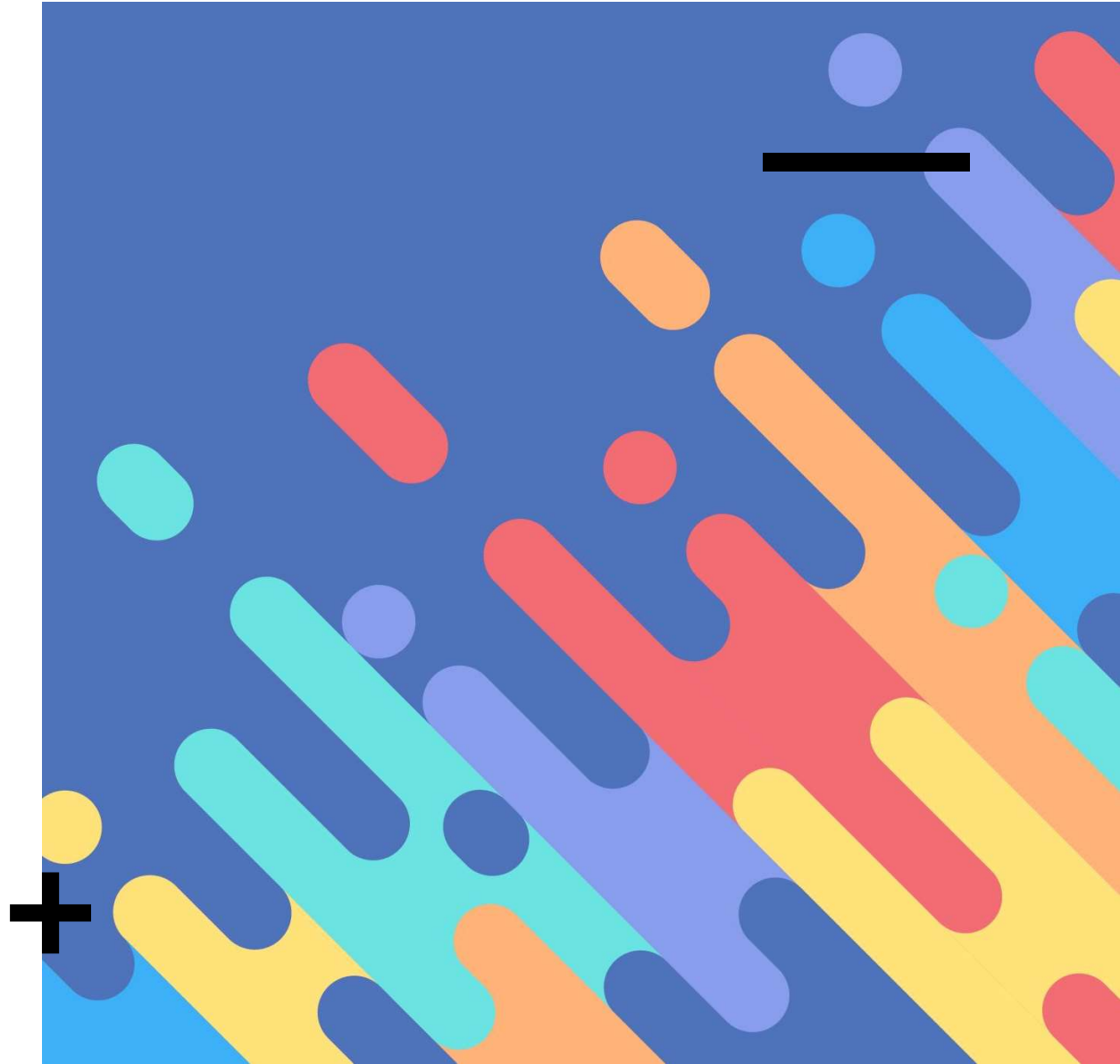


Worksy Technical Seminar

Technology overview



FRONTEND




What is Tailwind?

Tailwind is a utility-first CSS framework to build customize user interface

Allow us to write inline styling which then transformed into corresponding style and written to a CSS file

