Reptile Resource

Stelio Brooky | Arnan Jeram | Aaris Baskaran

Project Description

Reptile Resource is a wiki web page that allows users that are interested in reptiles to view and read about their favourite scaly creatures!

The page features basic but informative information about reptiles, primarily reptiles that are available as pets in New Zealand (e.g Blue Tongued Skink), although logged in users can add information about any reptile they want.

Users can login with Google or sign up through our own authentication system. The app is hosted on http://reptile-resource.herokuapp.com. Users can Add, Delete, Select, or Update the database (stored on Firebase).

High Level Design

Our project follows MVC architecture. The model.js file holds the Reptiles class which defines a reptile and its fields. It stores all the reptiles in an array which is then used to display the reptiles on the web page. The views folder contains all the ejs files which is what gets displayed to the user. The controller.js file defines the processes to be done when a button is pressed or when an HTML request is sent by the user. There is also the routes.js file, which handles the different URLs that can be accessed and directs incoming requests to the controller. Finally, there are auth and data folders which handle the authentication and database configuration respectively.

Workload Distribution

- Stelio
  - Database Functionality
- Arnan
  - Authentication Functionality
- Aaris
  - Code Architecture

Stelio’s Contribution

- CRUD Functions for the Application
    - Created a Firebase project and configured a Realtime Database (config.js)
    - exports.postAddPage in controller.js created with the ability to select an item from the database, add a new item from the database, add an item to the database and delete an existing item from the database.
    - Most of addPage.ejs to align the inputs with the data in the realtime database.
- Web Service
    - GET and DELETE functions for the web service. Done by editing routes.js to include the paths to the new functions in controller.js (getReptiles & getDeleteReptiles)
    - Send data in JSON format
- Populate the Local Array
    - Update the HTML on the client side (index.ejs) to display the wiki information. The code in the ejs will also determine if there are no reptiles to display.
- Hosting on a Cloud Platform
    - The application is hosted on Heroku
    - Changed the port number to support both localhost and the Heroku domain

Arnan’s Contribution

- Google Authentication
  - Configured to allow use of OAuth and Google Login for user registration and authentication for Web application.
- Reptile Resource Local Server Authentication
  - Basic registration and login functionality implemented.
- Setting Password Complexity
  - Password complexity specifications implemented for user registration functionality.
- Mocha
  - Mocha installed and test script created to test authorization middleware, however not fully functional.

Aaris’ Contribution

- Base Architecture for Application
    - I created the initial code template for the application, making sure that the file structure follows the MVC architecture. This included the model, views, controller and routes folders with the corresponding js/ejs files inside.
    - The model.js file contains a Reptile class which is used to store reptiles in an array once they are retrieved from the database. This array is used when displaying the reptile data on the web page.
    - The routes.js file handles the different URLs/requests that the app might receive and directs them to the controller to be processed. (the routes for the authentication and server were done by Arnan and Stelio respectively
    - The controller.js file processes the requests that are received by the routes.js. I wrote the basic template which was then used by my teammates.
    - I implemented different privileges for different users who are logged in or not logged in. This was done with EJS in the index.js file, and in the controller.js file.
    - I also did some major refactoring of the code so that it followed MVC architecture. Initially, most of the code in controller.js was in routes.js, so i refactored it accordingly.
- Bug Fixes
    - I also contributed with some bug fixes. Eg, a Google Authentication error.

