CSC 431

Coronapyrus

Software Architecture Specification (SAS)

Team 11

Alexander Claman	Scrum Master
Noah Jaccard	Team Member
James McSweeney	Team Member

Version History

Version	Date	Author(s)	Change Comments
1.0	04.01.2021	Team 11	Rough Draft #1
1.1	04.13.2021	Team 11	Rough Draft #2

Table of Contents

1. System Analysis

1.1. System Overview

The Coronapyrus system includes the Coronapyrus package itself, as well as applications built using the package deployed on communication services, spcifically either Slack or Discord. The package itself serves to process requests for COVID information, retrieving, locally storing, and formatting the data to save a developer-as-user time and effort. The package either sources data from the John Hopkins University coronavirus database or retrieves articles from the Internet in response to requests, then formats the retrieved information into a user-specified format. The package relies on other open-source packages, such as Pandas and Matplotlib, used to analyze and format the data.

The applications made using the Coronapyrus package, whether a Discord bot or a Slack app, represent an access point for the users of that platform to access the Coronapyrus package's features. The application will handle the requests for user information by handing them off to the package, then will convert the returned information into a message format applicable on the platform of implementation. The application will also handle requests for either help with the application, or for the source of the information.

The Coronapyrus package will be built in a manner most similar to the pipes-and-filters architecture, in that 1) a request will cause data to be retrieved, 2) the data will be analyzed, 3) the data will be formatted, and 5) the data will be returned to the user. The applications will function more like a two-tier or layered architecture with elements of an event-driven architecture as well; the application will process events in the form of user commands, request information from the Coronapyrus package to fulfill the commands issued, then send a message as an event in response.

1.2. System Diagram

Figure 1 – Coronapyrus System Diagram (Rough Draft)

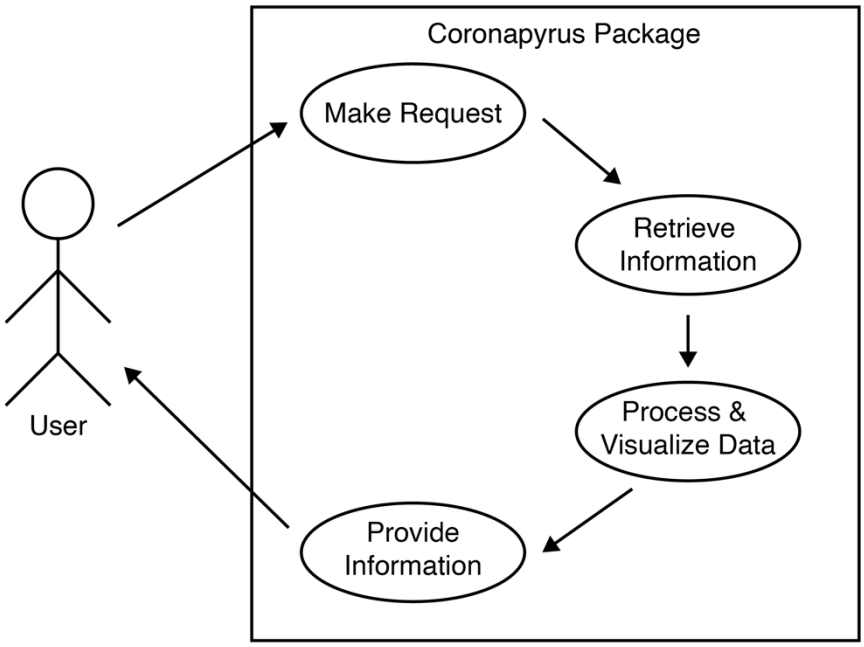
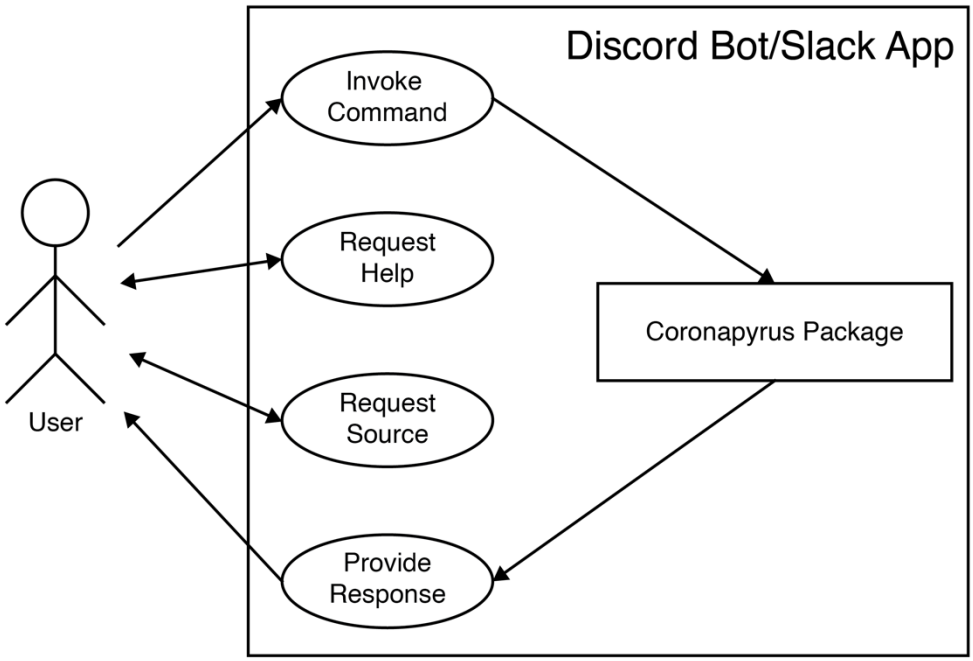


