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.
- UPDATED: Will use LocalStack for free local development and will create a mock DrugBank API service because I do not want to pay $5,000 dollars.

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:
        - Create the Terraform config files and LocalStack provider setup
        - Configure local state management stuff (local backend)
        - Set up Lambda functions with LocalStack IAM roles/etc.
        - Set up API Gateway with LocalStack (with Flask integration?)
        - Set up S3 bucket for static website w/ LocalStack (no longer CloudFront CDN)
        - Configure basic networking in LocalStack
        - 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:
        - Research/test out different database options such as DynamoDB, RDS, ElastiCache, etc.
        - Compare DynamoDB vs. in-memory storage for local development.

        - Research compute options as well (although I have mostly decided to go with Lambda over EC2)
        - Maybe test basic CRUD operations with different LocalStack DynamoDB
        - Evaluate pros/cons/tradeoffs for MVP and production
        - Update the infrastructure configurations as needed
3. Create Mock DrugBank API (POC)
    - As the developer, I need a fake DrugBank API so that I can simulate drug interaction data without waiting for API approval.
    - Tasks/sub-issues:
        - Research DrugBank API. NOTE: I applied for the research API and was unsuccessful.
        - Set up separate Python Flask (or FastAPI) service to mock DrugBank API.
        - Turn the provided drug interaction data from DrugBank into API-compatible formal (interactions.csv)
        - Create /drugbank/api/v1/drugs/search mock endpoint
        - Create drugbank/api/v1/interactions mock endpoint
        - Maybe even add stuff like response delays or errors to make things realistic (or at least be aware of those possibilities)
        - Create a test dataset with common drugs
        - Deploy the mock API service locally separate from the main Lambda
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:
    - Set up Flask app structure for LocalStack 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 LocalStack Lambda environment (import paths/static files)
    - Add basic CSS styling if I want things prettier
    - Test Flask app locally before LocalStack Lambda deployment
    - Deploy and test Flask Lambda integration via LocalStack API Gateway
    - Add connection to mock DrugBank API service
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:
        - Integrate the mock 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 mock drug interaction data
        - Add error handling for invalid drugs and any mock API failures
        - Design results template to show the interaction information
        - Test with basic/known drugs from existing dataset. Also test edge cases like with fake drugs or API issues.
6. Add basic caching (if doing this) with the DynamoDB (POC)
    - As the owner of this project, I would need caching to improve performance and reduce costs
    - Tasks/sub-issues:

        - Set up LocalStack DynamoDB connection from Lambda
        - Design the caching strategy stuff (like keys/TTL/structure for interactions/etc.)
        - Implement checking the cache before mock API calls in the Flask routes
        - If necessary, will need to configure Lambda IAM permissions for LocalStack DB access
        - SPIKE: add cache logging and basic performance metrics
        - Test caching behavior with repeating drug combinations
        - Handle cache errors
        - Create cache invalidation strategy?
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:
        - 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 mock API calls within the Lambda timeout restriction
        - Update results to show all interactions organized by drug pairs.
        - 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 (like 5-8 drugs)
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:
        - 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?
        - Maybe add severity-based sorting and filtering in the results if there's time.
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:
        - Set up LocalStack CloudWatch monitoring for Flask Lambda function
        - Implement stuff like Lambda function health checks (if not done already) and mock API service.
        - Maybe add Lambda timeout alerts or monitoring error rates or even cost estimation for AWS deployment
        - Test and document as needed
10. Security hardening as needed
    - As the owner of this project, I would want proper security measures before production
    - Tasks/sub-issues:

        - Look into and consider security measures such as validation for form inputs, securing 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`*