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 serverless infrastructure with Terraform (SETUP/INFRASTRUCTURE)
    - As the developer, I will need cloud infrastructure as code so that the drug interaction checker can be deployed.
    - Tasks/sub-issues might include:
        - Create the Terraform config files and AWS provider setup
        - Configure state stuff (S3 backend for Terraform state)
        - Set up Lambda functions with any additional IAM stuff as needed
        - Set up API Gateway (with Flask integration?)
        - Set up S3 bucket for static website w/ CloudFront CDN
        - Configure basic networking (security groups, VPC endpoints if they're needed--can test around)
        - Deploy and make sure it works
2. SPIKE: Evaluate the database/compute options (SETUP/INFRASTRUCTURE)
    - As the developer, I want to choose the best database option so that I can optimize performance, cost, and complexity.
    - Tasks/sub-issues might include:
        - Research/test out different database options such as DynamoDB, RDS, ElastiCache, etc.
        - Research compute options as well (although I have mostly decided to go with Lambda over EC2)

        - Maybe test basic CRUD operations with different options
        - Evaluate pros/cons/tradeoffs for MVP and production
        - Update the infrastructure configurations as needed
3. Research and integrate the DrugBank API (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). NOTE: I applied for the research API and am still waiting for their response.
        - Manually test some API calls and edge cases
        - Get a better idea on error responses and load limits for integration (document)
        - Need Python wrapper/client for API integration
4. Build the basic Flask web interface (POC)
    - As a healthcare provider, I would need simple web interface to enter drug names and view the results without technical knowledge.
    - Tasks/sub-issues might include:
    - Set up Flask app structure for Lambda deployment or whatever compute option was chosen (awsgi)
    - Make a very basic drug input form - jinja2 template?
    - Create the Flask routes for the drug input form (home page) and the results/form processing
    - Configure Flask for Lambda environment (import paths/static files)
    - Add basic CSS styling if I want things prettier
    - Test Flask app locally before Lambda deployment
    - Deploy and test Flask Lambda integration via API Gateway
5. 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:
        - Integrate the DrugBank API client with the Flask routes
        - Implement the form processing for the drug input validation part
        - Create a function that can query and parse the drug interaction data
        - Add error handling for invalid drugs and API failures
        - Design results template to show the interaction information
        - Test with basic/known drugs. Also test edge cases like with fake drugs.
6. Add basic caching (if doing this) with the chosen database (POC)
    - As the owner of this project, I would need caching to improve performance and reduce costs
    - Tasks/sub-issues might include:
        - Set up database connection from Lambda
        - Design the caching strategy stuff (like keys/TTL/structure)
        - Implement checking the cache before API call in Flask routes
        - If necessary, will need to configure Lambda IAM permissions for database access
        - SPIKE: add cache logging and basic performance metrics
        - Test caching behavior with repeating drug combinations
        - Handle cache errors
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 Flask 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 combinations... but I need to go back and study math :p)
  - Need to allow for multiple API calls within the Lambda timeout restriction
  - Make sure results will display the new interactions (organized by severity?)
  - NOTE: Might have to consider async processing or batch handling for large drug lists
  - 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?
     - Spike: A way to print out the reports for health documentation?
     - Test accuracy somehow
9. Monitoring and Optimization (MVP)
   - As the owner of this project, I would need monitoring and performance data to make sure the system performs reliably
   - Tasks/sub-issues might include:
     - Set up CloudWatch monitoring for Flask Lambda function
     - Implement stuff like health checks, Lambda timeout alerts, DrugBank API tracking, cost monitoring/usage alerts, basic analysis.
     - Test as needed
10. Security hardening as needed
    - As the owner of this project, I would want proper security measures before production
    - Tasks/sub-issues might include:
      - Look into and consider security measures such as validation for form inputs, securing DrugBank API keys, adding CSRF protection for flask forms, configuring CloudFront security headers and caching policies, fixing up Lambda IAM permissions, backup strategies, etc.
11. 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`*