Knowledge Graphs for Cybersecurity Reasoning

sdmay23-01

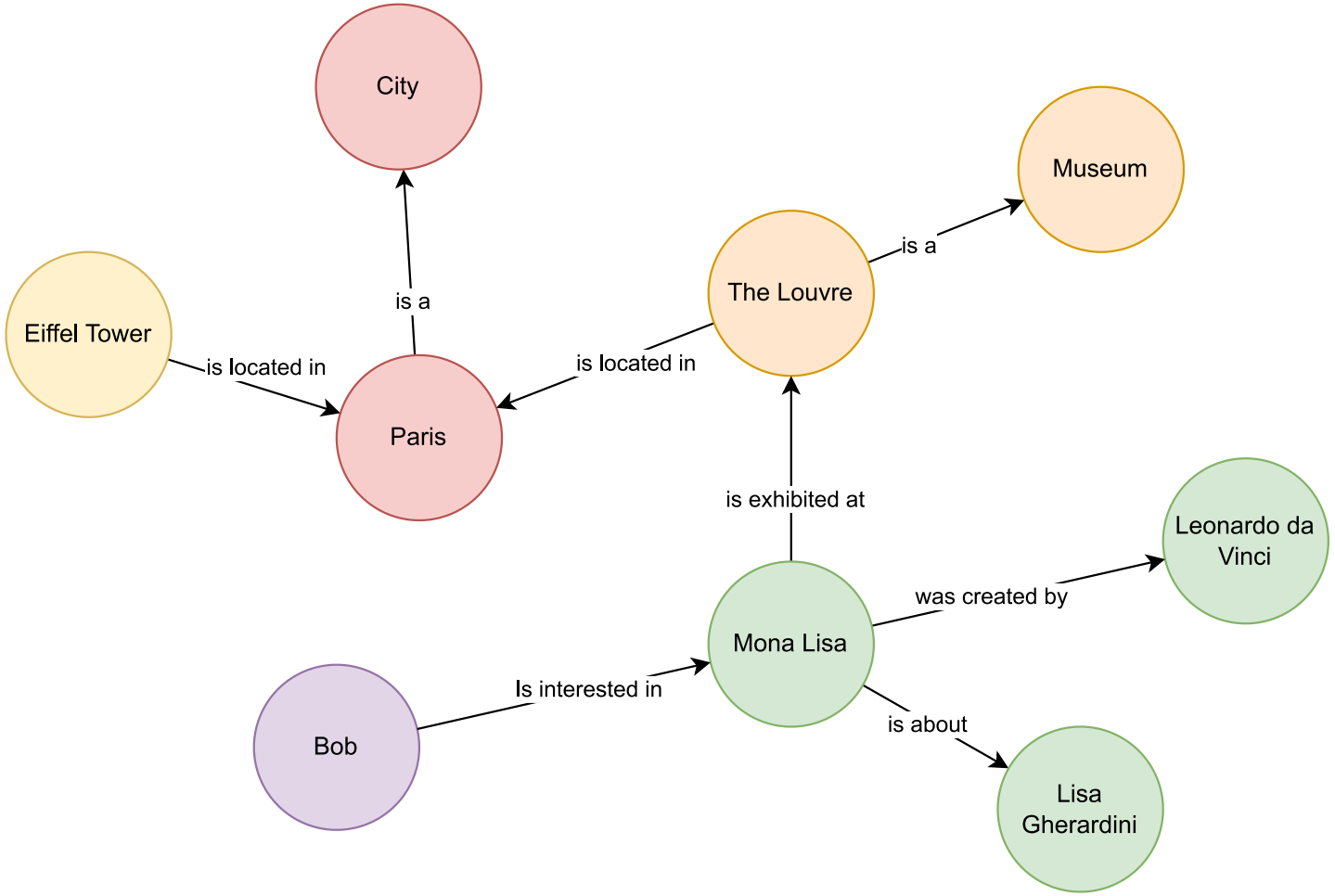
Alice Cheatum	Nicklas Cahill	Michael Watkins
Brandon Richards	Carter Kitelinger	Micah Gwin

