CSE 354: Distributed Computing Phase (4)

Presented to Dr. Ayman Bahaa Eldin, Dr. Hossam Mohamed AbdelRahman,

Eng. A'laa Hamdy, & Eng Mostafa Ashraf

September 2025

Team (29)

Name	**ID**
Omar Ahmed Ahmed AbdAlAal	**18P2998**
Ahmed Mohamed	**2100723**
Abdelrahman Khaled Alkawwa	**21P0285**
Kevin Sherif Safwat	**20P9405**
Mostafa Ayman Mostafa	**20P9883**

Contents

1 Code Repository

`https://github.com/omar4a/distributed`

2 Demo Video

`https://drive.google.com/file/d/15vGC4Gj4IQeDxQ8Zt-Y42wGoD1NvGHAj/view?usp=sharing`

3 Phase (4) Questions

3.1 Objective

To design & implement a scalable, fault-tolerant, distributed web crawling system capable of:

1. Efficiently crawling and extracting URLs from web pages.
2. Parsing web documents' HTML content and indexing it.
3. Search functionality where words are associated with URLs.
4. Using a distributed architecture, in which there's a master node, crawler nodes, and indexer nodes working on different cloud VMs.
5. Scaling easily: Adding more crawler & indexer nodes seamlessly as data size increases.
6. Being fault tolerant: Continued operation in case a node fails or a crawling error is encountered. [1]

3.2 Project Scope

The system is designed to handle large-scale data acquisition and indexing tasks, with flexibility to adapt to diverse workloads.
The system can be slightly modified and adapted to fit various real-world scenarios like:

- **Search Engine Development:** The system serves as the backbone by efficiently crawling, indexing, and organizing web content for quick and accurate search retrieval.
- **Competitive Market Intelligence:** Businesses can monitor competitors by collecting and analyzing public content such as pricing, promotions, or product catalogs.
- **E-commerce Platforms:** The crawler collects product data from marketplaces, enabling dynamic pricing strategies.
- **Big data analytics:** It gathers massive amounts of web data for insights, such as sentiment analysis, trend detection, or training AI models.
- **Cybersecurity Monitoring:** Crawls forums or websites to detect vulnerabilities, malicious content, or emerging cybersecurity threats.
- **Academic Research:** Researchers can use it to gather publicly available data to study trends or train algorithms.
- **Natural Language Processing Tasks:** It provides large textual datasets that can be indexed and used for training language models, creating conversational agents, or analyzing semantics.

And other applications.

3.3 Architecture

The system follows a **Publisher-Subscriber** architecture[2].

- The master publishes the seed URLs into a queue, which the crawlers are ”subscribed” to.
- The crawlers receive the seed URLs, and publish the crawled URLs into another queue, which only the master is subscribed to.
- The master then republishes these URLs into the first queue which the crawlers are subscribed to, implementing a simple workload-distribution mechanism.
- Similarly, the crawlers publish the crawled content into a queue which the indexers are subscribed to.
- The index is distributed among the indexer nodes. After crawling is done, the master receives a search query, sends it to all indexer nodes, aggregates the result, and displays it to the user.

This architecture provides a decoupled environment, in which more crawler and indexer nodes can be added seamlessly without having to change any system components. Also, more nodes can be added *while the system is in operation.* It also supports fault-tolerance, as the system will continue functioning (albeit with less performance) if a crawler or indexer node fails.[3]
Finally, to avoid the master node being a single point of failure, the system has a *second, inactive* master node, which takes over the master operations in case of master node failure, seamlessly continuing the workflow.

3.4 System Testing

3.4.1 Functional Testing

The system satisfies all user stories. The system implements crawling, indexing, searching, and respects robots.txt.
The system uses *Puppeteer* and a *chrome web driver* to fetch dynamic HTML content, as most modern web-pages don’t have static HTML content which can be crawled with the the static requests method.

3.4.2 Fault Tolerance Testing

- Handles HTML transient errors (404, 403, etc) gracefully.
- Handles arbitrary faults by using timeout and exception-handling mechanisms.
- Handles failures of crawler, indexer, and master nodes.
- The system can ”restart”, retrieving the crawled content and index from the last session from persistent cloud storage.