Benefits of using Tailwind CSS

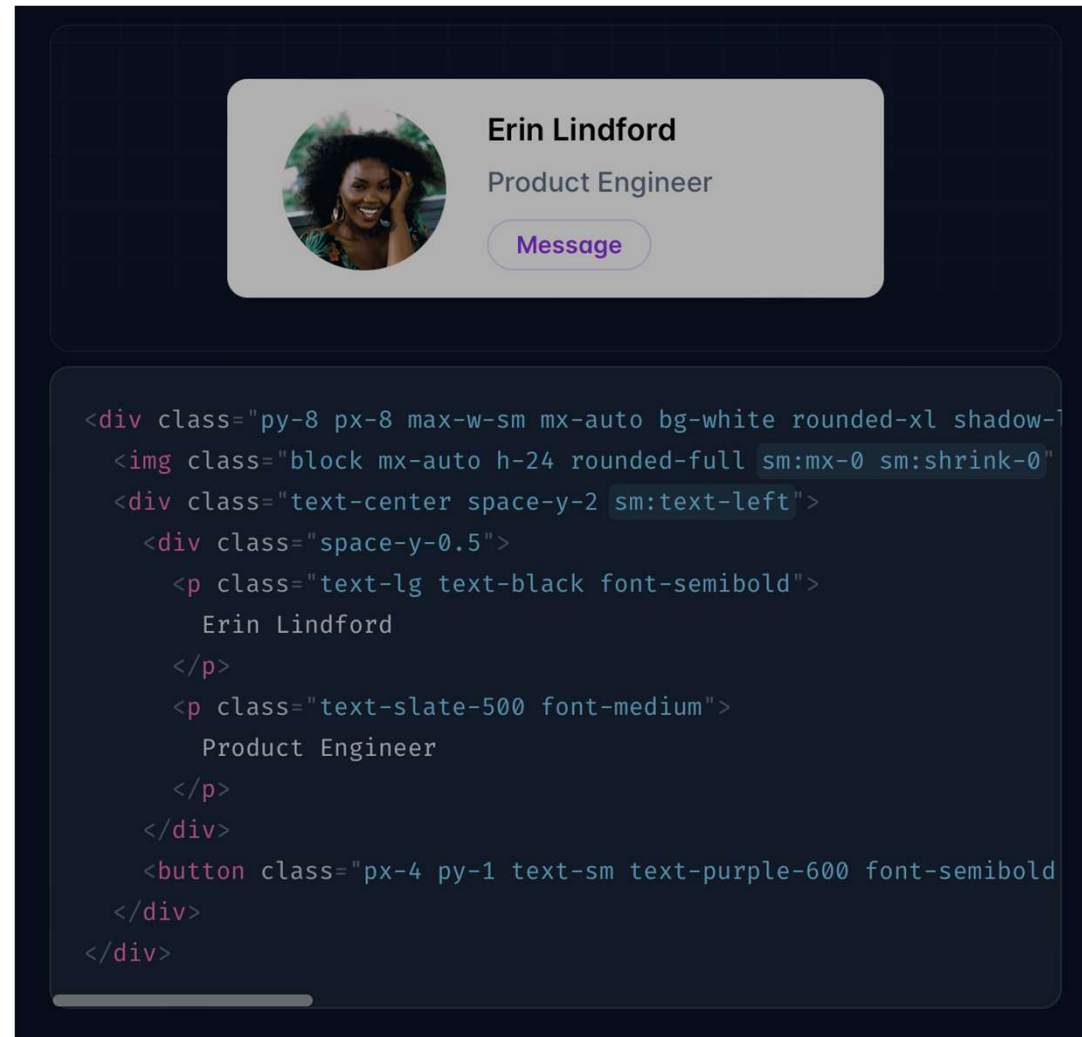


Unlike other CSS frameworks, where they provide prebuilt standard components, and we customize it. This result in very similar styles. Tailwind provides greater flexibility and control over how components appear and feel which create more unique looking UI.

Reduce the development time by using its abundance predefined classes

Optimization for unused CSS (PurgeCSS)

