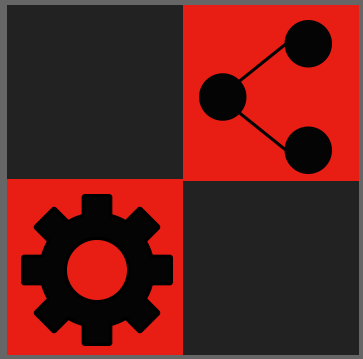
CHARTER AND GO

FRONT END TEAM

- ★ Dominic Prince
- ★ Gilbert Spietz
- ★ Sam Atienza
- ★ Steven Busateri

BACK END TEAM

- ★ Samuel Luke
- ★ Rickey Dendish
- ★ Ivan Barba
- ★ Peyton Triplett
- ★ Nicolas Banks



Final Product

FRONT END TEAM

Dominic Prince

Gilbert Spietz

Sam Atienza

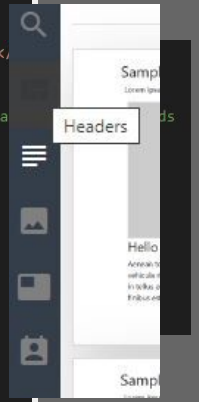
Steven Busateri

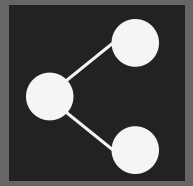
Dominic Prince

```
// dynamicaly adds areas to change blocks being edited in the inspector tab
return (
  <div>
    <div className='d-flex justify-content-between align-items-center'>
    <h5>Inspector</h5>
    <button className='btn btn-outline-danger btn-sm' onClick={() => this.handleDeleteBlock(blockUuid)}>Delete block</but
    </div>
    <hr />
    {Object.keys(config).map((el, index) => {
      if (config[el].type === 'string') {
        // names for the selected block
        return <div className='form-group' key={index}>
          <label>{config[el].name}</label>
          <DebounceInput
            debounceTimeout={500}
            type='text'
            className='form-control'
```

```
export default function NarrowSidebar(props) {
  return <div className="icons-wrapper bg-dark-blue d-flex flex-column justify-content-between">
  const submitButton = document.getElementById('submit-btn');     <button id="submit-btn">Submit</

  submitButton.addEventListener('click', () => {
    fetch(settings.url, {  //fetch returns a promise to server that is handled in the .then() a
      // methods, headers, and body from settings
      method: settings.method,
      headers: settings.headers,
      body: settings.body
    })
    .then(response => response.json())  //response sent to server as JSON
    .then(data => console.log(data))  //log to console
    .catch(error => console.error(error))  //catch any errors
  });
```

```
            <div>{this.props.name}</div>
            <button className='btn btn-outline-light btn-sm m-2'>Add block</button>
          </div>
        </div>
      </div>
    );
  }
}

BlockPreview.propTypes = {
  blockId: PropTypes.string,
  image: PropTypes.string,
  name: PropTypes.string,
  onPushBlock: PropTypes.func,
};

export default BlockPreview;
```





- Dynamically input forms to change data
- Displays name of block at the top
- Renders updates and deletions

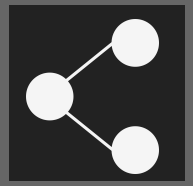
- Sends data to the database on click
- Adds text to the narrow sidebar buttons when you hover over them

- Renders the blocks into the preview bar

Sam Atienza

```
You, 3 days ago • Changed 'writeAPI' method, worked on integration

const postReq = 'This is a POST API Request';
const delReq = 'This is a DELETE API Request';
const getReq = 'This is a GET API Request';
const putReq = 'This is a PUT API Request';

function writeAPI() {
  fs.writeFile('/Users/Owner/Desktop/Backend.txt', postReq, err => {
    if (err) {
      console.error(err);
    }
  })
}


//Script attaches writeAPI function to an onClick event, sending info to Backend.txt
<script>
document.getElementById("article1").onclick = function() {writeAPI}
</script>
//console.log("The button has been clicked")
```

