Process Behavior Analysis with Focused Clustering: Milestone 2

Isaac Amann

3/31/2024

Contents

1 Progress

1.1 Backend Server

1.1.1 User Authentication

User authentication was initially done by providing a session token on login that would be stored in the database along with an expiration date. Instead of using simple generated tokens, I switched to using JWT. The keys are signed with a private key, so the server can grant access to resources without querying the database as long as the key is valid.

1.1.2 Page Requests

API calls were created that allows clients to request portions of the dataset providing a page number and page size. This is needed for the dataset explorer as once the dataset is large enough, sending the entire dataset all at once would not be practical. Page request API calls were also created for user accounts and registered collection clients for user in the admin dashboard on the frontend.

1.2 Data Analysis Job System

The analysis job system was setup such that jobs can be submitted through an API request. An entry for the job is posted to the database containing information likes its status, time started, time finished, analysis type, parameters, and the generated clusters. An abstract class for the analysis method was written so different clustering algorithms could be used in the system. Analysis using the SowAndGrow algorithm being developed by Dr. Che was implemented by creating input files and then executing the compiled program by using the Runtime.exec() Java method. The output could then be saved to the database by parsing the output files generated by the SowAndGrow program. See code section below.

```
private class SowAndGrowThread extends Thread
{
@Autowired
private DataPointRepository dataPointRepository;
@Override
public void run()
{
  String directoryPath = System.getProperty("user.dir") + "/analysis/
     job";
  directoryPath = directoryPath + parentJob.id;
  System.out.println(directoryPath);
  try
  {

    ArrayList<Integer> seedPointIndexes = new ArrayList<Integer>();
    //Create input CSV file from database
    File dataPointFile = new File(directoryPath, "input.csv");
    dataPointFile.getParentFile().mkdirs();
    dataPointFile.createNewFile();
    FileWriter dataWriter = new FileWriter(dataPointFile);

    ArrayList<DataPointEntity> dataPoints = new ArrayList<
        DataPointEntity>();
    for(DataPointEntity datapoint : parentJob.dataPointRepository.
        findAll())
    {
      dataPoints.add(datapoint);
      for(int i = 0; i < seedPoints.size(); i++)
      {
        if(datapoint.id == seedPoints.get(i))
        {
          //Add seed point's index into seed point indexes
          seedPointIndexes.add(dataPoints.size()-1);
        }
      }
    }

    for(int i = 0; i < dataPoints.size(); i++)
    {
      int column = 0;
      for(Float value : dataPoints.get(i).winAPIRatios.values())
      {
```

```
    if(column > 0)
    {
      dataWriter.write(",␣");
    }
    dataWriter.write(Float.toString(value));
    column++;
  }
  dataWriter.write("\n");
}
dataWriter.close();

//Create seed file
File seedFile = new File(directoryPath, "seeds");
//seedFile.getParentFile().mkdirs();
seedFile.createNewFile();
//Push seeds into seed file
FileWriter seedWriter = new FileWriter(seedFile);
for(int i = 0; i < seedPointIndexes.size(); i++)
{
  System.out.println(Integer.toString(seedPointIndexes.get(i)));
  seedWriter.write(Integer.toString(seedPointIndexes.get(i)));
  seedWriter.write("\n");
}
seedWriter.close();

//Run SowAndGrow
String command = "./analysis/bsng␣-z␣5␣-t␣1␣";
String commandDirectory = "./analysis/job" + parentJob.id +"/";
command = command + "-o␣" + commandDirectory + "out.txt␣";
command = command + "-i␣" + commandDirectory + "input.csv␣";
command = command + "-e␣" + epsilon +"␣";
command = command + "-m␣" + minPoints +"␣";
command = command + "-u␣" + commandDirectory + "clusterOut.csv␣";
command = command + "-l␣" + commandDirectory + "seeds␣";

System.out.println(command);

Process process = Runtime.getRuntime().exec(command);
process.waitFor();

//Parse output and write clusters
//Values should appear in the same order as the dataPoints
    arraylist
File outFile = new File(directoryPath, "clusterOut.csv");
Scanner outReader = new Scanner(outFile);
int currentIndex = 0;
while(outReader.hasNextLine())
{
  String currentLine = outReader.nextLine();
  //Get cluster label from line (first entry in csv row)
  String[] subStrings = currentLine.split(",");
  int clusterLabel = Integer.parseInt(subStrings[0]);
  System.out.println(clusterLabel);

  //If datapoint clustered, place it in clusters map
  if(clusterLabel != 0)
  {
    clusters.put(dataPoints.get(currentIndex).id, clusterLabel);
```

```
            }

            currentIndex++;
          }

        }
        catch(Exception e)
        {
          e.printStackTrace();
        }
        finish();
      }
    }
```

1.3 Frontend Server

The frontend is being implemented using React along with MaterialUI. MaterialUI provides several visually appealing components and makes it makes it easier to keep the design consistent throughout the application.

