CSC431

Download of Public-facing Data

System Architecture Specification

Team #3

Jerry Bonnell Gururaj Shriram Lixiong Liang Heyu Yao Re Chang

Version History

Version	**Date**	**Author(s)**	**Change Comments**
2	May 6, 2018	Jerry Bonnell, Re Chang, Lixiong Liang	Final Draft
1	April 4, 2018	Jerry Bonnell and Gururaj Shriram	First Draft

Contents

List of Figures

List of Tables

1 System Analysis

1.1 System Overview

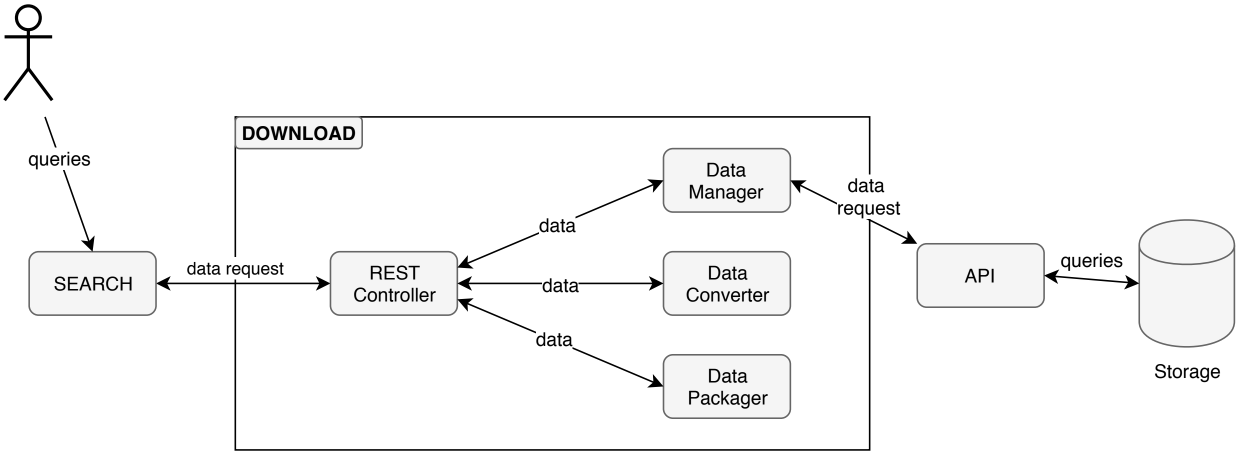
The System will be comprised of four main parts: a data manager, data packager, data converter, and REST controller. The data manager will control the connection to the database and handle database queries.[1] The data packager will package all requested pieces of data into a compressed file if multiple pieces of data are requested. The data converter will convert data from the database into the user's desired format. This format can be any of the supported file formats: kml, esri Shapefile, csv, or geoJSON. The REST controller will manage all of the other aspects of the system and manage the end-to-end pipeline of the downloading process.

This pipeline is as follows: (1) the browser submits a download request to the download server which the REST controller listens to, (2) the controller initiates the process of querying data from the database based on the list of layers provided by the browser request, (3) the data is then converted to the requested output format, and (4) the server packages data before submitting it to the browser for local download.

The three-tier architecture will be used for this project. The client-server tier allows the download server to handle requests from the search team. The database-centric tier queries the database to obtain geospatial and multimedia data. The business layer performs the conversion and archival of this data, before the client-server tier sends the packaged data back to client.

1.2 System Diagram

Figure 1: Download System Diagram



[1] This is later to be replaced by a call to a database API.

1.3 Actor Identification

There are three types of human actors: unauthorized users, authorized users, and the administrators. Unauthorized users may only download public pieces of data unless they are granted permission to specific pieces of data from an administrator. Authorized users are allowed to download any piece of data. Administrators are able to download any piece of data as well as approve requests for unauthorized users to download private pieces of data. The system must also support interaction with a non-human actor, i.e., an API, in order to retrieve required data.

1.4 Design Rationale

1.4.1 Architectural Style

This system, which is part of a larger system, solely exists on the back-end. In designing this, we are using the 3-tier architectural style because our system is a back-end API. This lends itself to a natural division into three tiers: client-server, business, and database-centric. In the client-server tier, the REST controller receives user input, validates the input, and passes that on to be processed by the business tier before responding to the client's request. The business tier requests data from the database-centric tier, converts that data and, finally, archives the converted data and sends it back to the client-server tier. The sole purpose of the database-centric tier is to interface with the database.

1.4.2 Design Pattern(s)

Express is a flexible Node.js framework that provides a robust set of features. We predict that the following design patterns will be primarily used in the following ways:

- Factory: To abstract objects that may share common tasks, such as data management and parsing objects
- Singleton: To maintain a single, consistent reference, such as a database reference
- Middleware: To handle flows that have middleware functions in the system's request-response cycle, such as in the data conversion process
- Stream: To process large amounts of flowing data, such as in the file writing/core download process

1.4.3 Framework

The web application will run on the `Node.js` JavaScript engine on the `Express` framework. `Express` allows for the successful implementation of a REST API on the backend and will facilitate a lightweight, efficient download pipeline.