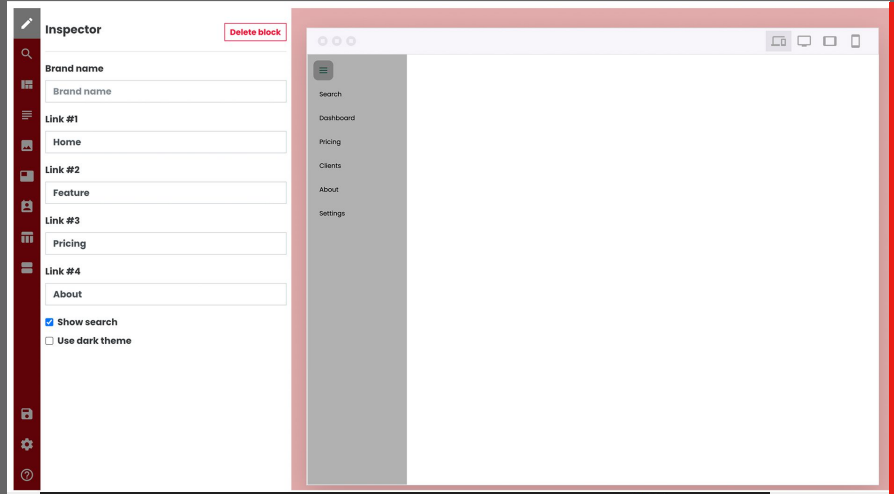
```
You, 22 hours ago | 2 authors (Dominic and others)
import React from 'react';

//Required to write to file
const fs = require('fs');

var settings = {
  "url": "https://vj6ip7fjl3.execute-api.us-east-2.amazonaws.com/dev/addSection",
  "method": "POST",
  "timeout": 0,  //believe this is to not tiimeout
  "headers": {
    "Content-Type": "application/json"  //indicate we are using JSON data
  },
  //Work on finishing body up      You, 3 days ago • Changes to integration on article1 …
  body: "Hello World",
  "data": JSON.stringify({
    "name": "article1"
  }),
};

fetch(settings.url, {  //fetch returns a promise to server that is handled in the .then() and .catch() methods
  // methods, headers, and body from settings
  method: settings.method,
  headers: settings.headers,
  body: settings.body
})
  .then(response => response.json())  //response sent to server as JSON
  .then(data => console.log(data))  //log to console
  .catch(error => console.error(error))  //catch any errors
```



- Writes to local text file on machine to simulate connection
- Sets up the settings variable
- Fetch function pulls in data set up in the settings variable for API requests
- Future improvements utilize a helper function to improve functionality of submit button in further versions

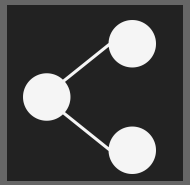
- Worked in Styles.css
  - Improving appearance and theme

- Worked in Navbar 1
  - Horizontal to Vertical
  - Retractable button
  - Matching theme




```css
body {
    background: #e3acac !important;
    width: 100vw;
    height: 100%;
    padding: 0;
    margin: 0;
    overflow-x: hidden;
}

.btn-sidebar:hover {
    background: #5c5b5b;
}

.active-button {
    background: #707070;
    color: #fff!important;
}

.btn-sidebar {
    color: #8d8d8d;
    padding: 0.75rem;
    display: flex;
    justify-content: center;
    align-items: center;
}

.btn-sidebar:hover {
    color: #fff;
}

.shadow {
    -webkit-box-shadow: 0px 0px 22px -1px rgba(0,0,0,0.25);
    -moz-box-shadow: 0px 0px 22px -1px rgba(0,0,0,0.25);
    box-shadow: 0px 0px 22px -1px rgba(0,0,0,0.25);
}

.icons-wrapper {
    width: 4rem;
    min-width: 32px;
    overflow-y: hidden;
    height: 100vh;
    background: rgb(96, 3, 8) none repeat scroll 0% 0%;
}

.inspector-wrapper {
    overflow-y: scroll;
    height: 100vh;
    width: 600px;
    background: white;
}
```

Steven Busateri



- Help with integrating front and back end code.
- Tidying syntax.
- Debugging.
- Helped out where I could as the least experienced programmer but watched others and provided input where I could.

