Digi Home' System Architecture

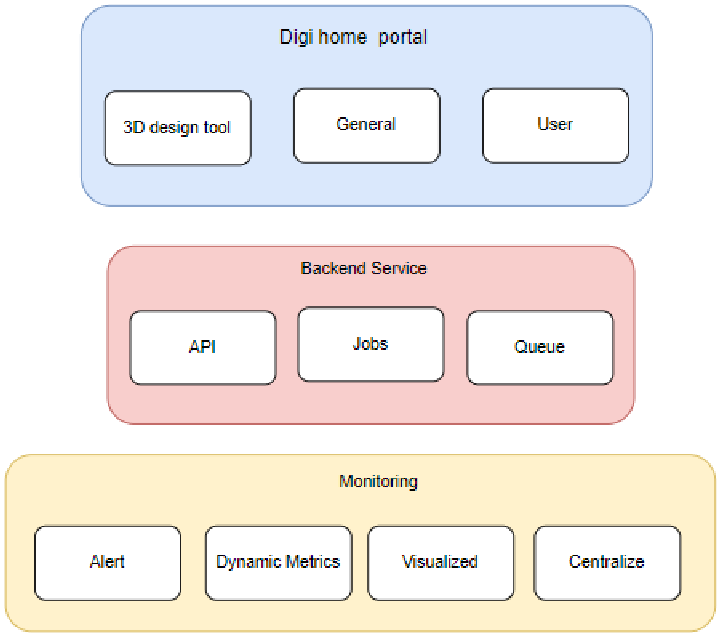
I. Instruction

The purpose of this document is to provide a comprehensive overview of the system architecture for Digi Home project. This document is intended for technical and non-technical stakeholders who are involved in the development, deployment, and maintenance of the system.

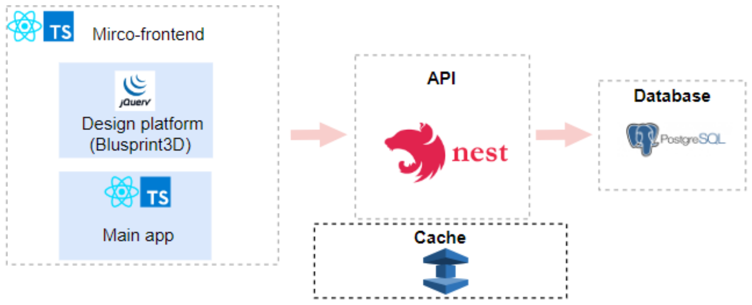
The system is designed to be highly scalable, reliable, and secure, with the ability to handle large volumes of data and traffic.

This document outlines the various components of the system architecture, including the software, hardware and infrastructure, network topology, data storage, and security measures. It also describes the interaction between these components, and how they work together to provide the functionality and performance required by the system

