First Interface Spring 2025 Report

By Lani Akinwale &
Daniel Kim

Goals

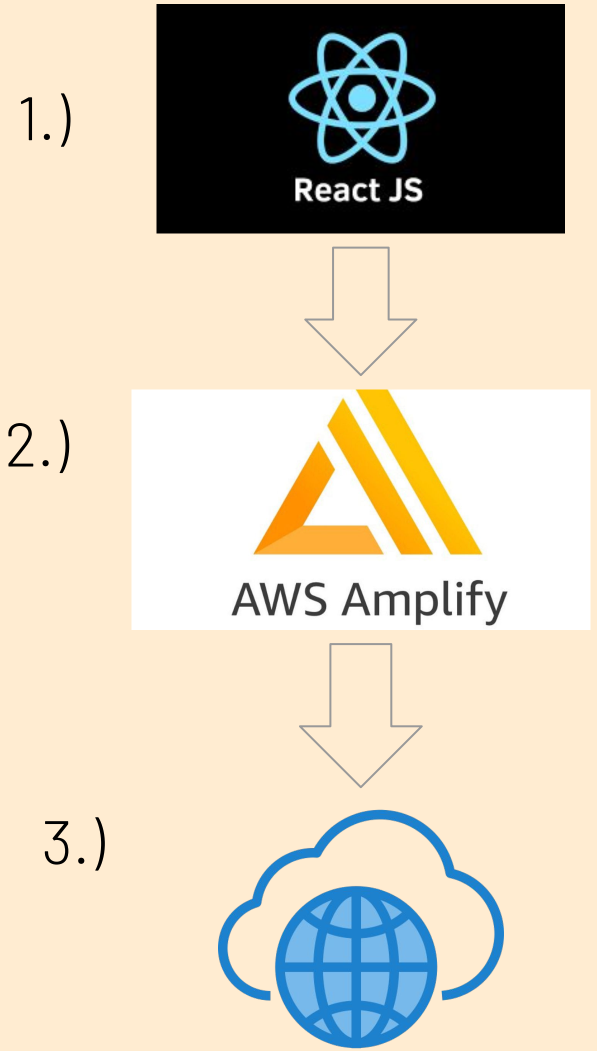
1. Create frontend (buttons, graphics, text) for Interface #1 website
2. Use RTMP to embed the OBS video feed into the website
3. Create backend (server and database that stores the website's data)
4. Create user authentication system for improved security

Revised Goals & Accomplishments

1. Created basic frontend user interface for UI (buttons, camera)
2. Created backend for website (onlines server, local server)
3. Connected local devices to backend servers
4. Created complex user interface for later user

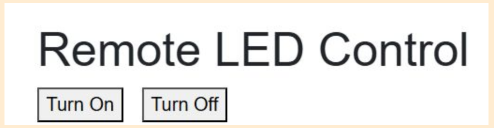
Frontend

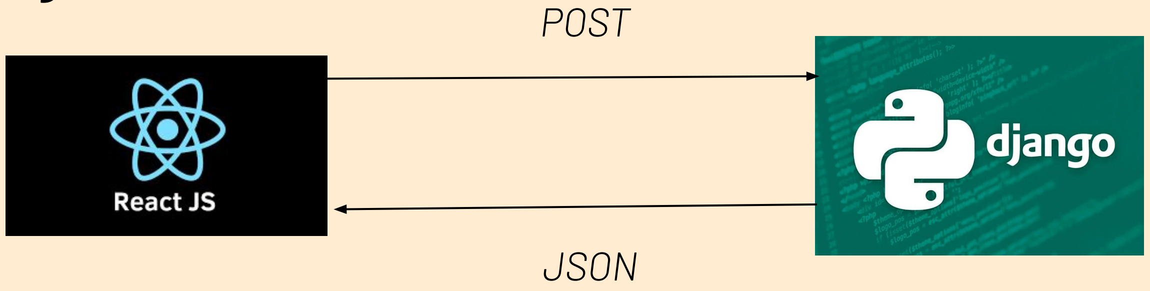
1. Used the React Javascript library to code the application
2. Amazon Amplify was used to build and deploy the application on the Internet
3. Users are then able to access the application on the Internet via a created URL



