Documentation

Version: 1.0 // Updated on 2022-06-19

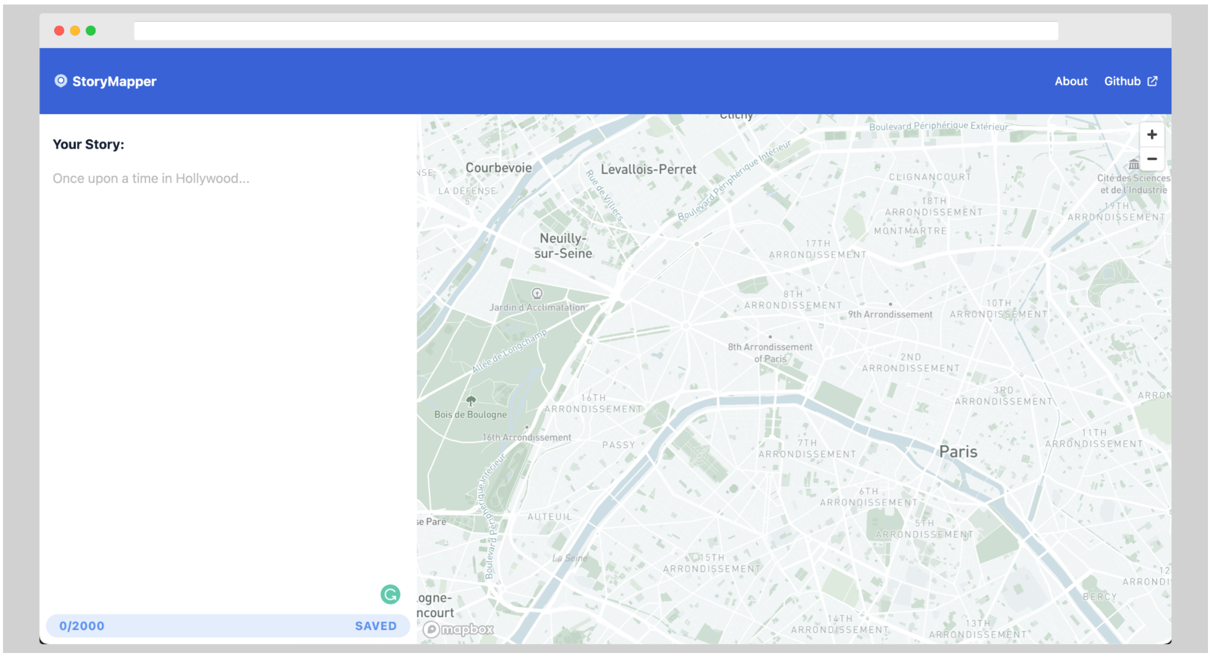
The Idea

I work part-time as a Frontend Developer in a travel agency. We often have requests from customers to build very specific holidays for them. This gave me the idea to create a tool that allows us to visualise the places they want to visit on a map. As a bonus, the route should also be visible to determine if a given journey is doable in a day or if they need another stop along the way.
I also wanted to incorporate NLP (natural language processing) to extract more context out of the customer request. For example, the number of days they want to stay in a place.
The interface should be very simple and minimalistic. A text input field and a map are all the tool needs.

Development process

I started with a blank Vite project. Vite offers a fast build tool and development server. It has presets for most frontend frameworks like Vue, Svelte and React. I chose React because that’s what I’m most fluent in. For the Backend, I wanted to work with serverless python functions. Unfortunately, I hit the limitations of those very

quickly. I switched to Flask which gave me more flexibility and also allowed me to use a bigger dataset for the NLP features.
Once I was happy with the Tech-stack, I began with simple test inputs that I sent to the Backend. The first version just tokenised the input and compared the token to a huge list of cities (over 100k). The results were not bad. The biggest problem I had was that there are many cities with the same name. Mostly American cities with their European counterparts. So somehow I had to guess which city the user truly means. Therefore I implemented more NLP than I initially anticipated. I made use of SpaCys Named Entity Recognition feature to detect cities, countries and points of interest. Once I had a list of all geographical entities I had to find the coordinates. I evaluated the Google Maps API and Mapbox. Mapbox seemed like the better fit, because of its more generous free tier of 100’000 Geocoding requests. It also offers a variety of settings to narrow down the search for a geographical entity. Now all I had to do was to display the coordinates on the map. And with that the prototype was ready.



