MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS

Computer Engineering Program

CNG 495 CLOUD COMPUTING

FALL – 2023

TERM PROJECT PROPOSAL

“CloudLink”

Prepared By

Buğra İLHAN (2315307)

Başak Özarslan (2385623)

İlayda Yağmur Karadağ (2315364)

Project Description:

CloudLink is a comprehensive URL shortener service designed for efficient link management and sharing. Deployed on the Heroku cloud platform, the application utilizes Heroku Postgres for secure and scalable storage of original and shortened URLs. To enhance performance, Heroku Redis is employed for in-memory caching, minimizing database queries and optimizing user experience. The primary goal is to provide users with a fast and reliable URL shortening service, ensuring seamless navigation through shortened links. The application will generate unique short codes for submitted long URLs, store them in the Heroku Postgres database, and utilize Heroku Redis for caching frequently accessed short URLs.

Cloud Delivery Models:

In the context of the CloudLink - URL Shortener Service project, we are leveraging a combination of cloud delivery models to build and deploy the application. Here's a more detailed explanation of how SaaS, PaaS, and IaaS are used together:

1) **SaaS (Software as a Service):**

CloudLink is delivered as a SaaS, where the URL shortener application itself is provided as a service to users. Users can access the functionality of the URL shortener without having to worry about the underlying infrastructure or software components. They interact with the application through a user interface to submit and retrieve URLs.

2) **PaaS (Platform as a Service):**

Heroku, the cloud platform of choice for this project, serves as the PaaS. It abstracts away the underlying infrastructure, providing a ready-made platform for deploying, managing, and scaling applications. With Heroku, we don't need to manage the servers or the operating system; instead, we focus on developing and deploying your application. Heroku also provides additional services and tools that simplify the deployment process.

3) **IaaS (Infrastructure as a Service):**

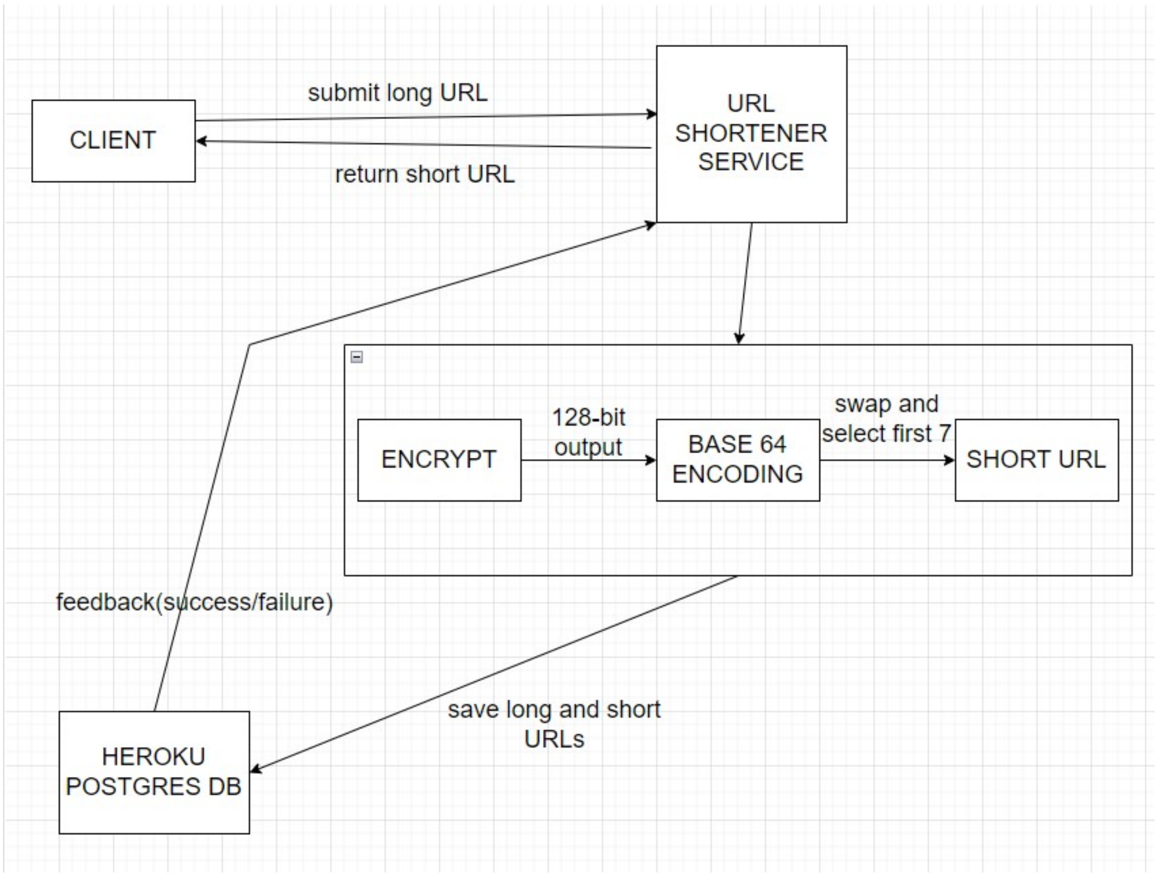
Although not explicitly managed by us, IaaS is implicitly part of the overall architecture. Heroku, while primarily a PaaS, is built on top of infrastructure provided by cloud providers (such as AWS, which provides the underlying infrastructure for Heroku). This infrastructure includes servers, storage, and networking components, but we don't need to directly manage or interact with them. Heroku abstracts away the complexities of infrastructure management, allowing us to focus on the application.

