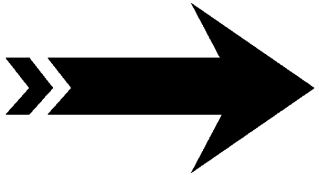

# Code Swap!



*The most entertaining way to learn the fundamentals of collaborative coding*

**Team Code Swap - 3/31/25**

Ashley Bhandari, Yona Voss-Andreae, Wanqi Li, and Viral Rathod

# Project Overview

**Code Swap** is a web application in which teams go head-to-head, competing to solve a simple coding problem the quickest. Players start in waiting room until there are enough people for two teams of two. Then, the game starts and they begin coding. Every 60 seconds, teams switch code editors such that each team builds on the other's work. The goal is to enhance players' understanding of CS 220 concepts and writing clear code.

Tagged Repository  Milestone 1

# Team Members

**Viral Rathod:** Created the test case component, which checks for an output equal to the expected value and returns feedback to the user.

**Ashley Bhandari:** Created the Waiting Room, wherein users can edit their team name and invite their friends via a QR code.

**Wanqi Li:** Created a markdown system that automatically applies custom styling when corresponding tags are being used.

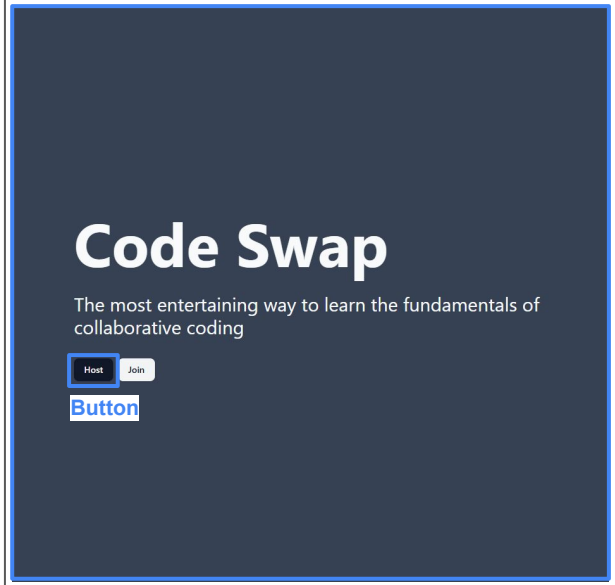
**Yona Voss-Andreae:** Created the Landing page and Challenge page (which contains the challenge question and a code editor).

# Software Architecture Overview

## Landing

Click "Host"

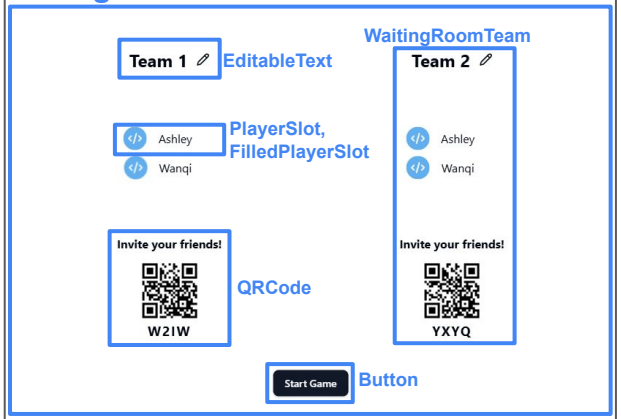
### Billboard




## Waiting Room

Click "Start Game"

### DialogBox



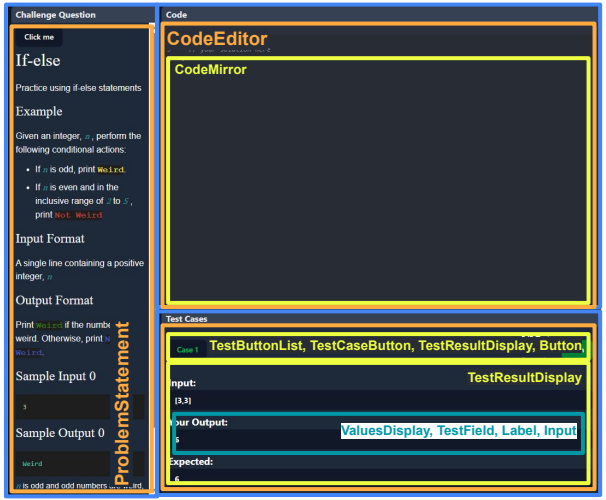

DialogBox's **isOpen** state controls whether the dialog is open; setting the state is exposed to parent (WaitingRoom)

WaitingRoomTeam's **player1** and **player2** states control which players are displayed on the team

