CSE186 Fall 2022

Assignment 8

In this assignment you create Slug Mail, a Google Mail style e-mail system, as a Single Page Full Stack Web App using the NERP Stack: Node.js, Express, React and PostgreSQL, plus Material-UI.

This assignment is worth 15% of your final grade.

Late submissions will not be graded.

Installation

See instructions in Assignment 2 and ensure you are running the current LTS version of Node.js. You will also need to have downloaded and installed Docker Desktop: https://www.docker.com/products/docker-desktop.

Setup

Download the starter code archive from Canvas and expand into an empty folder. We recommend, if you have not already done so, creating a folder for the class and individual folders beneath that for each assignment.

The starter code archive contains some files you will modify:

```
backend/api/openapi.yaml
backend/src/app.js
backend/sql/data.sql
backend/sql/schema.sql
backend/sql/indexes.sql

frontend/src/components/App.js
frontend/src/components/__tests__/App.test.js
```

Some you should ***not*** modify:

```
package.json

backend/.env
backend/docker-compose.yml
backend/package.json
backend/src/server.js
backend/src/dummy.js
backend/sql/database.sql
backend/src/__test__/db.js
backend/src/__test__/dummy.test.js

frontend/package.json
frontend/public/index.html
frontend/public/favicon.ico
frontend/src/index.js
```

And some you may want to remove after basing your tests on them:

```
e2e/tests/Dummy.test.js
frontend/src/components/Dummy.js
frontend/src/components/__tests__/Dummy.test.js
```


To setup the development environment, ***navigate to the folder where you extracted the starter code*** and run the following command:

```
$ npm install
```

This will take some time to execute as `npm` downloads and installs all the packages required to build the assignment.

To start the database Docker container, in the `backend` folder run:

```
$ cd backend
$ docker-compose up -d
$ cd ..
```

The first time this runs it will take a while to download and install the backend Docker PostgreSQL image and start the database service for your server to run against.

To start the frontend and backend dev servers, run the following command:

```
$ npm start
```

The frontend and backend servers can be run individually from their respective folders by running `npm start` in separate console / terminal windows.

To run the linter against your API or UI code, run the following command in the `backend` or `frontend` folder:

```
$ npm run lint
```

To fix linter errors, run the following command in the `backend` or `frontend` folder:

```
$ npm run lint -- --fix
```

To run API tests run the following command in the `backend` folder:

```
$ npm test
```

To run UI tests run the following command in the `frontend` folder:

```
$ npm test
```

To run end-to-end tests run the following command in the `e2e` folder:

```
$ npm test
```

To stop the database, run:

```
$ cd backend
$ docker-compose down
$ cd ..
```


Web App Specification

This system must be a Single Page Web Application using the following technologies.

Browser:	React & Material-UI
Server:	Node.js & Express
Storage Tier:	PostgreSQL

You can only use packages provided by the starter code. If you run `npm install` to use a 3rd party package, parts of your submission will fail to "compile" in the grading system and you will receive reduced marks for the assignment. In extreme cases you may receive no marks at all.

The server must present an OpenAPI constrained authenticated API to the SPA running in the browser and the server must store information other than its secrets in a PostgreSQL database.

Do NOT have the user interface simply download the contents of the database at startup and run all subsequent operations from an in-browser-memory cache. Submissions taking this approach will be *severely* marked down.

A "mobile first" approach should be taken; the principal operations being:

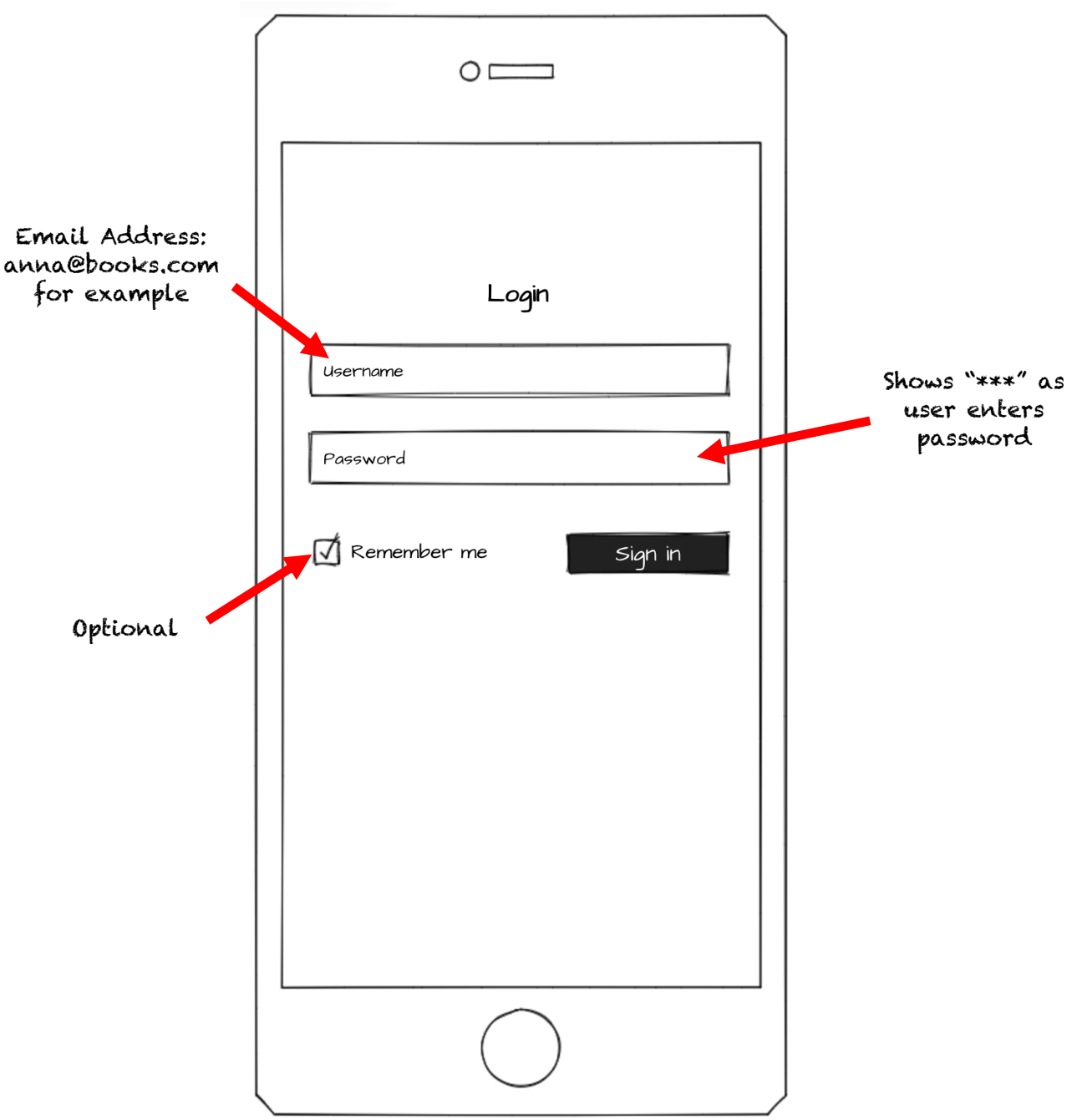
- Login Screen
- Mailbox Viewer
- Mailbox Selection
- Mail Viewer
- Mail Composer
- Search