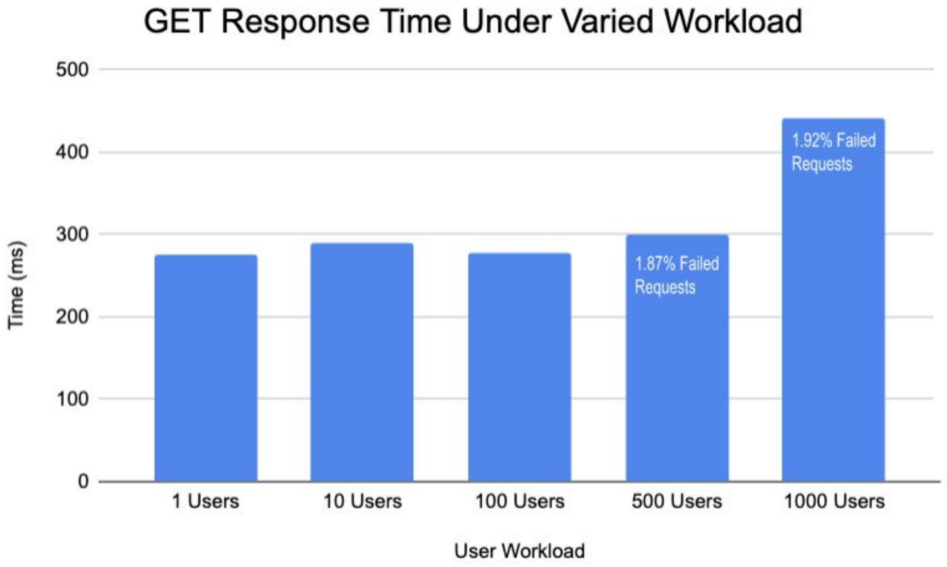
Performance Analysis

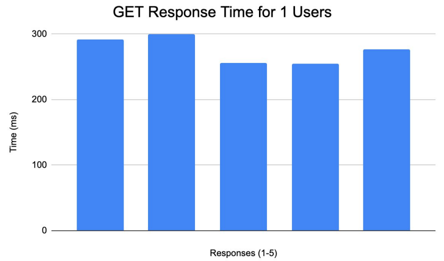
To do the performance analysis we used the K6 load testing tool to successfully undertake a performance analysis.

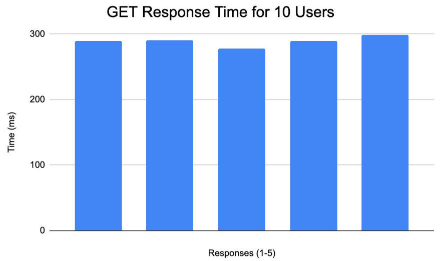
We tested the API functions done by the server, which in our case implements the GET and DELETE functionalities. We measured the average response time done for when only 1 virtual user is performing the function, and then compared that to how long it took for 10, 100, 500, and 1000 virtual users to complete that same function at the same time.

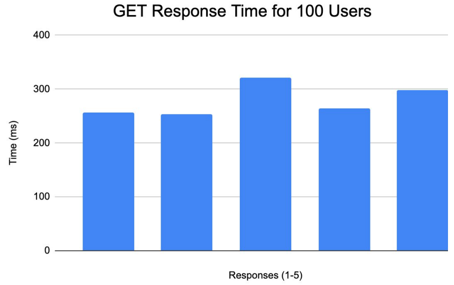
We ran the load testing tool for 10 seconds for each test and gathered that average response time. We repeated that 10 second process 5 times and gathered the response times into a combined average. We then compared the average response time of each amount of users with each other and noted down the results.