Frontend LED Component

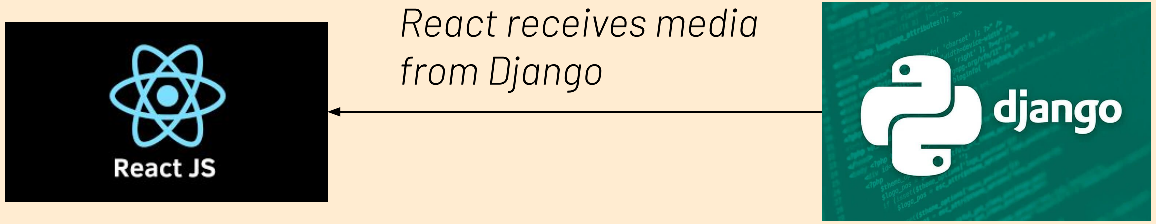
- React app sends a POST request (a type of HTTP request used to send data to a server) to either the */api/led/on/* or */api/led/off/* endpoint (or function) of the Django backend application depending on the button pressed on the UI
- When the Django application receives the POST request, it sends a JSON response containing data on the status of the ESP32 led back to the React application to confirm everything is working or signal an error





Frontend Camera Component

- The image HTML element uses its *src* attribute to retrieve the camera media displayed on the */vid_feed/* endpoint of the Django application

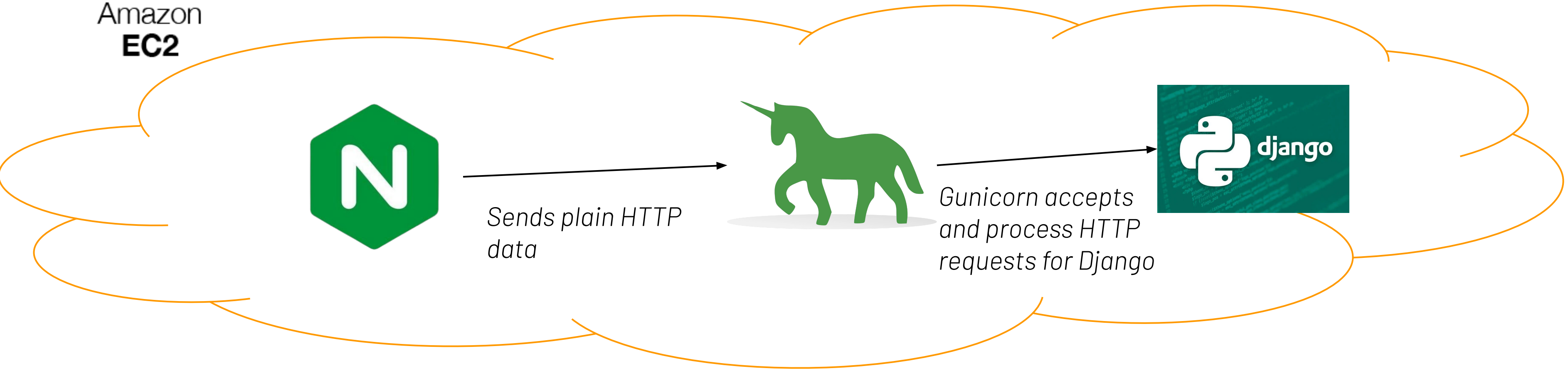


Backend

- Used the Django framework to code applications
- One Django application called *vistaBackend* is used to connect local server to the React application
- Second Django application called *vistaHardware* is used to connect vistaBackend and the local devices (ESP32 and camera)

