actions

~~Create label~~
Create draft emails or send emails + label them
Edit subject. - if draft, this is easy. if not, have to send new mail + tag it.

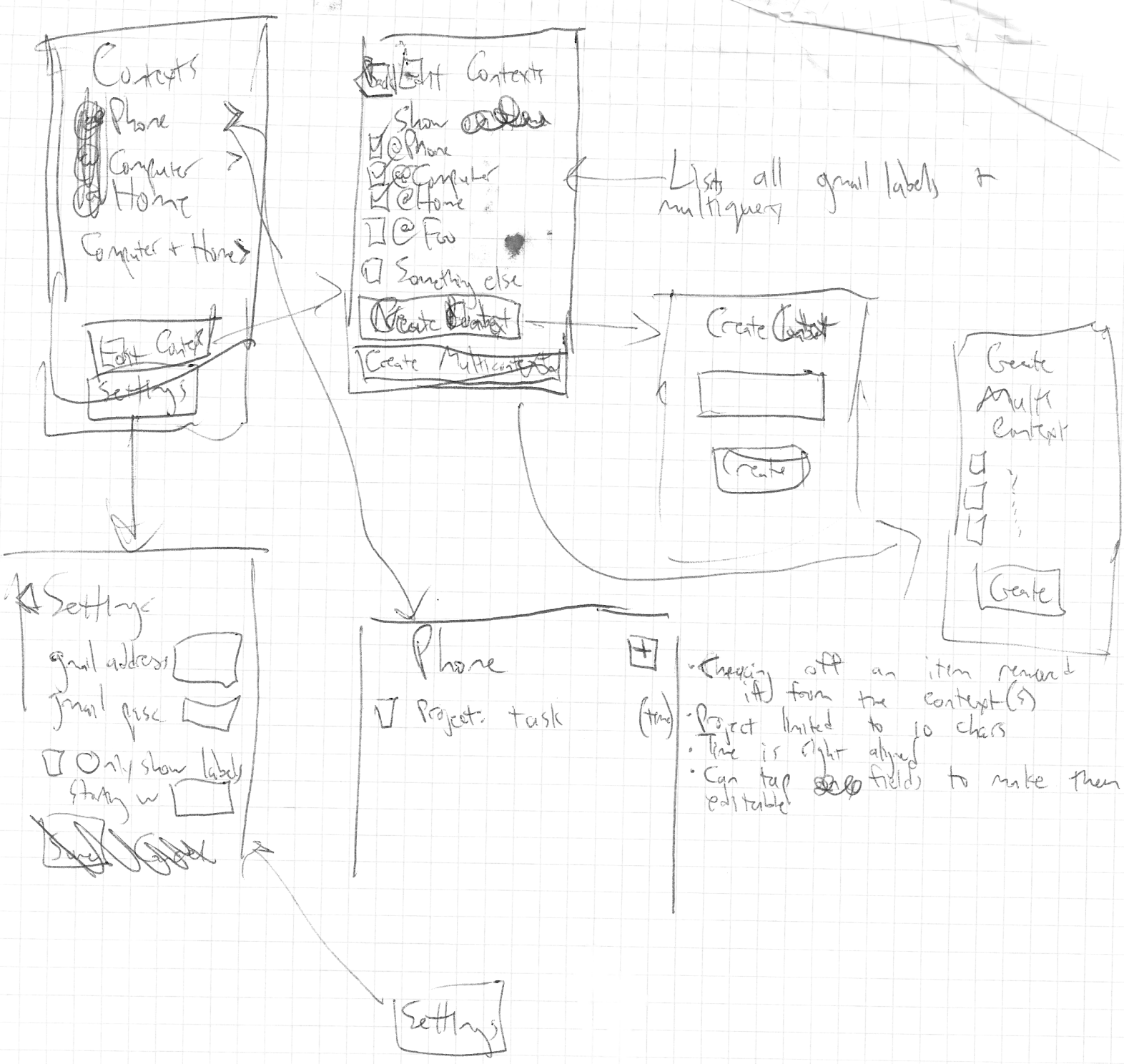
App ops

Get labels
Create label
Create task w context(s)
Edit task text - Create new task + remove old one from cxt
Add context to task
Remove context from task

IMAP

Get folders
Create folder
~~Send [illegible] it comes,~~ Create email/copy draft to folder, send
~~Move it to folder~~
Copy to folder
~~Move it to inbox~~ Remove

* Google label manipulation via IMAP



- Checking off an item removes it from the context(s)
- Project limited to 30 chars
- Time is right aligned
- Can tap fields to make them editable

to Do

Add context to existing task

GmailTaskStorage
Get Contexts(prefix="")
Get TasksWithinContext(context)
Create Context(context)
CreateTask(task)

EditOp?
op + param
data {}

~~Ops To Do List~~ commands

Syncer
ops To Do
task Storage

User types new task + clicks create.
UI sends event w op = create task + text = whatever
UI Updater responds to the event by adding it to the dom
Syncer responds by adding it to the ops to do list and ~~eagerly~~ tries
if online, starting a sync + resetting sync timer
When the sync starts, the syncer iterates through ops to do and
uses Gmail Task Storage to execute each.
~~Once that is~~ TODO: what about conflicts?
Then it pulls down all the tasks again and ~~either~~ sends
event(s) to update the dom

Django - HTTP/Stateless

3 Sections:	
Data storage	I/O from/to disk
action handling	Input from UI
Rendering	Output to UI

JS - Stateful

Can manage state explicitly:

User clicks add item
Event handler (controller) parses textbox via this, creates new task, added to list (model), redraws list (view).

Code Hierarchy:

```
{   settings: { StatusWidget, toggleWidget
    models: { // data classes
    }
    controllers: { // ui event handlers
        function addTask(event) { ...
            var text = event.widget.value;
            var task = parseTaskText(text);
            var list = server.addTask(task);
            view.list(list);

    state: { // anything that must be persisted over UI changes
    }
    server: { // functions that talk to server
    }
    views: {...} // draws and updates UI
}
```

comedy - homer - la dance mo flatbush choir monty python

```
function main(){
    views.app();
    setInterval(5000, controller.updateList)
    controller.updateList();
}


controller.updateList = function() {
    views.list(server.getList());
}

view.list = function(list) {
    var newBody = ... template...
    var body = topWidget.find('tbody.list');
    body.replace(newBody);
}

view.app = function() {
```