Bilkent University

Senior Design Project

Cerebro

Project URL: kyorgancioglu.github.io/CerebroApp/

Final Report

Project Members:

Kaan Yorgancıoğlu 21302439
Dilara Ercan 21201256
Özgür Taşoluk 21301674
Nihat Atay 21102292
Furkan Salih Taşkale 21300878

Supervisor: Selim Aksoy
Jury Members: Ercüment Çicek, Uğur Güdükbay

May 3, 2017

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

1. Introduction

Everyday thousands of new media are published in many different forms. Although there are some applications that help identify media by several ways, most of the mainstream apps fail to provide more than minimal information. Also most of them focus on a single type of media.

It is common that consumers come across a media product such as albums, movies, books in a shop and want to learn more about it. But there is no single easy way to access all the relative information in one place. Our solution to this problem is "Cerebro". Cerebro is a mobile app for users to take advantage of. It is a platform that searches the media and puts together desired information by gathering them from trusted third party sources.

Cerebro uses the camera of the smartphone to capture the cover art of media such as book covers, album covers, movie posters, etc. It then searches the database for any matches and gathers all the information from the internet such as the artist, author or the director of the media, casts, release date, customer reviews, brief summary and etc. and displays them in a tidy smart way. This would allow users to quickly find information from their favorite sources, all at once.

2. Current Architecture of the System

Our project contains an Android application where users make some image request to the server and a server application where users' image queries are processed and database transactions are handled. Because of this request-response cycle to be maintained properly, we have decided to use Client/Server architecture as our main software design pattern. Client part will be the Cerebro android application which is called Client Subsystem and Server part will be the Server Subsystem of our project.

2.1. Client Subsystem

Cerebro will contain an Android application where users make some image request to the server and server is responsible for processing the image and responding with information that is gathered using the results of the database queries. With respect to needs of the application, we decided to select MVC software design pattern as our architectural choice for the mobile application. MVC provides us to handle different parts of the application separately. As expected we have three main subsystems in our app. For model, corresponding

subsystem in our application is CerebroModels Subsystem; for view, we designed CerebroApp Subsystem and lastly, for controller we have CerebroController Subsystem.

In our application we have three main subsystems which are following:

- **CerebroApp Subsystem**: This subsystem is the view side of our application which is the user interface of our application.
- **CerebroController Subsystem**: This subsystem is the main controller subsystem of our project. Main task of this subsystem is handling interactions coming from the user and managing necessary processes.
    - **ServerController Subsystem**: This subsystem is responsible for connection with server and is in charge of API requests and responses. It maintains data transaction between our client application and server.
    - **SearchController Subsystem**: This subsystem maintains search operations performed by user through our mobile application.
    - **HistoryController Subsystem**: This subsystem takes care of saving and controlling search and view histories for the user.
    - SettingsController Subsystem: This subsystem is in charge of controlling application settings including user interface customization, selection of widgets etc.

CerebroModel Subsystem: This subsystem is the main data storage part of our project. It also maintains objects in the application.

- **ObjectModels Subsystem**: This subsystem deals with main info page objects for our application. It is responsible for creating and destroying them.
- **WidgetModels Subsystem**: This subsystem mainly works on widget objects which reside in an info page. It is responsible for creating and destroying them.

2.2. Server Subsystem

Our server subsystem is primarily works on processing given images and querying database with the achieved results. The server is written in Python using Flask library and is deployed on Heroku Cloud platform. We didn’t use any particular design pattern for this subsystem. It contains two inner subsystems that are following:

- **ImageProcessing Subsystem**: This subsystem is responsible for analyzing the images and creating identifying information for given image, for instance feature extraction. In this part a tool called Clarifai is used for image recognition.
- **DatabaseConnector Subsystem**: This subsystem is in charge of connecting the database and managing the database transactions. In the database image IDs are mapped to sources such as IMDB titles and Goodreads ids. We used Postgresql for the database.

3. Tools used

Several development tools were used during the implementation of the project. In this section these tools are introduced and how they are used is explained.

3.1. Scrappy

While building our dataset, we collected media information from IMDB. For this task we used a tool called Scrappy. It is a web crawler for Python that is easy to use and deploy. To use it, developers must extend a class called Spider. Spiders use the initial url list to access web pages and recursively reach more urls. After it loads a web page, developers can use the spider's methods to select and work with objects in the webpage, through CSS or XPATH. Once the development is finished, the scraping process can be started by using Scrappy's command line interface. Spiders can also be deployed to mainframes to complete tasks that would be nearly impossible to do on personal computers.

3.2. Flask

The library we used to build our REST API is Flask. We considered a few candidates including Django and Ruby On Rails but we decided that Flask is our best option. Flask is a web app library for Python that is a lot more lightweight compared to other similar libraries. It is dependent on only the minimal package required to build the web api. Other functionalities such as databases or security is provided in its side modules and they are all optional. The code required to handle a request is very compact and intuitive. Additionally its stateless nature is fully compatible with REST APIs.