For the desktop version, the principal operations are as for mobile, but composition should be a modal dialog rather than full screen.

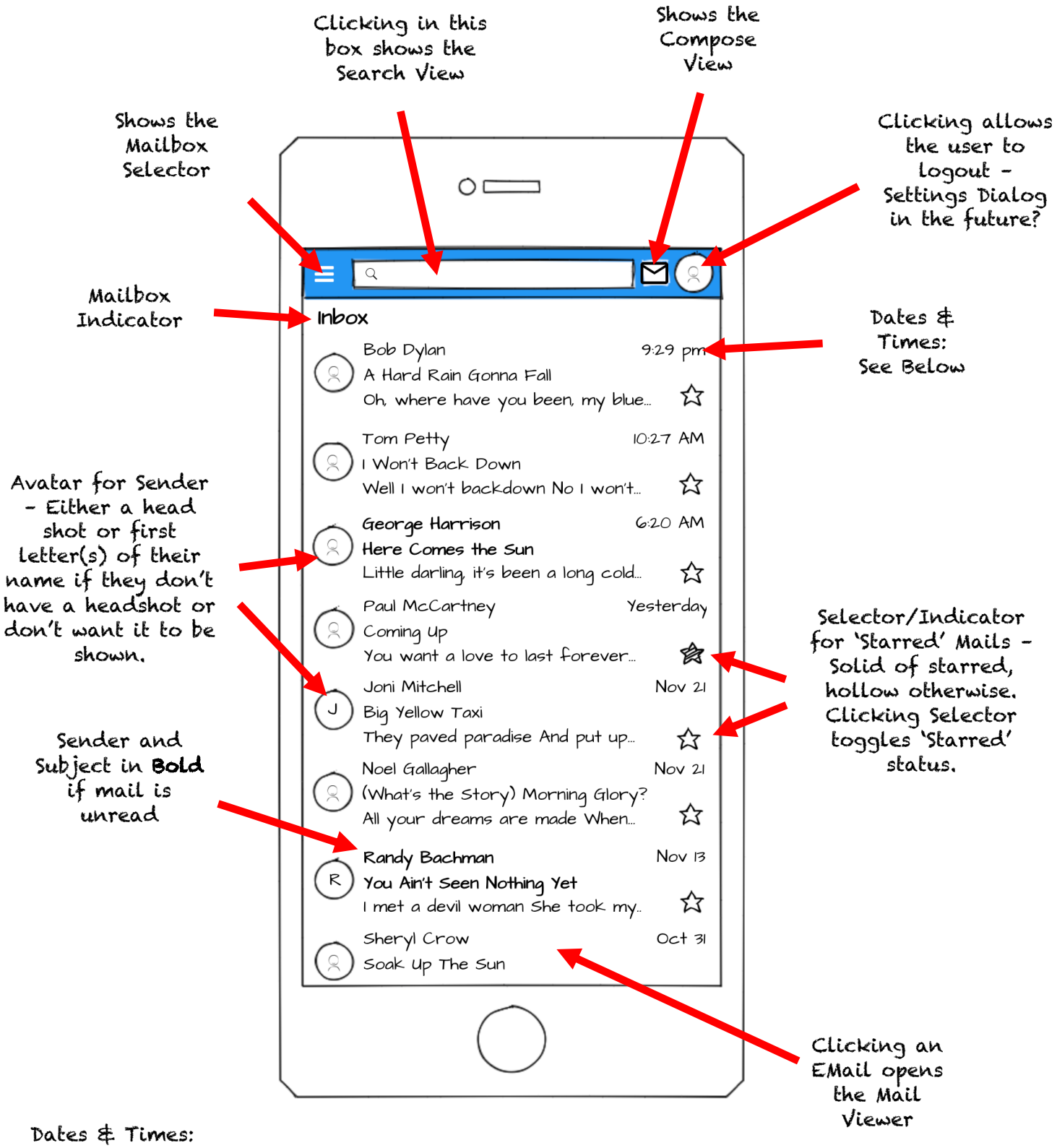
Each is examined in detail on the following pages.