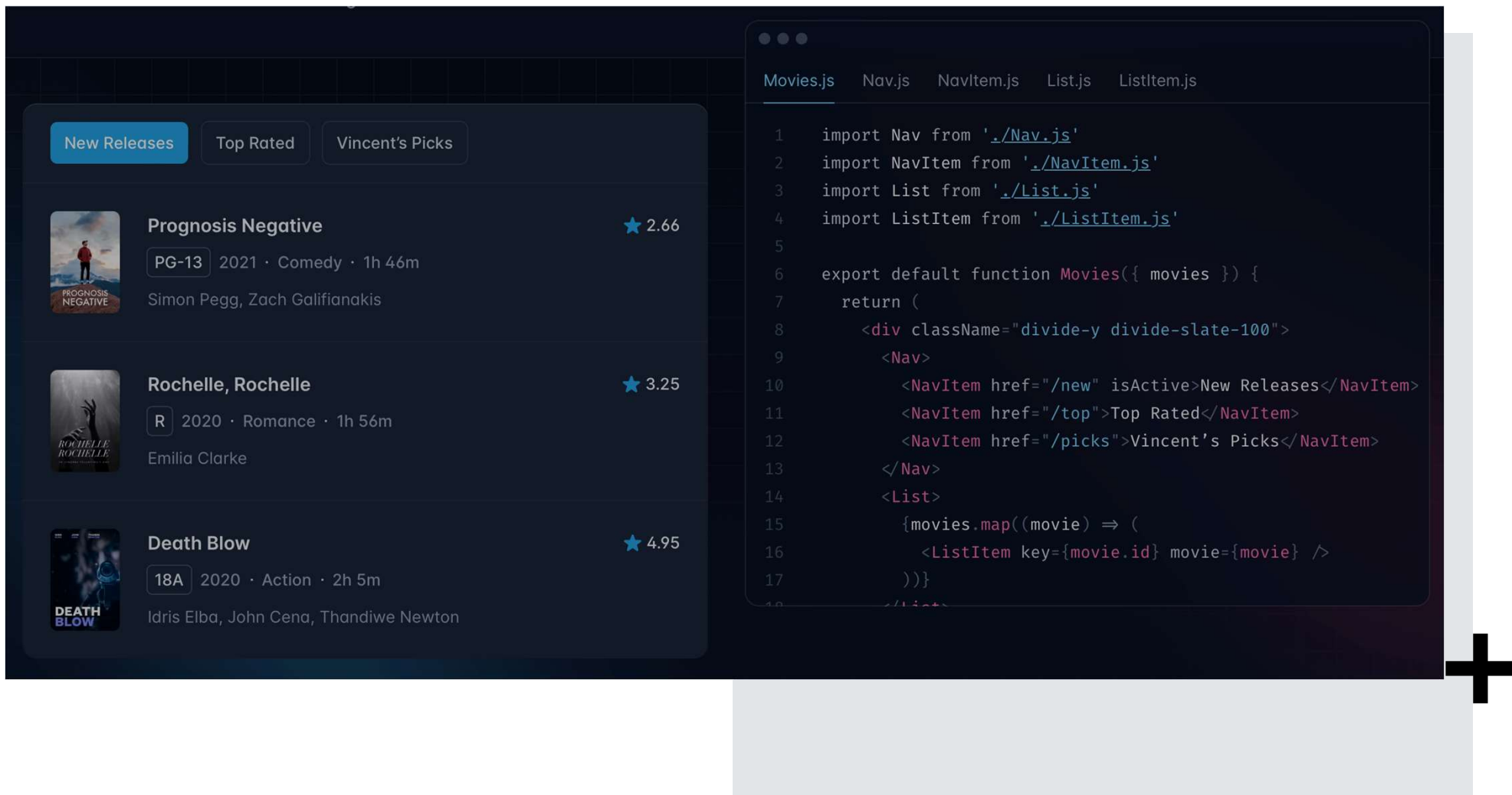
Quick view of Tailwind



Code duplication

- Extract them into component to reuse





For people who don't want to use component framework

Use @apply directive to extract into custom CSS classes.

styles.css

```
1  .btn {
2    @apply text-base font-medium rounded-lg p-3;
3  }
4
5  .btn--primary {
6    @apply bg-sky-500 text-white;
7  }
8
9  .btn--secondary {
10   @apply bg-slate-100 text-slate-900;
11 }
12
```

index.html

```
31   </dd>
32 </div>
33 </dl>
34 <footer class="grid grid-cols-2 gap-x-6">
35   <button class="btn btn--secondary">Decline</button>
36   <button class="btn btn--primary">Accept</button>
37 </footer>
38 </article>
39
```



```

<div class="w-full flex items-center justify-between block p-6 space-x-6">
  <div class="flex-1 truncate">
    <div class="flex items-center space-x-3">
      <h3 class="text-slate-900 text-sm font-medium truncate">Jane Cooper</h3>
      <span class="text-teal-600 bg-teal-600">Admin</span>
    </div>
    <p class="mt-1 text-slate-500 text-sm">Admin</p>
  </div>
  <img alt="Profile picture" class="w-10 h-10 bg-slate-300 rounded-full" />
</div>
<div class="border-t border-slate-200">
  <div class="-mt-px flex">
    <div class="w-0 flex-1 flex border-r border-slate-200">
      <a href="#" class="relative -mr-px flex items-center py-4 text-sm">
        <svg class="w-5 h-5 text-slate-400" />
        <span class="ml-3">Email</span>
      </a>
    </div>
  </div>
</div>

