Quaternary Star System
Elizabeth Doss, Yevgeniy Gorbachev, Kevin Li, & Emory Walsh
SoftDev1 pd1
P#01 -- ArRESTed Development
2019-11-14

APIs Using:

| | |
|---|---|
| Wikipedia | https://en.wikipedia.org/w/api.php |
| Wolfram Alpha | http://developer.wolframalpha.com/portal/myapps/index.html |
| NASA Exoplanet | https://exoplanetarchive.ipac.caltech.edu/docs/program_interfaces.html |

**Minimum Viable Product:**

The amount of APIs searched through depends on the keywords – something like "how long to reach {{exoplanet}} with/using Merlin 1C and 1000 tons of fuel" will involve all three APIs and work like this:

- Search NASA's exoplanets API for the planet
- If two exoplanets are named, send an equation to Wolfram|Alpha to get the distance
- Search Wikipedia's API for the engine/rocket - In particular, get information about thrust and vacuum specific impulse.
- Use the data gathered from the two above APIs to send a request containing an equation[1] to Wolfram|Alpha, which will return a result
- Return the result and statistics about the queried engine and exoplanet (underneath)

For every query, straight-line distance ignoring significant gravitational effects will be assumed - as though the spacecraft is starting in interstellar space near the Sun.

For queries for mass ratio, it will be assumed that maximum thrust will be maintained throughout the trip.

**If we have extra time:**

- Registration/login feature
- User profiles that can save specific pages to favorites
- Provide information that the user may be interested in based on their previous searches

[1] To be determined later - we will need to do some physics ourselves for that

**Task Division:**

Kevin - Project Manager

Emory - Frontend: creating templates, routing, styling with bootstrap

Dependencies:

- search.search(query: str) -> dict

Elizabeth - Connecting to APIs

Tasks:

- api_bus.wolfram(query: str) -> dict
- api_bus.wikipedia(query: str) -> dict
- api_bus.exoplanets(query: str) -> dict

Yevgeniy - Evaluating queries, storing in and searching cache

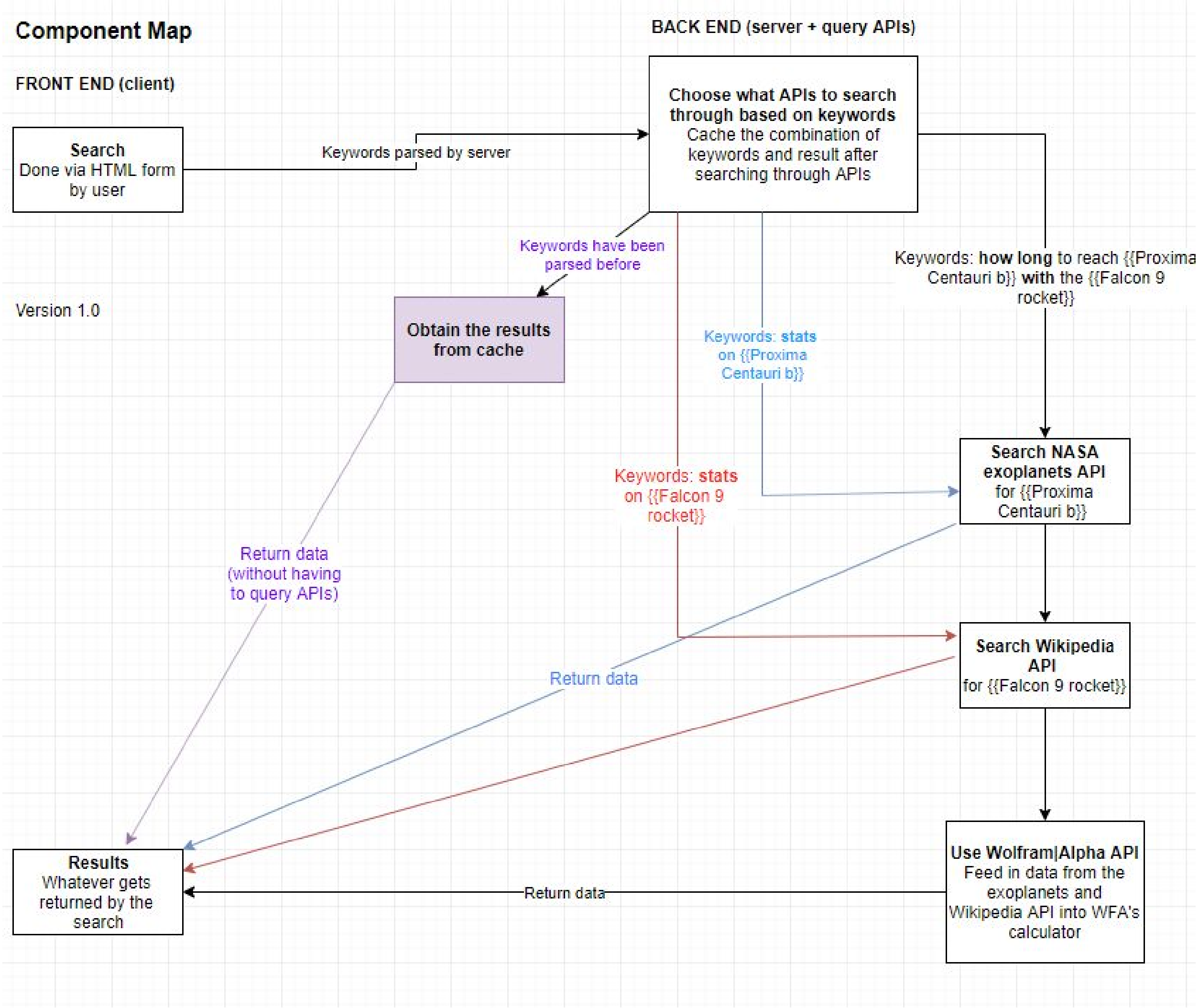
Tasks:

- search.search(query: str) -> dict
- cache.search(contents: dict) -> dict
- cache.store(contents: dict) -> dict

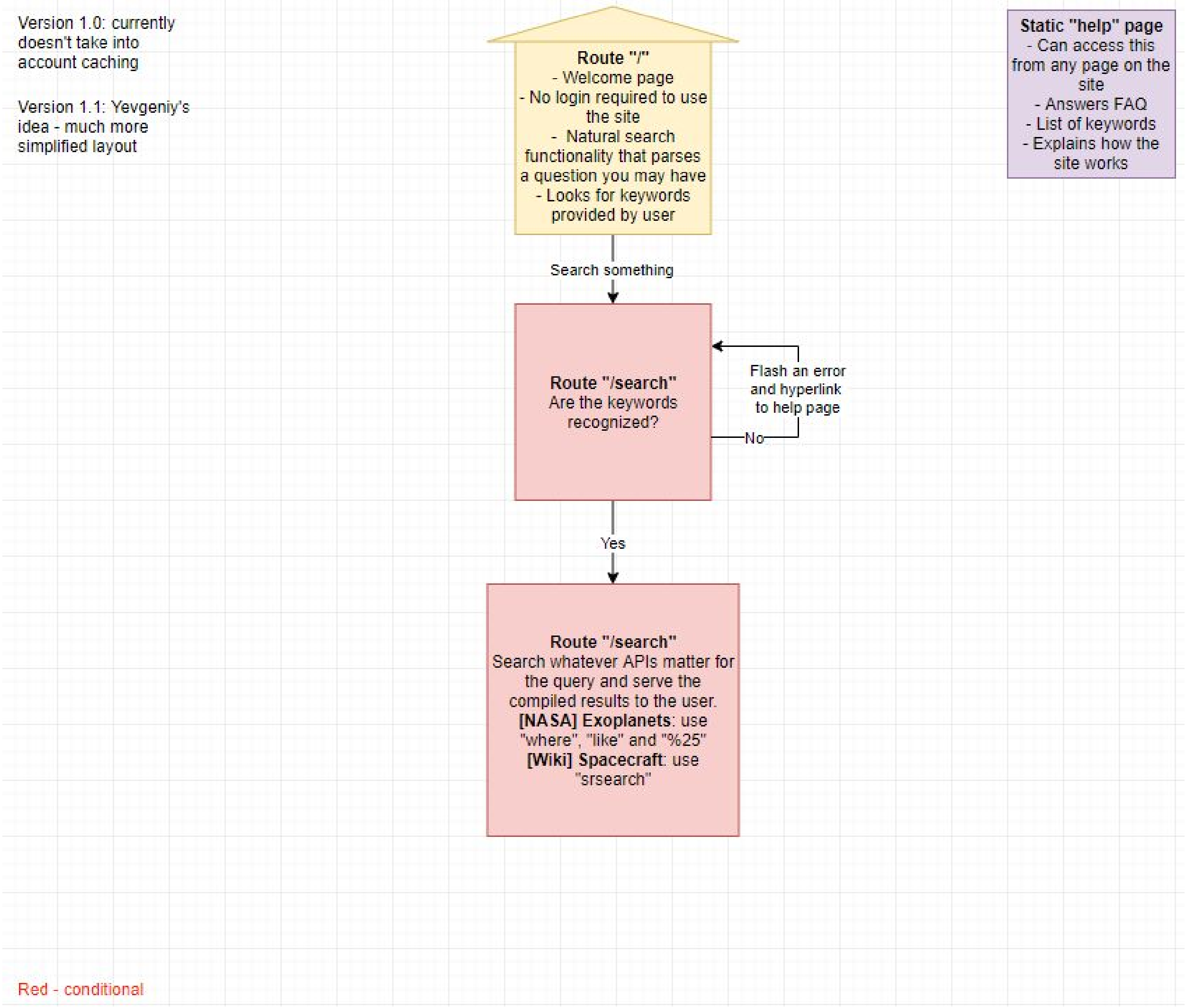
Dependencies:

- search.search(): api_bus.*

## Component Map


Component Map
FRONT END (client)
BACK END (server + query APIs)
Search
Done via HTML form by user
Keywords parsed by server
Choose what APIs to search through based on keywords
Cache the combination of keywords and result after searching through APIs
Keywords have been parsed before
Version 1.0
Obtain the results from cache
Keywords: how long to reach {{Proxima Centauri b}} with the {{Falcon 9 rocket}}
Keywords: stats on {{Proxima Centauri b}}
Keywords: stats on {{Falcon 9 rocket}}
Search NASA exoplanets API
for {{Proxima Centauri b}}
Return data (without having to query APIs)
Search Wikipedia API
for {{Falcon 9 rocket}}
Return data
Use Wolfram|Alpha API
Feed in data from the exoplanets and Wikipedia API into WFA's calculator
Results
Whatever gets returned by the search
Return data

## Site Map:


Version 1.0: currently doesn't take into account caching
Version 1.1: Yevgeniy's idea - much more simplified layout
Route "/"
- Welcome page
- No login required to use the site
- Natural search functionality that parses a question you may have
- Looks for keywords provided by user
Static "help" page
- Can access this from any page on the site
- Answers FAQ
- List of keywords
- Explains how the site works
Search something
Route "/search"
Are the keywords recognized?
Flash an error and hyperlink to help page
No
Yes
Route "/search"
Search whatever APIs matter for the query and serve the compiled results to the user.
[NASA] Exoplanets: use "where", "like" and "%25"
[Wiki] Spacecraft: use "srsearch"
Red - conditional

## Query Processing

Expected time query format (asterisks represent optional parameters) (case insensitive):

[(time|how long)] [to (reach|flyby)]* [from {planet}]* [to {planet}] [using {engine}] [and {fuel mass} of fuel]

Expected mass query format:

[how much (fuel|mass)] [to (reach|flyby)]* [from {planet}]* [to {planet}] [using {engine}] [in {years}]

Default values for optional parameters (*):

- [“reach”, “flyby”] defaults to “reach”, that is, a full deceleration at the end.
- [from {planet}] defaults to “Earth”

**Database Schema: (each row is a different table)**

| **Table** | **Contents**[2] |
|---|---|
| engines | name, mass, specific impulse[3], exhaust velocity, thrust, image link, propellant |
| planets | name, distance[4], right ascension, declination |
| queries | origin, method, goal, engine, mass, time |

[2] All numerics are in the standard international units for that corresponding measure

[3] Specific impulse in a vacuum

[4] light-years