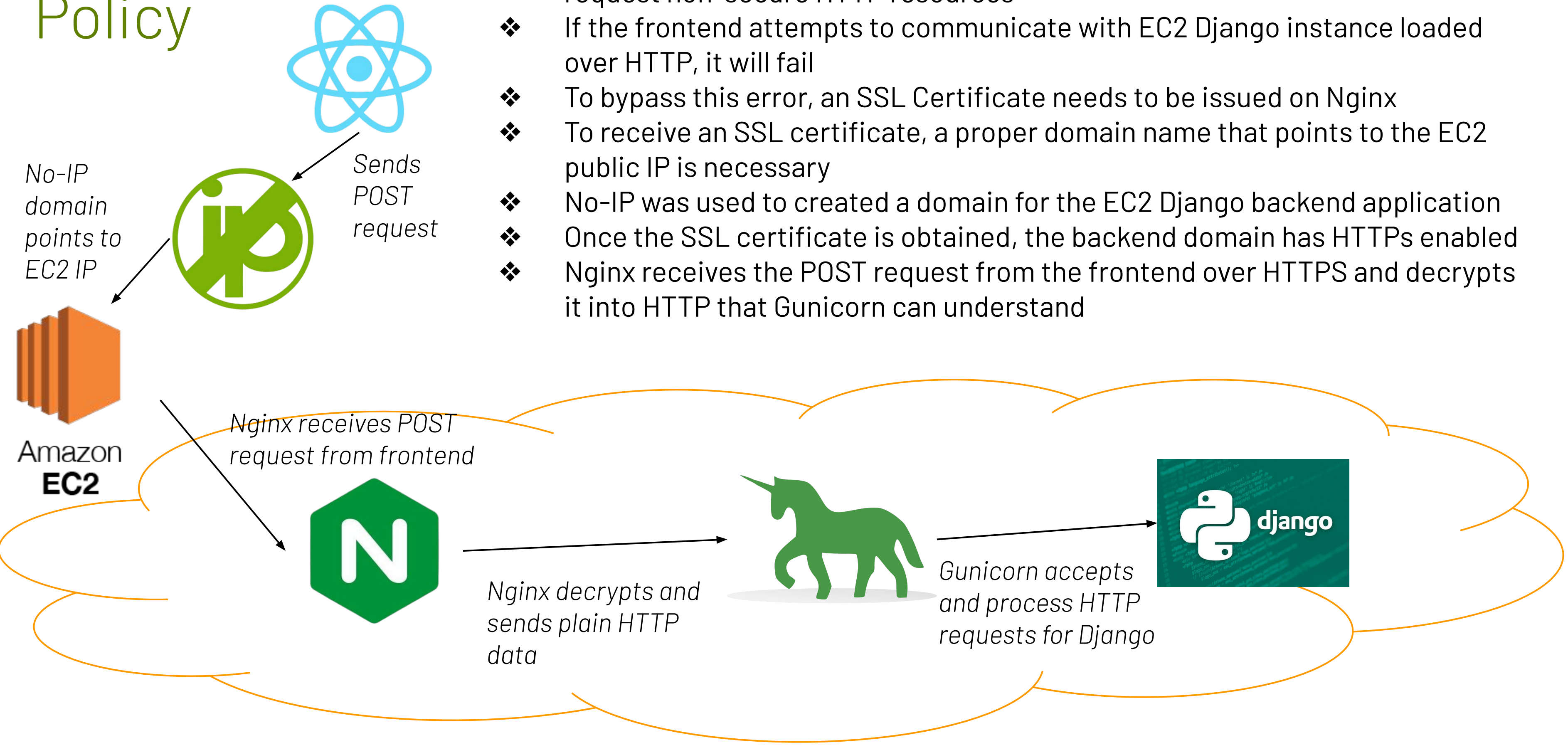
vistaBackend

- ❖ The vistaBackend uses an Amazon EC2 virtual machine. The virtual machine is in the cloud making it accessible from the Internet.
- ❖ Gunicorn is a Python WSGI HTTP server that allows the Django application handle HTTP requests (ex POST).
- ❖ Nginx is a high-performance web server and reverse proxy used to serve web applications:
  - ➢ Forwards requests to Django app running behind Gunicorn (turns HTTPs traffic into plain HTTP)
  - ➢ Decrypts HTTPs traffic using an SSL certificate