```

bg-transparent

bg-teal-50

bg-teal-100

bg-teal-200

bg-teal-300

bg-teal-400 background-color: #2dd4c1

bg-teal-500

bg-teal-600

bg-teal-700

bg-teal-800

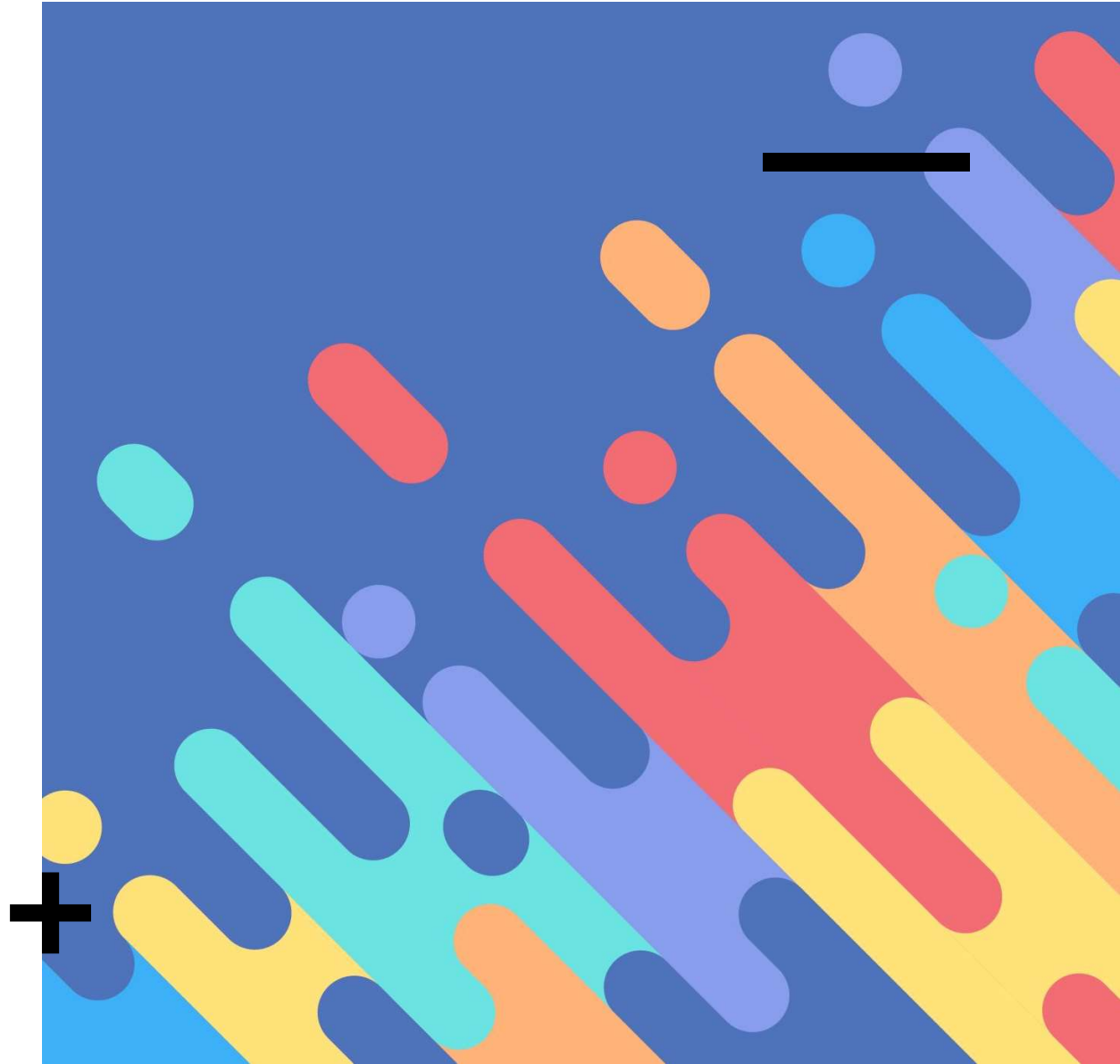
bg-teal-900

bg-top

Integrate well with your IDE choice



DATABASE



DATABASE :

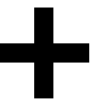


- Documented Oriented Database
- Basic Hierarchy :
 - Clusters
 - Databases
 - Collections
 - Documents / Objects
 - Data/Documents



DATABASE

- ❑ ISSUE : We set up class architecture for all the classes we were going to need beforehand. Some properties in some objects were instances of objects. Nested Documents in MongoDB were too large and took too long to query



DATABASE

- ISSUE : Since fronted uses JavaScript , we can't force types like with Typescript , tedious to do validation on frontend



DATABASE

- SOLUTIONS : USE ORMs (Technically an ODM)!
- ❑ Enforce schema at the application level

Mongoose {  }

+

DATABASE

SOLUTION :

- Using mongoose ,we were able to store references instead of the objects themselves. Since MongoDB indexes on id by default, we are able to access those documents in much faster time

```
chats: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Chats' }],  
services: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Services' }],  
bookmarks: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Services' }],  
orders: [{ type: mongoose.Schema.Types.ObjectId, ref: 'Orders' }],
```

UserModel.js

DATABASES

SOLUTION :

- Using Mongoose , we are able to define strict constraints for our datatypes, we can even match regex and formats of string, so our calls give us errors for invalid data

