A photograph of a railroad crossing at dusk or dawn. In the foreground, a white pole holds two large, circular flashing red lights. The lights are illuminated, and their red glow is visible. In the background, a train with several locomotives and freight cars is approaching the crossing. The train's headlights are on, and its lights are reflecting on the tracks. The sky is dark, and there are some trees and foliage in the background. The overall scene is dimly lit, with the primary light sources being the crossing lights and the train's headlights.

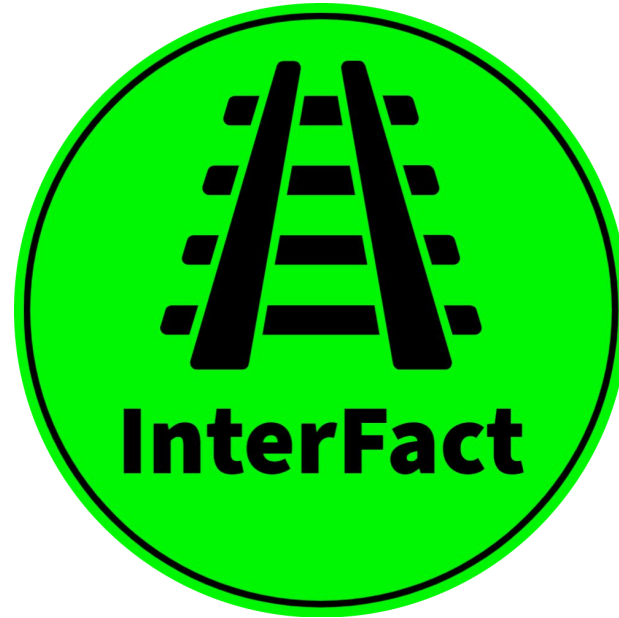
InterFact: Railroad Crossing Information System

InterFact Client & Information:



Huseyin Ergin

You know him



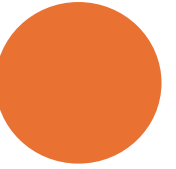
Determines if a train is blocking an intersection in Muncie, using cameras and machine learning



An Admin Dashboard for reviewing data about the InterFact system

3rd Iteration Features

- Interfact Admin Dashboard:
 - Camera Under Maintenance Sort Filter
 - Camera Status Filters
 - (Operational & Inactive)



Interfact Iteration 3 Demo!



Camera Maintenance Filter

Intersections can now be sorted
based on which cameras are under
maintenance!

```
case 'Maintenance':  
  if(isFilterOpen !== true || isFilterBlocked !== true){  
    setIsFilterMaintenance(!isFilterMaintenance)  
  }  
  break;
```



Camera Operation Status Filters

- Intersections can now be sorted based on if the cameras are being regularly updated!
 - If data is not received from a specific camera after 10 minutes of inactivity, the camera status is Inactive



```
// Cameras that have been updated within 10 minutes
if (isFilterWorking){
    if(calculateDifferenceInMinutes(item.timestamp) < 10){
        return item.status === "OPERATIONAL"
    }
}

// Cameras that have not been updated within 10 minutes
if (isFilterNotWorking){
    if(calculateDifferenceInMinutes(item.timestamp) > 10){
        return item.status === "INACTIVE"
    }
}
```

Ethan

The main thing I worked on was getting reports to show up on the intersection pages. I added the ability to approve or deny reports which would move images from one folder to another based on the response. I also added MySQL integration and a logs display.

```
1  import { NextResponse } from 'next/server';
2  import dbMS from '../../../../../../MySQLConfig';
3
4  export async function GET() {
5    try {
6      const [rows] = await dbMS.query("SELECT * FROM log ORDER BY timestamp DESC");
7      return NextResponse.json(rows);
8    } catch (error) {
9      console.error("Database error:", error);
10     return NextResponse.json({ error: "Database error" }, { status: 500 });
11   }
12 }
13
```