Figure 2 – Application System Diagram (Rough Draft)



1.3. Actor Identification

Actors for the Coronapyrus package will be a developer hoping to use Coronapyrus to save time in designing functions to request and format COVID information themselves. They will be able to make requests of specific scope and format and have data returned for applied use.

Actors for the applications will include the authenticated user members of each local environment on the communication service (i.e., members of the Discord server where the bot is installed and active). They will be able to issue commands to the bot/app and get a response in return.

1.4. Design Rationale

1.4.1. Architectural Style

Pipes-and-filters was chosen as an architectural methodology for the Coronapyrus package for its simplicity; the package itself simply removes layers of complexity which each programmer to use this set of functions would need to add. At its core, this is just data retrieval and formatting made easy.

A two-tiered, event-driven architecture was chosen for the applications out of necessity. The applications must respond to events (e.g., application users' commands) with events (messages) and they will use the Coronapyrus package's functions to do so.

1.4.2. Design Pattern(s)

Python conveniently lacks the need for strong typing and inheritance characteristic of object-oriented programming, but some design patterns will still be used for both convenience and necessity.

- Decorator: used in the coding of the Discord bot and likely in the coding of the Slack application as the preferred Python technique for executing a method in response to an event.

- Façade: used in the design of the Session class to minimize a programmer's need to call groups of methods to request and parse various types of data.

- Singleton: some characteristics used when creating the Session class – there should only be one Session object.

1.4.3. Framework

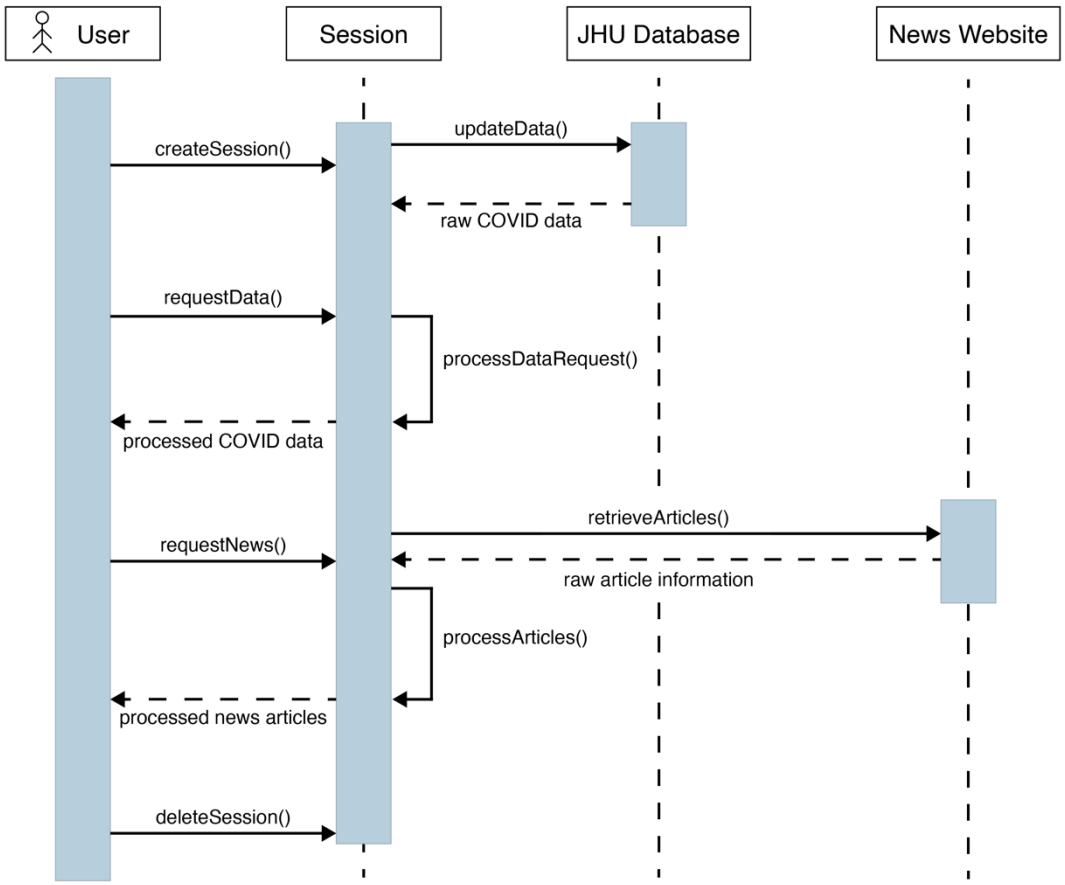
No framework will be used in the making of the Coronapyrus package or either application.