Introduction

- Project Overview
  - Perform information extraction on sources to determine cybersecurity named entities and relations between them
  - Build a knowledge graph from entities and relations
  - Continuously update knowledge graph with new information from sources
  - Query and visualize knowledge graph from frontend
- Objectives
  - Efficient querying of cybersecurity entities and their relations
  - Assist cybersecurity researchers and incident responders in contextualizing latest threats
  - Streamline decision-making for cybersecurity risk management

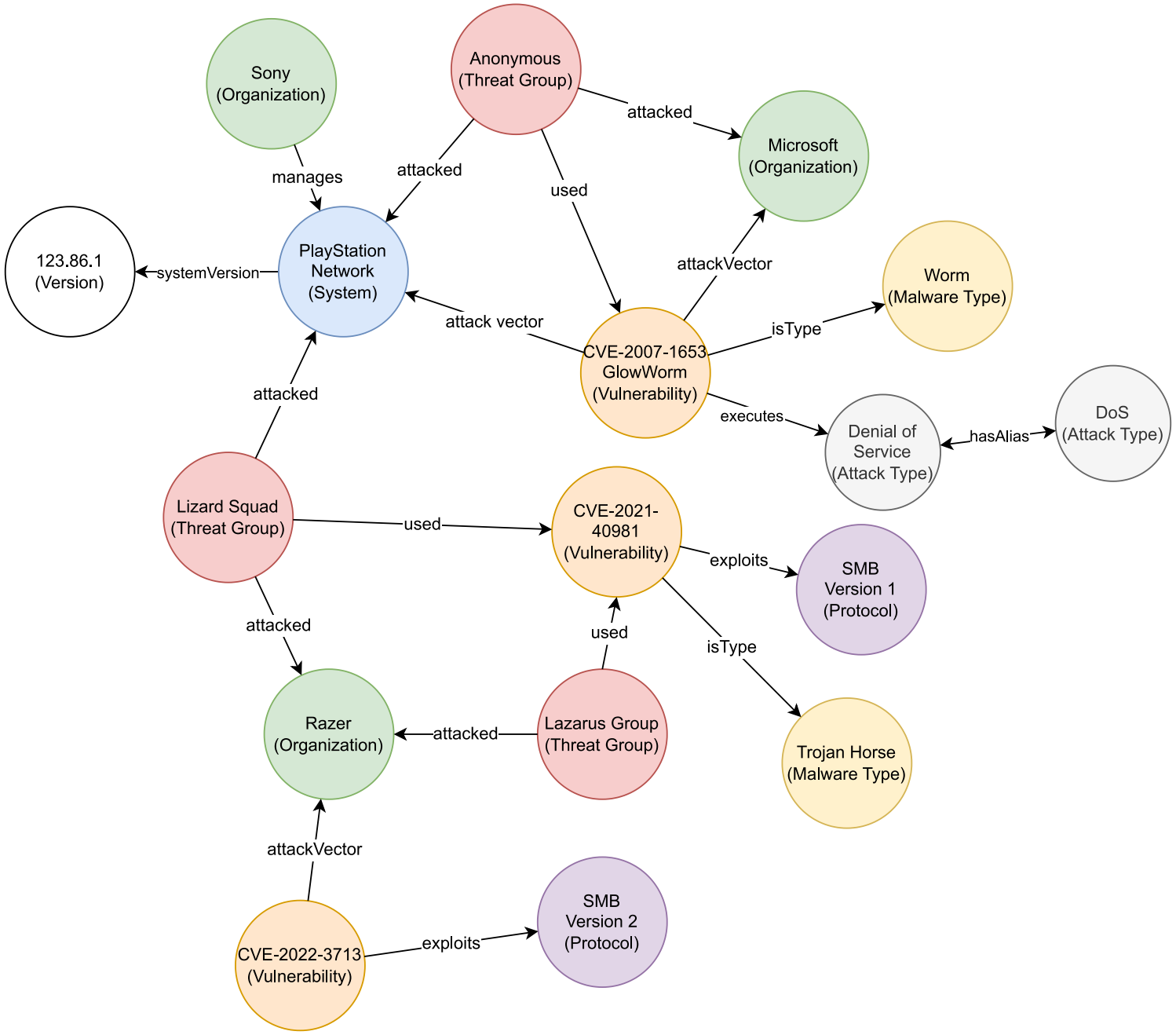
Knowledge Graph

- Represents a network of real-world entities and the relationship(s) between them.
- 3 main components:
  - Nodes
  - Edges
  - Labels
- Used by search engines to complete search queries to give expanded information.
- Example Query: Bob searches "Mona Lisa".