Milestones

The first milestone was reached when the first pins appeared on the map. The next milestone was to be able to remove places that are incorrectly interpreted. This wasn’t a big challenge because I used a very flexible text editor called Slate. It allowed me to highlight parts of the text and replace them with a custom React component. If the user now hovers over a geographical entity a tooltip appears with an option to remove the entity from the map and text. This works as a minimal version but I see the potential for more options like an edit feature where a user could add more context to the entity like for example the country. This would help the parser with the decision-making.

Edge cases

Most geographical entities weren't a problem for the parser. But once in a while, certain words posed a problem. For example places with a “Saint” in the name. The list of cities I use to look up if a place is a city uses a dashed convention. But in the UK we mostly shorten saint with “St”. To fix that I had to check each entity if there is a “St” in front of it. I did the same for the word “San”.

```
# handle prefixes:
prefixes = {
  "St": "Saint-",
  "San": "San ",
}

search_query = token.text
lefts = [t.text for t in token.lefts]

if len(lefts) > 0:
  prefix = lefts[0]
  # check if token has prefix
  if prefix in prefixes:
    search_query = prefixes[prefix] + token.text

city = _find_city(search_query)
```

Another issue I had was one I call the “(New) York” issue. It happened when a city had the same name as another one but with a prefix on it. The easy solution for that one was to order the list of cities by the number of characters. This means the string “New York” will always find the city New York before York.
The biggest issue I had were with cities that had the same name as another city. I found that the solution that worked the best was to set a point on the map from where the Mapbox would start the search for the place. Every time I parse a new place I would recalculate the starting point based on the centre point of all the already parsed entities.

```
def get_center_point(entities):
 if len(entities) == 0:
   return None

 lats = []
 lngs = []

 for entity in entities:
   if "lat" in entity and "lng" in entity:
     lats.append(entity["lat"])
     lngs.append(entity["lng"])

 if len(lats) == 0 or len(lngs) == 0:
   return None

 return {
   "lat": sum(lats) / len(lats),
   "lng": sum(lngs) / len(lngs)
 }
```

Final outcome

For the deployment, I initially wanted to use Vercel because they offer Python lambda functions. For the reasons I explained above in the text, I switched to a Flask application for the Backend. The easiest way to deploy a Flask app is in my mind Heroku. They have a generous free tier which allows me to host a relative fast application for free. The only drawback is that the server shuts down after a while, which means it takes a second to restart it once a user tries to parse their first text.
The Frontend is hosted on Netlify as a static React single-page application. Both Netlify and Heroku are connected to the GitHub Repository. When I now commit to the repository, both the Backend and the Frontend will redeploy and serve the most recent version of the app.

Roadmap

Initially, I wanted this to be a tool for travel agents to quickly visualize a customer's request for a complex holiday. This is still the goal but wanted to make sure that the parsing of places works well before I take care of the more complex NLP tasks. I certainly want to extract more data from the user input like dates they mention or the number of nights they want to stay in a certain place. I also want to add some good unit tests to ensure the good quality of results, while I add new features.

Reflections

Overall I am really pleased with the result. The process felt smooth and I was never stuck on a specific problem for a long time. When I hit a bump in the road I just started working on another part of the application and usually, I got an idea of how to solve the problem not too long after. I did learn a lot about NLP throughout the process. I'm now eager to go deeper down that route because I do see the benefits of it. I did especially like working with SpaCy. It allowed me to easily parse text and extract all sorts of information. I know that I only scratched the surface with this tool but this gives me confidence that I made the right decision in using it.

The application is available on https://storymapper.netlify.app/