II. Project overview



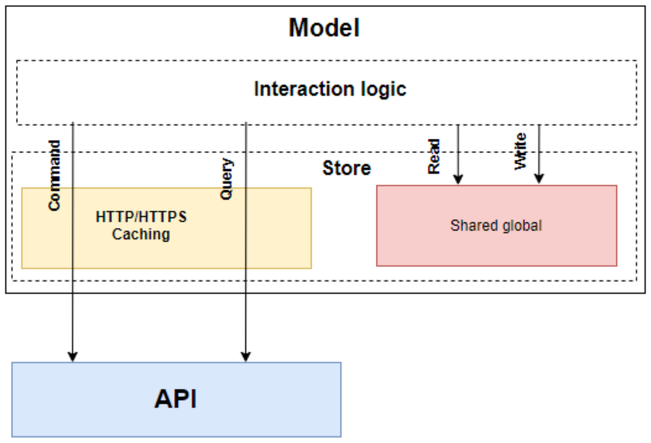
III. Software architecture



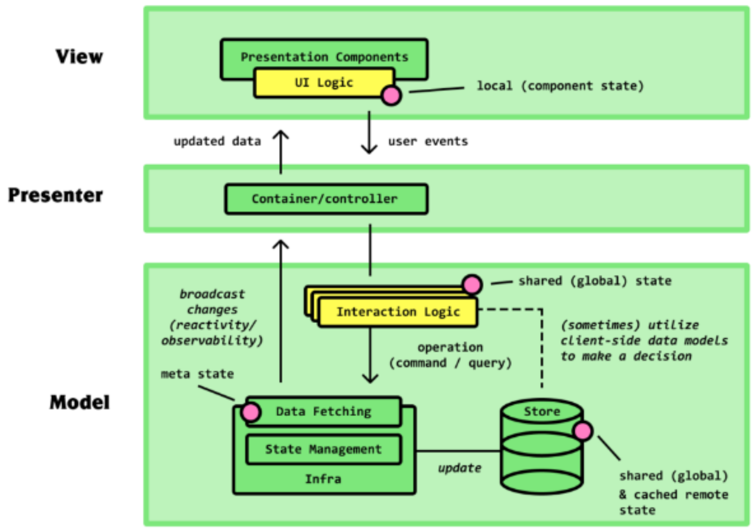
1. Client side

- Context:
    - We have an exist source code : **blueprint** which built in jquery
    - Need to build a SEO, friendly UI-UX web with many business goals and scope : React-app/Nextjs
- Solution: Morden web with micro-frontend architecture having many advantages:
    - Independent development and deployment:
        - Allow two team/member can work independently on their respective components includes develop, test and deploy.
        - Speed up development cycles and reduces the risk of conflict
    - Technology deversity
        - Enable the usage of different frontend technologies, frameworks and library within the same application to utilize the strengths to adapt business goals
    - Scalability: easier to roll back changes to specific parts of the application in case of issuesHigh-traffic components can be scaled up without affecting the performance of other components, ensuring optimal resource utilization
    - Team efficiency: Smaller, focused teams can be more productive and agile
    - Isiolation of failures: If an issue arises in one micro frontend, it is less likely to impact the entire application.
    - Modularity and reusebility:
      Micro frontends encourage a modular approach to frontend development, making it easier to create reusable components
    - Testing and QA:
      Testing becomes more focused and manageable with micro frontends, as teams can concentrate on specific areas of functionality
    - Improve UI and Performance:
      By breaking down the frontend into smaller parts, we can optimize the loading and rendering of the application. Users might experience faster load times, as they only load the parts they need, reducing the initial load size
    - Versioning and rollback:
      Easier to roll back changes to specific parts of the application in case of issues

- Architecture:
    - Main-app: React app with typescript
        - Latest stable version of all library
        - Clean architecture: SOLID, clean code.
        - Client side state management and server side statement management with CQS Principle:



- Modern MVP:



- Summary: With these architecture our app become :
  - Modularity and maintainability
  - Flexibility and Adaptability
  - Business focus
  - Testability
  - Performance optimization
  - Readability and understandability