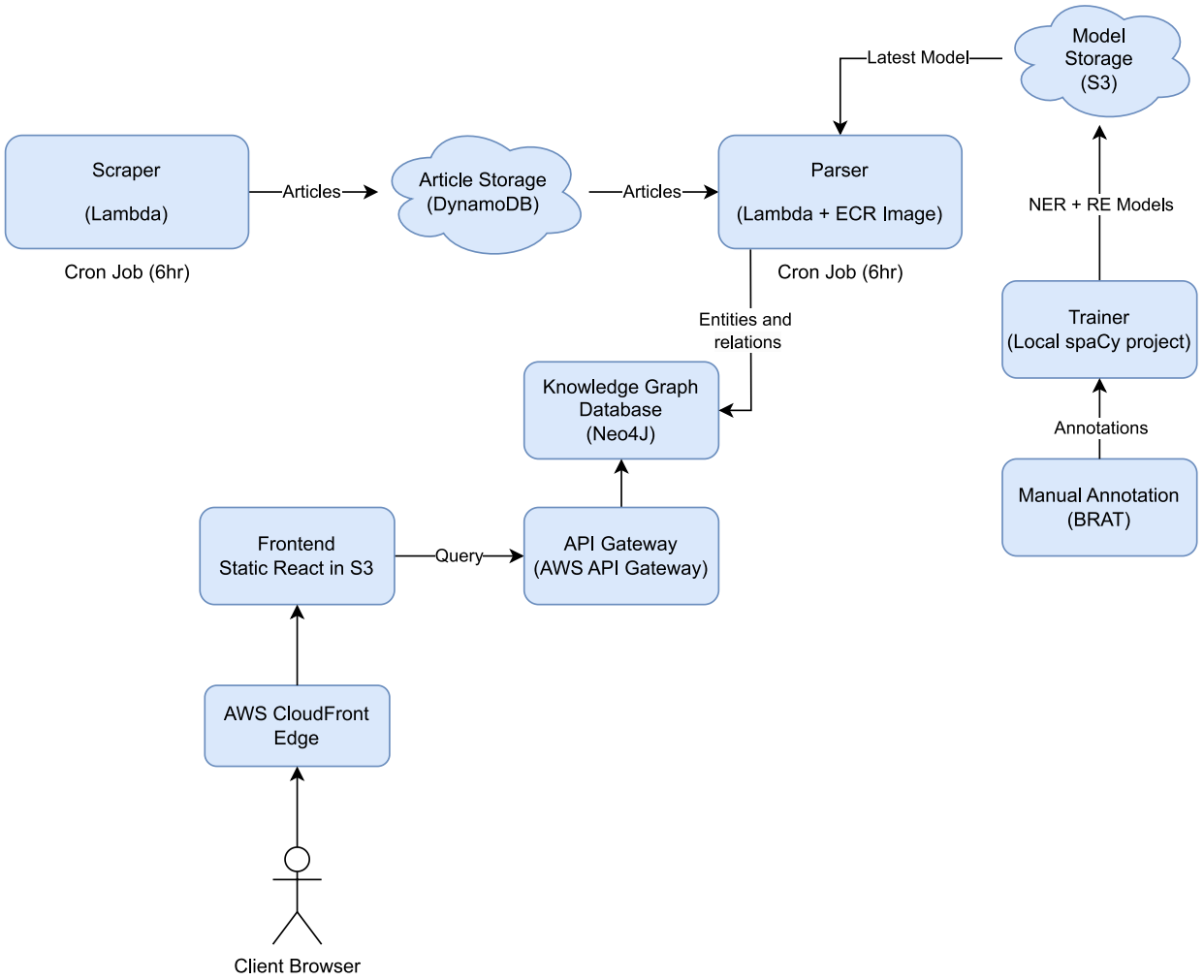
Cybersecurity Knowledge Graph (CSKG)

