Overview

Summary of Required Slides

Below is a summary of all the slides that must be included in the submission:

Team-Level Slides

1. **Title Slide** - Project name, team name, and date.
2. **Project Overview** - Brief description of the project and its primary goals.
3. **Team Members** - Names, roles, and primary contributions of each team member.
4. **Software Architecture Overview** - Diagram and explanation of the UI architecture.
5. **Historical Development Timeline** - Gantt chart or similar visual showing the timeline of key development phases.
6. **Design and Styling Guidelines** - Summary of the UI/UX design choices and style guide link.
7. **Component Documentation** - Screenshots and descriptions of major UI components.
8. **Performance Considerations** - Optimizations made and profiling results.

Individual Team Member Slides

9. **Assigned Work Summary** - Issues, commits, and PRs each member worked on.
10. **Code & UI Explanation** - Key code contributions, UI impact, and integration.
11. **Challenges and Insights** - Key takeaways, obstacles faced, and solutions.
12. **Future Improvements & Next Steps** - Proposals for future features and optimizations.

Example Breakdown for a 4-Person Team

A typical submission might include:

Team-Level Slides (6 slides)

- **Title Slide** (1 slide)
- **Project Overview & Team Members** (1 slide)
- **Software Architecture Overview** (1 slide)
- **Historical Development Timeline** (1 slide)
- **Design and Styling Guidelines** (1 slide)
- **Performance Considerations** (1 slide)

Individual Contributions (4 x 3 slides each = 12 slides)

- **Assigned Work Summary** (1 slide per team member)
- **Code & UI Explanation** (1 slide per team member)
- **Challenges & Insights** (1 slide per team member)

Closing Slide (1 slide)

- **Future Improvements & Next Steps**

Total estimated slides for a 4-person team: ~19 slides.

DefiGuard

Understand The Health of Your Lending Protocol

Chris Pickreign, Steven Arbo, Christian Noble Shriver

Project Overview

DeFiGuard

Our app aims to provide real-time risk monitoring and predictive analytics for DeFi lending protocols. By recognizing patterns in advance and proactively alerting teams, we help protocols mitigate risk exposure and ensure users are well informed. This helps lenders, borrowers, and platform operators make data-driven decisions to protect their assets.

