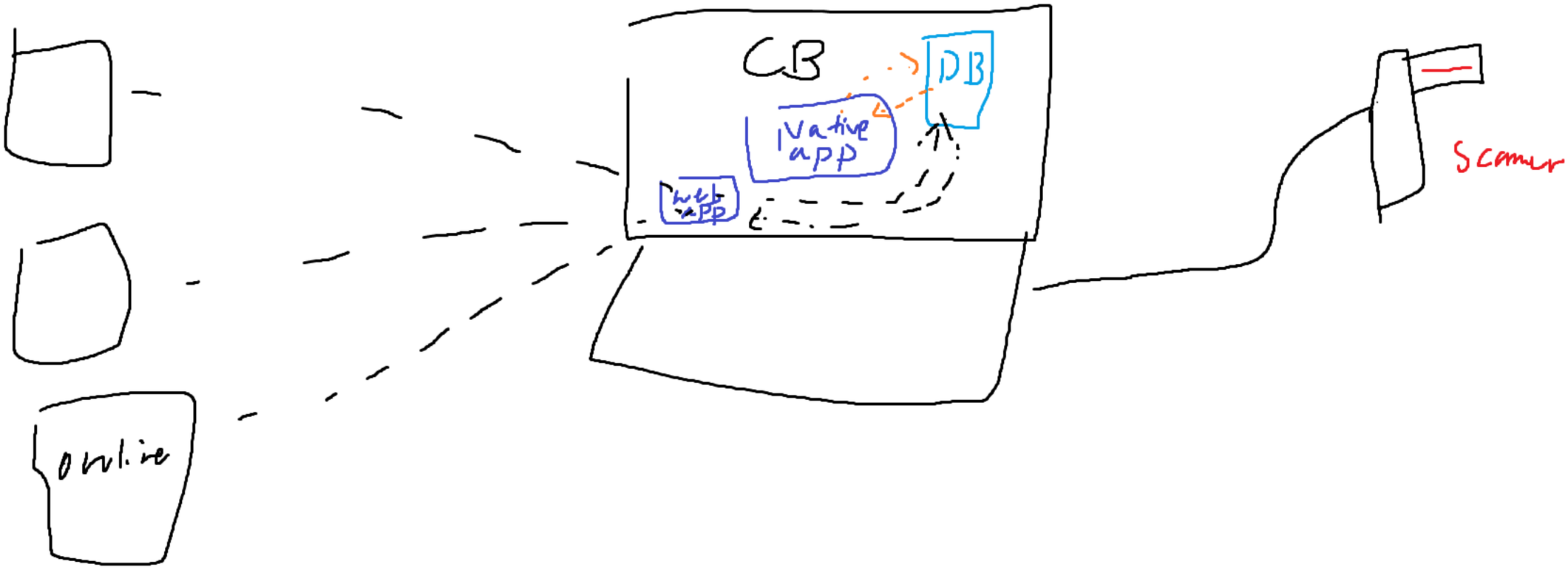


What is this?
This is an overview of what the system implemented would look like.

We would first hijack a chromebook, install linux onto it, then run the app on that chromebook to create the base station. This chromebook would be placed near the entrance of the makerspace with a barcode scanner plugged into it. (Which means that for this project we will need to “borrow” a chromebook 🙂)

With linux installed, it should be easy to run python, and host the server on the chromebook too.

The server has features

- **Archiving**
- **Admin panel**
- **api...**
- **db**

Then we will run the WEB version of the APP locally, exposing it to all ports (which purely means that anyone in the same wifi can connect to the app via wifi)

The barcode scanner is so that students can easily borrow and return items. The items/components will have a barcode with a uuid stuck onto them (this requires us to like completely cleanup and organize the makerspace).

APP Breakdown

Login

This page handles the Authentication of users, it does not support sign ups because accounts are privileged - will add guest-account support that will only allow users to view components

Sign up functionality *is implemented* and can be enabled anytime when necessary.