3.3. GUNICORN

Flask's own web server is built for hobby projects and it is single threaded. Therefore it is not well-suited for release builds. For this reason we used Gunicorn instead of Flasks own server. Gunicorn is a free tool that can be used for high performance web servers. Flask and Gunicorn are compatible and Flask's "app" object works as a wsgi (Web Server Gateway Interface) callable, which is used by Gunicorn.

3.4. Heroku

We used Heroku to deploy our web app. It is a freemium cloud deployment platform that is easy to use and manage. Users can deploy their apps simply by pushing the git repository of the project to heroku servers. Once pushed, the source code is complied into a slug that is run on a "dyno" in the cloud. It can be scaled by running several copies of the slug in different dynos. In its free model the app runs on a single dyno and goes to sleep mode after 30 minutes of inactivity. The free dyno time is limited to 550 hours. Since it is a cloud platform, the app cannot keep external files unless it can recreate those files on its own. For this reason Heroku does not support all database types. We used PostgreSQL as suggested by Heroku Developer's Guide.

3.5. Postman

To test our server during the development process, we used Postman. It is a Chrome app that provides a nice alternative to using curl. It can save requests, prepare templates, and show responses in a neat, visually appealing way. It is much faster to work with compared to curl. It also provides many features for development teams such as exporting and importing the working environment (saved request responses and templates).

3.6. Clarifai

We used Clarifai Video & Image Recognition API for the recognition of the covers of the DVD's. Clarifai is very easy to use for recognizing visual content. It saved us from implementing a recognition algorithm from scratch. It simply predicts the content of the

image user provides. Clarifai has some built-in models and concepts which are included in its predictions. For example, its predictions contains probabilities of the image containing concepts like water, sky, smoke, face etc. It also provides to create your own dataset for further search or other purposes. In our case, we created our dataset with posters of the movies. To search between the images in the dataset, users need to add the images to search index of the API. Also Clarifai allows users to create their own models and train the dataset according to those concepts. Since our search is done by getting an image from the user and finding the closest match for that input, we needed to use Clarifai's reverse image search functionality. Reverse image search basically tries to find the best match in the dataset for the given input. Since the main goal of our app is to find matching poster for given input, this functionality really helped us.

3.7. PostgreSQL

Since Heroku suggest its users to use PostgreSQL, we decided to use it. PostgreSQL is an object-relational database system. It contains all of the functionalities we need to keep our data safe and robust. Although our database will not be too complex, PostgreSQL will allow us to maintain the data of the application easily. Its implementation conforms to the ANSI-SQL:2008 standard. So it can be considered as a state-of-the-art database system. It provides us speed, robustness and reliability.

3.8. OMDB

OMDB is a open API for IMDB. It is built and maintained by a third party. We use it in our client to gather info about a media given its IMDB title.

4. Contemporary Issues Related with Project

- Cerebro does not own or claim rights to any of the media presented in the app. All rights belong to corresponding owners.
- Search history of users will only be stored locally and will not be accessed by the server in any condition, out of respect to privacy of users.
- Internet connection is a necessity for both parts of the project. So, anyone who has

problems with internet connection and speed will have problems using the program.

- Modern day mobile devices come in many types of operating systems. Yet, Cerebro currently only supports Android devices. This may cause some of the users to not be able to benefit from the program.
- Computer technology is rapidly moving towards more complex cloud systems. Future implementations of the project may require us to scale the cloud system we use to have higher performance.
- New technologies are emerging everyday in the area of image recognition. Utilizing these new technologies may enable cerebro to achieve better performance in matching images.

5. Impact of Engineering Solutions

Cerebro is a project that has been developed to ease the gathering information from different sources for the single media content. It requires internet connection for both server part and Android Mobile part. The four following impacts of the project will be explained in the following subcategories.

5.1 Global Impacts

The global world, technology has a huge impact on human's life and technology is changing rapidly. Today's world, many application needs the Internet connection, everyone is online therefore we want to create online media platform to gather almost all information related to single media content. In this sense, our application will ease searching information from different sources and putting together desired information by gathering them from trusted third party sources.

5.2 Economic Impact

With Cerebro, people can find films, movies or books that interest them easily, because they can find lots of information about these media easily. In this way, sales in cinema or book sector will increase because of satisfying people' curiosity more quickly.

5.3 Social Impact

With Cerebro, it is possible to access all wondered information about a media at the same time. In this way, user can make a more precise decision about quality of media and act according to this decision. Also, Cerebro will offer similar medias based on user's previous choices and this will ease to find new media according to taste of the user.

6. Resources Used While Building The Project

- **https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world** This is the resource we used to learn about using Flask.
- **https://docs.scrapy.org/en/latest/** This is the documentation of Scrappy. We used it to learn about the tool before using it.
- **https://developer.clarifai.com/docs/** We used the documentation of Clarifai to learn about it.
- **http://www.omdbapi.com/** We used OMDB website to learn how to use it.
- **https://devcenter.heroku.com/** We used Heroku documentation to learn how to use it.