Links:

- `https://nodejs.org`
- `https://expressjs.com`

1.5 Cross-cutting Requirements

We have identified two cross-cutting requirements that are involved in the download pipeline:

1. Performing a **download**.

   Performing a download requires the implementation of a download button on the UI, which calls our download web server, and, at some point, makes contact with the storage database. Therefore, this is a cross-cutting requirement involving three teams that are heavily dependent on each other: search/UI, download, and database.

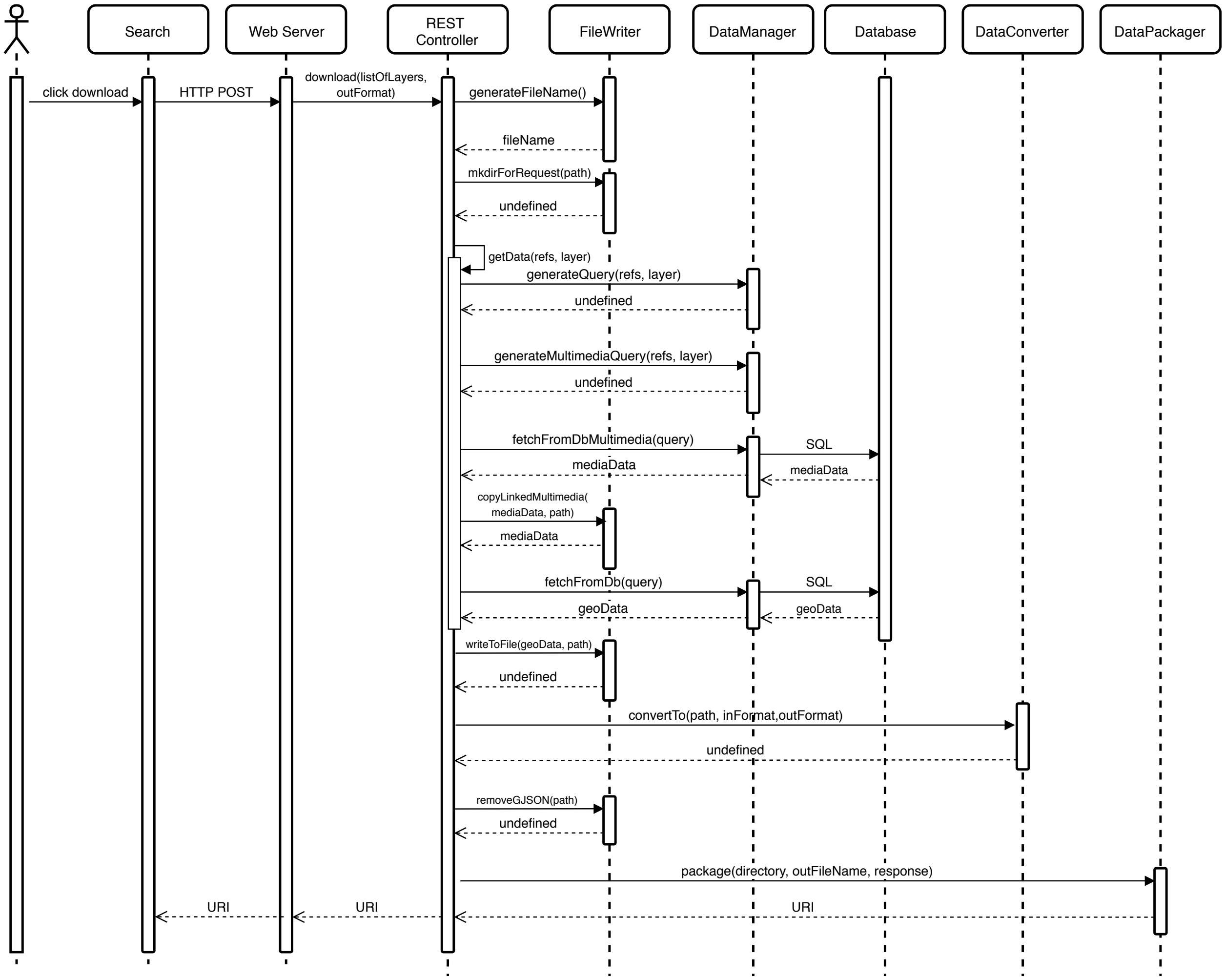
2. Gathering **geospatial data** from a database.

   Further, from the list of layers, our server must be able to gather geospatial data from the Postgres database. This is cross-cutting between the database and our download team.

2 Functional Design

Here we present the sequence diagram for the download use case. This outlines the core functionality for this system. Following it are clarifications regarding the diagram.