Link to repository: https://github.com/stevenarbo3/CS426-DeFiGuard

Link to Milestone: https://github.com/stevenarbo3/CS426-DeFiGuard/milestones

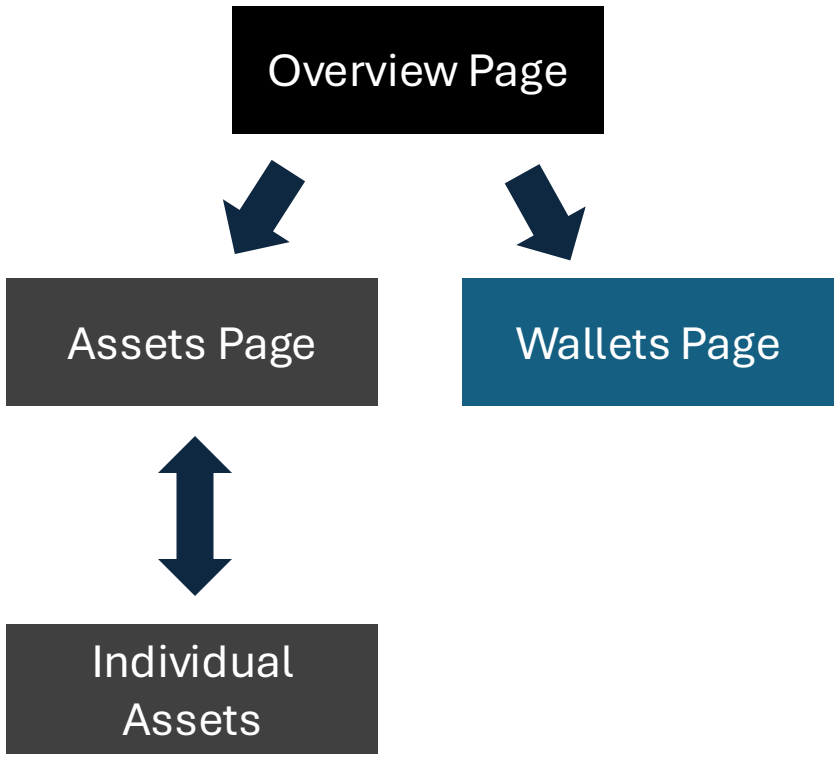
Team Members

TEAM MEMBER	ROLE THIS MILESTONE
Steven Arbo	Components and Design – Overview Page + Header
Chris Pickreign	Components and Design – Individual Asset Page
Christian Noble Shriver	Components and Design – Wallets / Assets Page

UI Architecture

Our UI architecture was built using Next.JS with a React Framework in TypeScript, NodeJS, Tailwind CSS for styling, and ShadCn for efficient component creation

Data Flow



Components

Created by Team:

1. **Header:** Persistent on all pages, provides routing capabilities
2. **StatBox:** On most pages, provides ability to display relevant information on
3. **StockChart:** Chart developed for statistics on
4. **data-table:** Table for displaying individual assets and user positions.

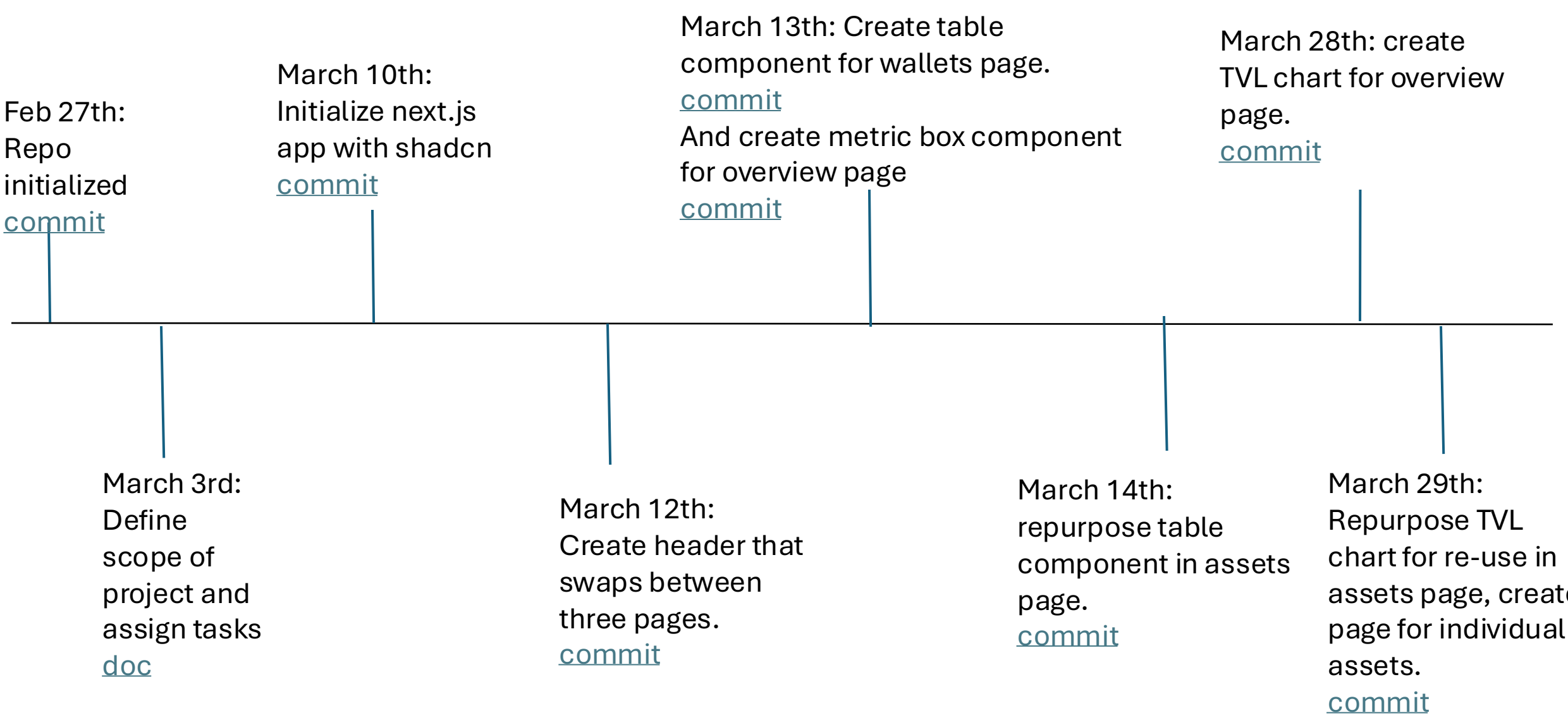
From ShadCN:

1. Button.tsx
2. Card.tsx
3. Chart.tsx
4. Select.tsx
5. Sheet.tsx
6. Table.tsx
7. Tooltip.tsx

State Management

- Since our application is a dashboard, there are no major state changes that are currently implemented
    - These state changes will occur with the implementation of user authentication, user type, etc.
- For the time being, the application is in one state with the user (everyone) being able to go through the relevant information about the lending protocol (wallets, assets, relevant metrics etc.)
- Some components are reused across the application and others occur in one instance, we have tried to make them reusable

Historical Development Timeline



Style Guidelines

Link to style guidelines

Style Guidelines

*Use TailwindCSS for Styling

Color Scheme:

Primary Background Color: #0a0e17
Primary Text Element Color: #ffffff (white)
Buttons, Links, etc. : Use judgement and let peers review. Use light gray / blue or white.

Typography:

Page Header: Use <h1> with text-5xl and font-bold
Subheadings and other text decrease in size and weight
Default font family for all (open to change)

Spacing and Layout Principles:

Page Layout: mx-auto max-w-7xl
Grid Layout for Stat Boxes: Will vary use judgement and let peers review

Component Behavior:

Use shadcn library for components such as charts, graphs, etc.
Hover: cursor-pointer, keep shad hover effects, lighten color / reduce opacity on custom hover
Transitions: keep shadcn transitions

Accessibility Standards:
Must adhere to WCAG standards

Responsive Design Considerations:

Header shrinks - keeping all links in view
Graphs should shrink staying in complete view
Grids should shrink depending on amount of elements and stay in view
Charts should shrink - allowing for horizontal scrolling

Component Documentation

Major developed components in our application include StatBox.tsx, Header.tsx, and StockChart.tsx

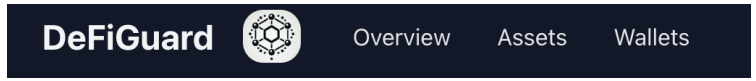
StatBox

A component developed to neatly display relevant statistics


```
  delta: number;
  positive?: boolean;
}) {
  const isPositive = delta >= 0;
  const color = isPositive ? "text-green-400" : "text-red-500";
  const arrow = isPositive ? "▲" : "▼";
  return (
    <div>
      <p className="text-sm text-gray-400">{title.toUpperCase()}</p>
      <p className="text-2xl font-bold">{value}</p>
      <p className={`text-sm ${color}`}>{arrow} {Math.abs(delta).toFixed(2)}%</p>
    </div>
  );
}
```