- Knowledge Graph with domain-specific nodes and edges
- Nodes (Entities)
  - Organization
  - System
  - Threat Group
  - Vulnerability
  - Malware Type
  - etc...
- Edges (Relationships)
  - attacked
  - exploits
  - used
  - attackVector
  - manages
  - isType
  - deployed
  - executes
  - etc...
- Use ontologies to perform queries efficiently
- Example Query: Attacks on PSN



Implementation Architecture

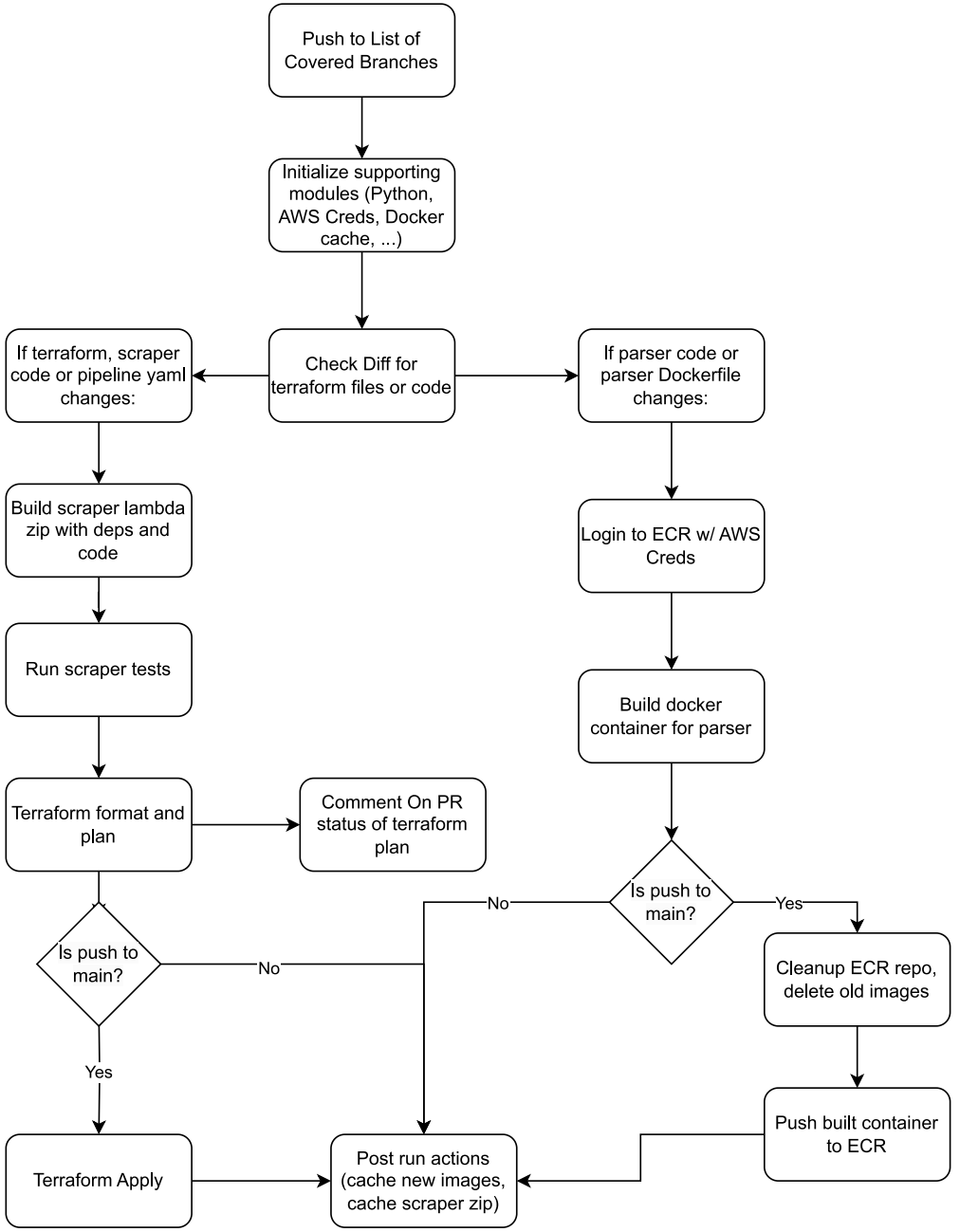
- Trainer
  - Python spaCy project, ran when annotations change
- Scraper
  - AWS Lambda, trigger every 6 hours
- Article storage
  - DynamoDB database
- Parser
  - AWS Lambda with ECR Image, trigger every 6 hours
- Knowledge Graph
  - Neo4J running on EC2, AWS API Gateway proxy
- Frontend
  - Static React webapp hosted with AWS S3 and AWS CloudFront