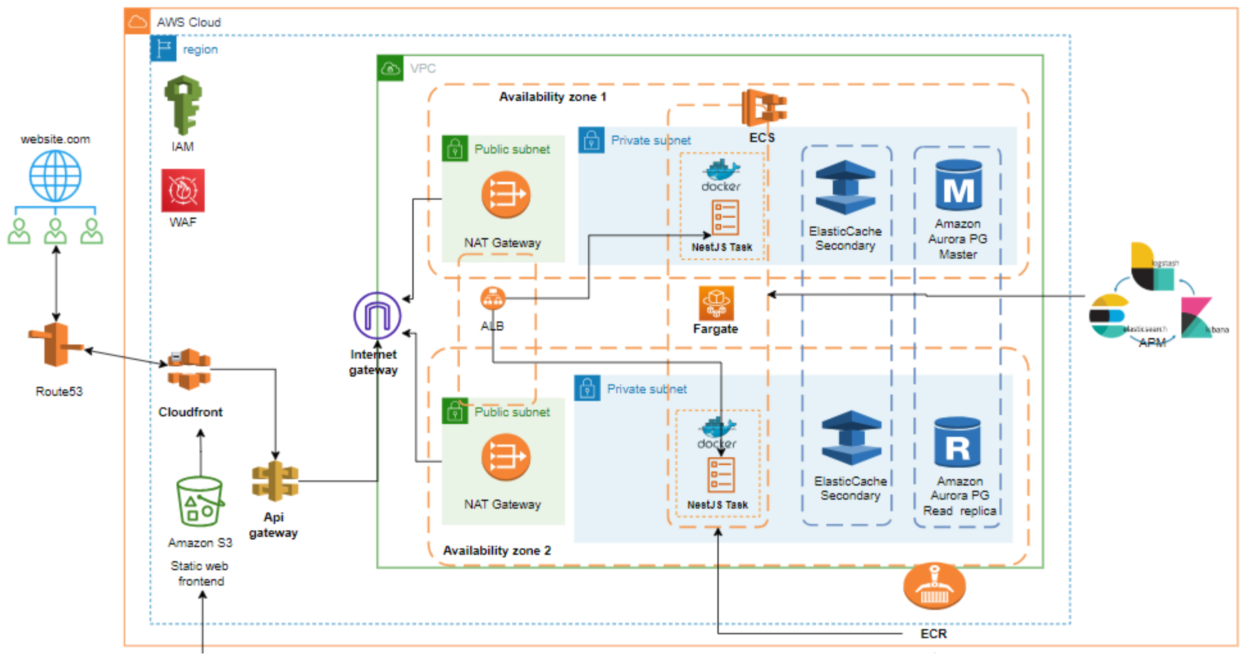
2. Server side
   - Nestjs framework:
     - Powerful and popular framework for building scalable and maintainable server-side applications using TypeScript.
     - It is built on top of Node.js and leverages various modern features and design patterns. Some of the advanced features and advantages of NestJS in powerful and popular framework for building scalable and maintainable

server-side applications using TypeScript. It is built on top of Node.js and leverages various modern features and design patterns.

- Postgres DB: Advanced open-source relational database management system (RDBMS) known for its robustness, extensibility, and rich feature set.
- Redis Cache: in-memory data for optimize speed, versatility, and rich feature.

3. Cloud: AWS
   - Comprehensive and advanced cloud computing platform that offers a wide range of services and features
   -