## Challenge

Finish game

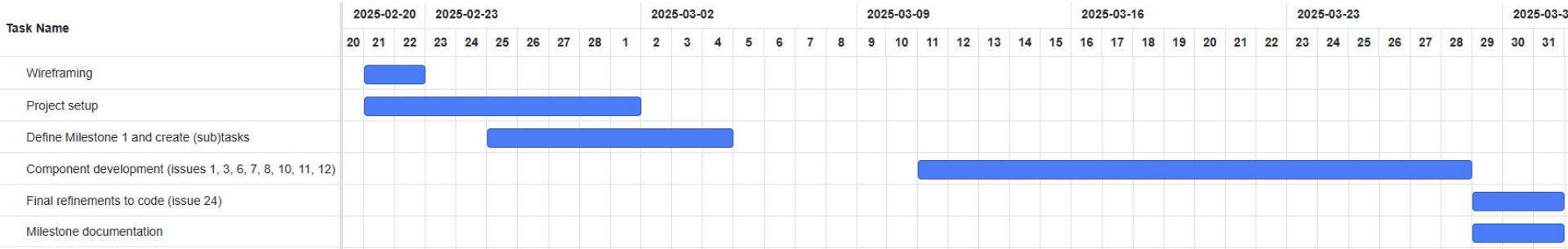
### TitledContainer




CodeEditor's **code** state controls the code editor's text value

TetsParent's **activeTab**, **testResult**, **testsRun**, and **loading** states control which test is viewed, how many tests passed/failed, and the loading indicator

# Historical Development Timeline



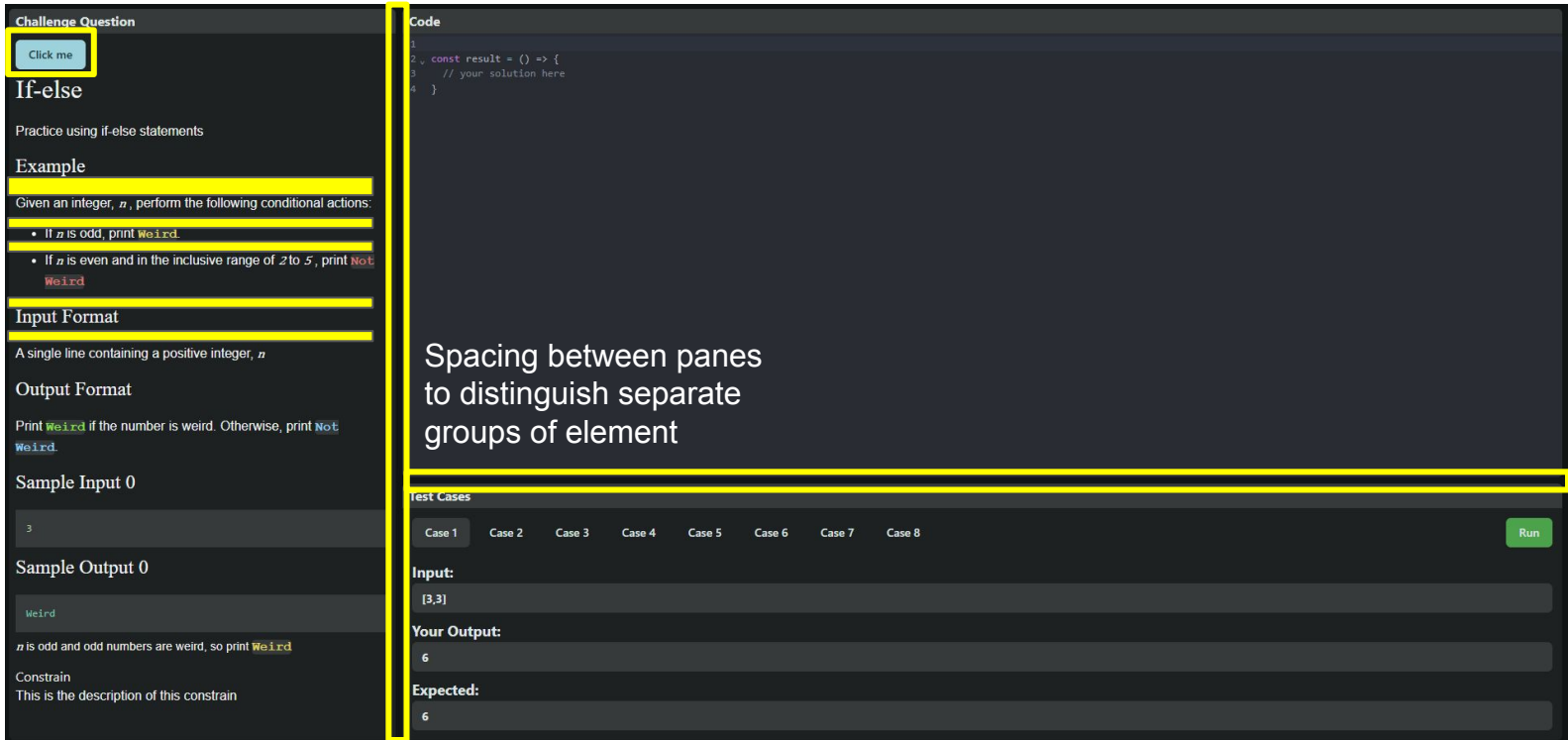
# Design and Styling Guidelines [Document]

UI elements have padding to make its content clearer. Buttons are properly labeled.