Deployment Architecture

GitHub Actions Pipeline:

- Check which files/folders have changes
- Build and test changed projects
- Terraform Plan changes to AWS infrastructure
- Terraform Apply on push to main branch (Production)
- If parser container or parser lambda code is modified:
  - Build/cache Parser container
  - Cleanup ECR repository
  - Push latest container to ECR (Production)



Testing

Scraper

- Unit tests
- E2E test
- 26 tests, 92% coverage

```
(base) ~/workspace/sdmay23-01/lambda_scraper >>> (br-parser-tests) make test
============================= test session starts ==============================
platform darwin -- Python 3.10.11, pytest-7.3.1, pluggy-1.0.0
rootdir: /Users/bqrichards/workspace/sdmay23-01/lambda_scraper
collected 26 items

tests/test_all_text_of_child_parser.py .........                        [ 34%]
tests/test_child_of_class_parser.py .......                             [ 61%]
tests/test_e2e.py .                                                     [ 65%]
tests/test_get_parsers.py ....                                          [ 80%]
tests/test_source_list_scraper.py .....                                 [100%]

============================== 26 passed, 3 warnings in 1.85s ==============================
```

Coverage report: 92%

coverage.py v7.2.3, created at 2023-04-29 11:28 -0500

Module	statements	missing	excluded	coverage
AllTextOfChildParser.py	26	0	0	100%
ChildOfClassParser.py	21	0	0	100%
__init__.py	10	0	0	100%
abstract.py	11	0	5	100%
bootstrap.py	25	1	0	96%
dynamo_output.py	21	11	0	48%
source_list_scraper.py	63	11	0	83%
tests/__init__.py	0	0	0	100%
tests/conftest.py	4	0	0	100%
tests/sites_server.py	22	0	0	100%
tests/test_all_text_of_child_parser.py	45	1	0	98%
tests/test_child_of_class_parser.py	42	1	0	98%
tests/test_e2e.py	38	0	0	100%
tests/test_get_parsers.py	17	1	0	94%
tests/test_source_list_scraper.py	39	3	0	92%
Total	**384**	**29**	**5**	**92%**

coverage.py v7.2.3, created at 2023-04-29 11:28 -0500

Parser

- Unit tests
- 23 tests, 81% coverage

```
test/test_app_pipelined.py .....                                        [ 21%]
test/test_blank_parser.py .                                             [ 26%]
test/test_composed_output.py ...                                        [ 39%]
test/test_cve_fetch.py ...                                              [ 52%]
test/test_dynamo_input.py ..                                            [ 60%]
test/test_neo4j_output.py ...                                           [ 73%]
test/test_nlp_parser.py ....                                            [ 91%]
test/test_parser_context.py .                                           [ 95%]
test/test_simple_input.py .                                             [100%]
```