3.4.3 Scalability Testing

- More crawler and indexer nodes can be integrated into the system seamlessly.
- No code modifications are needed. Simply plug-and-play.
- More crawler and indexer nodes can start operation while the system is inside a crawling session, effectively implementing dynamic workload handling.
- When more crawler/indexer nodes are added, the system finishes the crawling/indexing session by a factor of the number of added nodes compared to the original number. (E.g, 2 crawlers 2 minutes, 4 crawlers 1 minute, etc)

3.4.4 Crawl Quality

- Uses a chrome-driver to fetch dynamic JS HTML content for full coverage.
- Respects robots.txt and skips any web-pages on which crawling is not aollwed, most notably social media platforms.

3.5 Performance Tuning

- Each node's main loop runs on two separate threads to maximize resource usage, as AWS' cloud VMs free version provides 2 CPUs per VM.
- The biggest identified performance bottlenck is using the webdriver to fetch dynamic HTML content. No solution was found. The tradeoff is retrieving static HTML only, which renders some crawling sessions impractical and almost useless in a real-world scenario.

3.6 Security Review

Cloud resources (VMs, Queues, Database) can only be accessed by authorized users, using Amazon's built-in access keys and secret keys. The user must set up credentials on the local machine to access cloud resources.
Each user and VM has a unique key "ARN", which Amazon authenticates before giving access to any resource.

4 User Manual (README.txt)

To run the distributed web-crawling system and begin a crawling session, please do the following:

1. Launch the cloud VM with your favorite cloud provider.
2. Install python3, pip, and git.

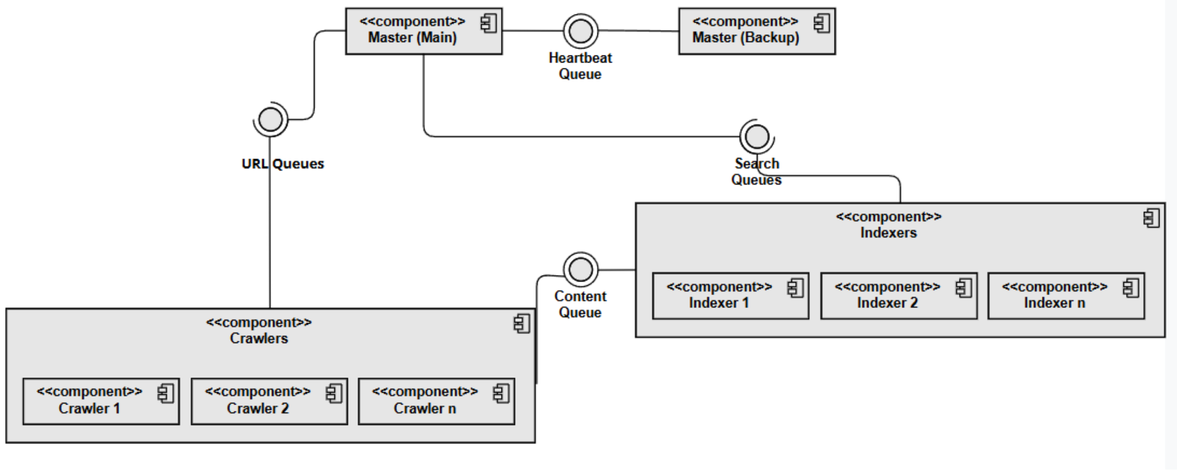
3. Clone the GitHub repo: https://github.com/omar4a/distributed.git
4. Run pip install requirements.txt
5. To run a crawler node: python3 crawler.py
6. To run an indexer node: python3 indexer.py
7. To run a master backup node: python3 $master_backup.py$
8. To run the master node and begin the crawling session: python3 master.py
9. You can run the nodes in any order. If you run the crawlers and indexers first, they will wait for a master node to provide seed URLs and crawling parameters.
10. When you run the master node, you will be prompted for seed URLs and depth parameters.
11. You can add more crawler and indexer nodes during a session. Simply follow the steps above.
12. When the crawling session is done, the master's CLI will prompt you for a search query. You can enter search queries and view search results.
13. If you want to exit, provide "/.quit" as the search query, all the nodes will shut down.

