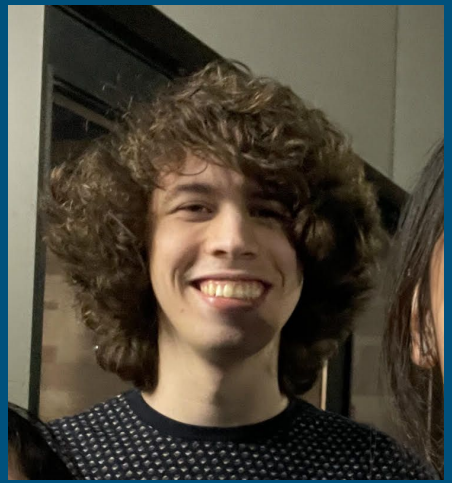
Team 510 Spreadsheet Project

Anthony Vo, Mateus Aurelio, Evan Gurry, Palak Tyagi

Meet the team members!

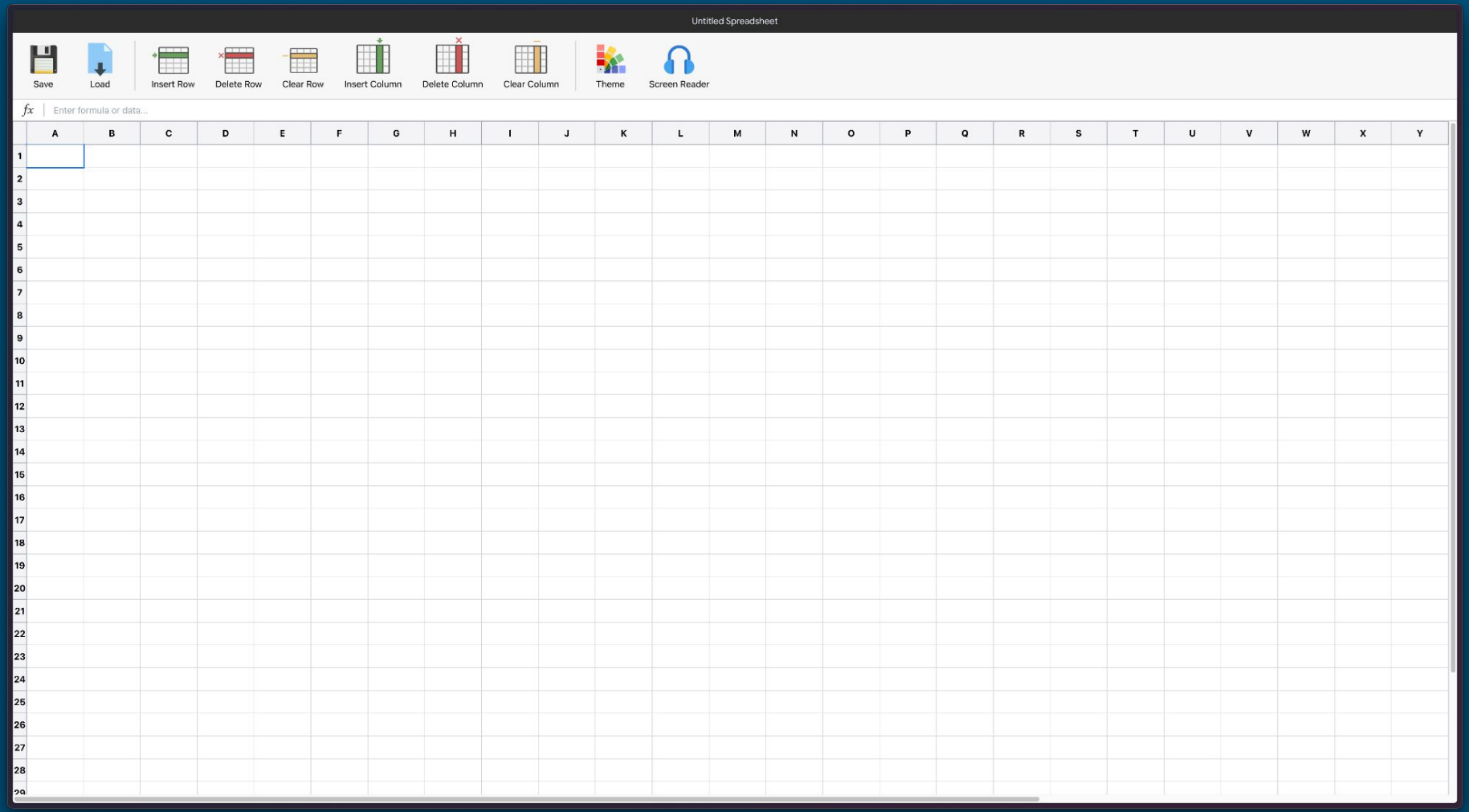
Evan Gurry

Anthony Vo



Mateus Aurelio

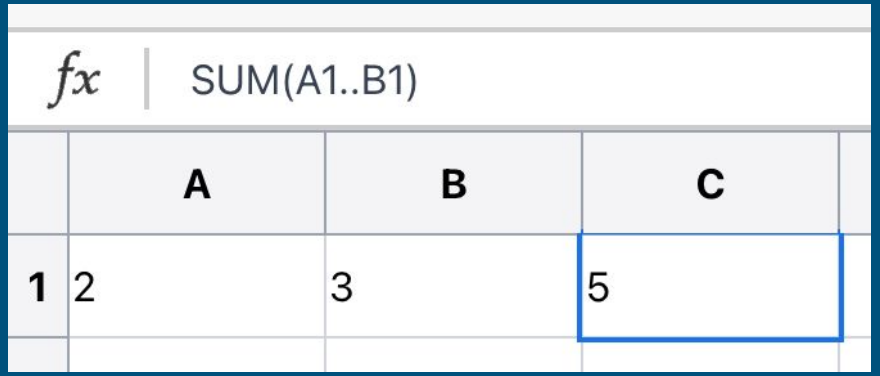
Palak Tyagi


Untitled Spreadsheet
Save
Load
Insert Row
Delete Row
Clear Row
Insert Column
Delete Column
Clear Column
Theme
Screen Reader
fx
Enter formula or data...
A B C D E F G H I J K L M N O P Q R S T U V W X Y