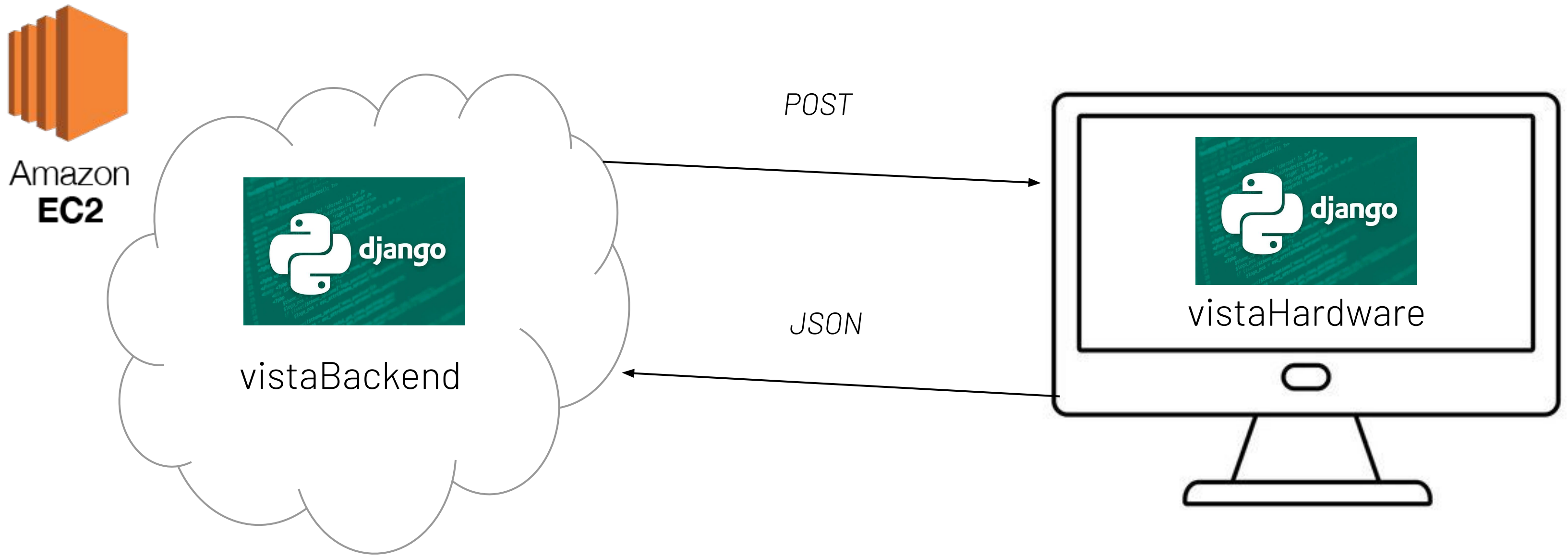
Mixed Content Policy

- ❖ Mixed Content Policy is an issue where an HTTPS webpage attempts to request non-secure HTTP resources
- ❖ If the frontend attempts to communicate with EC2 Django instance loaded over HTTP, it will fail
- ❖ To bypass this error, an SSL Certificate needs to be issued on Nginx
- ❖ To receive an SSL certificate, a proper domain name that points to the EC2 public IP is necessary
- ❖ No-IP was used to created a domain for the EC2 Django backend application
- ❖ Once the SSL certificate is obtained, the backend domain has HTTPs enabled
- ❖ Nginx receives the POST request from the frontend over HTTPS and decrypts it into HTTP that Gunicorn can understand


Sends POST request
No-IP domain points to EC2 IP
Amazon EC2
Nginx receives POST request from frontend
Nginx decrypts and sends plain HTTP data
django
Gunicorn accepts and process HTTP requests for Django