Note: You must either have access to the existing SQS queues, or create new SQS queues with these names, and provide access to them to your VMs:

- $crawled_content.fifo$
- $crawled_URLs.fifo$
- $crawler_completion.fifo$
- $index_completion.fifo$
- $master_heartbeat.fifo$
- $Queue1.fifo$
- $search_query.fifo$
- $search_results.fifo$
- $shutdown.fifo$

5 System Architecture

5.1 Component Diagram



5.2 Component Descriptions

Master:

- The master node serves as the central coordinator of the system.
- Provides CLI for user-interaction.
- Distributes tasks (URLs to be crawled) to the crawler nodes via the SQS queue.
- Processes results (extracted URLs and data) received from the crawler nodes via another SQS queue.
- Sends search queries to indexers, and aggregates search results from the distributed index.

Master (Backup)

- Serves as a backup Master in case of master node failure
- Sends a periodic heartbeat message to the master
- Takes over master operations in case master fails to respond to the heartbeat after the specified timeout period.

Crawlers:

- Independently fetch web pages and extract information (URLs, textual content)
- Use the SQS queues to receive tasks from the master and push extracted URLs back to the master for reassignment.
- Send the crawled content to the indexers for indexing.

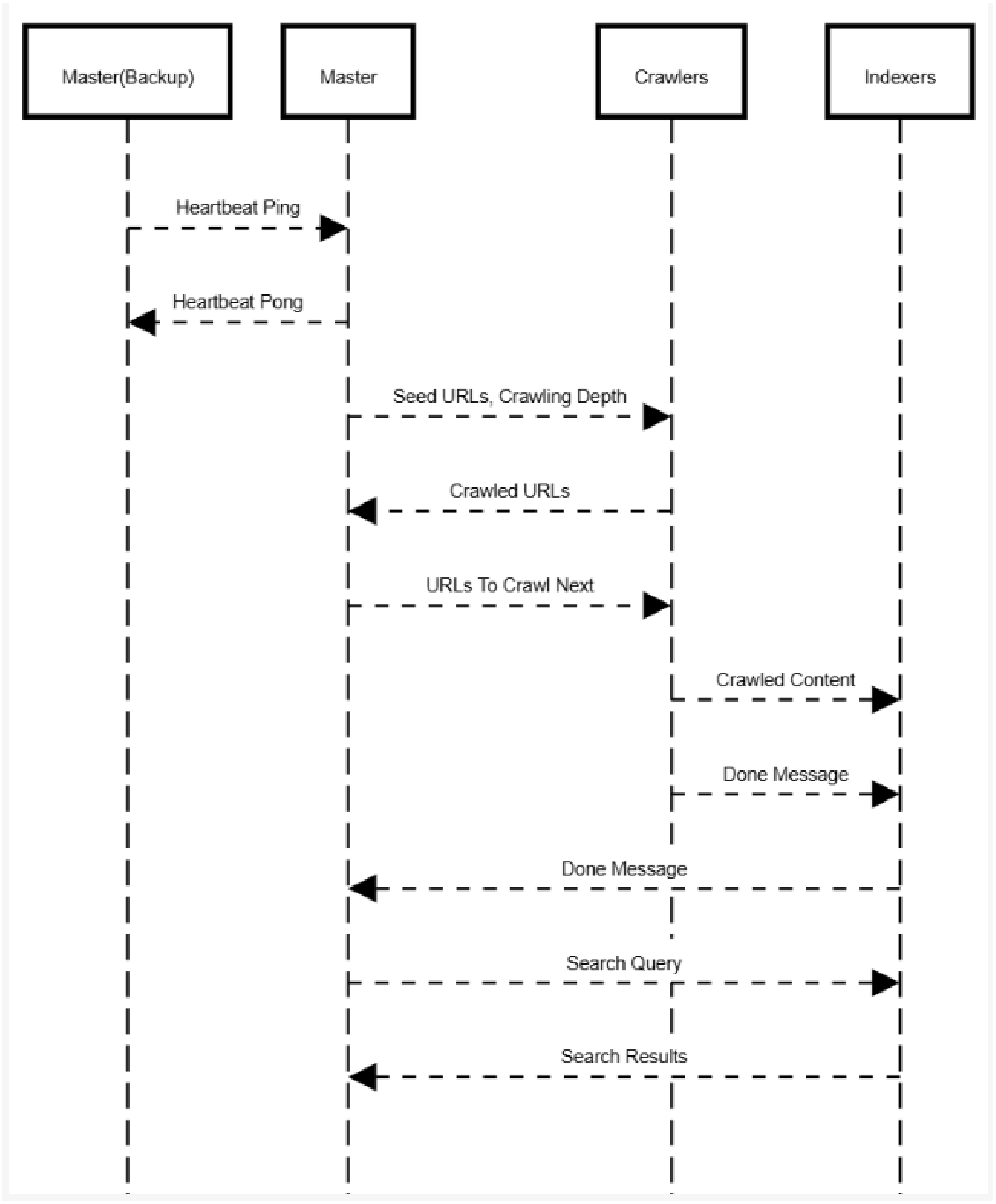
Indexers:

- Processes and organizes crawled data into a searchable whoosh index.
- Stores the processed data in an optimized format for querying.
- Processes search queries and sends results back to the master.

Client Interface

- Prompts the user for seed URLs and crawling depth, and passes them to the master to begin crawling.
- Has basic monitoring for crawling progress.
- After crawling & indexing is done, prompts the user for search queries and displays search results.

5.3 Sequence Diagram



6 Fault Tolerance Plan

- **Decoupled Communication:** Using a pub-sub based model to avoid connection-oriented communication. The rest of the nodes can continue to communicate if one or more nodes fail.
- **Node Redundancy:** Having more than one crawler/indexer node. Having an extra, inactive master node which runs if the main master node fails.
- **Crawling Error handling:** Graceful handling of transient HTTP errors (e.g., 403, 500), If a crawl consistently fails, the crawler will log the issue, skip the URL, and proceed with other tasks.
- **Data Duplication:** Crawled and indexed data is stored in S3 redundantly. And index is distributed among indexer nodes.
- **Graceful Shutdown:** Master sends a system-wide shutdown message, which initiates a comprehensive shutdown routine.
- **Exception Handling:** Abundant use of exception-handling mechanisms in the code to handle potential errors gracefully, log them, and continue operation.

7 Challenges, Lessons, & Future Work

7.1 Challenges Faced

- The biggest challenge was establishing comprehensive communication between the three main system components: master, crawlers, and indexers. In a way that allows seamless addition of more nodes, and fault-tolerance if a node fails.
- It was also challenging to implement the backup master mechanism. The "handoff" was especially challenging, making the backup master *start from where the main master stopped.* But we insisted on implementing this because one master node was SPOF and absolutely not fault-tolerant.
- Finally, it took us a while to figure out why web page's content wasn't being crawled properly, and understand the difference between static and dynamic content, and implement the web-driver mechanism on the limited resources of the free AWS VMs.

7.2 Lessons Learned

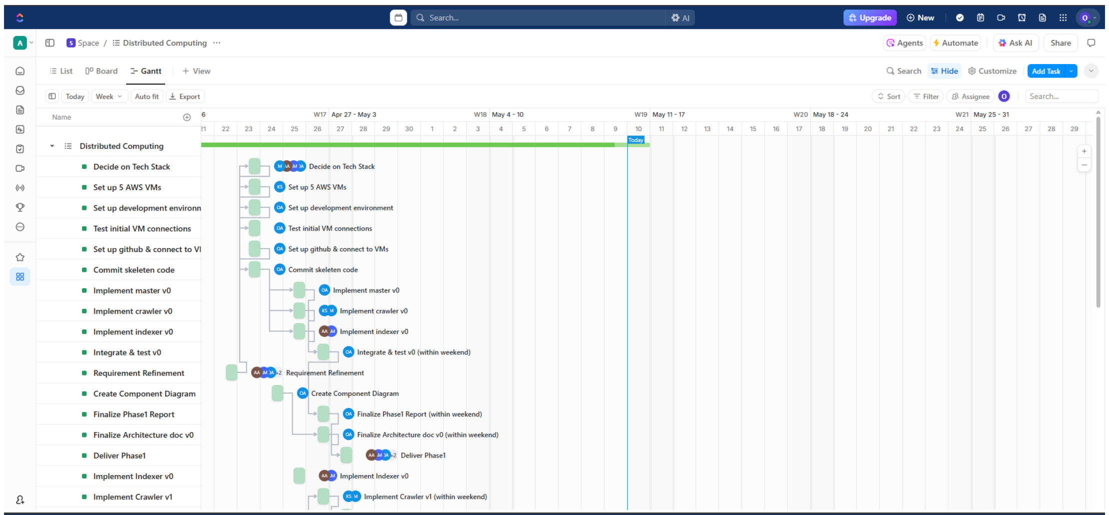
- How to launch cloud VMs, and deal with different cloud services like queues and databases.