Coverage report: 81%

coverage.py v7.2.3, created at 2023-04-29 22:17 -0500

Module	statements	missing	excluded	coverage ↑
kgfcrparser/outputs/dynamo_marked_parse.py	25	14	0	44%
kgfcrparser/parsers/nlp_parser.py	54	30	0	44%
kgfcrparser/outputs/neo4j_output.py	21	8	6	62%
kgfcrparser/app_pipelined.py	60	8	3	87%
kgfcrparser/inputs/dynamo_input.py	53	7	0	87%
kgfcrparser/cvefetch.py	41	1	0	98%
kgfcrparser/__init__.py	1	0	0	100%
kgfcrparser/context.py	13	0	0	100%
kgfcrparser/inputs/__init__.py	0	0	0	100%
kgfcrparser/inputs/parser_input.py	9	0	4	100%
kgfcrparser/inputs/simple_input.py	23	0	0	100%
kgfcrparser/outputs/__init__.py	0	0	0	100%
kgfcrparser/outputs/composed_output.py	11	0	0	100%
kgfcrparser/outputs/neo4jdrivers/__init__.py	0	0	0	100%
kgfcrparser/outputs/parser_output.py	6	0	2	100%
kgfcrparser/parsers/__init__.py	0	0	0	100%
kgfcrparser/parsers/blank_parser.py	6	0	1	100%
kgfcrparser/parsers/parser_parser.py	5	0	1	100%
kgfcrparser/parsertypes.py	28	0	0	100%
Total	**356**	**68**	**17**	**81%**

Frontend

- Unit tests
- UI tests
- 42 tests

```
(base) ~/workspace/sdmay23-01/frontend >>> (develop) yarn test
yarn run v1.22.19
$ jest
 PASS  src/tests/Queue.test.ts
 PASS  src/tests/Stack.test.ts
 PASS  src/tests/GraphMapper.test.ts
 PASS  src/tests/QueryMapper.test.ts
 PASS  src/tests/ExampleQueriesContainer.test.tsx
 PASS  src/tests/FilterSider.test.tsx
 PASS  src/tests/HomePageHeader.test.tsx

Test Suites: 7 passed, 7 total
Tests:       42 passed, 42 total
Snapshots:   0 total
Time:        4.115 s, estimated 6 s
Ran all test suites.
✨  Done in 4.50s.
```