Gilbert Spietz

Ricky Dendish

Peyton Triplett

Ivan Barba

Samuel Luke

Nicolas Banks

BACK END TEAM

Ricky Dendish

add_Section

We load data from the body and get the name from it.

Then that name gets put into the sections table on the condition it isn't already there.

We return status code, headers, and body.

```python
#These are our Dynamodb tables
SECTIONS_TABLE = os.environ['SECTIONS_TABLE']
IMAGES_TABLE = os.environ['IMAGES_TABLE']
TEXT_TABLE = os.environ['TEXT_ELEMENTS_TABLE']

dynamodb = boto3.resource('dynamodb', 'us-east-1')

client = boto3.client('dynamodb')

#this happens when the user creates a section
def add_section(event, context):
    data = json.loads(event['body'])

    name = '{}'.format(data['name'])  #retrieves the name that was passed into this function, and formats it to a string

    imgContent = []
    txtContent = []

    #here we are adding an item to the Sections table
    resp = client.put_item(
        TableName=SECTIONS_TABLE,
        Item={
            'name': {'S': name },
            'ImageContent': {'L': imgContent },
            'TextContent': {'L' : txtContent }
        },
        ConditionExpression='attribute_not_exists(#name)',    #this checks if the name already exists
        ExpressionAttributeNames={"#name": "name"}
    )

    body = {
        "name": name
    }

    response = {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json", "Access-Control-Allow-Origin": "*"},
        "body": json.dumps(body)
    }

    return response
```

insert_image

We grab section, image_id, name, description, width, height, and url data from the body and format it. We add all this to the images table and add it to the image section of the sections table.

We return status code, headers, and body.

```python
def insert_Image(event, context):
    data = json.loads(event['body'])

    #retrieves the information about the image that was passed into the function
    section = '{}'.format(data['section'])

    img_id = '{}'.format(data['img_id'])
    name = '{}'.format(data['name'])
    description = '{}'.format(data['description'])

    height = '{}'.format(data['height'])
    width = '{}'.format(data['width'])
    url = '{}'.format(data['url'])

    #the string representing our html element
    html = '<img src="'+ url +'" alt="'+ description + '" width="' + width + '" height="' + height + '" />'

    #here we are adding an item to the images table
    resp = client.put_item(
        TableName=IMAGES_TABLE,
        Item={
            'section': {'S': section},
            'img_id': {'S': img_id},
            'name': {'S': name},
            'description': {'S': description},
            'height': {'S': height},
            'width': {'S': width},
            'url': {'S': url},
            'html': {'S': html}
        }
    )

    #updating the content column of the section we placed the image in. It will hold the string representing the html image element
    client.update_item(
        TableName=SECTIONS_TABLE,
        Key={
            'name': {'S': section}
        },
        UpdateExpression='SET ImageContent = list_append(if_not_exists(ImageContent, :empty_list), :imgContent)',
        ExpressionAttributeValues={
            ':imgContent': {'L': [{
                'S': img_id
            }]},
            ':empty_list':{'L': []}
        }
    )

    body = {
            "section": section,
            "img_id": img_id,
            'html' : html
    }

    response = {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json", "Access-Control-Allow-Origin": "*"},
        "body": json.dumps(body),
    }

    return response
```