Header

A component created for navigation across relevant pages in the application




```
export default function Header() {
    return (
        <header className="flex items-center justify-center gap-5 bg-gray-900 p-4 shadow-
            <Link href="/" className="text-2xl font-bold text-gray-100 hover:text-white">
                DeFiGuard
            </Link>
            {/* Placeholder Image */}
            <Image
            src="/assets/temp-logo.png"
            alt="carbon logo"
            width={40}
            height={40}
            className="rounded-xl"
            />
            <nav className="flex list-none">
                <li><Link className={navButton} href="/">Overview</Link></li>
                <li><Link className={navButton} href="/assets_page">Assets</Link></li>
                <li><Link className={navButton} href="/wallets">Wallets</Link></li>
            </nav>
        </header>
    );
```

StockChart

A component developed to display charts and graphs of relevant info over time


```
return (
  <div className="flex w-full items-center justify-center p-6">
    <Card className="h-[60vh] w-[90vw] rounded-xl border border-gray-800 bg-[#0a0e17] text-white shadow-md">
      <CardHeader className="flex flex-row items-center justify-between p-5">
        <CardTitle className="text-xl font-bold">{title}</CardTitle>
      </CardHeader>
      <CardContent className="relative h-[calc(100%-70px)] p-0">
        <div className="h-full w-full px-5 pb-5">
          <div className="h-full w-full rounded-lg border border-gray-800 p-3">
            <ChartContainer config={chartConfig} className="h-full w-full">
              <ResponsiveContainer width="100%" height="100%">
                <LineChart
                  data={data}
                  margin={{ top: 20, right: 30, left: 20, bottom: 10 }}
                >
                  <CartesianGrid horizontal vertical={false} stroke="#333" strokeDasharray="3 3" />
                  <XAxis
                    dataKey="date"
                    axisLine={false}
                    tickLine={false}
                    tick={{ fill: "#9ca3af", fontSize: 12 }}
                    padding={{ left: 10, right: 10 }}
                  />
                  <YAxis
                    axisLine={false}
                    tickLine={false}
                    tick={{ fill: "#9ca3af", fontSize: 12 }}
                    tickFormatter={(value) => `$${value}B`}
                    domain={[0, 21]}
                    ticks={[0, 3, 6, 9, 12, 15, 18, 21]}
                    width={50}
                  />
                  <Tooltip
                    content={<CustomTooltip />}
                    cursor={{ stroke: "#666", strokeDasharray: "3 3", strokeWidth: 1 }}
                    wrapperStyle={{ zIndex: 100, outline: "none" }}
                    position={{ y: 0 }}
                  />
                  <Line
                    type="linear"
                    dataKey="value"
                    stroke={color}
                    strokeWidth={2}
                    dot={false}
                    activeDot={{ r: 5, fill: color, stroke: "#0a0e17", strokeWidth: 2 }}
                    isAnimationActive
```

Performance Considerations

- Main page components are implemented as server components, reducing the client-side burden.
- Server components don't send their code to the client, only the HTML output
- For example, the wallets page relies on an async function that will eventually be hooked up to a backend for data fetching:

```
export default async function WalletsPage() {
  const data = await getData();
  return (
    <div className="min-h-screen bg-[#0a0e17]">
      <main className="flex flex-col p-5 text-white mx-auto max-w-7xl w-full">
        <h1 className="text-5xl font-bold">Wallets</h1>
        <div className="container mx-auto py-10">
          <DataTable columns={columns} data={data} />
        </div>
      </main>
    </div>

  );
}
```

- For a data heavy component like this data table, using server components significantly reduces bundle size.