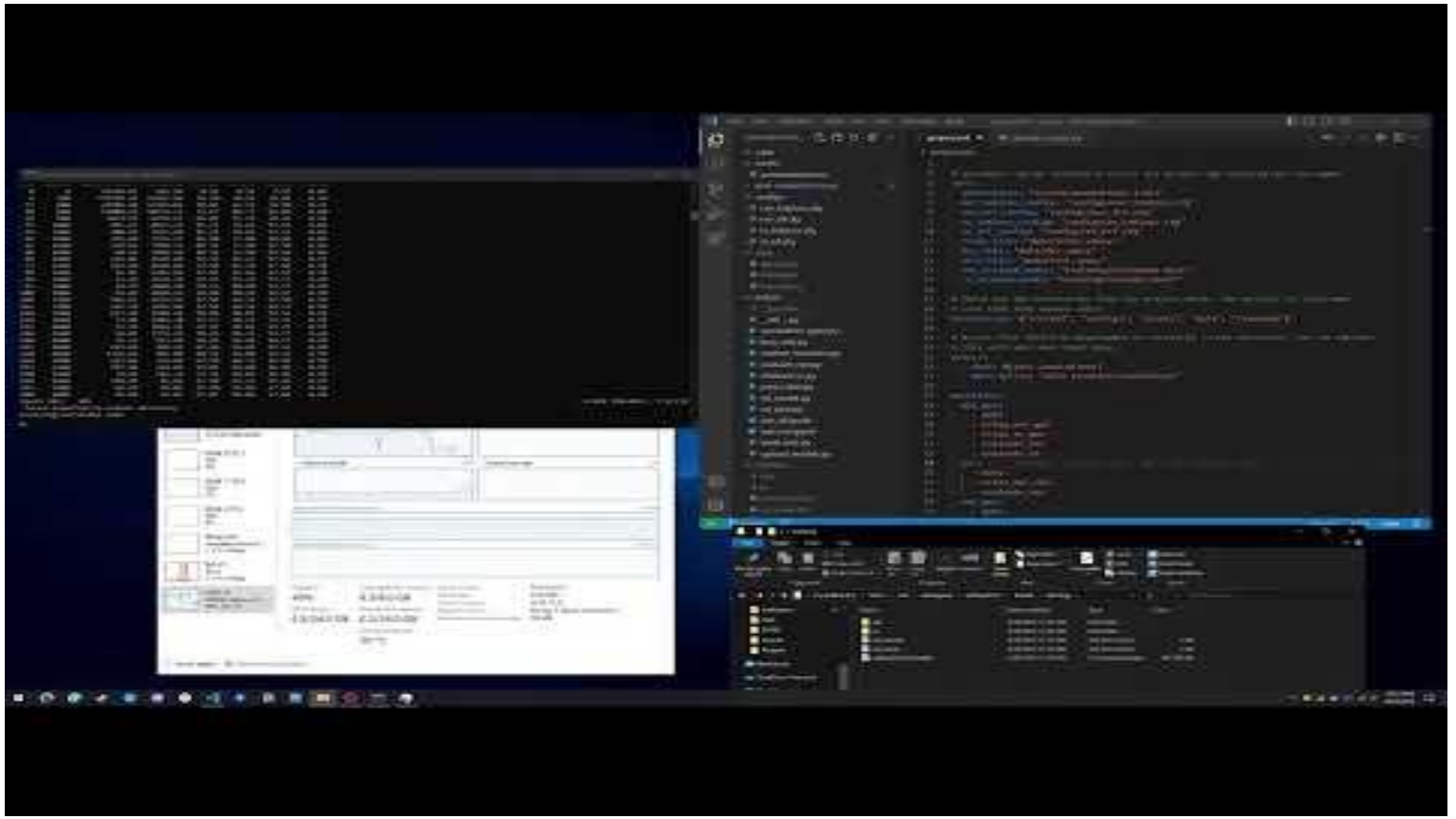
Key Contributions

- Brandon Richards
  - Create NER and RE model training software, configure for GPUs and transformers, upload to S3
  - Build and deploy React frontend for querying and displaying KG
- Micah Gwin
  - Setup the GitHub actions pipeline to monitor branches, then build and deploy IAC.
  - Cloud Infrastructure management and lambda development.
- Carter Kitelinger
  - Researched NER and RE training, training metrics
  - Created example Neo4J Cypher queries
  - Annotated articles for training models
- Alice Cheatum
  - Define entities and relations
  - Researched relationship extraction techniques
  - Wrote python script to fetch CVE data
  - Annotated articles for training
- Nicklas Cahill
  - Annotated Articles for training
  - Defined/Redefined entities and relations
  - Researched NER and RE training
  - Misc. supporting Python scripts
- Michael Watkins
  - Annotated articles for training
  - General infrastructure
  - Pipelined parser

Accomplishments

- NER and RE training pipeline using spaCy
- Scrape articles from source list on schedule and store in NoSQL database
- Multi-threaded modular parser that reads input from article storage, parses using trained models, and outputs to Neo4J Knowledge Graph
- React frontend with query builder, queries Neo4J and displays nodes and edges

Demo



Challenges and Solutions

- Entity and relationship definitions
  - Not too broad or too specific
  - Changes disrupted training and annotating
  - Solution: Rework several times as a team
- BRAT annotation format and spaCy
  - Solution: Write tool to convert annotation format (considering line endings)
- Limited pipeline build minutes
  - Solution: Caching artifacts, rebuild only if difference
- Building and running parser with dependencies
  - Dependencies of parser are 2GB+
  - Solution: Docker image uploaded to AWS ECR
- Converting graphical query into Cypher statement
  - Solution: BFS Algorithm

Future Work

- Training / Model
  - NER – Train on cybersecurity corpus
  - RE – Improve accuracy, more research
  - Combine into one model
  - Use MCC to score and evaluate models instead of F1
- CI/CD Pipeline
  - Split develop and production resources
- Parser
  - Test for valid relations before inserting into KG
- Knowledge Graph
  - Migrate from Neo4J on EC2 to AWS Neptune
- Frontend
  - Accounts
  - Monitor usage
  - Save queries
- Testing
  - Integration tests between components

Conclusion

- Many of the original project objectives have been met
  - Generate Cybersecurity Knowledge Graph
  - Create and train a machine learning model
  - Aggregate information from a variety of sources
- Additional stretch goals deemed important for operation were also met
  - Query KG from frontend
  - Deploy to AWS cloud infrastructure
  - Run automatically every 6 hours
  - Use interfaces in components for modularity
- Future enhancements
  - Larger article dataset for training/increase accuracy of NER and RE
  - Natural Language Processing for frontend queries
  - Parser constraints for valid relations
  - Overall improvement for code structure/pipeline