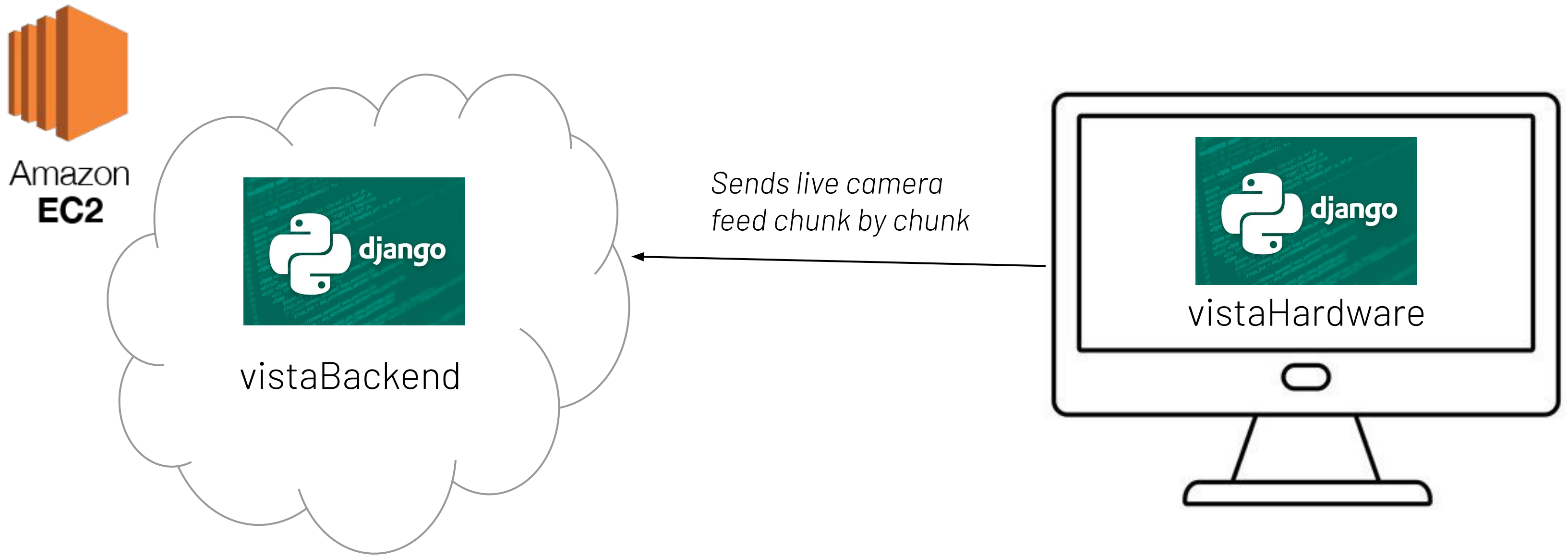
vistaBackend LED Component

- ❖ Either the *api/led/on* or *api/led/off* endpoint of vistaBackend receives a POST request from the frontend
- ❖ Once it receives a POST request from the frontend, vistaBackend sends a POST request to the */led/on/* or */led/off/* function of vistaHardware
- ❖ vistaHardware sends a JSON response containing the status of the ESP32 back