- How to distribute a large computational workload onto different machines and coordinate between them.
- How to crawl the internet appropriately and efficiently, and without getting banned.
- How to deliver a software system incrementally.

7.3 Future Work

- If we are to revisit the system, probably the first thing we would implement is a GUI application which runs locally and calls the VMs remotely. We thought about implementing this but there was not enough time.
- There's also plenty of room for performance optimization. We believe crawling one webpage takes too long now with the web-driver implementation.
- We can probably use 2 or 3 queues instead of 9 different queues, distinguishing between message groups with unique MessageGroupIDs, which is a feature provided by AWS.
- We can implement AWS' "dynamic worklaod" feature, which starts more VM instances dynamically based on the workload.
- Finally, we may modify and adapt this system for a real web-crawling application, like business intelligence or cybersecurity monitoring.

8 Full Project Plan



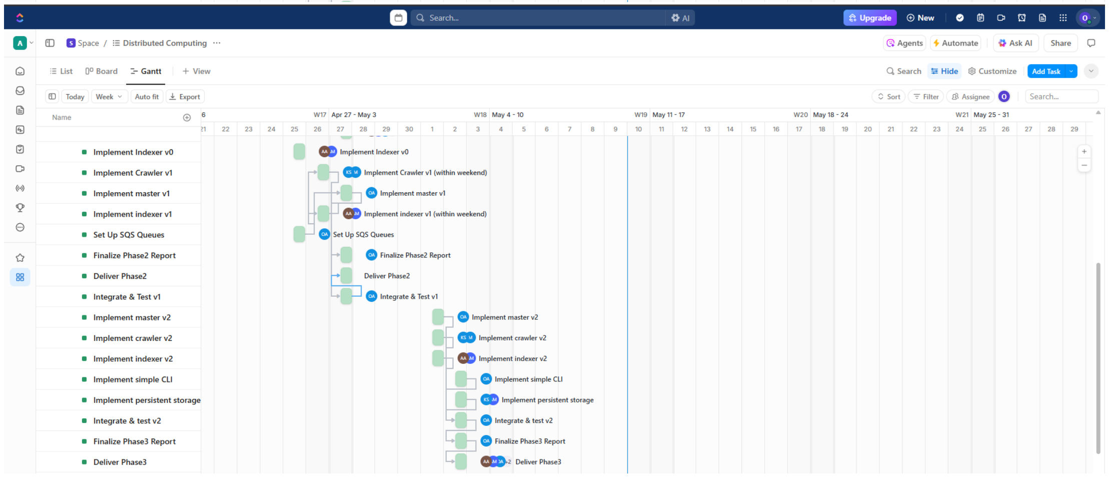
Space / Distributed Computing

List | Board | Gantt | + View

Name

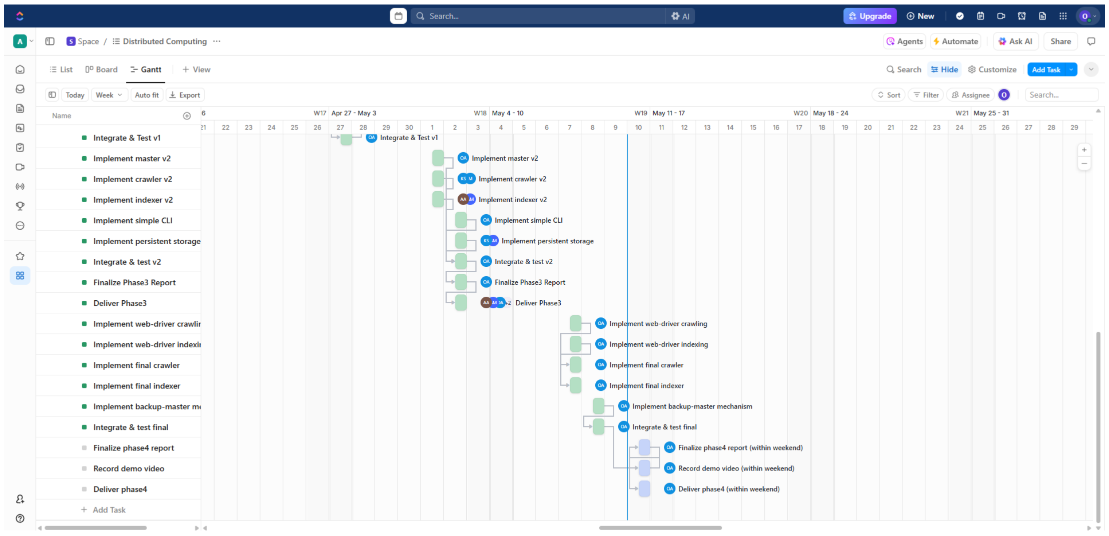
Distributed Computing
- Decide on Tech Stack
- Set up 5 AWS VMs
- Set up development environn
- Test initial VM connections
- Set up github & connect to VI
- Commit skeleton code
- Implement master v0
- Implement crawler v0
- Implement indexer v0
- Integrate & test v0
- Requirement Refinement
- Create Component Diagram
- Finalize Phase1 Report
- Finalize Architecture doc v0
- Deliver Phase1
- Implement Indexer v0
- Implement Crawler v1

Decide on Tech Stack
Set up 5 AWS VMs
Set up development environment
Test initial VM connections
Set up github & connect to VMs
Commit skeleton code
Implement master v0
Implement crawler v0
Implement indexer v0
Integrate & test v0 (within weekend)
Requirement Refinement
Create Component Diagram
Finalize Phase1 Report (within weekend)