Brooke









I focused on writing tests. The first week or so of the iteration was spent trying to figure out how to test with a database, but that was abandoned in favor of writing something I could make progress on.

```
// An array of UserFeedbacks
> const mockUserFeedback = [ ...
]

describe("Requests", () => {
  it('displays only requests for the correct intersection', () => {
    useParams.mockReturnValue({id: mockId});
    useIntersections.mockReturnValue(mockIntersections);
    useUserFeedback.mockReturnValue(mockUserFeedback);

    render(<IntersectionsPage/>)

    expect(screen.getAllByTestId("report").length).toBe(3);
  })
})
```

-  jest with **firebase** - Google Search — [google.com/search?client=firefox-b-1](https://www.google.com/search?client=firefox-b-1)
-  snippets-web/snippets/firestore-next/emulator-suite/fs_emulator_cc —
-  Connect your app to the Cloud Firestore Emulator | **Firebase** Local —
-  Setting Up the **Firebase** Emulators with Next.js — makerkit.dev/blog/tutorials
-  Build unit tests | **Firebase** Security Rules — firebase.google.com/docs/rules
-  View camera information from **firebase** (determine down) · Interfact —
-  Code coverage report for interfact-admin-dashboard/src/app/DAOs —
-  **Firebase** Experiment - Adding Tests with Jest - YouTube — [youtube.com](https://www.youtube.com)

^ search history

< example test

Tanner

My main work outside of recording minutes was adding the maintenance status to the intersections page, plus made the small fix to count the right amount of intersections in the city page.

```
case 'Blocked':  
    // If open filter is not active  
    if(isFilterOpen !== true || isFilterMaintenance !== true){  
        // Toggles filter to blocked  
        setIsFilterBlocked(!isFilterBlocked)  
    }  
    break;  
  
case 'Maintenance':  
    if(isFilterOpen !== true || isFilterBlocked !== true){  
        setIsFilterMaintenance(!isFilterMaintenance)  
    }  
    break;
```

```
.maintenance-indicator{  
    margin: 0 auto;  
    background-color: #d67309;  
    width: 50%;  
    height: 7px;  
    border-radius: 15px;  
}
```

Mason

This iteration, I implemented the working and inactive camera sort components, as well as help to implement the under-maintenance camera filter. I also figured out the process for adding keyboard shortcuts, did some refactoring of the dashboard file, and create the iteration presentation.

```
// Keyboard shortcuts
useEffect(() => {
  const handleKeyPress = (event: KeyboardEvent) => {
    // If the c key is pressed
    if (event.key.toLowerCase() === 'c') {
      // Redirect to add camera page
      router.push('/add_camera');
    }
  };
  // EventListener is needed for keydown events
  document.addEventListener('keydown', handleKeyPress);

  return () => {
    document.removeEventListener('keydown', handleKeyPress);
  };
}, [router]);
```

```
// Cameras that have been updated within 10 minutes
if (isFilterWorking){
  if(calculateDifferenceInMinutes(item.timestamp) < 10){
    return item.status === "OPERATIONAL"
  }
}

// Cameras that have not been updated within 10 minutes
if (isFilterNotWorking){
  if(calculateDifferenceInMinutes(item.timestamp) > 10){
    return item.status === "INACTIVE"
  }
}
```

Bella

This iteration, after dealing with some technical difficulties, I spent my time researching and working on the requests page. As well as updating the user.md file

```
export default function requests() {

  const userFeedback = useUserFeedback();
  const params = useParams();
  const [requests, setRequests] = useState<string[] | null>([]);
  const id = Array.isArray(params.id) ? params.id[0] : params.id;

  const getRequests = (id: string): Request[] => {
    return userFeedback.flatMap((user) => {
      if (user.requests) {
        return user.requests.filter((request) => request.reportid === id);
      }
      return [];
    });
  };

  useEffect(() => {
    if (userFeedback.length > 0 && id) {
      const fetchedRequests = getRequests(id);
      setRequests(fetchedRequests);
    }
  }, [userFeedback, id]);
}
```

Mentor Feedback

- Feedback :
 - Positive feedback. Complimented the UI and the ideas we had for the following iteration features



Client Feedback

- Feedback:
 - Not very usable in its current state
 - Integration with mysql for the reports feature is essential.
- Changes Made:
 - Feedback is first priority for the 4th iteration



Planned **Iteration 4 Features:**

- MySQL integration for Reports feature
- Camera requests page UI and functionality