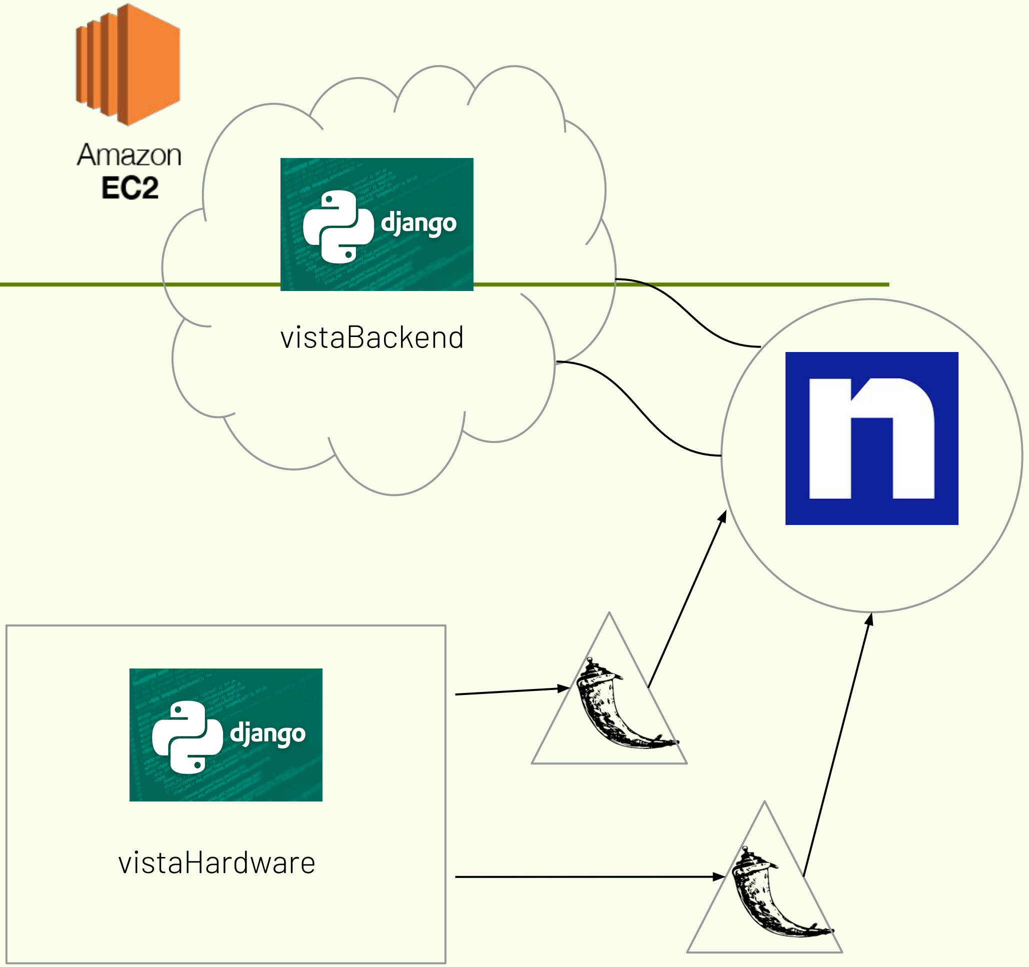
vistaBackend Camera Component

- The live camera feed on the local Django server, vistaHardware, is forwarded to the vistaBackend server
- The streamed content is sent directly to the vistaBackend chunk by chunk
- Going to the the online vistaBackend domain would show you the feed obtained from the local camera



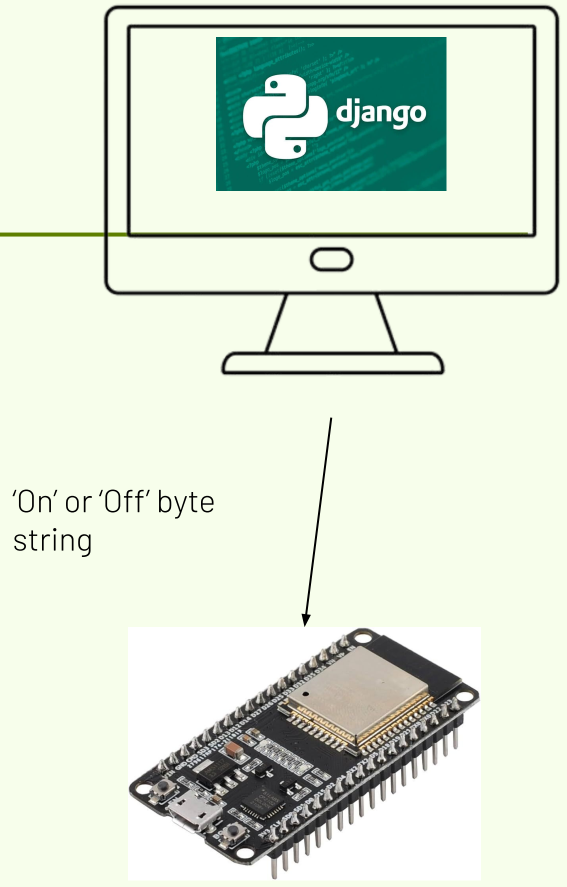
vistaHardware

- ➔ The reverse proxy *ngrok* is used to expose the vistaHardware servers to the Internet. This allows vistaHardware to receive the POST requests from vistaBackend.
- ➔ Two Flask servers are run locally. One for the camera component of vistaHardware and the other for the LED.
- ➔ Flask servers are used instead of Django servers due to their minimal memory usage and simple hardware integration



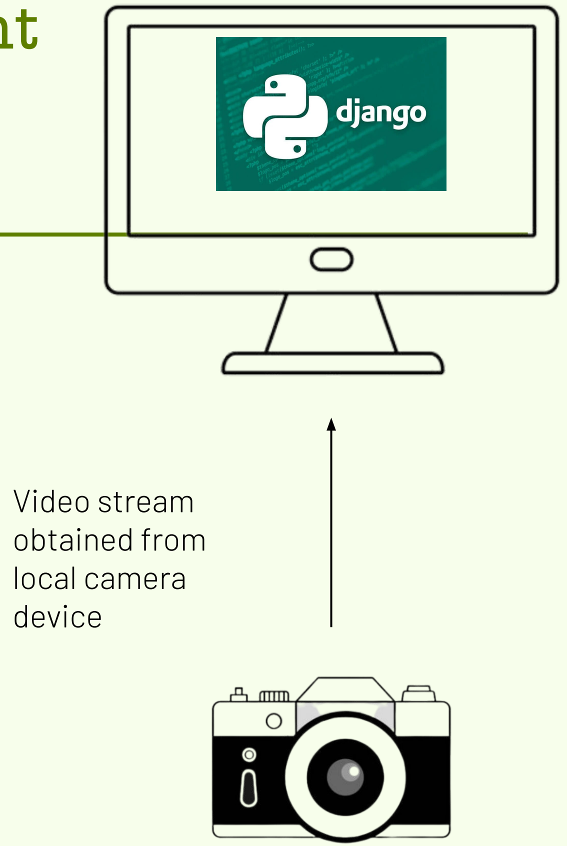
vistaHardware LED Component

- The LED component server receives a POST request from vistaBackend
- Either the */led/on/* or */led/off/* function receives and processes the POST request
- The two functions will send/write a binary string (‘On’ or ‘Off’) to the local ESP32 device