Assigned Work Summary

Contribution Description

Over the course of this milestone, I had two major PRs. In the first PR, major updates included the creation of the assets page and the data table to display the relevant asset information. In the second, major updates included the creation of the individual assets page linked to each of the assets in the asset table as well as the updating of the previous TVLChart to be a StockChart that we can reuse. All along the way I have continued to make updates to the CSS across a number of main pages as well as minor updates to the project

Linked Contributions:

Link to pull requests

Link to assigned issues

Code & UI Explanation

- A key contribution of mine was the linking of the assets page to the individual assets that were listed in the data table
- Before, the table was populated with all of the relevant data, but with no way to see each of the individual assets info on a page
- With the updates in the Columns page, I was able to configure a link that, when clicked, routes to a link for each of the individual assets
- In total, as you can see on the table, all of the asset names are now blue and when they are clicked, it brings the user to the page on the bottom where they can view all of the relevant information for an asset in more detail

Code

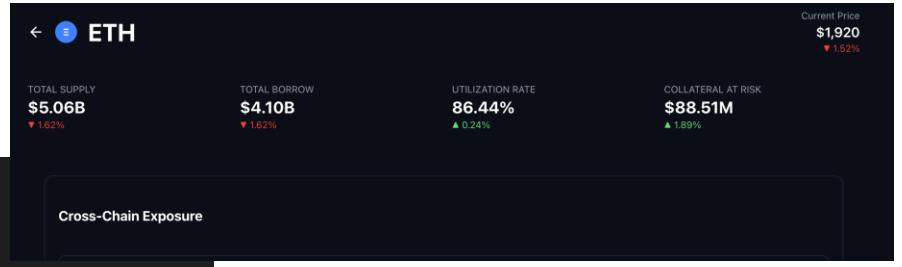
```tsx
export default async function AssetPage({ params }: AssetPageProps) {
  // Fetch data and grab the asset ID
  const data = await getData();
  const assetId = params.asset.toLowerCase();

  // Find the asset data based on the asset ID
  const asset = data.find((a) => a.assetType.toLowerCase() === assetId);
  if (!asset) return notFound();

  return (
    <div className="min-h-screen bg-[#0a0e17]">
      <main className="min-h-screen bg-[#0a0e17] text-white px-8 py-10 space-y-
        {/* Header section */}
        <div className="flex items-center justify-between">
          <div className="flex items-center gap-4">
```

```tsx
// Function to export the data for our table
export async function getData(): Promise<Position[]> {
    // Mock data
    return [
        {
            assetType: 'ETH',
            price: 1920,
            collateralAmount: 4940000000,
            totalBorrow: 4100000000,
            collateralAtRisk: 88510000,
            walletsAtRisk: 13760,
```

```tsx
export const columns: ColumnDef<Position>[] = [
    {
        accessorKey: "assetType",
        header: "Asset",
        enableSorting: true,
        cell: ({ row }) => {
          const asset = row.getValue("assetType") as string;
          return (
            <Link href={`/individual_assets/${asset.toLowerCase()}`} className="text-blue-500 hover:underline">
              {asset}
            </Link>
          );
        },
      },
```

UI Impact

Asset	Price	Total Collateral
ETH	1.92K	4.94B
BTC	84.44K	3.91B
stETH	2.29K	3.74B
USDC	0.9999	3.66B
USDT	0.9998	3.45B
weETH	2.04K	2.99B



Challenges & Insights

Obstacles

There were many obstacles faced in this sprint from a technical and planning perspective as I assumed the role of Sprint Lead for our latest sprint. The biggest obstacle with this role was assessing the current state of our project as well as identify and assign tasks for our group members. From a technical perspective, figuring out how to configure the routing to the individual asset page proved challenging in this milestone

Lessons

1. Overall, the biggest lesson I learned was how to develop an application efficiently. I did not know about ShadCn going into this project, and learning how to use it to quickly develop has proven invaluable for my abilities to create applications
2. I also learned many skills related to how to properly plan a sprint, which in the past I have been given in my previous SWE roles. Understanding the current state, what features are high priority, and developing tasks have all proven helpful

Future Improvements and Next Steps

Improvements

1. **Development of Charts and Graphs:** We currently have mock data in for our charts and graphs, hooking up our current infrastructure to an external data source is first on my mind
2. **Page to compare lending protocols against one another:** I think it would be interesting and relevant to include information/a page that compares relevant statistics of different lending protocols against one another