1.3.1 Main Page

The main page provides the project description and a sample of the dataset within a DataGrid component. This page acts as a landing page allowing users to login.

1.3.2 Dataset Explorer

The dataset explorer was implemented using a DataGrid component. DataGrid also supports server-side pagination, so I was able to take advantage of the page request API calls for dataset access that I had previously written.

1.3.3 Admin Dashboard

The link to the admin dashboard is only displayed to users with the correct user role. The page currently only contains a command shell for the server. The implementation for the command shell is mostly taken from some of my previous work on another project with some alterations to work with the user authentication system. The command shell API call exposes Java methods within the ShellCommands class. These methods are able to be invoked when running the server application from the command line, but connecting over SSH and trying to bring the server application into the foreground so that commands can be entered is inconvenient. I plan on changing the frontend command shell component to something more visually appealing and convenient to use, right now it is just a text area element with a text field for entering commands. This page will also include system statistics, collection client info, and a log viewer in the future.

1.4 Data Analysis Page

This page contains a DataGrid showing previously submitted jobs with their information. Jobs can also be selected and inspected to show more detailed information including the clusters that were generated by the clustering algorithm used. There is also a form for submitting new jobs to the server.

1.5 Project Deployment

The project has been deployed to AWS using EC2 to create a Ubuntu Linux virtual machine to run on and Route 53 for DNS. Both the backend and frontend server software are managed as a Systemd service so that they are ran in the background on server startup. I also purchased the domain name winapimonitoring.com to route traffic to my server.

2 Challenges

2.0.1 Number of fields in Windows API counts

The data points created use a large number of fields so working with them can be inconvenient. For the Windows API monitor program, creating separate named fields for each API count was unavoidable. I was able to avoid this issue by using hashmaps in Java on the backend. This does create a potential problem, a modified client with a valid key could submit data points to the dataset with an unexpected number of fields. I may add more validation on data points before allowing them to be posted to the database.

3 Work for Next Milestone

3.1 Backend

- API rate limits / brute force attack protection
- Implement DBSCAN with cosine similarity for data analysis
- Maybe switch to HTTPS over HTTP

3.2 Frontend

- Improved analysis job viewer
- More admin dashboard features
- More responsive login form

3.3 Windows API Monitor

- Further reduce the system instability caused by function hooking

4 Testing

4.1 Unit Tests

Project Name: Process Behavior Analysis	
Test Case 1	
Test Case ID: User Login	Test Designed by: Isaac Amann
Test Priority (Low/Medium/High): High	Test Designed Date: 3/29/2024
Module Name: User Authentication Service	Test Executed by: Isaac Amann
Description: Show that user authentication only issues token when given correct credentials	
Pre-conditions: System connected to Database with a test user already created	

Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status Pass/Fail**	**Notes**
1	Call login API function	Correct username and correct password	System provides a JWT for the target user	System provided a JWT for the target user	Pass	none
2	Call login API function	Correct username and bad password	System returns bad username or password error	System returns bad username or password error	Pass	none

Project Name: Process Behavior Analysis						
Test Case 2						
Test Case ID: Request Authentication			Test Designed by: Isaac Amann			
Test Priority (Low/Medium/High): High			Test Designed Date: 3/29/2024			
Module Name: User Authentication Service			Test Executed by: Isaac Amann			
Description: Show that restricted API functions can only be called when provided a valid token						
Pre-conditions: System connected to Database with test users for each user role						
Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status Pass/Fail**	**Notes**
1	Call authenticateRequest function	Signed JWT with matching user role to the requiredUserRole parameter	authenticate Request returns true	authenticate Request returns true	Pass	none
2	Call authenticateRequest function	Signed JWT with mismatching user role to the requiredUserRole parameter	authenticate Request returns false	authenticate Request returns false	Pass	none
3	Call authenticateRequest function	JWT signed with the wrong key	authenticate Request returns false	authenticate Request returns false	Pass	none
4	Call authenticateRequest function	Malformed JWT	authenticate Request returns false		Pass	none