2. Functional Design

2.1. Sequence Diagram: Coronapyrus Package

The below sequence diagram represents the events needed to use the Coronapyrus package, ordered vertically with respect to time. Further explanation is included below the diagram.

Figure 3 – Coronapyrus Sequence Diagram (Rough Draft)



A developer will be able to create a session; upon creation, the session will retrieve and locally cache data from the JHU database.

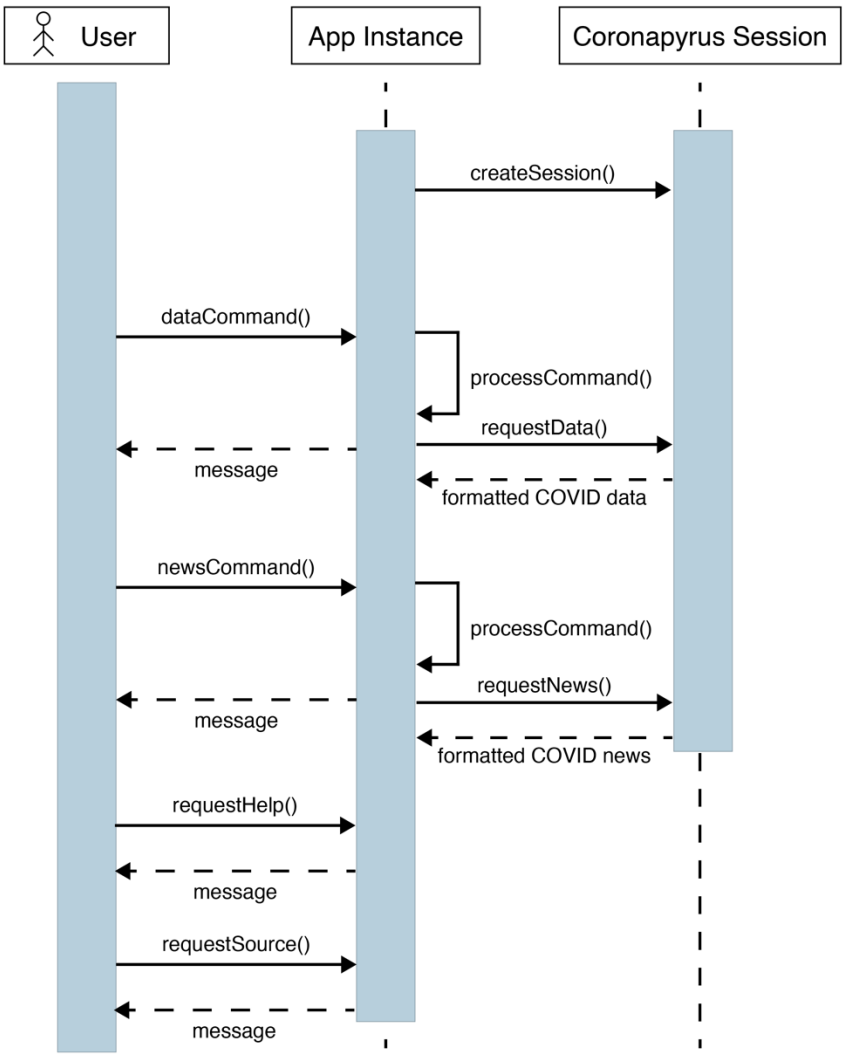
Any data request passed from the developer's code to be processed will be processed by methods contained within the Session object, using the locally cached data.

Any request for media passed from the developer's code to be processed will be processed by methods contained within the Session object, including downloading the relevant articles from the Internet.

2.2. Sequence Diagram: Discord Bot/Slack App

The below sequence diagram represents the events needed to use the applications made using the Coronapyrus package, ordered vertically with respect to time. Further explanation is included below the diagram.

Figure 4 – Application Sequence Diagram (Rough Draft)



Upon instantiation, the application will create a Coronapyrus Session object, which will in turn dowload and locally cache COVID data.

Any command that requires COVID information will be processed to parse its arguments, then have those passed as parameters to the Coronapyrus package. Any other command will be processed by the bot itself.

3. Structural Design

Figure 5 – Coronapyrus Class Diagram (Rough Draft)