vistaHardware Camera Component

- ➔ The */vid_feed* function of vistaHardware generates a real-time video stream from the local camera device
- ➔ vistaBackend then retrieves the HTTP-served video stream from the ngrok exposed vistaHardware server



Pros and Cons of Virtual Lab Demo Structure

Pros:

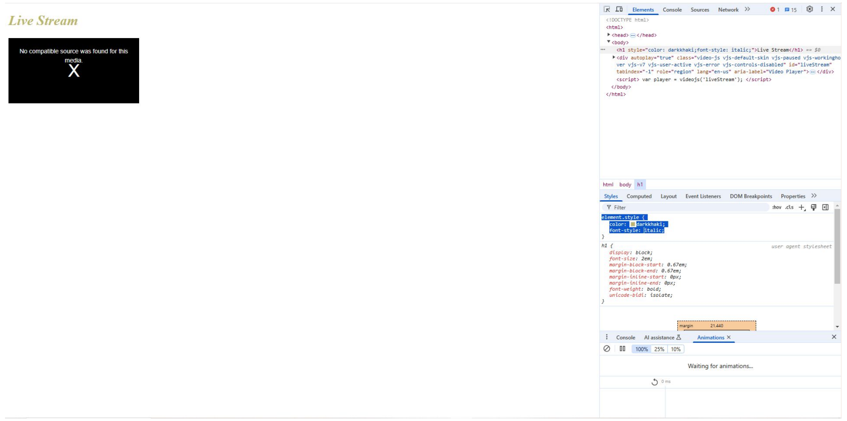
★ Latency speed of camera video feed and led control is relatively fast
★ Robust structure that allows for future scalability
★ Transmission of data works correctly
★ Can be used on campus wifi
★ Does not require additional devices or apps to operate

Cons:

★ Ngrok has a data limit (1 GB) which is easily reached when running the demo
★ Data transfer is inefficient
★ Structure is complex and prone to potential breakdowns

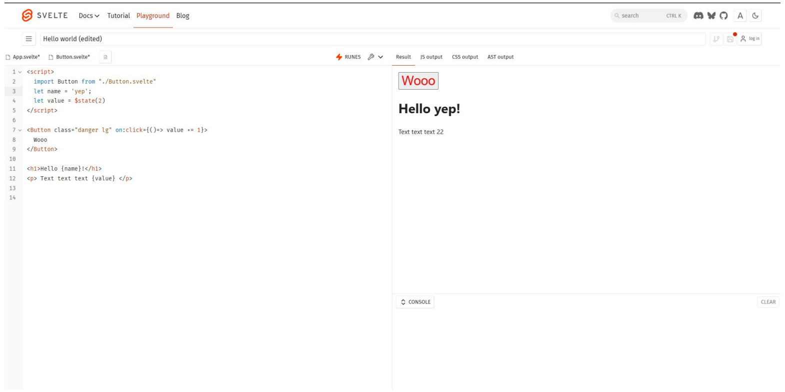
Design Document and UI Research - Daniel

- ❑ Searched for development framework (primarily tested Chrome Dev Tools and Svelte before team settled on React)
- ❑ Created Design Document for UI (with assistance from Alex)
- ❑ Started work on UI that incorporates the following from design doc:
  - ❑ Dedicated button for camera controls
  - ❑ Retractable menu for providing information
  - ❑ Controls such as buttons and sliders