Login Screen

The concept of a user is important when designing Web Applications. There, the logged in user should only be able to see their own e-mails.

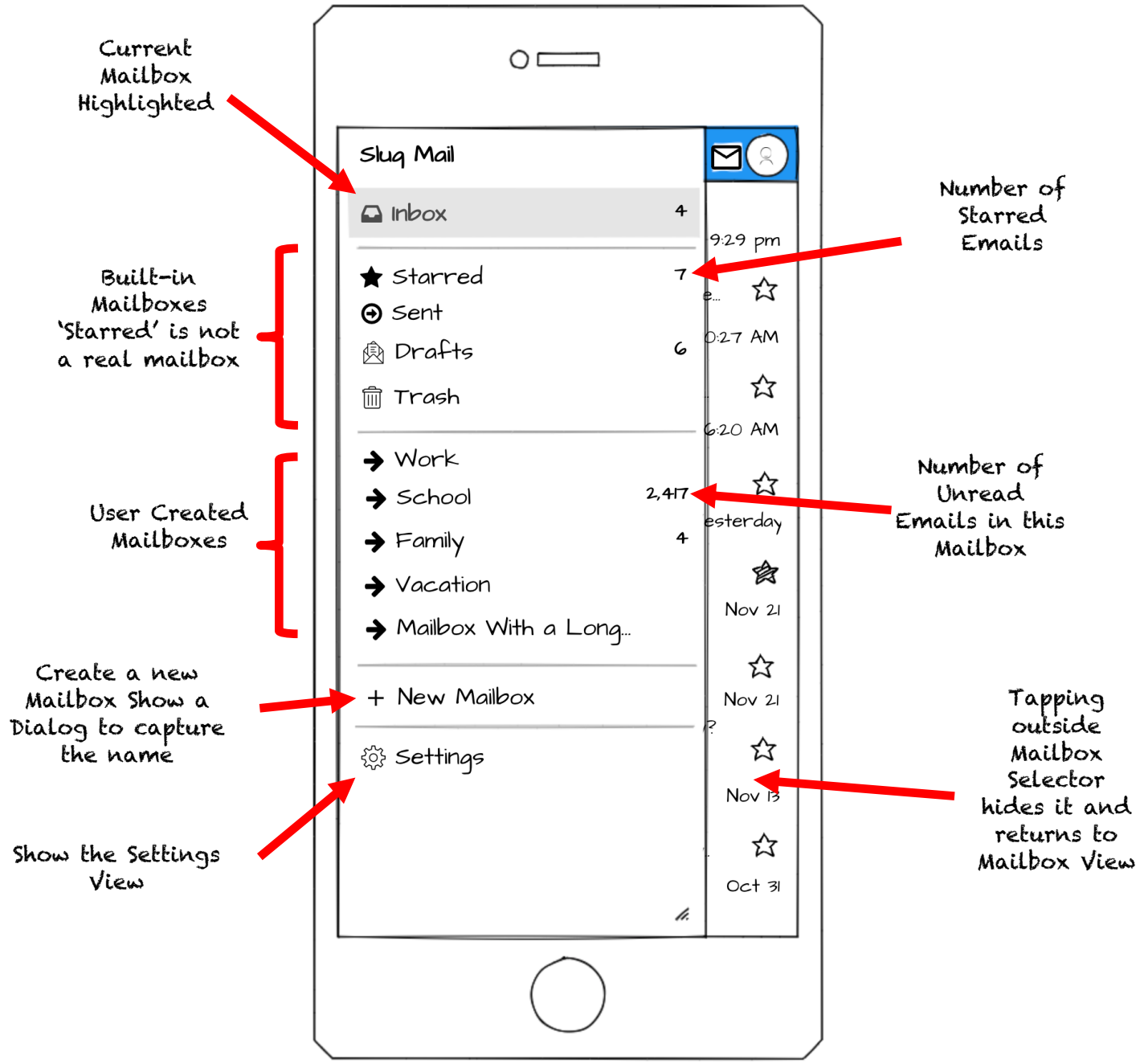


Mobile: Mailbox Viewer

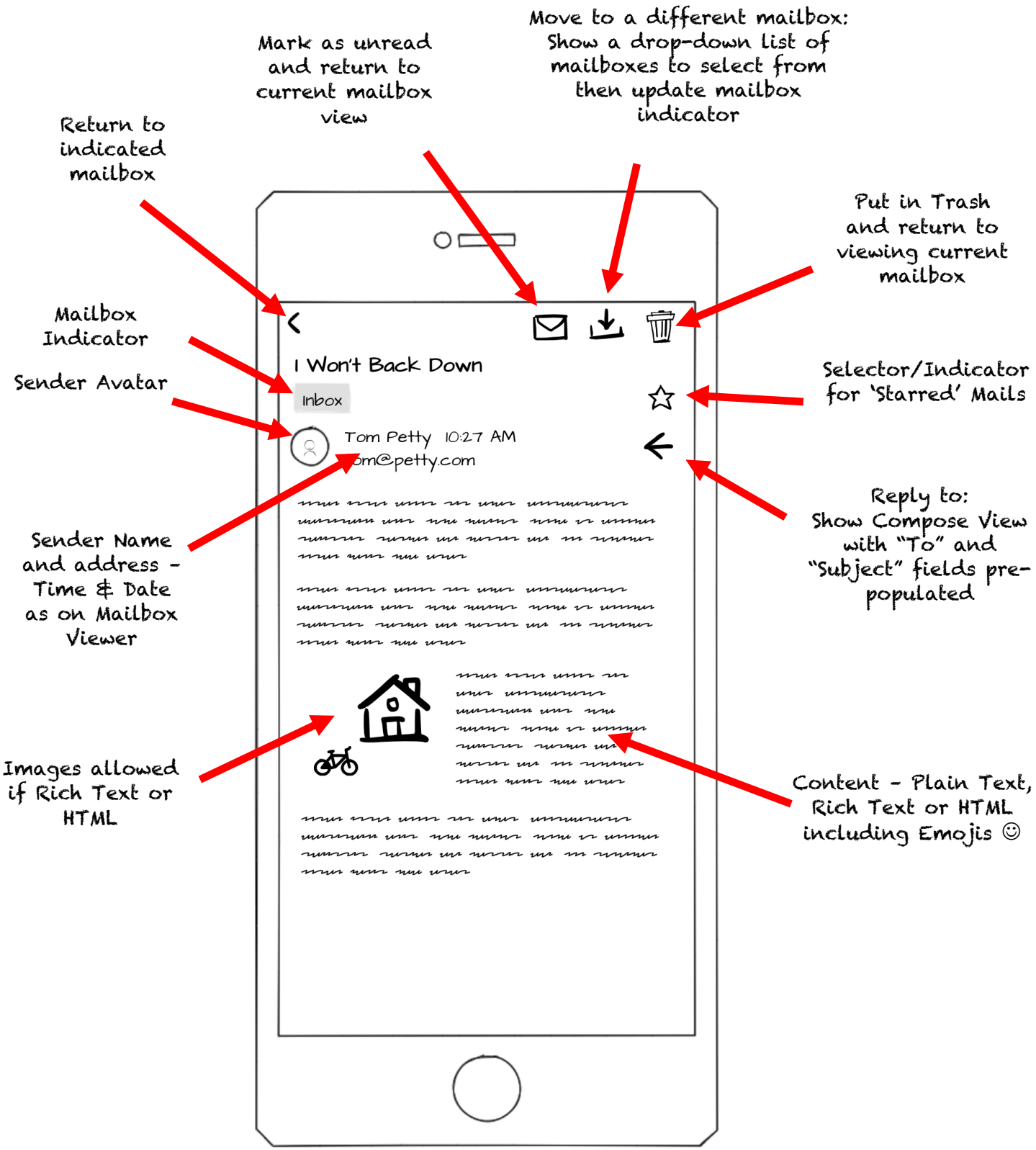


Dates & Times:
- Today => Time
- Yesterday => 'Yesterday'
- Previous 12 months => Month and Day
- > 12 moths agon => Year only