Ivan

```python
def insert_Text(event, context):
    data = json.loads(event['body'])

    #retrieving the information about the text element from the data passed into the function
    section = '{}'.format(data['section'])
    text_id = '{}'.format(data['text_id'])

    content = '{}'.format(data['content'])

    unorderedList = data['ul'] #checks if the user requested an unordered list
    orderedList = data['ol']
    header = data['header']

    #here i'm splitting the string on the comma, so that we can access each element individually
    #if there is a better way to do this feel free to make changes
    if orderedList or unorderedList:
        content = content.split(", ")

    list_html = ''
    html = ''

    #checks if a list was requested. we are generating li elements for each element in the list
    if unorderedList or orderedList:
        for item in content:
            list_html += '<li>' + item + '</li>\n'
```

insert_Text pt. 1

First, we parse the the following from the data loaded into this function: section, text_id, and content. Using the given content we can format it according to the user's preference.

```python
#the html string will be made with either <p>, <ul>, or <ol> tags depending on what was requested
if unorderedList == False and orderedList == False and header == False:
    html = '<p>' + content + '</p>'
elif unorderedList:
    html = '<ul>' +  list_html + '</ul>'
elif orderedList:
    html = '<ol>' +  list_html + '</ol>'
elif header:
    html = '<h3>' + content + '</h3>'

#places the item in the text table
resp = client.put_item(
    TableName=TEXT_TABLE,
    Item={
        'section': {'S': section},
        'text_id': {'S': text_id},
        'header': {'BOOL': header},
        'ul': {'BOOL': unorderedList},
        'ol': {'BOOL': orderedList},
        'html': {'S': html}
    }
)
```

insert_Text pt. 2

After we have the content from the front end, we can place it into one of four possible designations. Paragraph ( ), unordered list, ordered list, or header.

Then we just add it to our database's text table.

Ivan

client_update_item

After adding the content with all the parameters we looked at into the database, then we can update the sections table to reflect the addition of the new text.

```
#updates the section table and its content attribute. We place the html string in there
client.update_item(
    TableName=SECTIONS_TABLE,
    Key={
        'name': {'S': section}
    },
    UpdateExpression='SET TextContent = list_append(if_not_exists(TextContent, :empty_list), :txtContent)',
    ExpressionAttributeValues={
        ':txtContent': {'L': [{
            'S' : text_id
        }]},
        ':empty_list':{'L': []}
    }
)

body = {
        "section": {'S': section},
        "text_id": {'S': text_id},
        "html": {'S': html}
}

response = {
    "statusCode": 200,
    "body": json.dumps(body),
}

return response
```

Peyton Triplett

getHandler.py

```python
#These are our Dynamodb tables
SECTIONS_TABLE = os.environ['SECTIONS_TABLE']
IMAGES_TABLE = os.environ['IMAGES_TABLE']
TEXT_TABLE = os.environ['TEXT_ELEMENTS_TABLE']

dynamodb = boto3.resource('dynamodb', 'us-east-1')

client = boto3.client('dynamodb')

def get_Sections(event, context):

    #Retrieves all the names of the sections in the section table that have the 'name' attribute (all of them)
    table = dynamodb.Table(SECTIONS_TABLE)
    resp = table.scan(ProjectionExpression = '#nme', ExpressionAttributeNames = {'#nme': 'name'})['Items']

    response = {
        'statusCode': 200,
        "headers": {"Content-Type": "application/json", "Access-Control-Allow-Origin": "*"},
        'body': json.dumps(resp)
    }

    return response
```