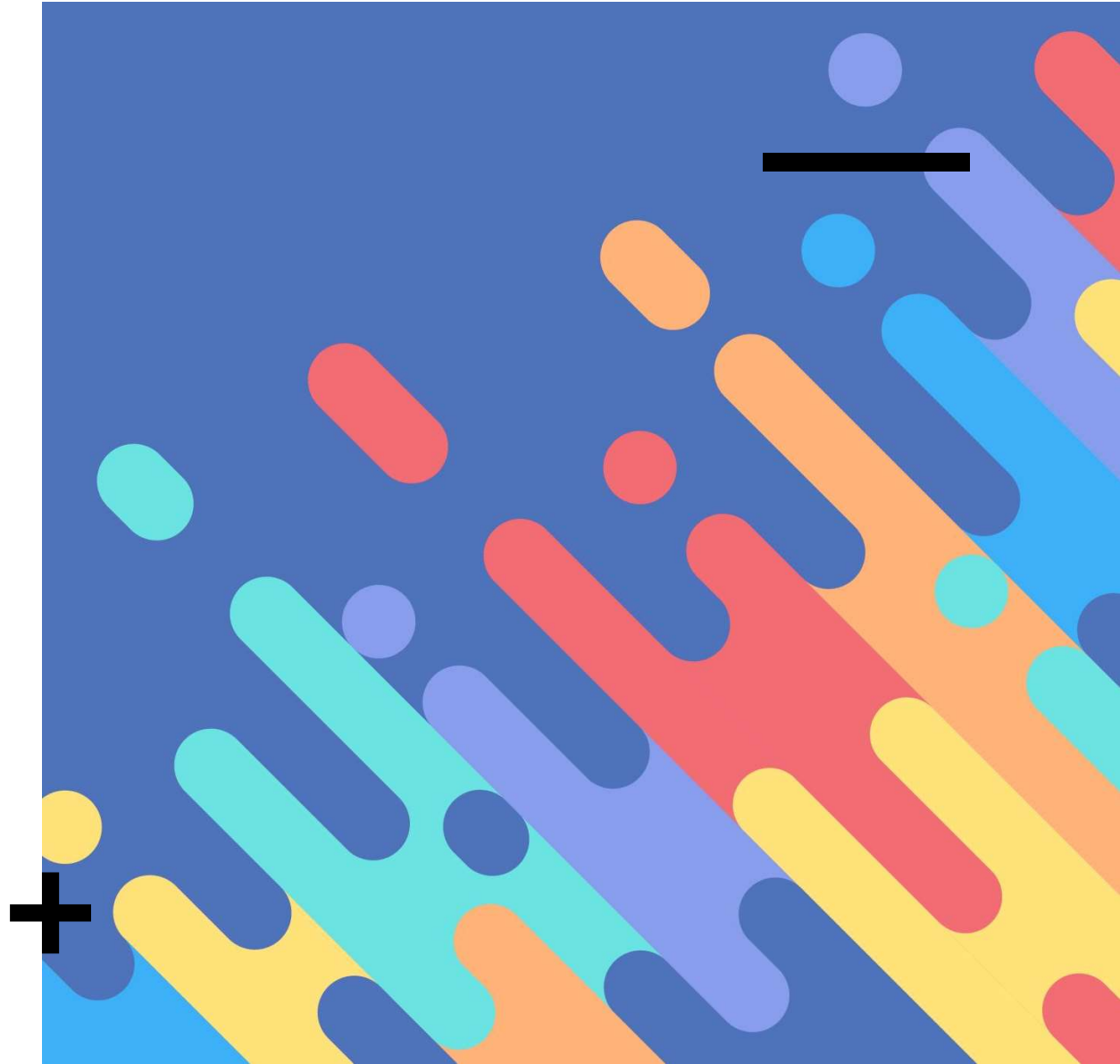
```
const userSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: true,
    match: /^[A-Za-z]+$/,
  },
  lastName: {
    type: String,
    required: true,
    match: /^[A-Za-z]+$/,
  },
});
```

UserModel.js



USE ORMS!

TESTING



TESTING (UNIT) :

- (So far) We have implemented Unit tests for the backend



TESTING (UNIT) :

- Set up an instance of a test database in our MonogDB Cluster

WORKSY'S ORG - 2023-10-03 > WORKSY > DATABASES

 Cluster0

VERSION
6.0.11

REGION
AWS N. Virginia (us-east-1)

Overview

Real Time

Metrics

Collections

Search

Profiler

Performance Advisor

Online Archive

Cmd Line Tools

DATABASES: 3 COLLECTIONS: 7

 VISUALIZE YOUR DATA

 REFRESH

+ Create Database

 Search Namespaces

worksy
worksy_dev
worksy_test

worksy_test

LOGICAL DATA SIZE: 432B STORAGE SIZE: 100KB INDEX SIZE: 156KB TOTAL COLLECTIONS: 3

CREATE COLLECTION

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
Categories	0	0B	0B	32KB	2	64KB	32KB
Services	2	432B	216B	36KB	1	36KB	36KB
Users	0	0B	0B	32KB	2	56KB	28KB

TESTING (UNIT) :

- Using package.json, environment files and variables we specify our test database at runtime

```
▶ Debug
"scripts": {
  "dev": "NODE_ENV=dev nodemon Server.js",
  "test": "NODE_ENV=test nyc --reporter=text mocha --exit",
  "prod": "NODE_ENV=prod nodemon Server.js"
},
```



TESTING (UNIT) :




- Unit Tests , tests follow a logical order so we know where to add and remove tests if we need to

```
USER API TEST
Connected to the database
✓ Create a user (424ms)
✓ Get created user (96ms)
✓ Update user Bio (78ms)
✓ Update user (254ms)
✓ Testing login (223ms)
✓ Testing login for non-existent user (52ms)
✓ Testing login with invalid password (249ms)
✓ Testing get user with invalid token
✓ Testing get user by token (53ms)
✓ Create a service for user (85ms)
✓ Add Service to User (112ms)
✓ Get services by user (121ms)
✓ Delete created service (61ms)
✓ Delete user (56ms)
✓ Update user (53ms)
✓ Get services for deleted user (69ms)
Closed database connection
```