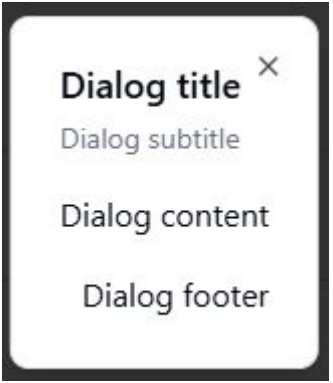
Margins separate different UI elements.

Font styles are consistent throughout the application. rem is used for font sizing.

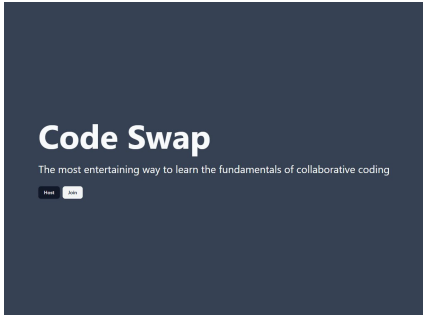
Colors are part of a cohesive overall color scheme. Text and background colors have accessible contrast.



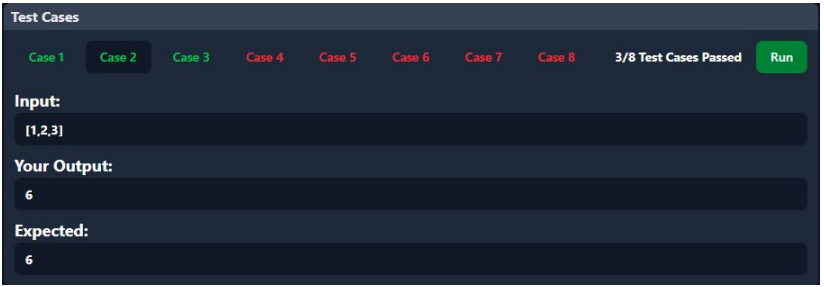
# Component Documentation




**DialogBox** is a dialog with an optional trigger, subtitle, and footer; as well as options to start up open, disable being closed by the user, and hide the title and subtitle. The parent may use a ref to forcefully close the dialog




**Billboard** that takes up the entire screen to aesthetically show information about the application




**TestCases** compare output to an expected value and returns feedback to the user


**WaitingRoomTeam** allows users in the Waiting Room to edit their team name and invite their friends to join

# Performance Considerations

- Using immutable data structures by treating state as read-only and only updating it through the state setting function
- Minimize unnecessary re-renders by splitting components up and only coupling necessary dependencies
- Using React Fragments over unnecessary HTML element wrappers
- Using arrow functions over inline functions for props
- Avoid using index as a list key

[Performance Checklist]

```
<>
  <Button
    onClick={() => navigate('challenge')}
    className="text-on-surface bg-surface hover:bg-surface/80"
  >
    Host
  </Button>
  <Button className="text-surface bg-on-surface hover:bg-on-surface/80">
    Join
  </Button>
</>
```

```
{testCases.map((tc) => {
  const isPassed = tc.output === tc.expected;
  return (
    <TestCaseButton
      key={tc.id}
```

Local metrics

Largest Contentful Paint (LCP)

0.92 s

Your local LCP value of 0.92 s is good.

LCP element h1.text-8xl.font-bold

Cumulative Layout Shift (CLS)

0

Your local CLS value of 0 is good.

Interaction to Next Paint (INP)

40 ms

Your local INP value of 40 ms is good.

INP interaction pointer

# Individual Team Member Contributions

# Assigned Work Summary - Viral Rathod

- Issue #3: Create a test case component: Commits, PR
  - Created a test case component that checks equality between output and input from mock data
  - Added value display component to modularize the “input, output, and expected” values
  - Added case button component to allow for as many test case buttons without repetitive code
- Issue #5: Add Functionality to the test case component: Issue
  - Moved to milestone 2, commented on issue

# Test Cases Component

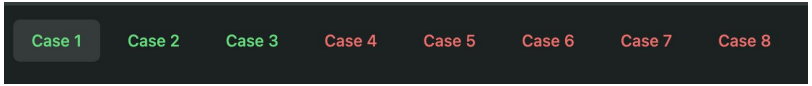
Case buttons at top change to red or green based on whether test case passed or failed. Clicking them changes values of the input and expected fields to show different test cases.