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

Demo Time!

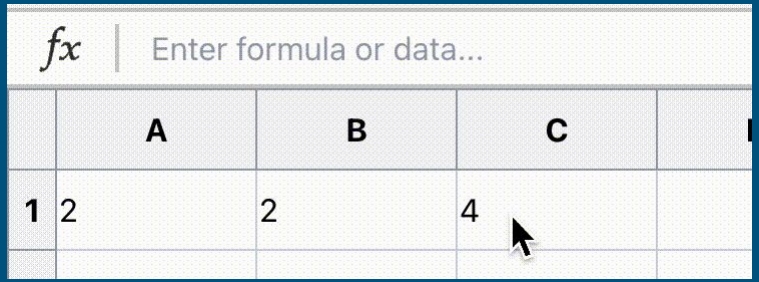
How a Formula Works

- A formula is invoked by a keyword.
- Example keywords:
  - REF()
  - AVERAGE()
  - SUM()
  - TOTAL()
  - MAXIMUM()
  - MAX()
  - MINIMUM()
  - ...



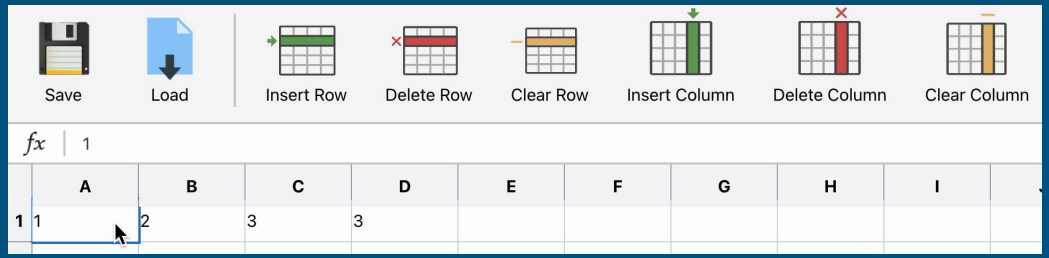
How a Formula is Recalculated

- The formula is recalculated with all newly provided values.
- Example:
  - If A1 = 2 and B1 = 2, SUM(A1..B1) = 4
  - If B1 = 4, now SUM(A1..B1) = 6 (automatically)