TESTING (UNIT) :

- Test Reporting : Use mocha , nyc/istanbul to define test dependencies, report speed for tests (and API calls) and code coverage in our server Directory as well

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	85.35	77.58	95.83	85.31	
Server	90.9	100	0	90.9	
app.js	90.9	100	0	90.9	13
Server/Config	82.35	100	100	82.35	
db.js	82.35	100	100	82.35	21-22,31

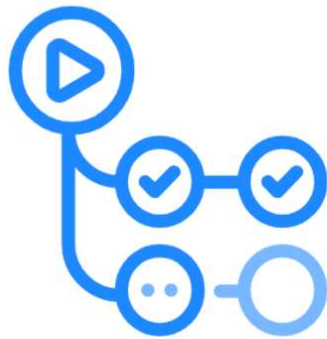


C.I



CI :

- Github actions with yaml files



GitHub Actions



CI :

- Triggered on pull request into main branch

```
on:  
  pull_request:  
    branches:  
      - main  
      - develop
```

CI :

- Sets up node environment

```
steps:
  - name: Checkout code
    uses: actions/checkout@v2

  - name: Set up Node.js
    uses: actions/setup-node@v2
    with:
      node-version: 16
```

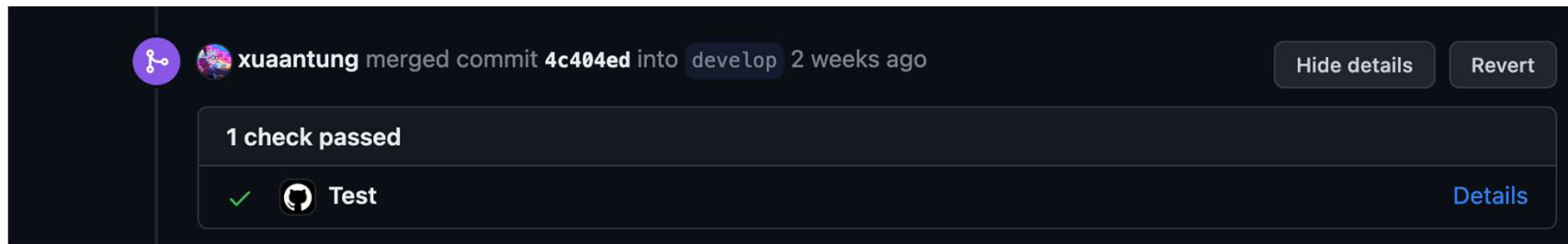
- Installs dependencies and runs tests

```
- name: Install dependencies
  run: npm install
  working-directory: Server

- name: Run Mocha tests
  run: npm run test
  working-directory: Server
```

CI :

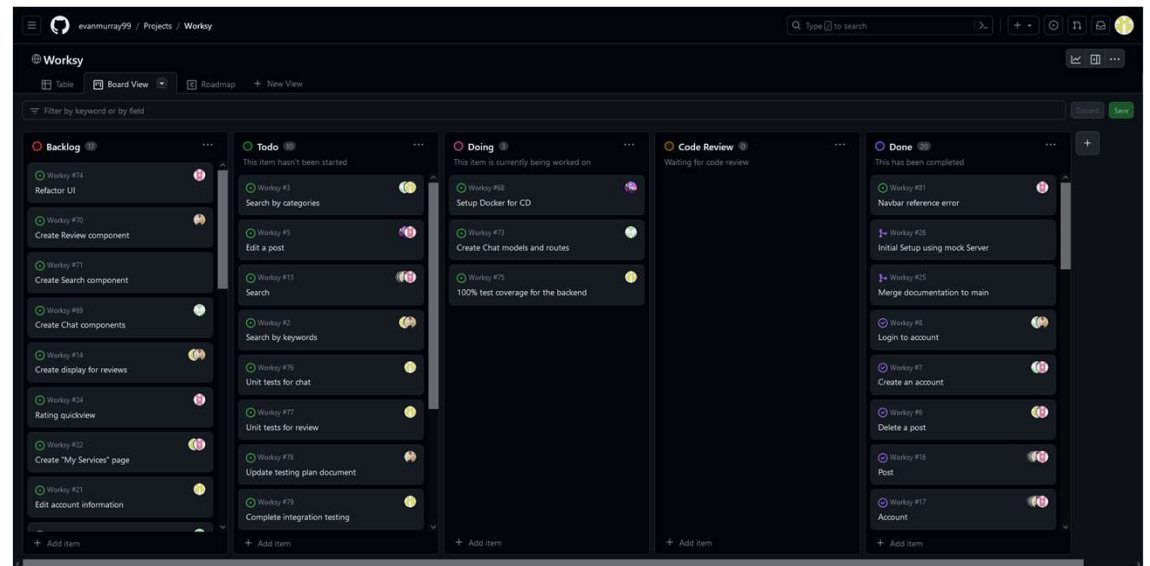
- In Github – Checks require all tests to pass before code review and finally merging into branch