Technical Debt

1. **Development of reusable components:** We made some mistakes in which a component was configured for a single use case. I hope to refactor other and future components to be reusable for efficient development
2. **Inclusion of time frames:** For each of the statistics, providing the user with options to view the data in different times (1 mo, 6 mo etc) to make the data more tailored for user needs

Assigned Work Summary

Contribution Description

In this milestone, I worked on component development and styling. Specifically, I took charge on the overview page, header component, and general styling. I created reusable stat box components along with Chris Pickreign that could be used in both the overview and asset page. I developed the header component that allows for navigation between pages. Finally, I standardized CSS among pages and created the style guidelines.

Here, I will link my contributions:

Link to commits and merged pull requests

Link to assigned issues

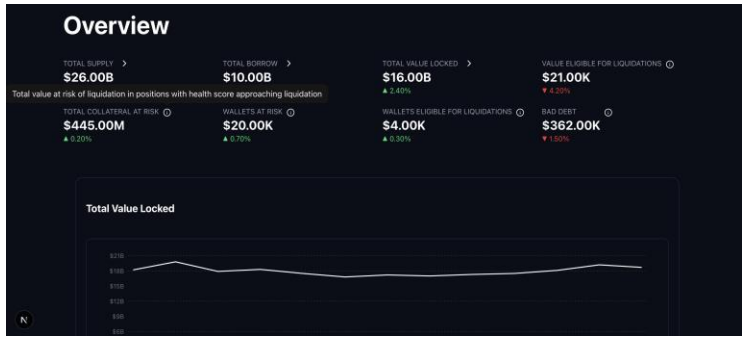
Code & UI Explanation

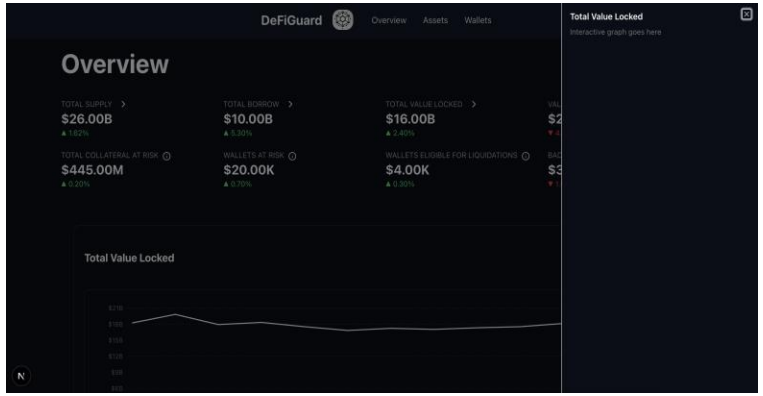
- The screenshots depict the overview page before and after clicking on a metric box. (The graph is not in the sidebar and this is explained in the issue comments).
- This is a key functionality in that it allows us to display more than just the basic information displayed (such as graphs)
- A challenged faced is that the first three metric boxes have the sidebar functionality while the other have an info button. I used a ternary to solve this.
- The component follow the dark colored styling guidelines by keeping the background color and using white text.