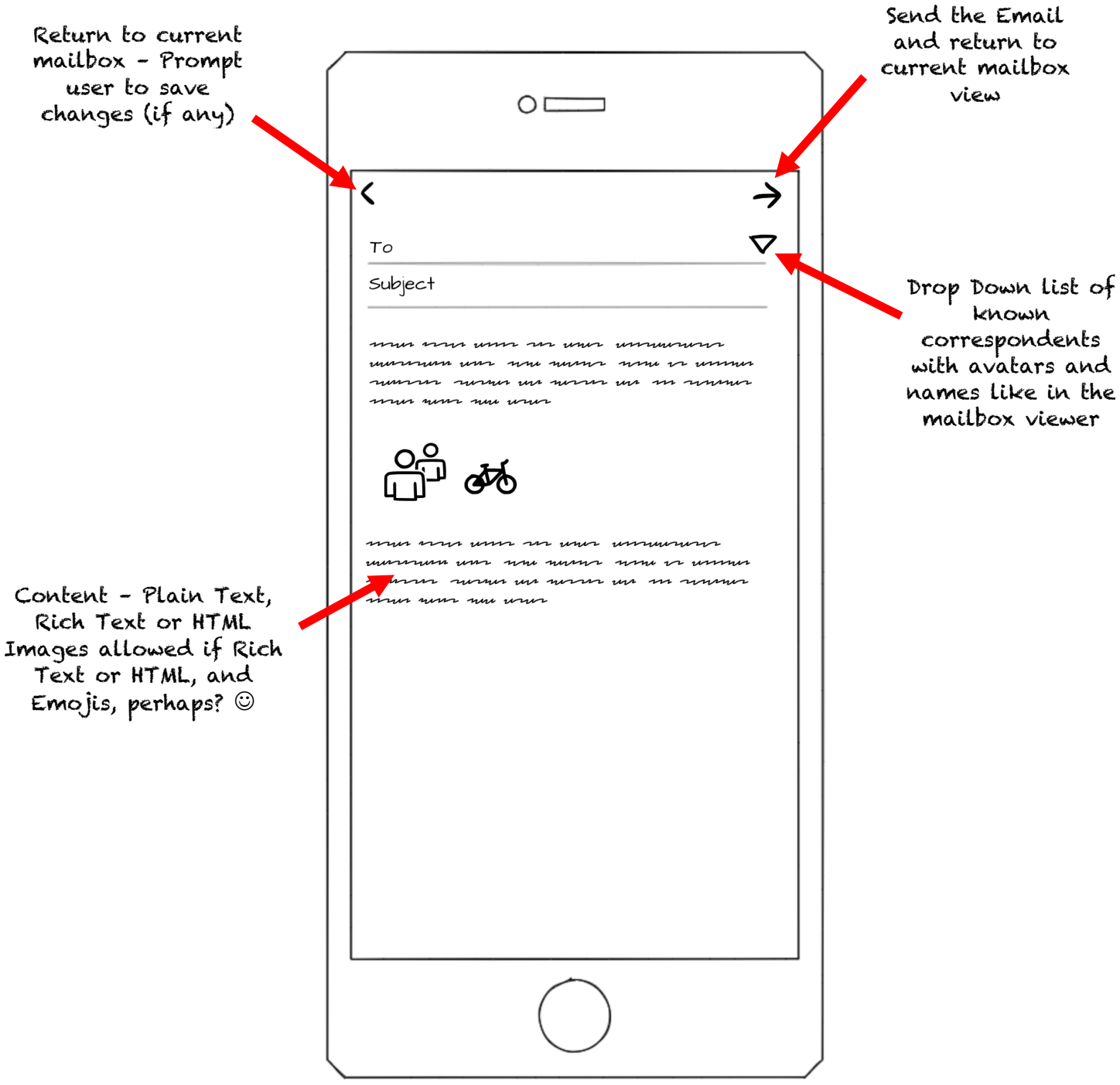
Mobile: Mailbox Selector



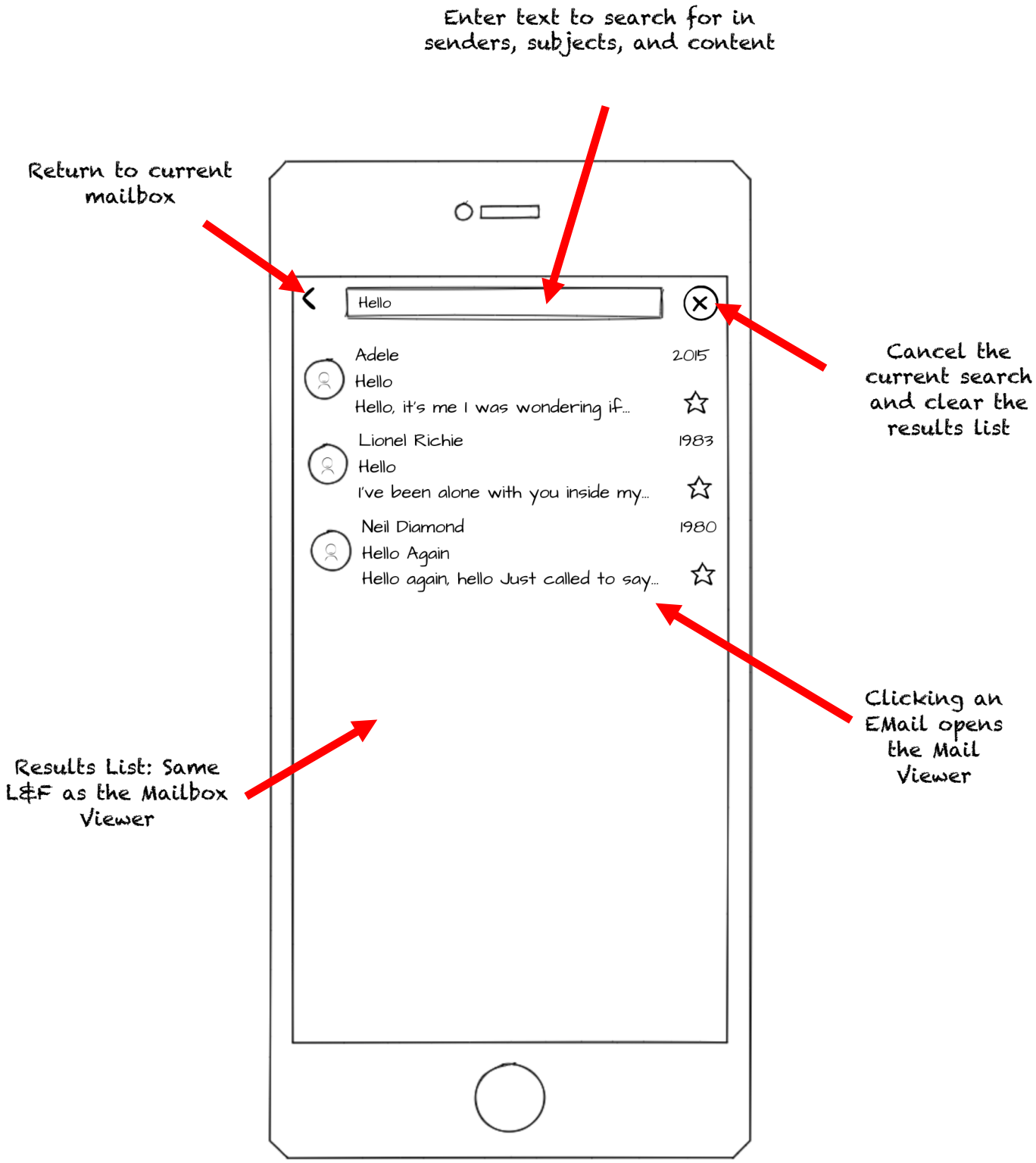
Mobile: Mail Viewer



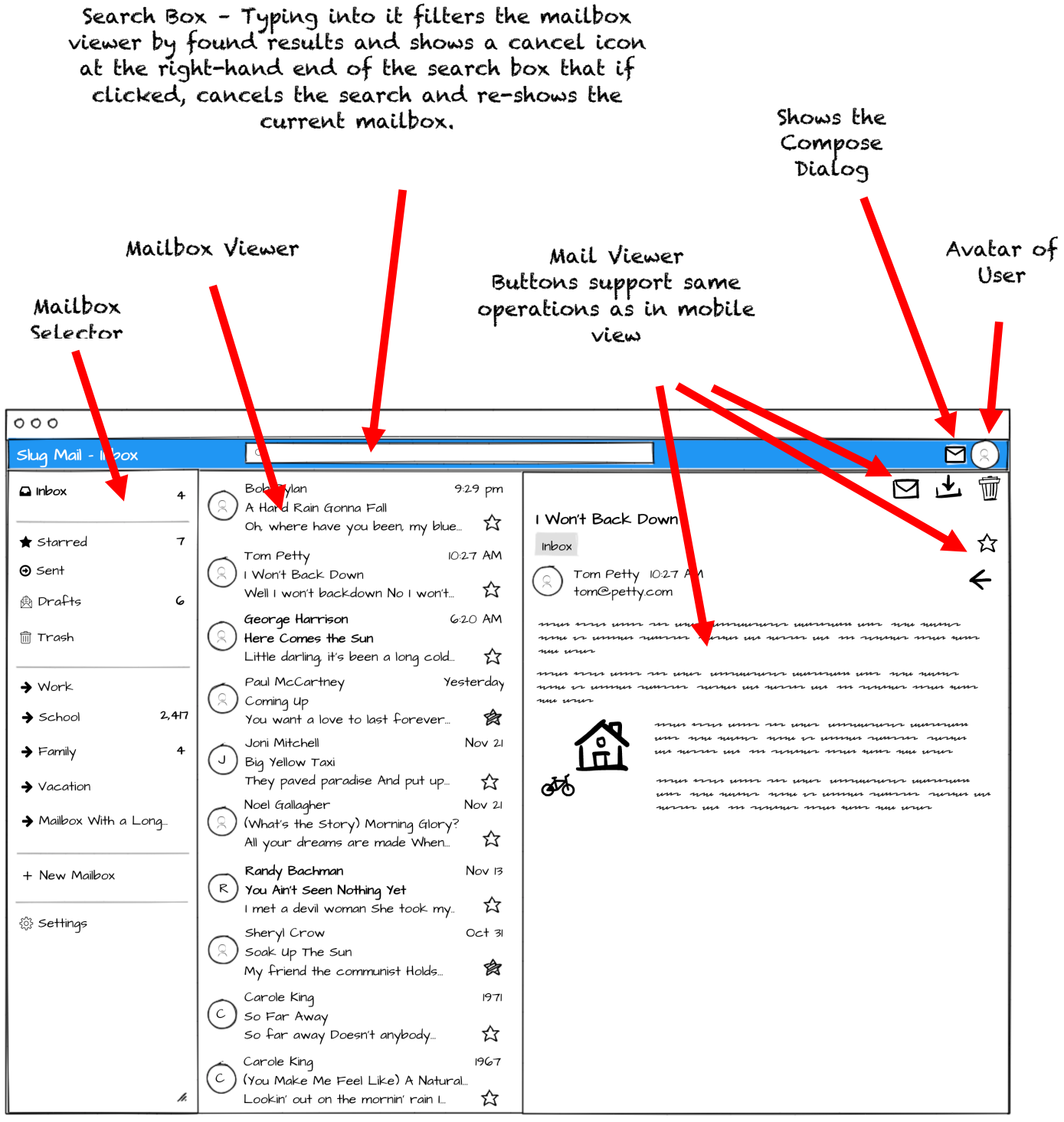
Mobile: Composer



Mobile: Search



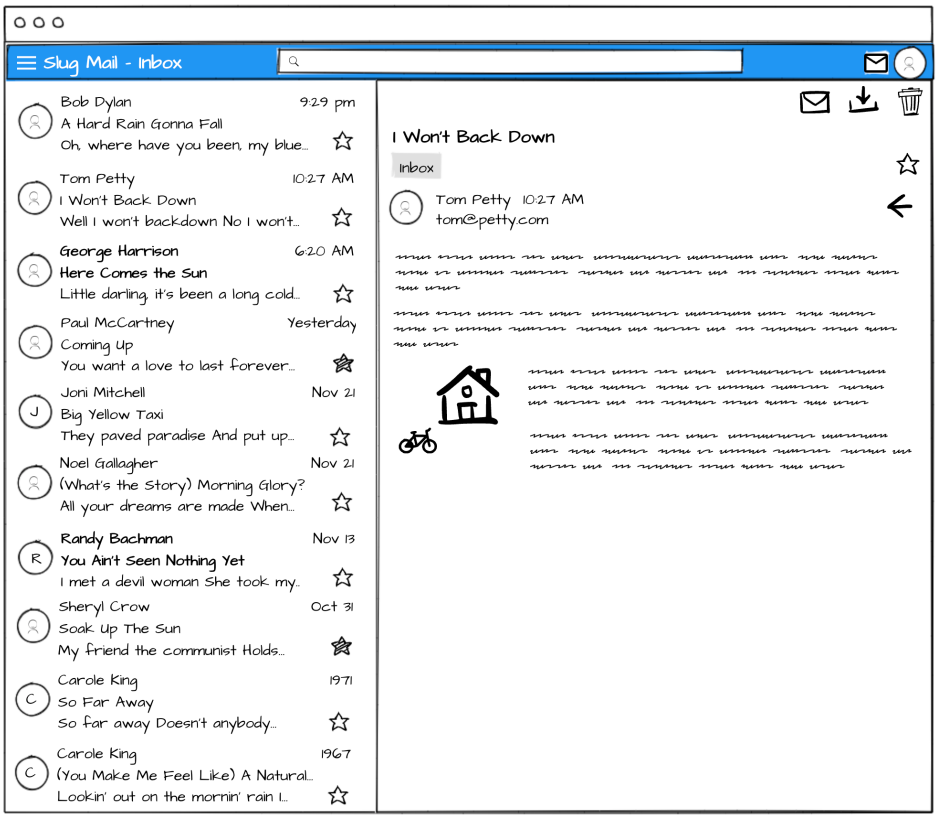