WORKFLOW

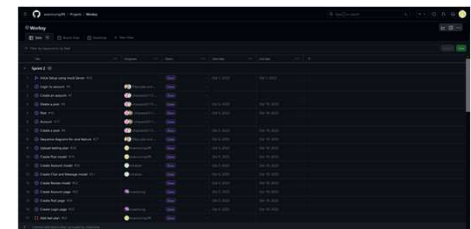
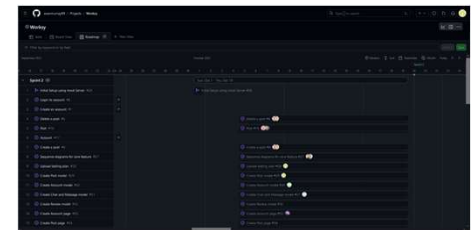
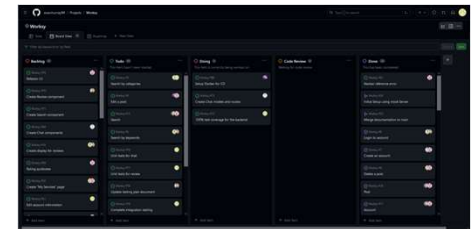


GitHub Projects



What is GitHub Projects?

- A project is an adaptable spreadsheet, task-board, and road map that integrates with your issues and pull requests on GitHub to help you plan and track your work effectively.



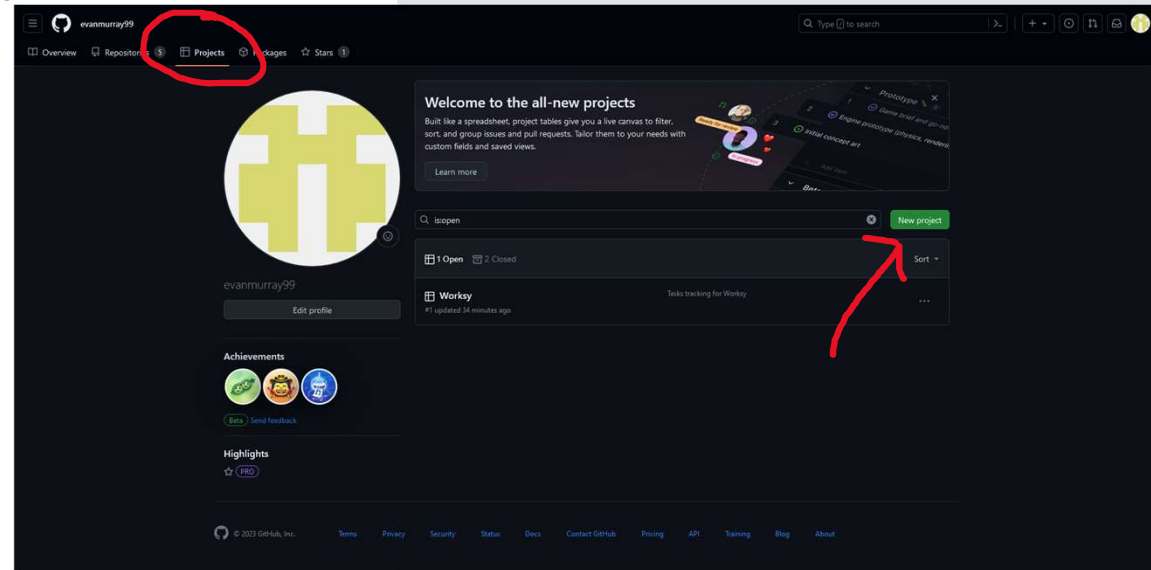
Why use GitHub Projects?

- Rather than enforcing a specific methodology to project management, a project provides flexible features you can customize to your team's needs and processes.
- Your projects are built from the issues and pull requests you add, creating direct references between your project and your work. Information is synced automatically to your project as you make changes, updating your views and charts.
- You can use custom fields to add metadata to your issues, pull requests, and draft issues and build a richer view of item attributes. You're not limited to the built-in metadata (assignee, milestone, labels, etc.) that currently exists for issues and pull requests.