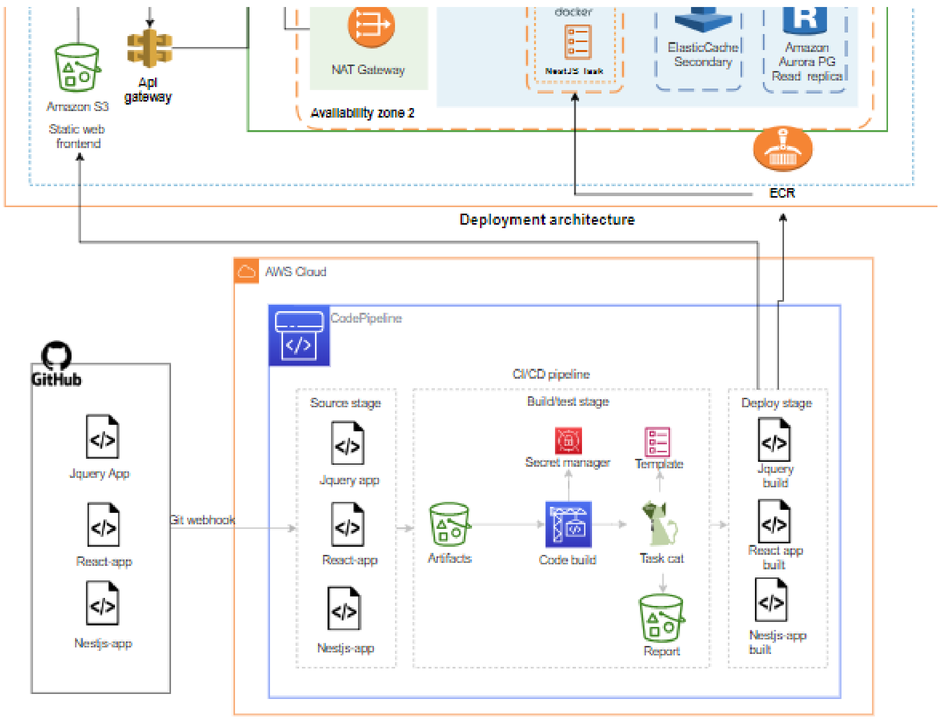
IV. Infrastructure architecture



- This architecture approach adapt the **Twelve-Factor** methodology which is a set of best practices for building modern cloud-native applications that are scalable, maintainable, and portable
  Reference: https://aws.amazon.com/vi/blogs/compute/applying-the-twelve-factor-app-methodology-to-serverless-applications/
- Using AWS Fargate service:
  - Scalable, cost-effective, and easy-to-use solution for running containerized applications in a serverless environment
  - Simplifies container management, reduces operational overhead, and allows developers to focus on building and deploying applications with confidence
  - Allow enterprise can forcus on business without spend much time and effort on infrastructor.
- Mutil AZ: Base on business and application context , we can set up this or not to reduces cost, effort

V. Deployment architecture

- Using cloudformation:
  - Define and provision infrastructure as code (Iac).
  - Allow manage AWS resources in a safe, predictable and automated.
  - Full CI/CD : init resource, build, test,deploy, report



VI. Data management

1. Data stored in a secure, scalable, and cost-effective manner. AWS S3 and AWS RDS will be used for data storage.
2. The system ensure data integrity, availability, and confidentiality. Data backups should be taken regularly to ensure availability and recovery.
3. The data processing are performed using appropriate services, such as Fargate, to ensure scalability and efficiency.
4. Access to data are controlled and authenticate using AWS IAM. Appropriate access policies will be defined to restrict unauthorized access to data.
5. A monitoring and alert mechanism will be implemented to detect and respond to data-related issues. ELK would be good

VII. Performance

1. Scalability: Able to handle increasing levels of traffic and workload. This can be achieved through techniques such as horizontal scaling, load balancing, and auto-scaling.
2. Availability: be highly available, meaning that it is accessible and function at all times. This can be achieved through techniques such as redundancy, fault tolerance, and disaster recovery.
3. Latency: Have low latency, meaning that it responds quickly to user requests. This can be achieved through techniques such as caching, data partitioning, and optimizing network latency.

4. Efficiency: Be efficient, meaning that it uses resources (such as CPU, memory, and disk space) effectively. This can be achieved through techniques such as optimization, compression, and resource pooling.

VIII. Maintenance and support

1. Scalability: The system designed to allow for easy scaling, upgrades, and maintenance without disrupting service availability.
2. Monitoring: The system have robust monitoring ELK stack tool to track performance metrics, detect issues, and alert support teams in case of any failures.
3. Logging: The system have APM to log all transactions, including errors, for troubleshooting and audit purposes.
4. Backup and Disaster Recovery: The system should will implement robust backup and disaster recovery solution place to ensure the continuity of service in the event of a catastrophic failure.
5. Testing: Have testing plan consists of unit test, integration, penetration test and user acceptance test to validate its performance, security, and functional.
6. Support: Able to respond quickly to issues, provide Emely updates, and communicate effectively with stakeholders.
7. Documentation: The system will be implemented clear and comprehensive documentation that outlines its architecture, functional, and maintenance procedures