Component shows how many cases were passed in total

Run button has a loading indicator when clicked, and case button test returns to white text while tests run.

# Code & UI Explanation - Viral Rathod

```typescript
import { Button } from '../ui/Button';

interface TestCaseButtonProps {
  id: number;
  isActive: boolean;
  isPassed: boolean;
  testsRun: boolean;
  onClick: () => void;
}

export const TestCaseButton = ({
  id,
  isActive,
  isPassed,
  testsRun,
  onClick,
}: TestCaseButtonProps) => {
  return (
    <Button
      key={id}
      className={`default bg-surface-bright hover:bg-surface-bright
        active:bg-surface-bright/80 shadow-none ${
        isActive ? '' : 'bg-transparent'
      } ${testsRun ? (isPassed ? 'text-green-400' : 'text-red-400') : ''}`}
      onClick={onClick}
    >
      Case {id}
    </Button>
  );
};
```




**TestCaseButton.tsx -** a button component that is rendered at the top of the test case component through TestButtonList.tsx. These are rendered at the top of the test case component, allowing for the user to swap between test cases quickly. The button background changes based on whether it is selected or not, which makes it easy to see which test case is currently being shown.

A challenge I faced was getting the button to reset to white text while the tests were running. They would already be white before any cases were run but they would stay set to a color even while running the tests. I fixed this by passing a prop called testsRun which would keep the state of the whether tests were run or not in the parent component, and this would be changed with the press of the run button. Passing this prop to the test case buttons would now change the text to white while the tests would run.

# Component Hierarchy & Interaction - Viral Rathod

- The test case buttons adhere to the style guidelines since they follow the same font and colors as the rest of the challenge view. They also have appropriate spacing against each other as well as the edges of the test case component. The buttons also all have appropriate feedback by highlighting the background whenever one is selected.
- These buttons appear within the test case component, which is shown on the last page of the application (after the waiting room). The test case component sits below the code editor, and to the right of the problem statement component. The buttons themselves appear at the top of the test case component, above the input and output values, and to the left of the run button.
- These buttons and the test case component in general interact with the rest of the UI by sitting to the bottom right of the challenge view. This allows for the user to keep their focus on the code editor, and to use the test cases when they are finished with writing their code. It is a streamlined process from top to bottom.

# Challenges & Insights - Viral Rathod

- An obstacle I faced was making my code much more readable. I had most of my early code within one tsx file, which meant that the test case component wasn't componentized.
  a. I was able to fix this by making much smaller components to incorporate into my parent component, making the test cases modular and the code for it much more readable and easier to work with. From this I learned the importance of componentization and modularity.
- A key takeaway from this collaborative environment was working efficiently with issues and PR's in github. We had a good amount of documentation for our issues and our PR's were reviewed by at least one person, which was a great learning experience for using version control efficiently.

# Future Improvements & Next Steps - Viral Rathod

- For the next milestone I think that we need to work on actually handling test cases by testing the user’s code on the back-end and checking it’s output against our own values. Right now our mock data can only do so much in terms of testing.
- The top of the test case component doesn’t work well when the user makes the screen smaller. This could be optimized in the future to allow for better compatibility.

# Assigned Work Summary - Ashley Bhandari

Issue #11: Create the Waiting Room dialog: Commits, PR

- Created a customizable, responsive dialog component
- Created a text component that turns into an input field when a button is clicked
- Created component that displays a user's name and profile picture