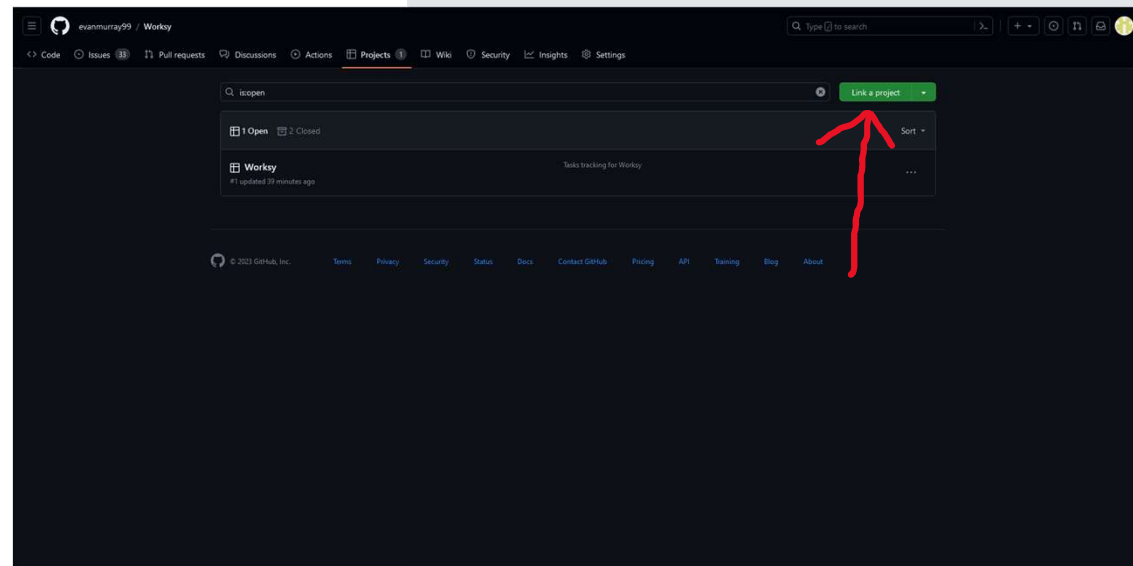
How to use GitHub Projects (step 1)

- Create a new project from your GitHub profile and add status tags to your project.



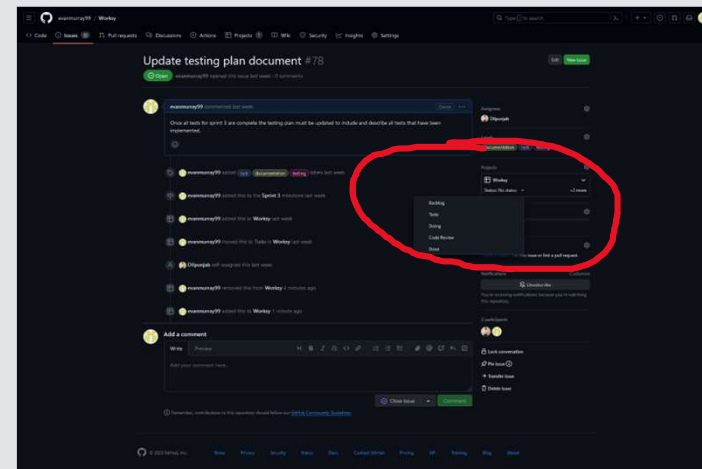
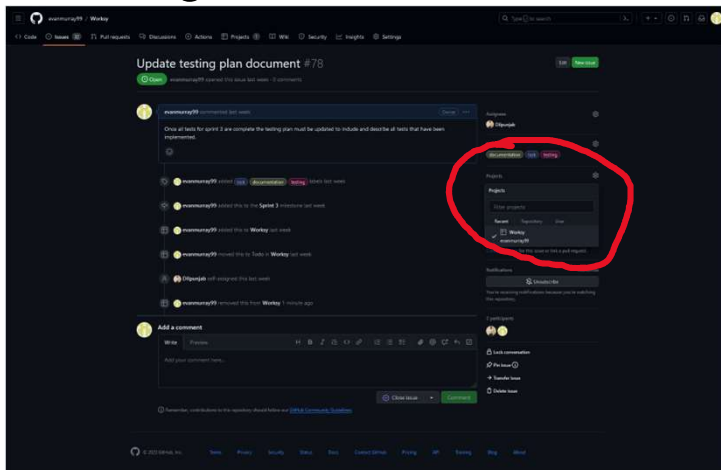
How to use GitHub Projects (step 2)

- Link project to GitHub repo



How to use GitHub Projects (step 3)

- Once issue has been created add it to your project
- Select issue status
- Don't forget to select a milestone for the issue!



GitHub Projects

- It really is as easy as 1-2-3.
- Helps communication between group members and tracking what your peers are working on.
- After initial project is setup all it takes is a couple dropdown menu clicks to add items to your project overview, so why not use it!



Challenges and Solutions



Challenge 1: Communication

- We had issues where the code for the backend didn't meet the frontend expectations.
- At times we also weren't aware of what our teammates were doing leading to repeated work.

Solution: Is to discuss any changes to our plans as well as to have meetings twice a week.



Challenge 2: Merge Conflicts

- In Sprint 2 we ran into many merge conflicts because our tasks were too big resulting in changes to the same files. Additionally, since tasks were too big our pulls were few and far between further increasing the risk of merge conflicts.

Solution: Our solution is to break our project into smaller pieces so that we are doing more frequent pulls.



Challenge 3: Code Review

- Some of the pull requests were not reviewed or were self-merged.
- Pull requests often remained unchecked for multiple days.

Solution: Is to assign someone to review pull requests and code should be reviewed by the end of the following day.



QUESTIONS

+



—