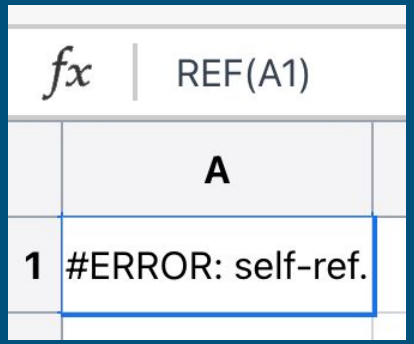
How a Formula is Affected upon Deletion

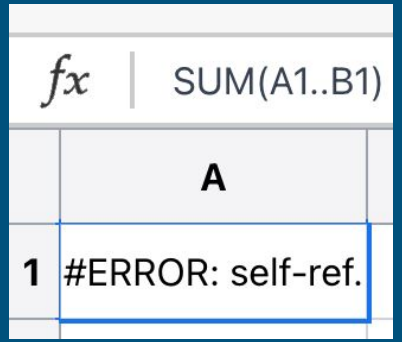
- When a row or column is deleted, the formula is **immediately** re-calculated with the new indexes.
- Example:
  - If a range references A1..B1, and B1 is deleted, the value of C1 will be shifted into B1.
  - The range would still be A1..B1, but it would be like calculating “A1..C1”



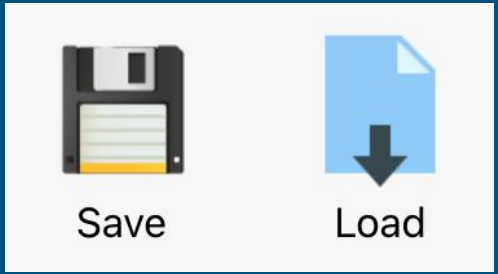
How Error Handling Works

- When a cell references itself (alone or in a range), it throws an error (#ERROR: self-ref.)





Feature 1: Save and Load



The GUI contains both a Save and Load button.

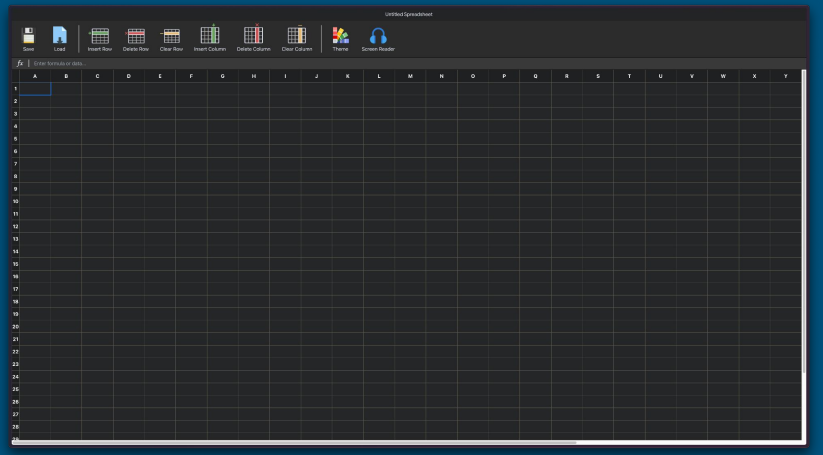
- The Save button exports the data structure as a JSON file, mapping key (cell location) to value (string).
- The Load button would import the exported JSON data back into the program.


```
{"RowCount":"35","ColCount":"35","A1":"","B1":"","C1":"","D1":"","E1":"","F1":"","G1":"","H1":"","I1":"","J1":"","K1":"","L1":"","M1":"","N1":"","O1":"","P1":"","Q1":"","R1":"","S1":"","T1":"","U1":"",
"V1":"","W1":"","X1":"","Y1":"","Z1":"","AA1":"","AB1":"","AC1":"","AD1":"","AE1":"","AF1":"","AG1":"","AH1":"","AI1":"","A2":"","B2":"","C2":"","D2":"","E2":"","F2":"","G2":"","H2":"","I2":"","J2":"",
"K2":"","L2":"","M2":"","N2":"","O2":"","P2":"","Q2":"","R2":"","S2":"","T2":"","U2":"","V2":"","W2":"","X2":"","Y2":"","Z2":"","AA2":"","AB2":"","AC2":"","AD2":"","AE2":"","AF2":"","AG2":"","AH2":"",
"AI2":"","A3":"","B3":"","C3":"","D3":"","E3":"\"sahdiusadhsa\"","F3":"","G3":"","H3":"","I3":"","J3":"","K3":"","L3":"","M3":"","N3":"","O3":"","P3":"","Q3":"","R3":"","S3":"","T3":"","U3":"","V3":"",
"W3":"","X3":"","Y3":"","Z3":"","AA3":"","AB3":"","AC3":"","AD3":"","AE3":"","AF3":"","AG3":"","AH3":"","AI3":"","A4":"","B4":"","C4":"","D4":"","E4":"","F4":"","G4":"","H4":"","I4":"","J4":"","K4":"",
"L4":"","M4":"","N4":"","O4":"","P4":"","Q4":"","R4":"","S4":"","T4":"","U4":"","V4":"","W4":"","X4":"","Y4":"","Z4":"","AA4":"","AB4":"","AC4":"","AD4":"","AE4":"","AF4":"","AG4":"","AH4":"","AI4":"",
"A5":"","B5":"","C5":"","D5":"","E5":"","F5":"","G5":"","H5":"","I5":"","J5":"","K5":"","L5":"","M5":"","N5":"","O5":"","P5":"","Q5":"","R5":"","S5":"","T5":"","U5":"","V5":"","W5":"","X5":"","Y5":"",
"Z5":"","AA5":"","AB5":"","AC5":"","AD5":"","AE5":"","AF5":"","AG5":"","AH5":"","AI5":"","A6":"","B6":"","C6":"","D6":"","E6":"","F6":"","G6":"","H6":"","I6":"","J6":"","K6":"","L6":"","M6":"","N6":"",
"O6":"","P6":"","Q6":"","R6":"","S6":"","T6":"","U6":"","V6":"","W6":"","X6":"","Y6":"","Z6":"","AA6":"","AB6":"","AC6":"","AD6":"","AE6":"","AF6":"","AG6":"","AH6":"","AI6":"","A7":"","B7":"","C7":"",
```