Home

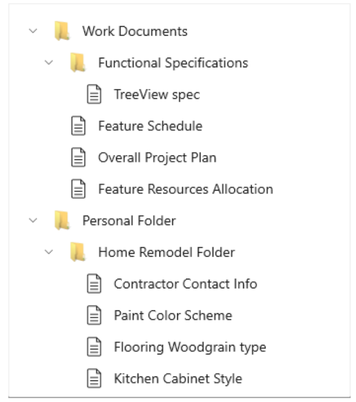
This page displays options for
1. User profile
2. Logout
3. Display and search of historical records of the logged-in user's borrowed items

User profile option displays user data

Search algorithm can be customized *easily* through object inheritance

A feature that I am looking at implementing in the home specifically is that each component cell (the ones that display component data) will have a short description of *where* the component is stored (currently not implemented)

Example Image:



Components

Future features that will be on these pages

- search filter options will include borrow initiation time, return time, and if item is being borrowed (please reference the API screenshot of what filters are possible) (currently only text search is possible)
  - To implement time search I will create a time-selector that will make sure that users select appropriately formatted time. (ISO 800 standard)
- Display filters will also be implemented (such as completed borrow/return transactions and ongoing ones being separated)
- From the video, we can see the items are clickable, with a click, they would then display a detailed view of the components that are not displayed in the normal view

Date - something like this

This page's main feature is to display the Maker Space's entire inventory with a search functionality.
I plan to create another view mode called "map view" which will be something like a tree view

Borrow and Returns

This page will be the page where logged in users can borrow and return items through entering a uuid (most likely through the barcode scanner hardware (if you are on a laptop) or using the user's built in camera (if you are on the app)), searching up the component, or being redirected from the home pages' item's button that allows the users to return that component.

- Another feature that I am going to implement is a feature that enables other users to, underneath a different user's account, return and borrow items.
- How this works is that there will be an additional field in the borrow/return form that *can* toggled true and false (this field would be named "the user under this account is preforming this account" and will be "true" by default)
  - If it is true, we will display two more additional fields that requests for the user-performing-action's credentials (username and password)
  - Else, we use the logged in user's creds

Sign in and Sign out

This page has not been implemented yet, but the main idea is that if the account is an admin account, an additional page in the navbar will appear (this one)

This page will handle (by that i mean send to the db and display data) student sign in and sign outs.

(Kind of like the genesis sign in and sign outs but without the pfp and all the jazz)

Noteworthy mentions
- Can compile to code that can run on ios, android, linux, and windows.
- that's it ig..

Text

Name

Database + API BREAKDOWN

Database

MakerSpaceAPI has the concept of Buildings, Rooms, Storage Units, Bins, and Components (in descending size order).

A component is the smallest unit of measurement and could be anything from a bolt of cloth or a spool of 3D printer filament through to SMD resistors, LED's, or linear actuators.

Components live in "Bins". A "Bin" is a sub-division of a Storage Unit and could be a box, a drawer, or a specific location on a peg board.

Bins live inside "Storage Units" (chests of drawers, toolboxes, peg boards etc), and Storage Units live inside "Rooms".

Finally, Rooms live inside "Buildings".

This may feel like overkill for a small home setup, but if you're working in a Makerspace that has multiple units on a yard or similar then it could be incredibly useful!

Building

```
  name = models.CharField(max_length=200)
  address = models.CharField(max_length=200, null=True, blank=True)
  postcode = models.CharField(max_length=20, null=True, blank=True)
```

Room

```
name = models.CharField(max_length=200)
short_code = models.CharField(max_length=5, null=True, blank=True)
building = models.ForeignKey(Building, on_delete=models.CASCADE)
```

Storage Unit

```
 name = models.CharField(max_length=200)
 short_code = models.CharField(max_length=5, null=True, blank=True)
 room = models.ForeignKey(Room, on_delete=models.CASCADE)
```

Storage Bin

```
 name = models.CharField(max_length=200)
 short_code = models.CharField(max_length=5, null=True, blank=True)
 unit_row = models.CharField(max_length=5, null=True, blank=True)
 unit_column = models.CharField(max_length=5, null=True, blank=True)
 storage_unit = models.ForeignKey(StorageUnit, on_delete=models.CASCADE)
```

Component

```
name = models.CharField(max_length=200)
sku = models.CharField(max_length=100, default="", blank=True)
mpn = models.CharField(max_length=100, default="", blank=True)
upc = models.IntegerField(default=0, blank=True)
storage_bin = models.ManyToManyField(StorageBin)
measurement_unit = models.ForeignKey(
    ComponentMeasurementUnit, on_delete=models.CASCADE
)
qty = models.IntegerField(default=0, validators=[MinValueValidator(0)])
description = models.TextField(default="", blank=True)
```