- Define constants for our table names
- Connect to DynamoDB using imported library

```python
def get_Section_Content(event, context):

    #we get the name of the section that was passed to this function
    section_name = '{}'.format(event['pathParameters']['secName'])

    #we retrieve the item from the section table that has the same name as what was passed into the function
    table0 = dynamodb.Table(SECTIONS_TABLE)
    resp = table0.query(KeyConditionExpression=Key('name').eq(section_name))
    items = resp.get("Items", None)

    #these are the lists that hold the id's of the Image and Text Elements
    ImageContent = items[0]['ImageContent']
    TextContent = items[0]['TextContent']

    #these lists will be storing the id's of the images and text items that are in the section
    imgList = []
    txtList = []

    table1 = dynamodb.Table(IMAGES_TABLE)
    table2 = dynamodb.Table(TEXT_TABLE)

    #Searching through the Image Table to find the element id that matches the current value in ImageContent. We add the html of that element to imgList
    for val in ImageContent:
        resp1 = table1.query(KeyConditionExpression=Key('img_id').eq(val))
        items1 = resp1.get("Items", None)
        imgList.append({'id':val, 'html':items1[0]['html']})

    #Searching through the Text Table to find the element id that matches the current value in TextContent. We add the html of that element to txtList
    for val in TextContent:
        resp2 = table2.query(KeyConditionExpression=Key('text_id').eq(val))
        items2 = resp2.get("Items", None)
        txtList.append({'id':val, 'html':items2[0]['html']})

    #This is what we will be returning in the response
    sectionContent = {'images':imgList, 'text':txtList}

    response = {
        'statusCode': 200,
        "headers": {"Content-Type": "application/json", "Access-Control-Allow-Origin": "*"},
        'body': json.dumps(sectionContent)
    }

    return response
```

- get_Section_Content retrieves the name and item from the selection table that was passed into the function
- Searches for the matching Image and Text id stored in the table and adds the corresponding HTML to the list
- Returns the updated imgList and txtList

Nicolas Banks

delHandler.py > ...

```python
import json
import os
import boto3
import botocore
from boto3.dynamodb.conditions import Key

#These are our Dynamodb tables
SECTIONS_TABLE = os.environ['SECTIONS_TABLE']
IMAGES_TABLE = os.environ['IMAGES_TABLE']
TEXT_TABLE = os.environ['TEXT_ELEMENTS_TABLE']

dynamodb = boto3.resource('dynamodb', 'us-east-1')

client = boto3.client('dynamodb')
```

- Define constants for our table names
- Connect to DynamoDB

```python
def del_Section(event, context):
    #retrieving what was passed in to this function
    section_name = '{}'.format(event['pathParameters']['secName'])

    #deleting the item from the section table that has the same name as what was passed in as secName
    table = dynamodb.Table(SECTIONS_TABLE)
    resp = table.delete_item(
        Key={
            "name": section_name
        }
    )

    response = {
        'statusCode': 200,
        "headers": {"Content-Type": "application/json", "Access-Control-Allow-Origin": "*"},
        'body': "Successfully deleted the section"
    }

    return response
```

- del_Section(), del_Text(), del_Image()
- Deletes item matching section_name from SECTIONS_TABLE
- Latter two are similar, but they also update SECTIONS_TABLE after deleting the text/image

Issues Encountered

Front End Team

- Correct connection of front-end to back-end using different programming languages.
- Navbar vertical placement
- Lack of familiarity with AWS and serverless framework.
- Too clunky

Back End Team

- Lack of familiarity with AWS and serverless framework.

Future Improvements

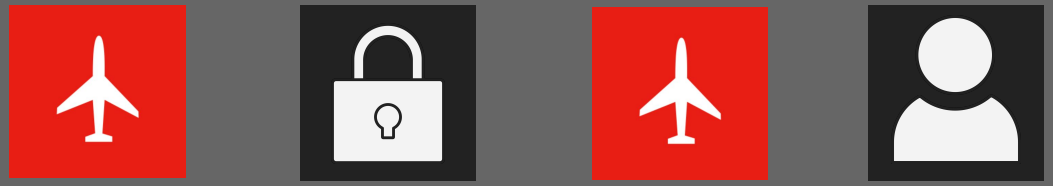
Front End Team

- Front end Navbar Integrated
- More decorated UI and menu for website.
- Help and questions tab.

Back End Team

- Improved integration with the use of environment variables.

Thank you!



Questions?