Code

```
const statBoxElements : Element[] = mock_data.map((data : { title: string; value: number…,index) => {
  return (
    <React.Fragment key={data.title}>
      {index < 3 ?
        <Sheet>
          <SheetTrigger asChild className="cursor-pointer">
            <div className="flex gap-3">
              <StatBox
                  key={data.title}
                  {...data}
                  value={`$${formatNumber(data.value)}`}
                />
              <ChevronRight className="h-5 w-5" />
            </div>
          </SheetTrigger>
            <SheetContent className="bg-[#0a0e17] text-white">
              <SheetHeader>
                <SheetTitle className="text-white">{data.title}</SheetTitle>
                <SheetDescription>
                  Interactive graph goes here
                </SheetDescription>
              </SheetHeader>
            </SheetContent>
          </Sheet>
          :
          <div className="flex gap-2 flex-wrap items-start">

            <StatBox
                {...data}
                value={`$${formatNumber(data.value)}`}
              />
              <TooltipProvider>
                <Tooltip>
                  <TooltipTrigger className="flex flex-start mt-1"><InfoIcon className="h-4 w-4"/></TooltipTrigger>
                  <TooltipContent>
                    <span className="text-base">{data.description}</span>
                  </TooltipContent>
                </Tooltip>
              </TooltipProvider>
            </div>
      }
    </React.Fragment>
  )
})
```

UI Impact





Challenges & Insights

Obstacles

The major obstacles faced were time related. With other classes and projects such as the Carbon Footprint homework, it made it so work on this milestone was done sporadically. There was no way to build momentum and there was a lot of troubleshooting. Code related obstacles included getting used to shadcn component library and making responsive pages.

Lessons

I learned a lot in this milestone.

1. More preparation is required – it's easy to go straight to coding but preparation is key when collaborating with others
2. More communication is required – we talked enough and when necessary but more dedicated meeting times are needed on scheduled basis.

Future Improvements and Next Steps

Improvements

1. Sidebar Graph (unfinished Issue) - I believe this would really complete the overview page for reasons specified earlier
2. More graphs / charts – our project's goal is to convey information thus the more ways we can do that the better
3. Analysis – right now we display plenty of information and I wonder if some more in depth analysis of this information would be helpful for the user

Technical Debt

1. I believe the header could be improved as it wasn't a priority in this milestone - I just wanted functionality
2. The responsiveness could be looked at more – it is currently responsive but there could be a better way especially with the grids

Assigned Work Summary

Contribution Description

- My responsibilities included:
  - Creating the "Wallets" page, with a table component to display user positions and relevant information. In includes sorting by column and pagination functionality.
  - Creating the "Total Value Locked" chart on the "Overview" page.

  PRs:
  - https://github.com/stevenarbo3/CS426-DeFiGuard/commit/e0eb2452541013243ec979870821b35ce682700b
  - https://github.com/stevenarbo3/CS426-DeFiGuard/commit/c13f28f3663a6e1eeb1c89e10c84ba71747a5227
  - https://github.com/stevenarbo3/CS426-DeFiGuard/commit/b1710f5a0d4b1b30b167ce3ed70a090ba6598945