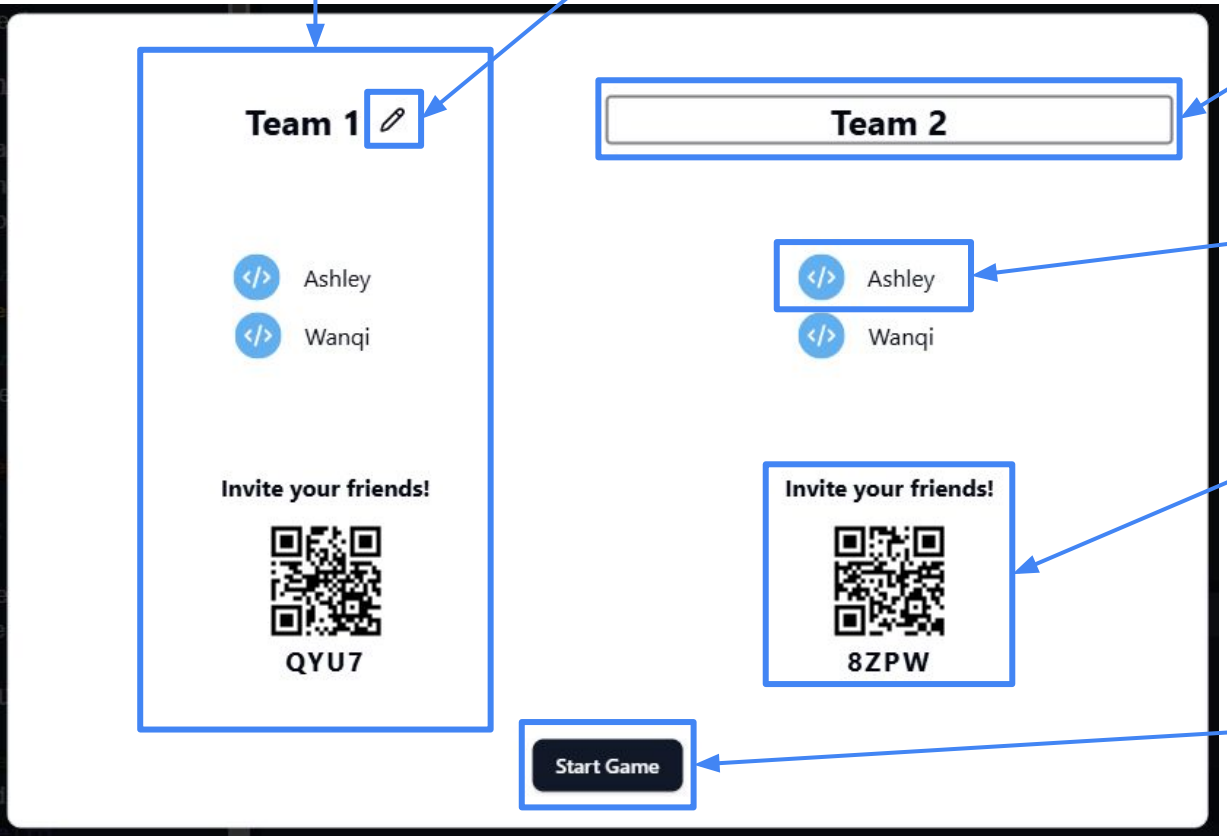
Issue #12: Create a QR component: Commits, PR

- Created component that displays a string and a QR code generated from that string that links to the application

# Waiting Room

Reusable so it can be used for both teams without duplication

Clicking this button allows you to edit the team name




Clicking somewhere else, ENTER, or TAB saves your changes. ESC undoes them.

Displays the name and profile picture supplied to it

Generates a random string and a QR code linking to localhost:5173/{random_string}

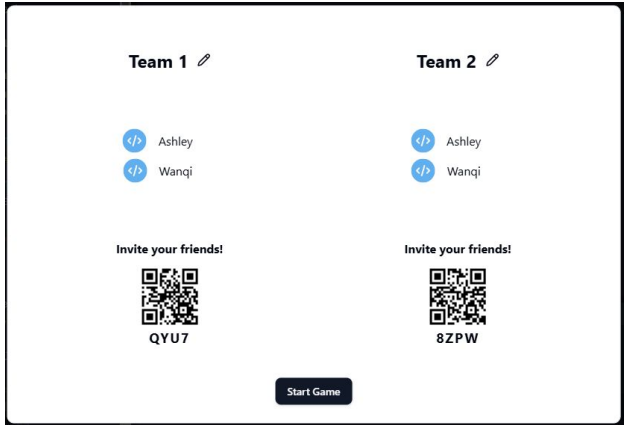
Closes the dialog and reveals the Challenge page (this instance of the dialog component cannot be closed any other way)

# Code & UI Explanation - Ashley Bhandari

```
const [open, setOpen] = useState(isOpen);

const handleClose = (e) => {    Parameter 'e' implicitly has an 'any' typ
  if (disableClose) {
    e.preventDefault();
  }
};

const HeaderContent = (
  <DialogHeader>
    <DialogTitle>{title}</DialogTitle>
    <DialogDescription>{subtitle}</DialogDescription>
  </DialogHeader>
);

useImperativeHandle(ref, () => ({
  closeDialog() {
    setOpen(false);
  },
}));

return (
  <Dialog
    open={open}
    onOpenChange={setOpen}
  >
    {trigger && <DialogTrigger>{trigger}</DialogTrigger>}
    <DialogContent
      onInteractOutside={handleClose}
      onEscapeKeyDown={handleClose}
      className={cn('!max-w-fit', { '[&>button]:hidden': disableClose })}
    >
      {hideHeader ? (
        <VisuallyHidden>{HeaderContent}</VisuallyHidden>
      ) : (
        HeaderContent
      )}
      <div {...props}>{children}</div>
      {footer && <DialogFooter>{footer}</DialogFooter>}
    </DialogContent>
  </Dialog>
);
```

**DialogBox:** A highly customizable dialog component. WaitingRoom uses an instance of this component which starts open, cannot be closed by the user through typical means (clicking outside, ESC, etc.; instead closed by parent WaitingRoom with a ref), and has a hidden header. DialogBox is overlaid on a page (WaitingRoom on the Challenge page).