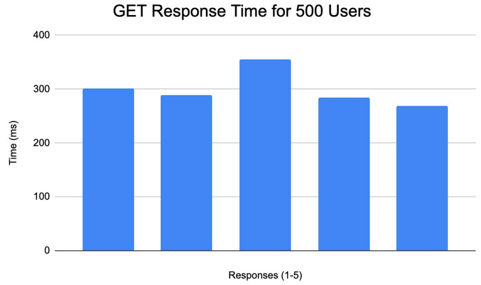
Performance Analysis - GET

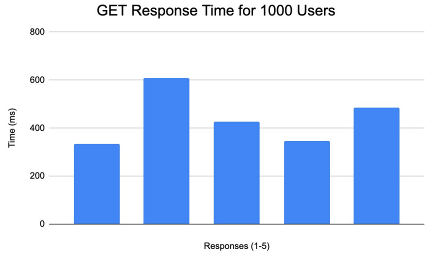




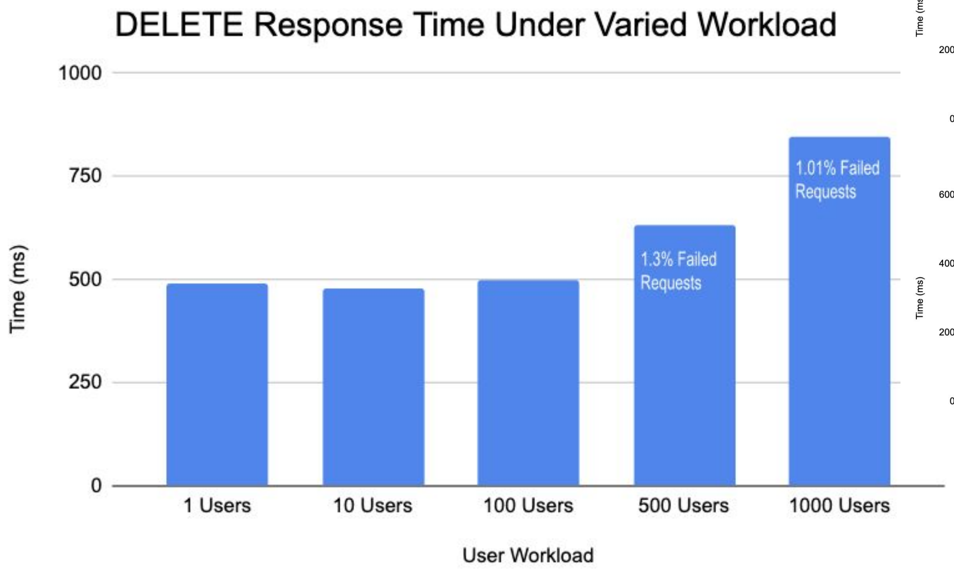


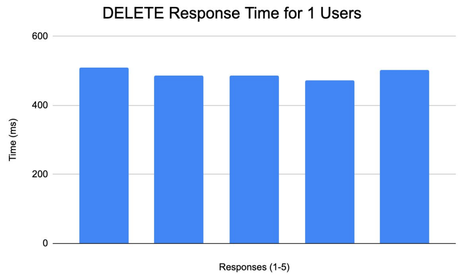


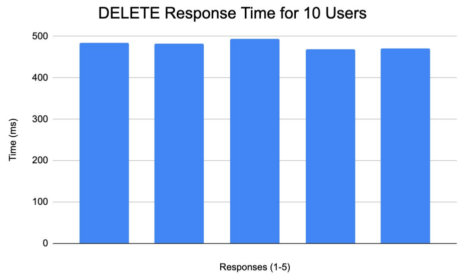


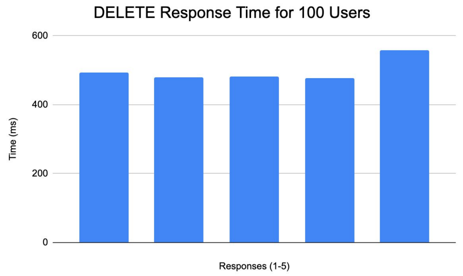


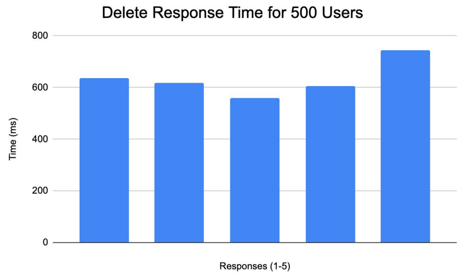
Performance Analysis - DELETE

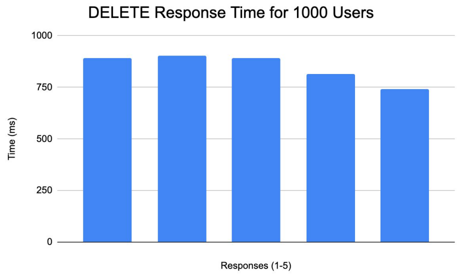












Performance Analysis - Conclusion

- For both functions there was not much variation when using anywhere between 1-100 users at the same time
- The average response time was notably slower for both functions at around 500 users, with failures existing in some cases
- At 1000 active users, the requests would produce some failures at all times
- When there is a large amount of failures (>90% failure rate), the response time is very small and can skew results
- DELETE requests seem to take slightly longer than GET requests for a response
- DELETE requests may change server state, which is one reason it may take longer than GET requests
- DELETE requests are idempotent