Project Name: Process Behavior Analysis						
Test Case 3						
Test Case ID: Collection Client Authentication			Test Designed by: Isaac Amann			
Test Priority (Low/Medium/High): Medium			Test Designed Date: 3/29/2024			
Module Name: User Authentication Service			Test Executed by: Isaac Amann			
Description: Show that only collection clients can only post data points using valid API tokens						
Pre-conditions: Test client created with generated ID and API token						
Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status Pass/Fail**	**Notes**
1	Call isValidAPIToken on test client	Correct token	isValidAPI Token returns true	isValidAPI Token returns true	Pass	none
2	Call isValidAPIToken on test client	Incorrect token	isValidAPI Token returns false	isValidAPI Token returns false	Pass	none
3	Call isValidAPIToken on test client	Malformed token	isValidAPI Token returns false	isValidAPI Token returns false	Pass	none

4.2 System Tests

Project Name: Process Behavior Analysis						
Test Case 4						
Test Case ID: Datapoint Posting				Test Designed by: Isaac Amann		
Test Priority (Low/Medium/High): Medium				Test Designed Date: 3/29/2024		
Module Name: PostDataPointController				Test Executed by: Isaac Amann		
Description: Show that registered clients can post data points to the database through HTTP						
Pre-conditions: Server running and client software installed on Windows machine						
Step	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status Pass/Fail**	**Notes**
1	Execute system monitor on client machine	Valid credentials with a data point containing at least 1 counted API call	code 200, datapoint posts to database	code 200, datapoint posts to database	Pass	none
2	Execute system monitor on client machine	Valid credentials with a data point containing all 0's	code 200, datapoint posts to database	code 500, server error	Fail	Server also stores ratios of called Windows API functions. Passing 0 causes a division by 0 throwing an exception
3	Execute system monitor on client machine	Invalid credentials	code 200, Failed to authenticate	code 200, Failed to authenticate	Pass	none

Project Name: Process Behavior Analysis	
Test Case 5	
Test Case ID: Frontend User Login	Test Designed by: Isaac Amann
Test Priority (Low/Medium/High): Medium	Test Designed Date: 3/29/2024
Module Name: LoginController	Test Executed by: Isaac Amann
Description: Show that registered users can login from the webpage	
Pre-conditions: Server running and client connected through web browser	

Step	Test Steps	Test Data	Expected Result	Actual Result	Status Pass/Fail	Notes
1	Submit login form	Valid login credentials	Login form closes and page state changes to indicate login	Login form closes and page state changes to indicate login	Pass	none
2	Submit login form	Invalid login credentials	User not given access	User not given access	Pass	Need to display error message on form to indicate the wrong credentials entered. Should also clear the form

5 Work Log

5.1 2/25/2024

Fixed Windows API Monitor program overwritting credentials retrieved from the Windows Registry. Implemented JWT user authentication and updated other classes to support JWT on the backend server.

5.2 2/27/2024

Created API requests providing pageing for datapoints, user accounts, and registered collection clients.

5.3 3/1/2024

Generated React project for frontend

5.4 3/2/2024

React setup

5.5 3/3/2024

Added frontend pages

5.6 3/4/2024

Frontend changes: Added navbar, login form, profile menu with logout button, and cookies for storing session token after login

Backend changes: Added API call for getting a sample of the dataset accessible to unauthenticated users

5.7 3/5/2024

Frontend changes: Created sample dataset viewer. Created pages for dataset explorer and data analysis. Started dataset explorer component.

5.8 3/12/2024

Frontend changes: Set up server side pagination for dataset explorer. Set fixed page size on dataset explorer

Backend changes: Created shell command to create test data points for testing the dataset explorer

5.9 3/14/2024

Backend changes: Began work on the data analysis system

5.10 3/15/2024

Backend changes: More work on data analysis system. Began implementing RunSowAndGrow class.

5.11 3/16/2024

Backend changes: Finished implementing RunSowAndGrow class.

5.12 3/17/2024

Backend changes: Created DataAnalysisController class to contain REST controllers for data analysis API calls

5.13 3/23/2024

Backend changes: Implemented API calls for data analysis job access

5.14 3/24/2024

Frontend changes: Added admin dashboard with command shell. Added analysis job submit form. Added button for inspecting selected analysis jobs. Created data analysis job table

Project Deployment: Purchased domain name from AWS. Created EC2 instance running Ubuntu Linux. Basic system setup. Created bash script for launching server and Systemd service for executing script on startup.

6 Meetings

6.1 2/28/2024

- Discussed results of implementation of solution to load balancing problem
- Discussed speed loss from threads exchanging points for load balancing when number of points is too small
- Discussed my senior project and motivation for implementation using another distance function than euclidean distance to support high dimensional data.

6.2 3/8/2024

- Discussed work for over Spring break
- Discussed implementation of seed input files