Finalize Architecture doc v0 (within weekend)
Deliver Phase1
Implement Indexer v0
Implement Crawler v1 (within weekend)



- Implement Indexer v0
- Implement Crawler v1
- Implement master v1
- Implement indexer v1
- Set Up SQS Queues
- Finalize Phase2 Report
- Deliver Phase2
- Integrate & Test v1
- Implement master v2
- Implement crawler v2
- Implement indexer v2
- Implement simple CLI
- Implement persistent storage
- Integrate & test v2
- Finalize Phase3 Report
- Deliver Phase3

Implement Indexer v0
Implement Crawler v1 (within weekend)
Implement master v1
Implement indexer v1 (within weekend)
Set Up SQS Queues
Finalize Phase2 Report
Deliver Phase2
Integrate & Test v1
Implement master v2
Implement crawler v2
Implement indexer v2
Implement simple CLI
Implement persistent storage
Integrate & test v2
Finalize Phase3 Report
Deliver Phase3



- Integrate & Test v1
- Implement master v2
- Implement crawler v2
- Implement indexer v2
- Implement simple CLI
- Implement persistent storage
- Integrate & test v2
- Finalize Phase3 Report
- Deliver Phase3
- Implement web-driver crawlir
- Implement web-driver indexir
- Implement final crawler
- Implement final indexer
- Implement backup-master me
- Integrate & test final
- Finalize phase4 report
- Record demo video
- Deliver phase4
- Add Task

Integrate & Test v1
Implement master v2
Implement crawler v2
Implement indexer v2
Implement simple CLI
Implement persistent storage
Integrate & test v2
Finalize Phase3 Report
Deliver Phase3
Implement web-driver crawling
Implement web-driver indexing
Implement final crawler
Implement final indexer
Implement backup-master mechanism
Integrate & test final
Finalize phase4 report (within weekend)
Record demo video (within weekend)
Deliver phase4 (within weekend)

References

[1] Dr. Ayman Bahaa-Eldin. "Dr. Ayman's Slides". In: *ASU Engineering* (2025).

[2] Andrew S. Tanebaum Maarten Van Steen. *Distributed Systems 4th Edition.* Maarten van Steen, 2023.

[3] Eng. A'laa Hamdy. "Eng A'laa Hamdy's Slides". In: *ASU Engineering* (2025).