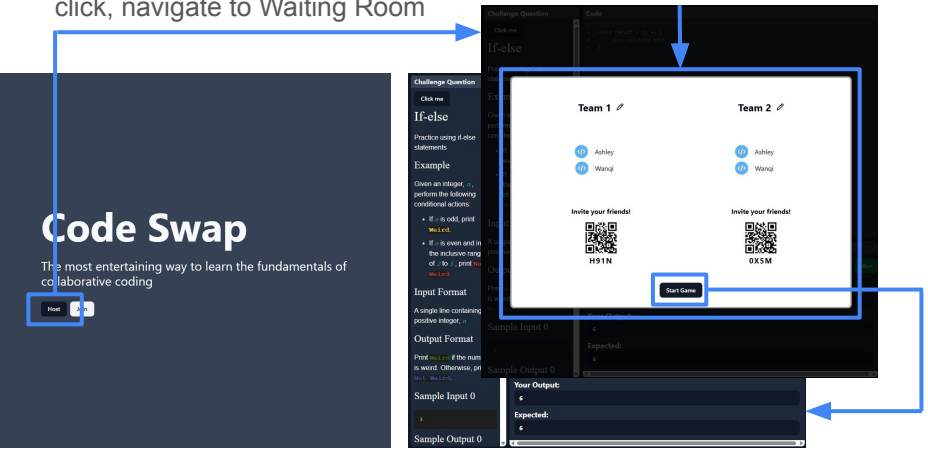
A challenge I faced was how to manage opening/closing the dialog. Initially, I managed state in DialogBox, but this made it difficult to programatically close when closing by the user is disabled (as in WaitingRoom). So, I lifted the state up to DialogBox's parent, but this wasn't very intuitive for any use case other than WaitingRoom. I finally moved state back to DialogBox and exposed a closeDialog method to the parent with useImperativeHandle.



**Adherence to style guidelines:** Dialogs follows the color scheme to ensure accessible color combinations. Breakpoints have been set such that DialogBox and WaitingRoom look good at every screen size. Opening and closing the dialog trigger an animation so that the change in UI looks smooth.

"Host" button (Landing): on click, navigate to Waiting Room

My contribution




"Start Game" button (Waiting Room): on click, close dialog to reveal Challenge page

**User flow:** DialogBox sits on top of other components. WaitingRoom, which uses DialogBox, sits on top of the Challenge page. Inside WaitingRoom are four additional components: editable text, profile picture & name, QR code, and button (clicking the "Start Game" button closes the dialog and reveals the Challenge page).

# Challenges & Insights - Ashley Bhandari

Reflection on obstacles faced and lessons acquired:

- Not everything has to be general and reusable: the Waiting Room holds two teams of two people and that will never change, so simple yet slightly repetitive code is clearer than overly-general code.
- Sometimes escape hatches are okay. I avoided using one until I realized it was the simplest, most effective solution that existed exactly for my use case.

Key takeaways from working within a collaborative team environment.

- Working with better programmers than myself exposes me to better programming practices and encourages me to write clearer code.
- We can all rely on and learn from each other, especially since we all have different backgrounds and expertise.

# Future Improvements & Next Steps - Ashley Bhandari

We need to improve routing. Adding route parameters (such as Room ID and Team Code) will give components more data to work with and better distinguish pages from one another.

We should also improve UI responsiveness; right now our Challenge page isn't great on smaller screen sizes.

# Assigned Work Summary - Yona Voss-Andreae

- I set up the repo, I built the routing in, I set up tons of ui component for common use in from shadcn, I built a dialog to input room codes, I built the landing page billboards, I built the layout of the challenge screen, and wrote a set of challenge questions. I'll save you the screenshots and just direct you to the github page where you can see the commit history and issues for yourself.
- As for the two issues I haven't done yet, the test case one is partially done (see code challenge questions for example) and the other for mocks was added and assigned to me but I am not sure what It is supposed to mean.

# Code & UI Explanation - Yona Voss-Andreae

- A slide showcasing a key piece of code you contributed to, alongside a screenshot of how it impacts the UI.
- Explanation of how the code integrates into the larger UI architecture.
- Discussion of challenges faced and solutions implemented.
- Illustrate and explain how the component design and style adheres to the application's style guidelines.
- A diagram or brief write-up explaining where your work fits within the larger structure of the application.
- User Flow Representation: A description of how your contribution interacts with other UI components in terms of user experience.