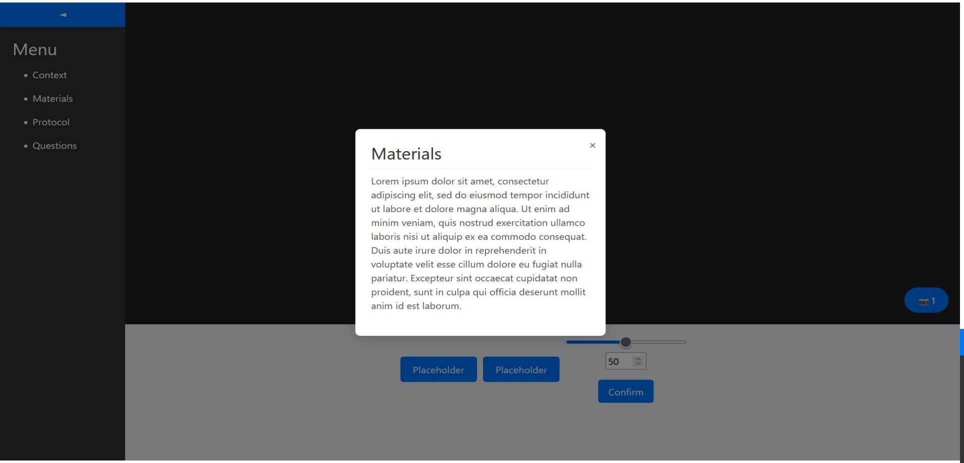
```
<!DOCTYPE html>
<html>
▶<head> … </head>
▼<body>
    <h1 style="color: darkkhaki;font-style: italic;">Live Stream</h1> == $0
  ▶<div autoplay="true" class="video-js vjs-default-skin vjs-paused vjs-workingho
    ver vjs-v7 vjs-user-active vjs-error vjs-controls-disabled" id="liveStream"
    tabindex="-1" role="region" lang="en-us" aria-label="Video Player"> … </div>
    <script> var player = videojs('liveStream'); </script>
  </body>
</html>
```

```
element.style {
    color: darkkhaki;
    font-style: italic;
}
h1 {                                          user agent stylesheet
    display: block;
    font-size: 2em;
    margin-block-start: 0.67em;
    margin-block-end: 0.67em;
    margin-inline-start: 0px;
    margin-inline-end: 0px;
    font-weight: bold;
    unicode-bidi: isolate;
}
```



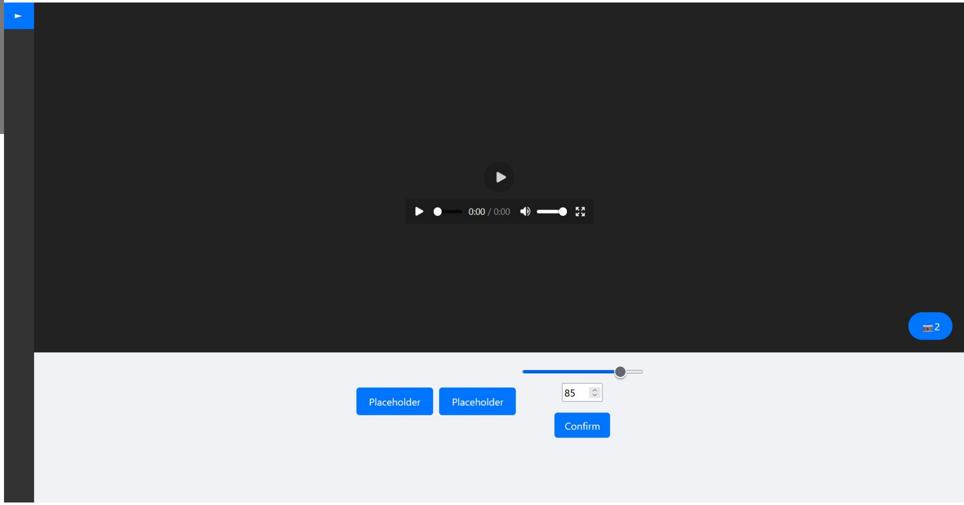
```
<script>
  import Button from "./Button.svelte"
  let name = 'yep';
  let value = $state(2)
</script>

<Button class="danger lg" on:click={()=> value += 1}>
  Wooo
</Button>

<h1>Hello {name}!</h1>
<p> Text text text {value} </p>
```

User Interface - Daniel



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Below: Demonstrates slider functionality and camera button usage (currently switches between two hypothetical cameras, more to be added)



Above: Demonstrates usage of submenus

Future Goals:

- ★ Change the structure of our virtual lab demo to make it more data efficient
- ★ Find alternatives to *ngrok* through the potential use of a router
- ★ Create a more complex frontend UI
- ★ Create a user authentication system for our frontend and backend systems
- ★ Overall, we want to simplify this process to improve scalability in the future