Desktop: Wide View

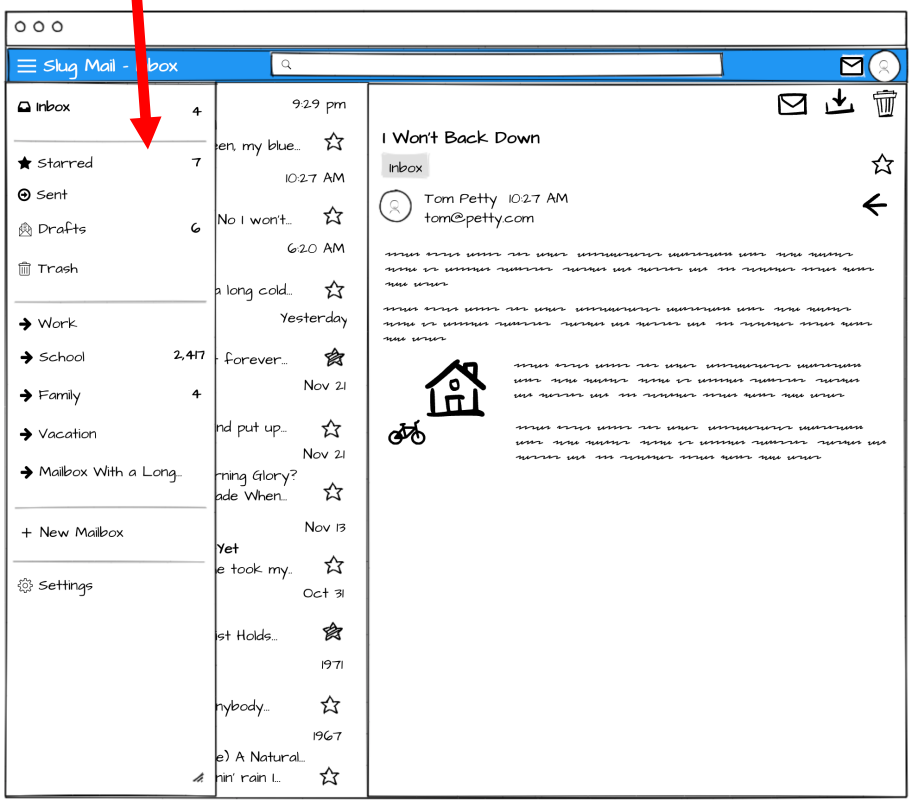


Desktop: Narrow View & Mailbox Selection Popup

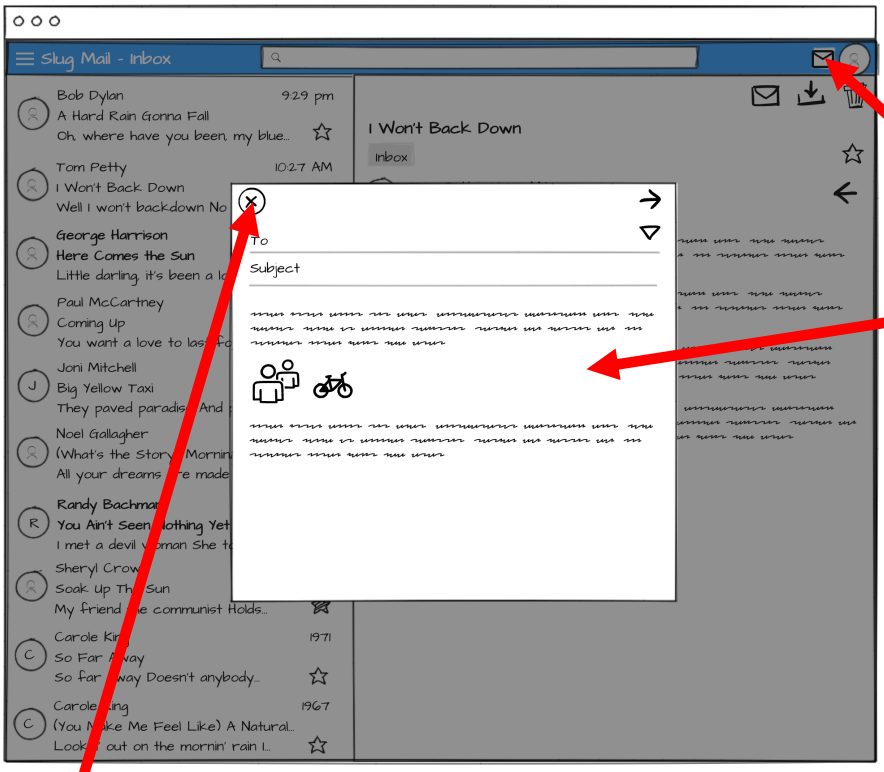
Mailbox Selector is hidden, and Menu Button is shown when browser is too narrow

Clicking the Menu Button shows the Mailbox Selector above the Mailbox Viewer.





Desktop: Compose Dialog



Clicking the Compose Button shows the Compose Dialog - buttons have same operations as in mobile view

Dialog Close Button

Development Guidelines

Whilst you can construct the user interface any way you like, this finished front end must be a Single Page, Material UI, React Application; React Routes may prove useful for the mail reading and composition pages.

