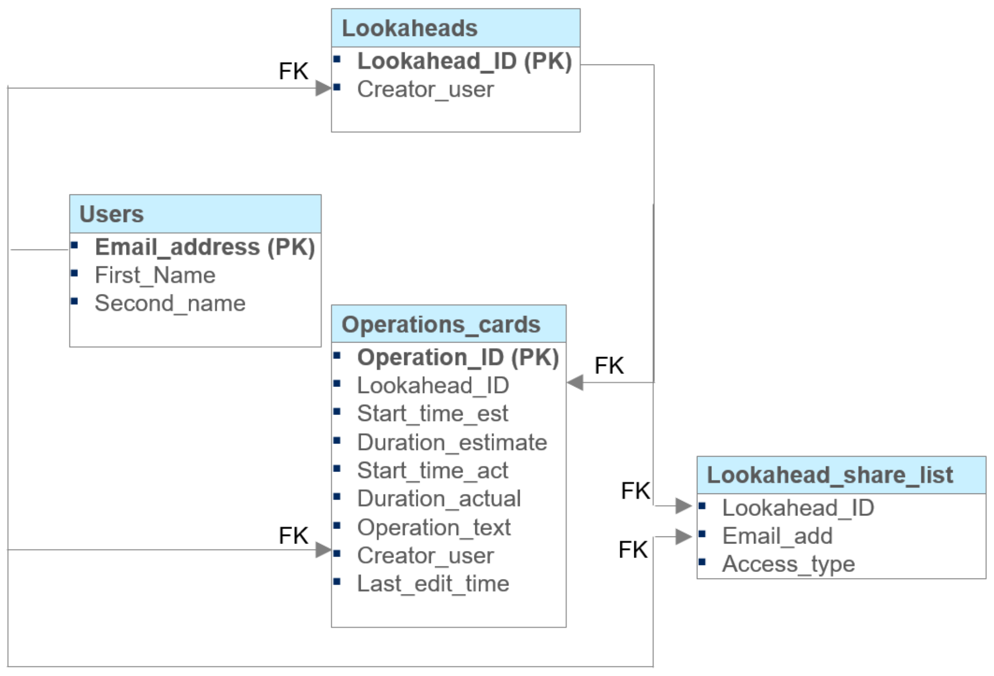
4.4.9. The view needs to be responsive, with each day occupying the screen on standard mobile size view

4.4.10. The operations lookahead view should cover a period of 6 days, but it must be possible to scroll to the right to see more days in the future. It should also be possible to scroll to the left to go "back in time"

Note: in general there should be no "save" buttons, all entered data should save when the user clicks off "goes out of focus" i.e. similar to a google docs experience, the state should be managed with Redux to achieve this (a debounce function might be needed the throttle updates if on a slow connection)

5. Schema design

In general, there are 4 entities. One lookahead instance contains many operations/task_cards. A user can create many lookahead instances, and each operations/task_card is owned by the user that created it. Each lookahead instance can be viewed by multiple users, and a user may have access to multiple lookahead instances: this many-to-many relation is the reason for the lookahead_share _list table, tracking each user's access permission to each lookahead instance.



Initial schema idea – feel free to deviate from this if needs be. I missed some fields like lookahead name and description.

6. Authentication and user permissions

6.1. Authentication should be JWT (jot token) based, passwords hashed and not stored in the DB

6.2. All views should be protected routes, with a particular lookahead instance only being accessible to users that have been explicitly included in the sharing list as described in the settings view functionality

6.3. There are only two types of user, as described earlier. The available functionality of these two users is summarized below

Function	**Read-Write**	**Read Only**
View lookahead (including scrolling through time)	X	X
Edit lookahead (including adding actual operations/task times or changing operations/task cards in any way)	X	
Inviting new users to share the lookahead instance with	X	

7. Infrastructure/hosting

7.1. App should be deployed to Heroku, (preferred) AWS EC2 or google cloud, small scale is fine here – no load balancer etc. required (will only be for small scale personal use)
7.2. Database should also be hosted on one of the above
7.3. I have the domain livelookahead.com, this should be configured for the app (I can supply and update CNAME records etc). root page (livelookahead.com) should be the **simple static landing page view**.
7.4. Would like the code to be hosted in a private repo on Bitbucket or similar (I will pay costs for this)