# Challenges & Insights - Yona Voss-Andreae

- The biggest challenge is dealing with this bloat that is added to the project by making us jump through these hoops.
- I dont have any key takeaways from this experience if im being honest

# Future Improvements & Next Steps - Yona Voss-Andreae

- Add more components
- Make those components a bit better and more flexible
- Make the game work by adding the functionality that will actually make ti run

# Assigned Work Summary - Wanqi Li

- Developed a markdown convention that will be used for displaying challenge questions. This markdown system applies custom styling automatically when corresponding tags are being used.
- Links to PR and issues: PR, issue

viralrathod10 opened 28 days ago · edited by ashleybhandari  Edits

Use custom HTML tags to build a style system for our question markdown.

Create sub-issue

viralrathod10 added this to the Milesone 1 - Front-end UI milestone 28 days ago

viralrathod10 assigned wLMEB 28 days ago

viralrathod10 changed the title ~~Build a style system for our question markdown~~ Questions Styling 28 days ago

ashleybhandari changed the title ~~Questions Styling~~ Styling & Responsiveness 28 days ago

ashleybhandari added enhancement last month

Yonava changed the title ~~Styling & Responsiveness~~ Markdown templating for challenge questions last month

wLMEB mentioned this yesterday

Custom styling #22

wLMEB closed this as completed in #22 48 minutes ago

Assignees
wLMEB

Labels
enhancement

Projects
No projects

Milestone
Milestone 1 – Front-
Past due by 4d, 60% co

Relationships
None yet

Development
Open in

Custom styling
Yonava/code-swap

Notifications

Commits on Mar 29, 2025

Merge branch 'main' into
wLMEB authored 50 minutes a

Commits on Mar 28, 2025

Merge branch 'main' into custor
wLMEB authored yesterday

adding adjustment to fonts colo
wLMEB committed yesterday

Commits on Mar 14, 2025

Adding code snippet, adjust tex
wLMEB committed 2 weeks ago

Commits on Mar 13, 2025

extract color sytles
wLMEB committed 2 weeks ago

## Custom styling #22

Merged  wLMEB merged 9 commits into `main` from `custom-styling`  53 minutes ago

Conversation 0  Commits 9  Checks 0  Files changed 8

wLMEB commented yesterday

Created customs with associated styles

- : challenge title
- : sub-title for different sections such as examples, inputs, constraints, etc.
- : description of the previous title or subtitle
- : secondary description
- *: math notations/variables*
- : list container
- *: list item*
- *: code variable in color yellow, blue, green, or red*
- *: code snippet in block code format*

*Resolves #1*

Commits on Mar 12, 2025

Edited tag description to be more clear
wLMEB committed 2 weeks ago

Changed font style and made code more modular
wLMEB committed 2 weeks ago

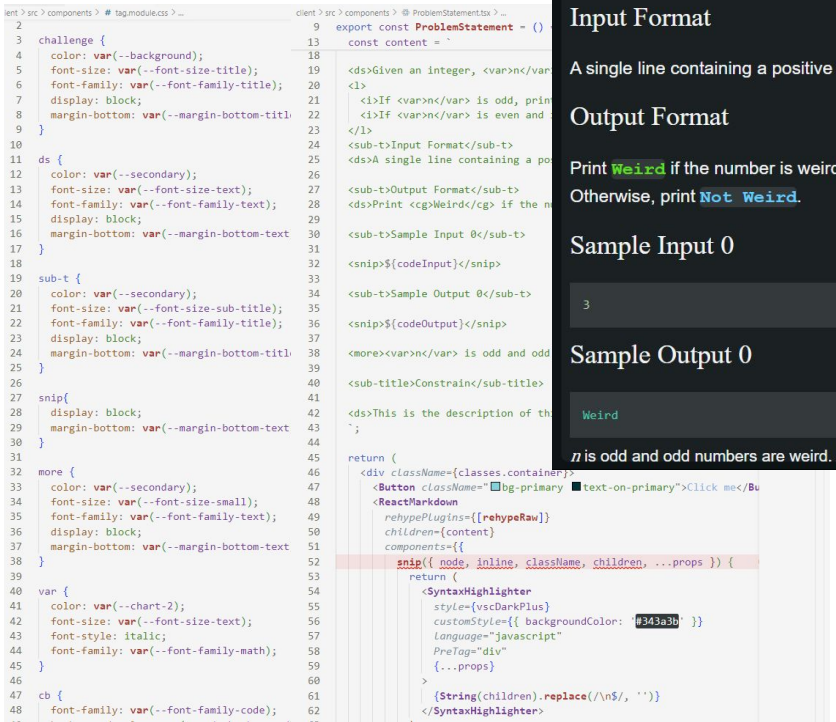
Commits on Mar 3, 2025

Merge branch 'main' of https://github.com/Yonava/code-swap into
wLMEB committed last month

