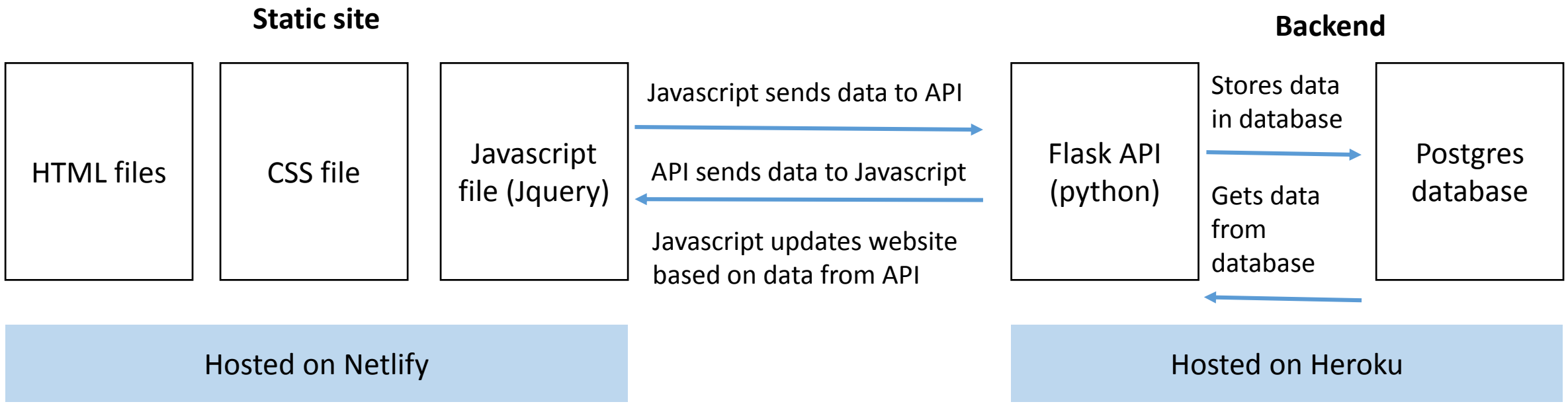
Interactive blog

www.baybridgebio.com

Structure

- Static site
  - HTML, CSS and JS files
  - Upload to github repository
  - Hosted on Netlify
- Flask backend (Flask is a Python “micro-framework”, a set of pre-made tools for making web apps, written in Python)
  - Flask-RESTful API framework
  - Flask-SQLAlchemy ORM
  - Postgres database
  - Hosted on Heroku
- Static site is “dumb”; can’t interact with a user, process data, store data
- Think of HTML like Microsoft Word, CSS like a way to format that word doc in Powerpoint, and Python like Microsoft Excel that does the math / logic that goes into the Word / Powerpoint


Static site
Backend
HTML files
CSS file
Javascript file (Jquery)
Javascript sends data to API
API sends data to Javascript
Javascript updates website based on data from API
Flask API (python)
Stores data in database
Gets data from database
Postgres database
Hosted on Netlify
Hosted on Heroku

Static site

- Make a site with HTML, CSS and Bootstrap
  - Learn how to build sites with HTML and CSS in a day: https://www.codecademy.com/learn/make-a-website
  - Use a free text editor like Atom (https://atom.io/) or an Integrated Development Environment like Cloud9 (can get access for free with EdX's CS50 course https://www.edx.org/course/cs50s-introduction-computer-science-harvardx-cs50x)
  - Create a folder on your computer and save your HTML and CSS files there (as well as any images or other files you'll include on your site)

Static site

- Make your site accessible on the internet (“deploy” it)
  - Set up a Github account and create a “repository” for your project (many tutorials online)
  - Add your HTML, CSS and other files to your repository (helps to learn basic terminal commands for this)
  - Create a Netlify account, a new Netlify site, and connect your Netlify site to your github repository
  - Can post any updates to your site just by pushing them to github
- Optional: set up a custom domain name
  - Buy a domain name from namecheap or another provider

API – concept

- An API is basically a way for computers (as opposed to human users) to interact with a website
- In our case, the API makes a “dumb” static site “smart” by adding logic and data handling behind the scenes
- Imagine you are leading a meeting, a colleague is at the meeting listening, and another colleague is at the home office on standby
  - The colleague with you at the meeting listens for any questions, texts the colleague at the home office, who finds the answers and responds
  - You are the HTML / CSS, your colleague at the meeting is the Javascript, your colleague at the home office is the API

API – concept

- The static site is hosted on Netlify; when people go to your site, Netlify sends the user's browser your HTML / CSS, then the browser displays that as a site
- The HTML and CSS files have no "logic" and can't store or process user-provided data
- A separate file, written in Python (or another programming language), can collect, process and store user data
- We create a Javascript file that sends data to and receives data from the API, then changes the static site based on data from the API
- The Javascript file acts as a "messenger" sending data to and from the static site and the API

API: implementation

- Written in Python, using Flask micro-framework
  - http://flask.pocoo.org/
- Uses Flask-RESTful for handling API “requests” and “responses”
  - https://flask-restful.readthedocs.io/en/0.3.5/quickstart.html#endpoints
- Uses Flask-SQLAlchemy for communicating between API and database
  - http://flask-sqlalchemy.pocoo.org/2.3/
- Restful Flask API example
  - https://codeburst.io/this-is-how-easy-it-is-to-create-a-rest-api-8a25122ab1f3
  - https://www.sqlalchemy.org/
- To learn about these concepts, you can check out CS50’s lectures http://cs50.tv/2017/fall/ (taking the whole course is highly recommended if you want to learn about programming)
  - Basic web programming concepts: lecture 6
  - Intro to python: lecture 8
  - Web programming with Python and Flask: lecture 9 (note this project uses Flask in a different way than the lecture does)

Connecting static site and API

- With this structure, the static site and API are separated and hosted on different servers
- We need a way to get data from the static site that the user interacts with in the browser, to the API, so we can store and process that data
- We send data back and forth with a Javascript file
- The Javascript file is hosted with the HTML and CSS and sent to the browser along with the other static site files
- The Javascript file “listens” for a user’s interaction with specific parts of the static site, and then sends data to the API via an “HTTP request”
- The Javascript file waits for a “response” from the server, then updates the static site accordingly
- We use an Ajax library called axios to handle these “asynchronous” requests and responses with Javascript

One note: CORS

- Usually, servers can only handle requests that come from the same “origin”
- With our structure, our static site and API have a different origin, so the requests from our static site to our API will be blocked by the browser
- We need to tell our API that requests from our static site are safe
- Helpful resources
  - https://stackoverflow.com/questions/10636611/how-does-access-control-allow-origin-header-work
  - https://stackoverflow.com/questions/26980713/solve-cross-origin-resource-sharing-with-flask

Deploying the API

- We will deploy the API using Heroku
  - Good tutorial: https://devcenter.heroku.com/articles/getting-started-with-python
- Deploying to Heroku is somewhat similar to deploying on Netlify
  - Connect to github repo and deploy from terminal
- However there are differences
  - Need a Procfile (very simple file)
  - Make sure you have an __init__.py file (just a blank file with that title)
  - Need a requirements.txt file with all your app's dependencies, and need to install gunicorn
    - https://devcenter.heroku.com/articles/python-gunicorn
- If you want a database, you can easily provision a Postgres database with Heroku