CS6620 Final Project: Problem Statement and Constituent Tasks

Problem Statement

Healthcare providers often need a fast and reliable, yet easily comprehensible drug interaction checking system in order to analyze complete patient medication lists to prevent adverse drug reactions. Doing so allows them to optimize treatments, especially in cases where patients are taking multiple drugs at once (which is almost always the case). My brother's experience with epilepsy has also shown me the same. The current systems are often slow or require manual drug comparisons one at a time, risking human error or missing potential oversights that can lead to adverse drug reactions such as when medications make others less or more effective.

Overall Goals

- Allowing for input of multiple medications
- Querying a database or drug API to check for interactions
- Returning interaction details that may or may not include severity level, type of interaction, potential alternative drugs, etc.
- System should be fairly simple and usable for those without technical prowess.
- Ideally, system would be secure and show proper error handling, fairly quick, and have a potential for scaling.
- NOTE: Goals will be open to adjustments depending on what is technically feasible given the timeline.

Issues/Sub-Issues (Constituent Tasks)

1. Set up basic AWS infrastructure with Terraform (SETUP/INFRASTRUCTURE)
    - As the developer, I will need AWS infrastructure as code so that the drug interaction checker can be deployed.
    - Tasks/sub-issues might include:
        - Create the Terraform files
        - Set up the AWS provider/state/etc.
        - Make sure VPC /networking stuff is all set
        - Set up EC2 instance
        - Set up DynamoDB table
        - Set up S3 bucket
        - Deploy and make sure it works
2. Integrate DrugBank API (also study DrugBank API further) (POC)
    - As the developer, I would need to understand how the API works so that it can then be integrated for interaction checking
    - Tasks/sub-issues might include:
        - Research DrugBank API (I'll use the free version, but also consider the pro version for the final product consideration)
        - Maybe manually test some API calls
        - Get a better idea on error responses and load limits for integration

3. Build the basic Flask web interface (POC)
    - As a healthcare provider, I would need simple web ui to enter drug names and view the results without technical knowledge.
    - Tasks/sub-issues might include:
    - Set up Flask app on EC2 to create basic app structure (requests, python3, app.py, folders, etc.)
    - Make the basic HTML drug input form (should be a simple template)
    - Create the Flask routes for the drug input form (home page) and the results (POST route to handle form submission and show the results)
    - Test locally and on EC2 to make sure the forms and routes work
4. Implement basic two-way drug checking (POC)
    - As a healthcare provider, I would need to check if two specific drugs interacted so that I could identify possible issues.
    - Tasks/sub-issues might include:
        - Create the DrugBank API client
        - Create a function that take two drug names and can return the information
        - Integrate client w/ Flask
        - Parse or format the DrugBank responses so that you have the relevant information.
        - Maybe have a basic HTML template to show the results better
        - Test with basic/known drugs. Also test edge cases like with fake drugs.
5. Error handling and validation (POC)
    - As a user, I would need the system to handle errors and validate input.
    - Tasks/sub-issues might include:
        - Add input validation (like correct drug names, whitespace stripping, etc.)
        - Also make sure API error handling is there
        - Cache errors handled
        - Maybe create the error display pages?
        - Test for stuff like invalid drugs, network failures, etc.
6. Caching with DynamoDB (POC)
    - As the owner of this project, I would need caching so that the app will perform well and reduce API costs.
    - Tasks/sub-issues might include:
        - Setting up boto3 w/ DynamoDB. Will need to test connection from Flask.
        - Plan how the caching will work (or at least have a basic logic for checking and writing).
        - NOTE: I did consider adding ElastiCache for persistent storage with Dynamo. Doing so would mean implementing ElastiCache redis earlier. ElastiCache would provide faster caching (microsecond vs millisecond latency) but it adds complexity with the extra service to configure/monitor. DynamoDB will have sufficient caching performance for this use case while also serving as persistent storage for query history. * NOTE: Alternative database storage option are also possible for at least the POC if issues arise.
        - Statistics and testing
7. Expand POC to multiple drug checking (MVP)
    - As a healthcare provider, I would need to check interactions across a full patient medication list so that I can analyze complex drug interactions.
    - Tasks/sub-issues might include:
        - Update the HTML form to accept multiple drug names (like a text area or a dynamic input field).

        - Pairwise interaction algorithm needed to check all possible drug combinations (I think this is n choose 2… but I need to go back and study math)
        - Need to allow for multiple API calls
        - Make sure results will display the new interactions
        - Test with an example list.
8. Severity classification and results (MVP)
    - As a healthcare provider, I would need to have interaction severity levels and detailed information to make better decisions for my patient.
    - Tasks/sub-issues might include:
        - Extracting and standardizing the severity levels from the API responses. Implement some sort of organization method from minor to major severity.
        - Maybe add color coding in the results?
        - Interaction summaries?
        - Test accuracy somehow
9. SPIKE: Setup Lambda infrastructure for better auto-scaling (NOTE: I'm not sure if this is necessary or if I should be implementing with Lambda from the start. I can make adjustments to this depending on what people think or how much time I have and how much I learn about Lambda)
    - As the developer, I would want to have Lambda functions so that the drug checking can auto-scale independently.
    - Tasks/sub-issues might include:
        - Adding Lambda to Terraform files, create the function resources, IAM roles, and etc.
        - Need REST API with proper CORS to connect to Lambda functions
        - Configure the Lambda/DynamoDB integration (set up IAM permissions and test the access)
        - EC2 will need to communicate with the API Gateway endpoints
10. SPIKE: Migrate the drug logic to Lambda
    - As the developer, I would want to move the drug checking logic to Lambda so that it can scale automatically.
    - Tasks/sub-issues might include:
        - Would need to create Lambda function for drug checking and configure API access to handle authentication in Lambda.
        - Need to handle Lambda caching
        - Need to update Flask to call lambda instead of DrugBank calls
        - Test everything
11. Monitoring and Optimization (MVP)
    - As the developer, I would need monitoring and performance data to make sure everything is running okay.
    - Tasks/sub-issues might include:
        - Set up Cloud watch
        - Implement the health checks
        - Test everything
12. Documentation and Architecture
    - As the reviewer, I would need good documentation and architecture to understand and evaluate the product.
    - Tasks/sub-issues might include:

- Finish the README and have the architecture diagrams ready along with all the tradeoffs/explanations
- Prepare the presentation

**for architecture diagrams, descriptions, and tradeoff options, please see* `docs/architecture.pdf`