Figure 2: Sequence Diagram for the Download Use Case

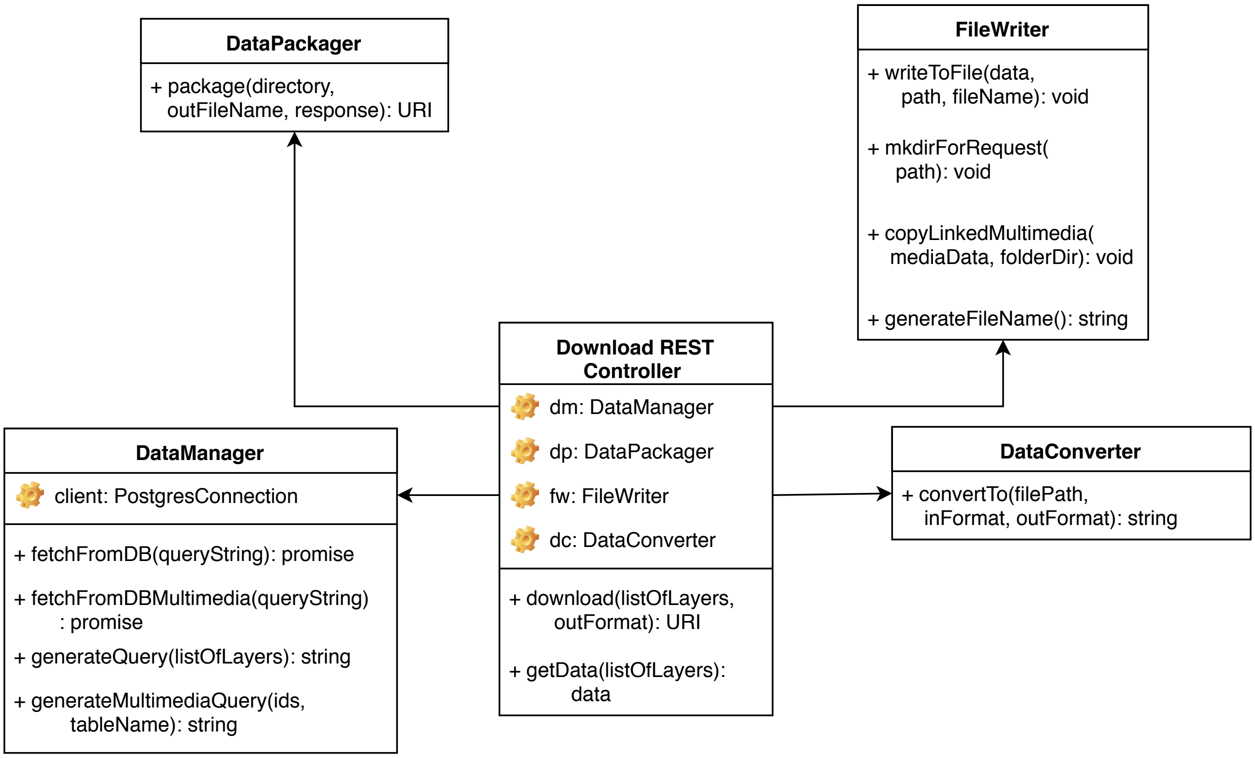


- When a user clicks the download button on the search page, an HTTP POST request is sent to our web server, which calls on our REST controller to handle the request and retrieve the required data.
- The controller will then call FileWriter to generate a unique name for the request directory and the files contained inside. Using this name, a folder is created.
- For each layer, a call is made to DataManager to generate queries that fetch the desired data (both multimedia and geospatial) for the given references from the storage database. On the production machine, a call would instead be made to the backend's API for retrieval.
- The database returns data in its stored format, and FileWriter writes this data to file.
- For geospatial data, DataConverter is invoked to convert the data from its stored format to the desired output format (if necessary). No conversion is necessary for returned multimedia data.
- GeoJSON files are deleted from the directory if not requested. At the very end, DataPackager is called to package all of the download files into a single URI that is returned to the search.

3 Structural Design

The following class diagram represents an overview of the component breakdown in our system. A primary focus for this design is modularity.

Figure 3: Download Class Diagram


DataPackager
+ package(directory, outFileName, response): URI
FileWriter
+ writeToFile(data, path, fileName): void
+ mkdirForRequest(path): void
+ copyLinkedMultimedia(mediaData, folderDir): void
+ generateFileName(): string
Download REST Controller
dm: DataManager
dp: DataPackager
fw: FileWriter
dc: DataConverter
+ download(listOfLayers, outFormat): URI
+ getData(listOfLayers): data
DataManager
client: PostgresConnection
+ fetchFromDB(queryString): promise
+ fetchFromDBMultimedia(queryString) : promise
+ generateQuery(listOfLayers): string
+ generateMultimediaQuery(ids, tableName): string
DataConverter
+ convertTo(filePath, inFormat, outFormat): string

An asterisk is here required. The DataManager outlined here interfaces with the database and, as such, carries the responsibility for generating queries. Strictly speaking, this does not adhere to the requirements. In its place, a call should be made to the backend's API for retrieval of the data. We expect most of the DataManager's methods to be deprecated and replaced by an API call on the production machine.