adding custom tags with styles
wLMEB committed last month

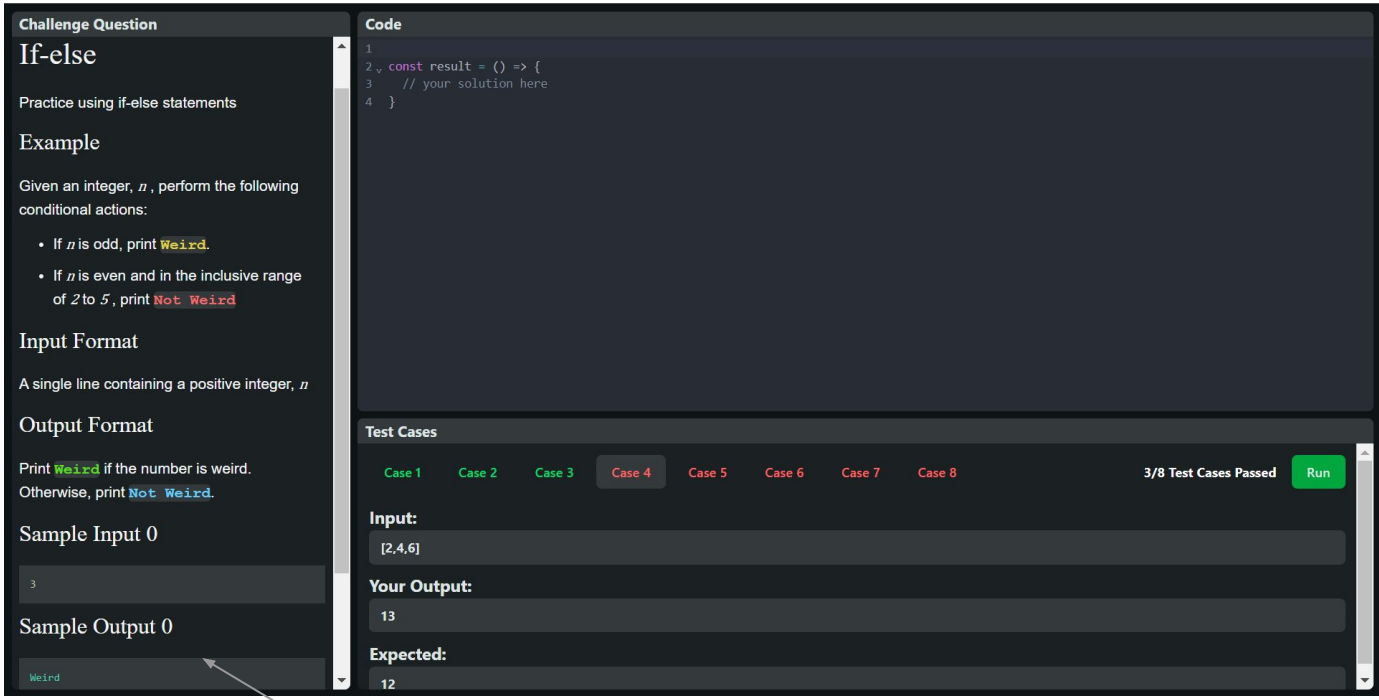
# Code & UI Explanation - Wanqi Li

- **Integration**:The piece of code I worked on was to create custom tags for components that will be part of the challenge questions. The intention behind this is to avoid constantly applying styling to HTML tags so that displaying different problem descriptions will be more consistent and more streamlined.
- **Challenge**: The initial challenge was how to link the styles with the custom tags created. At first, I tried to create another js file that exports each tag with their corresponding styles as a regular .css file doesn't automatically apply the custom styling. But this is very inefficient for making changes and test. My teammate has pointed out to me that I should read on how vite handle css imports. I tried to name the css file end with .module.css and it worked very well.



# Code & UI Explanation - Wanqi Li

**User Flow Representation:** There is not a lot of user interaction intended for the portion I was contracting. The main thing is that the user will be able to scroll up and down to view the full description of the challenge problem.




Scrollable components

# Challenges & Insights - Wanqi Li

- Doing front end work especially making components look nice and user friendly was never a thing for me, but through my experience in creating the components I have more experience in making a scalable front end, at least in the aspect of scalable styles. I now know what are the points to look out for a descent webpage.
- Look things up in documentations more than simple google searches as documentations are more relevant with our development environment.

# Future Improvements & Next Steps - Wanqi Li

- For the later contributions to the application, I want to handle the rendering of different challenge questions from some form of storage.
- The divider between the challenge question and coding panel is currently not adjustable, I'd like to make it more robust for viewing.