In summary, we are using SaaS for delivering the URL shortener application, PaaS for the underlying platform that simplifies deployment, and IaaS as the foundation that supports the entire architecture, even though we don't directly interact with or manage the infrastructure. This combination allows for a streamlined development and deployment process, where you can concentrate on building and improving the application without getting bogged down by infrastructure management.

Diagrams/Figures:

1) **Client-Service-Interaction:**

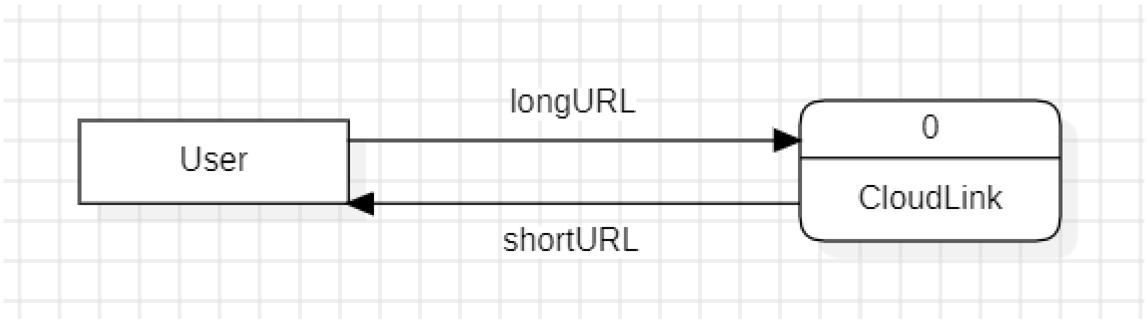
Illustrate how users interact with the URL shortener service, including the submission of long URLs and accessing shortened URLs.



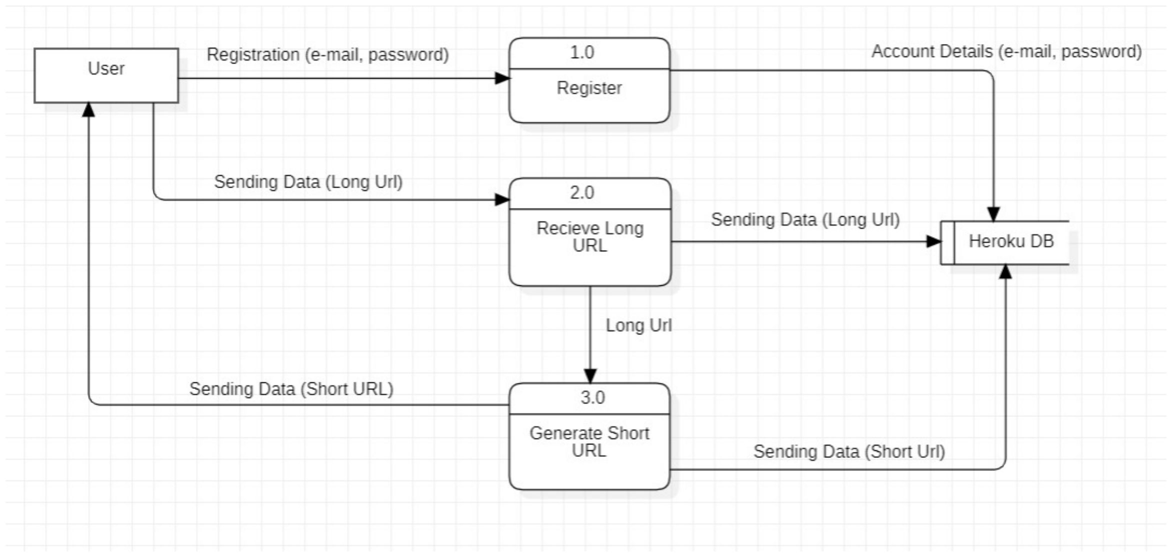
2) Data Flow:

Diagram showcasing the flow of data from user input to storage in Heroku Postgres, with caching in Heroku Redis for improved performance.

Context Data Flow:



Level-0 Data Flow:



3) Data Types:

Specify the types of data involved, distinguishing between binary (e.g., images) and text (URLs, metadata) data.

- Text Data:
- ❖ Long URLs: The original URLs submitted by users are text data. These can be stored as strings in the database.

- ❖ Short Codes: The unique identifiers generated for the long URLs are also text data, typically alphanumeric strings.
- ❖ Metadata: Any additional information associated with the URLs, such as creation date, user details, or analytics data, is text-based.

- Binary Data:
- ❖ Images (if applicable): If your URL shortener service supports custom URL short codes with images, the image data would be binary. This could include user-uploaded images associated with custom short URLs.
- ❖ Cached Data: Data stored in the Redis cache, such as frequently accessed short URLs, might include binary representations for images or other binary content to improve retrieval speed.
- Mixed Data:
- ❖ Short URLs: While the short URLs themselves are often constructed as text (e.g., "https://yourdomain.com/abc123"), they can be considered as a mix of text and symbolic data.

4) Computation:

- URL Shortening Logic:

Description: The core computational process involves taking a long URL submitted by the user and generating a unique short code. This process requires algorithms to create a unique identifier, which can be a combination of alphanumeric characters, ensuring uniqueness and collision prevention.

- Database Interaction:

Description: The application logic interacts with the Heroku Postgres database to store and retrieve data. This includes computational processes for inserting new records (long URL and short code) into the database and querying the database to retrieve the original URL based on the short code.

- Caching Mechanism:

Description: The caching process utilizes Heroku Redis to store frequently accessed short URLs. The computational aspect involves checking the cache to determine if the short URL is present. If it is, the original URL is retrieved directly from the cache. If not, the application queries the Heroku Postgres database to fetch the required data.

- Short URL Construction:

Description: Once the short code is generated, the application constructs the short URL by combining it with the domain of the URL shortener service. This process is a simple concatenation, representing a straightforward computational task.

- User Interface Logic:

Description: If the application includes a user interface for submitting and retrieving URLs, there will be computational processes related to handling user input, managing form submissions, and presenting information to users.

- Error Handling and Logging:

Description: Computational processes should include error handling mechanisms to manage potential issues, such as invalid input or database errors. Additionally, logging processes may be implemented to record events and monitor system behavior.

5) Expected Contribution for Each Project Group Member:

- Backend Developer(Bugra Ilhan): Responsible for implementing the URL shortening logic, database interactions, and integration with Heroku Postgres and Redis.
- Frontend Developer(Basak Ozarslan): Designs the user interface for URL submission and retrieval, ensuring a seamless experience for users.
- DevOps Engineer(Ilayda Yagmur Karadag): Manages deployment on Heroku, configures environment variables securely, and ensures the overall system reliability.

6) References:

- Heroku. (n.d.). Heroku Documentation. Retrieved from https://devcenter.heroku.com/
- PostgreSQL Global Development Group. (n.d.). PostgreSQL Documentation. Retrieved from https://www.postgresql.org/docs/
- Redis. (n.d.). Redis Documentation. Retrieved from https://redis.io/documentation