Feature 2: Theming

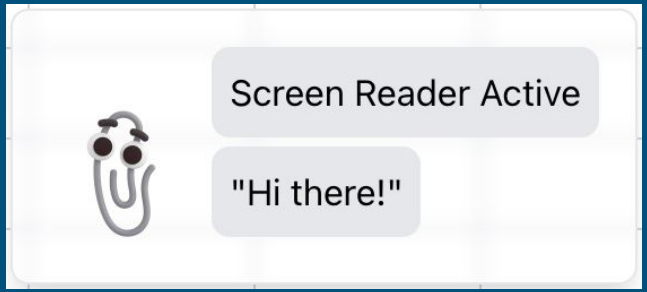
The program supports 3 themes:

- Light Mode (default)
- Dark Mode
- High Contrast Mode



Feature 3: Screen Reader

- Every interactable element in the spreadsheet has text-to-speech!
- Elements that would invoke TTS:
  - Renaming file, when done
  - Clicking a button
  - Editing in the formula bar, when done
  - Clicking on a cell
  - Editing a cell, when done



System Architecture

- Frameworks and Components:
  - Front-end
    - React, TailwindCSS
  - Back-end
    - TypeScript

System Architecture

- Design Patterns:
  - Observer Design Pattern in Data
    - When data structure updates, the CellGrid React Component should update.
    - Data structure subscribes to Spreadsheet class.
    - Spreadsheet class notifies subscribers when a function is invoked that changes data.

System Architecture

- Design Patterns:
  - Observer Design Pattern in Text-To-Speech
    - Text-to-speech stores a log of spoken messages
    - This log is visualized in a separate React component called “ScreenReaderLog”
    - The log subscribes to the ScreenReader class so that:
      - When a message is spoken and the log is updated, ScreenReaderLog is notified

System Architecture

- Design Patterns:
  - Singleton Design Pattern
    - Spreadsheet should only be invoked once
    - To create a Spreadsheet object, call “getInstance()”, which either:
      - creates a new instance
      - reuses an existing instance

Code Exploration

```
function App() {

  // Instantiate Spreadsheet model (Singleton Design Pattern)
  // const spreadsheet = Spreadsheet.getInstance();
  // const screenReader = new ScreenReader();

  // User states: selected cell, cell being edited, and edit value
  const [activeCell, setActiveCell] = React.useState<string>("A1"); // Active cell
  const [activeEditCell, setActiveEditCell] = React.useState<string>(""); // Active cell being edited
  const [editValue, setEditValue] = React.useState<number|string>(""); // Value of the cell being edited
  const [fileName, setFileName] = useState<string>("Untitled Spreadsheet"); // Name of the file when saving and loading
  const [theme, setTheme] = useState<string>("defaultTheme"); // Theme of the app
  const [screenReaderUIActive, setScreenReaderUIActive] = useState<boolean>(false); // Whether screen reader is active

  return (
    <div className={`flex flex-col h-screen ${theme}`}>
        <div className="sticky top-0 z-10">
            <FileHeader fileName={fileName} setFileName={setFileName} />
            <OptionsPane activeCell={activeCell} setEditValue={setEditValue} fileName={fileName} setFileName={setFileName} theme={theme} setTheme={setTheme}
            screenReaderUIActive={screenReaderUIActive} setScreenReaderUIActive={setScreenReaderUIActive}/>
            <FormulaBar activeCell={activeCell} activeEditCell={activeEditCell} editValue={editValue} setEditValue={setEditValue} />
        </div>
        <div className="flex-grow overflow-auto">
            <CellGrid activeCell={activeCell} setActiveCell={setActiveCell} activeEditCell={activeEditCell} setActiveEditCell={setActiveEditCell} editValue={editValue}
            setEditValue={setEditValue} />
        </div>
        {screenReaderUIActive && <ScreenReaderLog />}
    </div>
  );
}
```