Background

Listings in this system can have any properties you deem necessary to complete the assignment.

Note that your API must be constrained by an OpenAPI version 3.0.3 schema. You should endeavor to make this schema as restrictive as possible.

For example, your schema should specify the acceptable format of any e-mail property. If an e-mail property in any other form is presented, your system should reject it simply by performing the validation provided in the starter code.

When you start the backend, use a browser to visit http://localhost:3010/v0/api-docs/ where you can manually test the API as demonstrated in class.

Initially, the only operation available will be:

Which if executed will return the current data and time and the date and time the backend database was initialised.

Database Schema

The supplied database schema can be found in `backend/sql/schema.sql`. It defines a single table '`dummy`'. You will want to define the tables you need in this file.

Optionally, you can also define indexes to be built against your tables in `backend/sql/indexes.sql`.

Resetting Docker

If you run into problems with your dev database, or simply want to re-set it to its initial state, issue the following commands (you will need to be in PowerShell on Windows):

```
$ docker stop $(docker ps -aq)
$ docker rm $(docker ps -aq)
```

What steps should I take to tackle this?

You should base your implementation on your solutions for Assignments 5 and 7 plus the Database Book Example and the Authenticate Books Example.

There is a huge amount of information available on OpenAPI 3.0.3 at https://swagger.io/specification/ and http://spec.openapis.org/oas/v3.0.3. Also consult the Petstore example at https://petstore3.swagger.io/ and make good use of the on-line schema validator at https://editor.swagger.io/.

A good resource for querying JSON stored in PostgreSQL is: https://www.postgresqltutorial.com/postgresql-json/

Many Material-UI Examples and sandboxes for experimentation are available: https://mui.com/

A plausible development schedule

There are any number of ways of approaching this task, but items that need consideration in approximately the order they need considering are:

- Decide on your data entities
  - For example (not an exhaustive list, not necessarily complete lists of attributes)
    - User
      - Name
      - E-Mail address
      - Password hash
    - Mailbox
      - Owner
      - Name
    - Mail
      - Mailbox
      - From
      - To
      - Content
      - Sent
      - Received

- Decide on the end points for your API
  - Login
  - Mailbox
  - Mail

- Create UI Components
  - Basic:
    - Login Dialog / Screen
    - Logout Button
    - Mail List
  - Advanced:
    - Mailbox List
    - Search
    - Compose
    - Etc. etc.

Remember, this is only a guideline. You can develop in any sequence you like.

Most importantly, keep on top of your code coverage. A submission that meets all the Basic Requirements and all the Quality Requirements (see below) and has comprehensive end-to-end tests will score 50% so write tests as you go. A very simple implementation with perfect code coverage is likely to score better than a fancy one with poor code coverage.

How much code will you need to write?

This is a difficult question to answer, but not including your OpenAPI Schema, something in the order of 1,000 to 2,000 lines of JavaScript & JSX might be a reasonable estimate for a well-tested basic implementation.

Grading scheme

The following aspects will be assessed:

1. (10%) **Basic Requirements**

    - All data is stored in PostgreSQL
    - Server exposes an authenticated OpenAPI restricted RESTful interface
    - UI Is written in React and Material UI
    - Users can log in and see a list of their emails

2. (40%) **Advanced Requirement**

    - UI is as specified in the wireframes above - see “handwritten” notes for details
        - The more features you implement, the more marks will be awarded
        - But remember, do not implement features at the expense of code coverage

2. (10%) **Stretch Requirement**

    - End-to-End tests exist for all implemented functionality

3. (10%) **Extreme Requirement**

    - Multiple users can log in, send each other email and have it appear without manual refresh

4. (30%) **Quality Requirements**

    - No linter warnings/errors 5%
    - Mean of Statement, Branch, Function, and Line Code Coverage across frontend and backend
        - 100% 25%
        - >= 99% 15%
        - >= 98% 5%
        - < 98% 0%

5. (-100%) **Did you give credit where credit is due?**

    a. Your submission is found to contain code segments copied from on-line resources and you failed to give clear and unambiguous credit to the original author(s) in your source code (-100%). You will also be subject to the university academic misconduct procedure as stated in the class academic integrity policy.

    b. Your submission is determined to be a copy of a past or present student’s submission (-100%)

    c. Your submission is found to contain code segments copied from on-line resources that you did give a clear an unambiguous credit to in your source code, but the copied code constitutes too significant a percentage of your submission:

        - < 35% copied code No deduction
        - 35% to 50% copied code (-50%)
        - > 50% copied code (-100%)

        Note that code distributed in class does not count against the copied total but must be cited.

Additional Requirement

You are also required to make a short YouTube video of your Web App demonstrating the features you have successfully implemented.

- Create it under your UCSC CruzID Account
- Make it “unlisted” - only those with the link can see it
- No more than three minutes
- Demonstrate all the features you successfully implemented
- Screen capture from your laptop is fine
- Keep it simple - no marks for fancy effects and/or editing

Mandatory User Accounts

You can have as many users as you like in your submission, but you must have these two and they must have at least three mailboxes (Inbox/Sent/Trash) with at least four emails in each:

molly@books.com / mollymember
anna@books.com / annaadmin

You can use their password hashes from The Authenticated Book Example; do NOT store plain text password in the database!

What to submit

Run the following command to create the submission archive:

```
$ npm run zip
```

You must also paste a link to your YouTube video as a comment on the canvas submission.

****** UPLOAD** `Assignment8.Submission.zip` **TO THE CANVAS ASSIGNMENT AND SUBMIT ******