Component Measurement

```
unit_name = models.CharField(max_length=30)
unit_description = models.TextField(max_length=200, null=True, blank=True)
```

Borrow

```
qty = models.IntegerField(default=0, validators=[MinValueValidator(0)])
person_who_borrowed = models.ForeignKey(User, on_delete=models.CASCADE)
timestamp_check_out = models.DateTimeField()
timestamp_check_in = models.DateTimeField(null=True, blank=True)
borrow_in_progress = models.BooleanField(default=True)
component = models.ForeignKey(Component, on_delete=models.CASCADE)
```

User

```
ss User(AbstractUser):
 user_id = models.PositiveIntegerField(
    default=00000, validators=[MaxValueValidator(999999)], unique=True
 )
```

KEY:

- the letters/word BEFORE the "=" is a field/attribute of that database model
- On the RIGHT, if null=True, that means that it is OK for the attribute to be NONE (which is nothing)
- On the RIGHT, if blank = True, that means that there is no default option, which means the user has to input something if None is not allowed
- On the RIGHT, if default = "something" that means that that's the default value given to the attribute in its creation if the user doesn't specify a specific value to that attribute
- On the RIGHT if unique = true, that means that that attribute can only be held by one instance of the model
- models.ForiegnKey, or models.ManyToManyField means that that attribute is "linked" to another model available in the database that has the specified class.
  - example: *models.ForeignKey(StorageUnit...)*
    - This means that the model has an attribute that is an instances of the class storage unit
-

- The USER class/model IS special because it inhereits attributes from the class Abstract User, or the built in user. This is not a big worry because we won't be using most of the attributes given.
  - tdlr; ^why theres not a lot of fields underneath the User database model

As shown, there is a timestamp attribute.

This is implemented because we want to keep the students held responsible for not return items after a long while.

API - please view video to check out main features such as how the db's datas are returned based on different request types (get, post, patch, etc)

Building

Room

Storage Unit

Storage Bin

Component

Component Measurement

All of these have basically the same thing, except for the search fields

Building

- You can search using the name, address, or postcode

Room

- name only

Storage unit

- Short code and name

Bin

- Short code and name

Component Measurement Unit

- No search fields (because honestly there's no scenarios where we need to search for the measurement unit specifically)

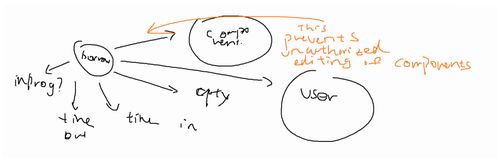
Component:

- Search fields:
  - name, description

Borrow

Borrow View endpoint has the feature called filterset, rather than search_filter.
filterset means I can use *one attribute* to filter the entire search results given.
for example i can search for only objects that have "is_borrowed" set to true.

```
search_fields = [
    "person_who_borrowed__username",
    "component__name",
    "component__description",
    "timestamp_check_out",
]
filterset_fields = [
    "borrow_in_progress",
    "person_who_borrowed__user_id",
    "person_who_borrowed__email",
]
```



Right now, I am planning to create a custom end point for this model so that it takes into account the qty the student/user borrows (qty is indeed a field/attribute of the borrow class).
This endpoint would make sure the student doesn't borrow over the *qty* number in the component field

User

Search filters

- **user id**
- **email**
- **username**

Custom Endpoint 1

- **register**
  - **Registers the user based on the following exmpale fields:**

Give:

```
{
 "username": "bob",
 "user_id": 108503,
 "email": "bob@gmail.com",
 "password":"test123"
}
```

Return: User creation code 200

Checks (if any violations):

```
{
 "username": [
  "A user with that username already exists."
 ],
 "user_id": [
  "user with this user id already exists."
 ]
}
```

Custom Endpoint 2

- **api-user-login**
  - **Logs the user in based on credentials (username and email)**

Give:
Username, email
Response (retusna token that will be used later in authentication):

```
{
"token": "58843bcfea231da973acd770c1fa154527a39253",
"id": 1,
 "username": "eddie",
 "user_id": 0,
 "email": "eddietang2314@gmail.com"
}
```

R/W Permission Overview

Model	IsAdmin	IsAuthenticated user	Guest User
Building	R,W	R	R
Room	R,W	R	R
Storage unit	R,W	R	R
Storage Bin	R,W	R	R
Component	R,W	R	R
Borrow	R,W	W	R
Users	R,W	N/A	N/A