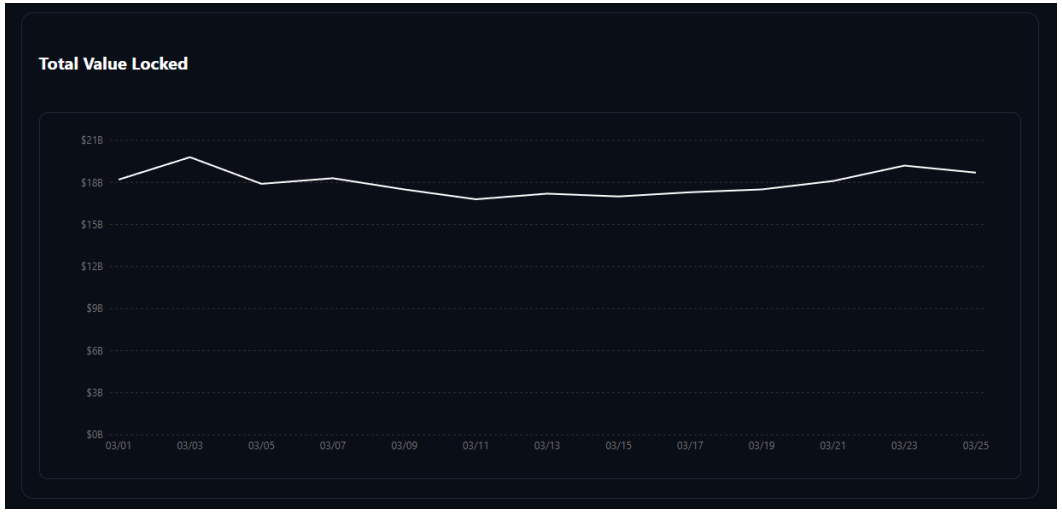
Code & UI Explanation

Code

```
{
  accessorKey: "healthFactor",
  header: "Health Factor",
  cell: ({ row }) => {
    const healthFactor = parseFloat(row.getValue("healthFactor"));

    // Determine color class based on health factor
    let bgColorClass = "bg-green-500/20 text-green-500 border-green-500/50";

    if (healthFactor < 1.3 && healthFactor >= 1.1) {
      bgColorClass = "bg-orange-500/20 text-orange-500 border-orange-500/50";
    } else if (healthFactor < 1.1) {
      bgColorClass = "bg-red-500/20 text-red-500 border-red-500/50";
    }

    return (
      <div className="flex justify-center">
        <span
          className={`px-3 py-1 rounded-full text-xs font-medium ${bgColorClass} border`}
        >
          {healthFactor.toFixed(2)}
        </span>
      </div>
    );
  },
  enableSorting: true,
},
];
```

UI Impact

- The wallets page contains a table component that shows each user's position, the collateral type they supply, the amount supplied and the amount borrowed. It also shows the "health" of their position, that is, how close their position is to liquidation (<1.1 unhealthy and >1.3 healthy)
- The table includes sorting and pagination functionality. You can sort each column in ascending or descending order. Also if the data exceeds a set amount (like 10), wallets are organized into separate pages, this saves the site from having to load too many positions on the same page.
- The Total-value-locked (TVL) chart shows how much capital there is locked up in a lending protocol's smart contracts. The y-axis is dollars, and the x-axis is time.
- The code shows the logic for changing the colors of the health factors in the health factor column of the table.



Challenges & Insights

Obstacles

- At first, I didn't know how to best go about displaying the data. Luckily I was browsing through the "blocks" tab on https://ui.shadcn.com/ , and found a table component that I liked. All I had to do was basically change the column names, and a bit of the styling.
- It was a similar scenario for the TVL chart, I was looking for ways to make neat graphs given data, and luckily shadcn provided a library of chart components that I could easily repurpose for this app.

Lessons

- A lot of the problems I wanted to solve already had solutions in the form of pre-built components. All I had to do was understand how they worked, and I was able to change them for my own purpose. This took a lot of the heavy lifting out of the process, and allowed me to build things quicker.
- Having a tool like shadcn in the project was very nice for streamlining the process of building components, its like they provide the lego bricks (nice looking buttons and stuff) and all you have to do is put them together.

Future Improvements and Next Steps

Improvements

1. Currently, the app uses sample data from local objects directly in the component files. It uses async functions to "fetch" the data, which will allow us to easily hook things up to a database or API down the road. So actually doing this is the logical next step.
2. On the wallets page, I want to make the wallet addresses clickable, so you're brought to a site like https://etherscan.io/ to view more info on that wallet.

Technical Debt

1. Need to hook up a database or API to fetch protocol data.
2. Need to implement error handling if data fails to get retrieved, such as displaying loading bars, or error messages

Future Improvements and Next Steps

- On the UI side of things, we want to be able to display data by different time periods (e.g. 30d, 90d, 1yr)
- Another big hurdle will be to hook everything up to a backend. We suspect that some of the info we want to display, we might not be able to get, so some decisions will have to be made on what we want to include or discard based on the data available to us.
- The table component we used for the Assets page and Wallets are independent, and should be edited to a single re-usable component.
- Besides these main things, other small things might be changing the styling to make things look nicer.

Thank You!