Code Exploration

```
18  interface UserProps {
19    activeCell: string;
20    setActiveCell?: (cell: string) => void;
21    activeEditCell?: string;
22    setActiveEditCell?: (cell: string) => void;
23    editValue?: string | number;
24    setEditValue: (value: string | number) => void;
25    fileName: string;
26    setFileName: (name: string) => void;
27    theme: string;
28    setTheme: (theme: string) => void;
29    screenReaderUIActive: boolean;
30    setScreenReaderUIActive: (screenReaderActive: boolean) => void;
31  }
32
33  export const OptionsPane: React.FC<UserProps> = ({activeCell, setEditValue, fileName, setFileName, theme, setTheme, screenReaderUIActive, setScreenReaderUIActive}) => {
34
```

Code Exploration

```tsx
function App() {

  // User states: selected cell, cell being edited, and edit value
  const [activeCell, setActiveCell] = React.useState<string>("A1"); // Active cell
  const [activeEditCell, setActiveEditCell] = React.useState<string>(""); // Active cell being edited
  const [editValue, setEditValue] = React.useState<number|string>(""); // Value of the cell being edited
  const [fileName, setFileName] = useState<string>("Untitled Spreadsheet"); // Name of the file when saving and loading
  const [theme, setTheme] = useState<string>("defaultTheme"); // Theme of the app
  const [screenReaderUIActive, setScreenReaderUIActive] = useState<boolean>(false); // Whether screen reader is active

  const userProps: IUserProps = {
    activeCell,
    setActiveCell,
    activeEditCell,
    setActiveEditCell,
    editValue,
    setEditValue,
    fileName,
    setFileName,
    theme,
    setTheme,
    screenReaderUIActive,
    setScreenReaderUIActive
  };

  return (
    <UserContext.Provider value={userProps}>
      <div className={`flex flex-col h-screen ${theme}`}>
          <div className="sticky top-0 z-10">
              <FileHeader />
              <OptionsPane />
              <FormulaBar />
          </div>
          <div className="flex-grow overflow-auto">
              <CellGrid />
          </div>
          {screenReaderUIActive && <ScreenReaderLog />}
      </div>
    </UserContext.Provider>
  );
}
```

Code Exploration

```ts
export default interface UserProps {
    activeCell: string;
    setActiveCell: (cell: string) => void;
    activeEditCell: string;
    setActiveEditCell: (cell: string) => void;
    editValue: string | number;
    setEditValue: (value: string | number) => void;
    fileName: string;
    setFileName: (name: string) => void;
    theme: string;
    setTheme: (theme: string) => void;
    screenReaderUIActive: boolean;
    setScreenReaderUIActive: (active: boolean) => void;
}
```

```ts
import { createContext } from "react";
import IUserProps from "interfaces/IUserProps";

// Define default values for UserProps
const defaultUserProps: IUserProps = {
  activeCell: "A1",
  setActiveCell: () => {},
  activeEditCell: "",
  setActiveEditCell: () => {},
  editValue: "",
  setEditValue: () => {},
  fileName: "Untitled Spreadsheet",
  setFileName: () => {},
  theme: "defaultTheme",
  setTheme: () => {},
  screenReaderUIActive: false,
  setScreenReaderUIActive: () => {},
};

// Create the context with default values
const UserContext = createContext<IUserProps>(defaultUserProps);

export { UserContext };
```

Lessons Learned

- Projects take time… make sure to distribute work wisely!
- It can be difficult to coordinate meeting up to work with a team. It is worthwhile to find ways to split up and distribute work between programmers
- It is important to split the work early before it becomes hard to find where tasks can be split.
- Refactor and test early!