Project Ragnarok:
Post-Quantum Cybersecurity with Lightning data
Milestone 1 Presentation

Team Members: Joanna Zhang,

Gianni Bubb, Aidan Nelappana

Faculty Advisor and Client: Dr. Bhattacharyya

Goal Reintroduction

- **Goal: Extend Project Thor capabilities by adding new cryptographic algorithms, easier pulling of keys, and database security**
- Project Thor used the natural randomness in lightning data produce cryptographic keys. Project Ragnarok will improve on Project Thor by adding new features.
- Originally keys were generated directly on the website. For convenience, users will be able to instead request keys from the website via an API.
- Project Thor provided MD5 and AES. We want to provide cryptographic algorithms for a post-quantum age to further future proof Project Thor's original aspirations.

Task 1: Technical tools

- Python API frameworks
  - FastAPI, web2py, flask and py4web

Compare and Select Collaboration tools

- Github as the code repository
- Google drive for sharing files
  - Docs for writing collaboration
  - Slides for presentation collaboration
- Text message group chat, Discord server for general communication
- Google Calendar for scheduling meetings

Technical Challenges

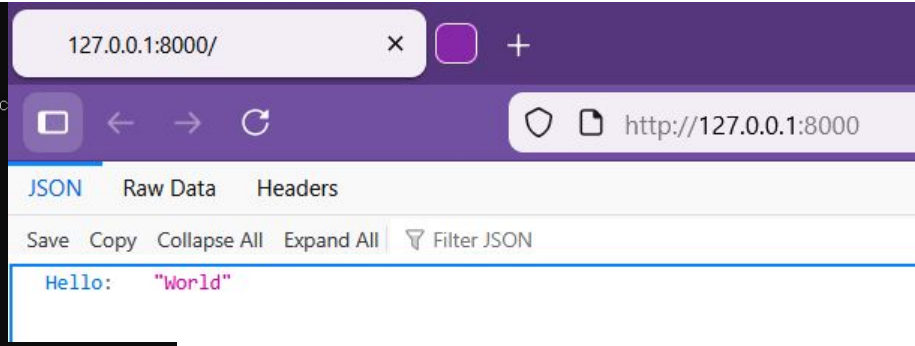
- Working with Project Thor’s original website
  - Key generation
  - Connecting the database

Task 2: Demos

- API Frameworks
  - Security Features
  - Documentation quality
  - Database connection
  - Performance

Demo 1: API Hello World via FastAPI

```
(env) D:¥Sakarain Ult Folder¥stuff¥school¥senior design¥framework tests¥fastest¥src>fastapi dev main.py

 FastAPI   Starting development server 

           Searching for package file structure from directories with __init__.py files
           Importing from D:¥Sakarain Ult Folder¥stuff¥school¥senior design¥framework tests¥fastTest¥src

  module   main.py

    code   Importing the FastAPI app object from the module with the following code:

           from main import app

     app   Using import string: main:app

  server   Server started at http://127.0.0.1:8000
  server   Documentation at http://127.0.0.1:8000/docs

     tip   Running in development mode, for production use: fastapi run

           Logs:

  ·[37;100m INFO ·[0m  Will watch for changes in these directories: ['D:¥¥Sakarain Ult Folder¥¥stuff¥¥school¥¥senior
       design¥¥framework tests¥¥fastTest¥¥src']
  ·[37;100m INFO ·[0m  Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
  ·[37;100m INFO ·[0m  Started reloader process [23368] using WatchFiles
  ·[37;100m INFO ·[0m  Started server process [12200]
  ·[37;100m INFO ·[0m  Waiting for application startup.
  ·[37;100m INFO ·[0m  Application startup complete.
  ·[37;100m INFO ·[0m  127.0.0.1:53318 - "GET /items/5?q=somequery HTTP/1.1" 200
  ·[37;100m INFO ·[0m  127.0.0.1:53319 - "GET /favicon.ico HTTP/1.1" 404
  ·[37;100m INFO ·[0m  127.0.0.1:53321 - "GET /items/5?q=somequery HTTP/1.1" 200
  ·[37;100m INFO ·[0m  127.0.0.1:53322 - "GET /favicon.ico HTTP/1.1" 404
  ·[37;100m INFO ·[0m  127.0.0.1:53345 - "GET / HTTP/1.1" 200
  ·[37;100m INFO ·[0m  127.0.0.1:53346 - "GET /favicon.ico HTTP/1.1" 404
  ·[37;100m INFO ·[0m  127.0.0.1:53347 - "GET / HTTP/1.1" 200
  ·[37;100m INFO ·[0m  127.0.0.1:53347 - "GET /favicon.ico HTTP/1.1" 404
  ·[37;100m INFO ·[0m  127.0.0.1:53349 - "GET / HTTP/1.1" 200
  ·[37;100m INFO ·[0m  127.0.0.1:53352 - "GET /docs HTTP/1.1" 200
  ·[37;100m INFO ·[0m  127.0.0.1:53352 - "GET /openapi.json HTTP/1.1" 200
  ·[37;100m INFO ·[0m  127.0.0.1:53367 - "GET / HTTP/1.1" 200
  ·[37;100m INFO ·[0m  127.0.0.1:53368 - "GET / HTTP/1.1" 200
  ·[37;100m INFO ·[0m  127.0.0.1:53368 - "GET / HTTP/1.1" 200
  ·[37;100m INFO ·[0m  127.0.0.1:53368 - "GET / HTTP/1.1" 200
```



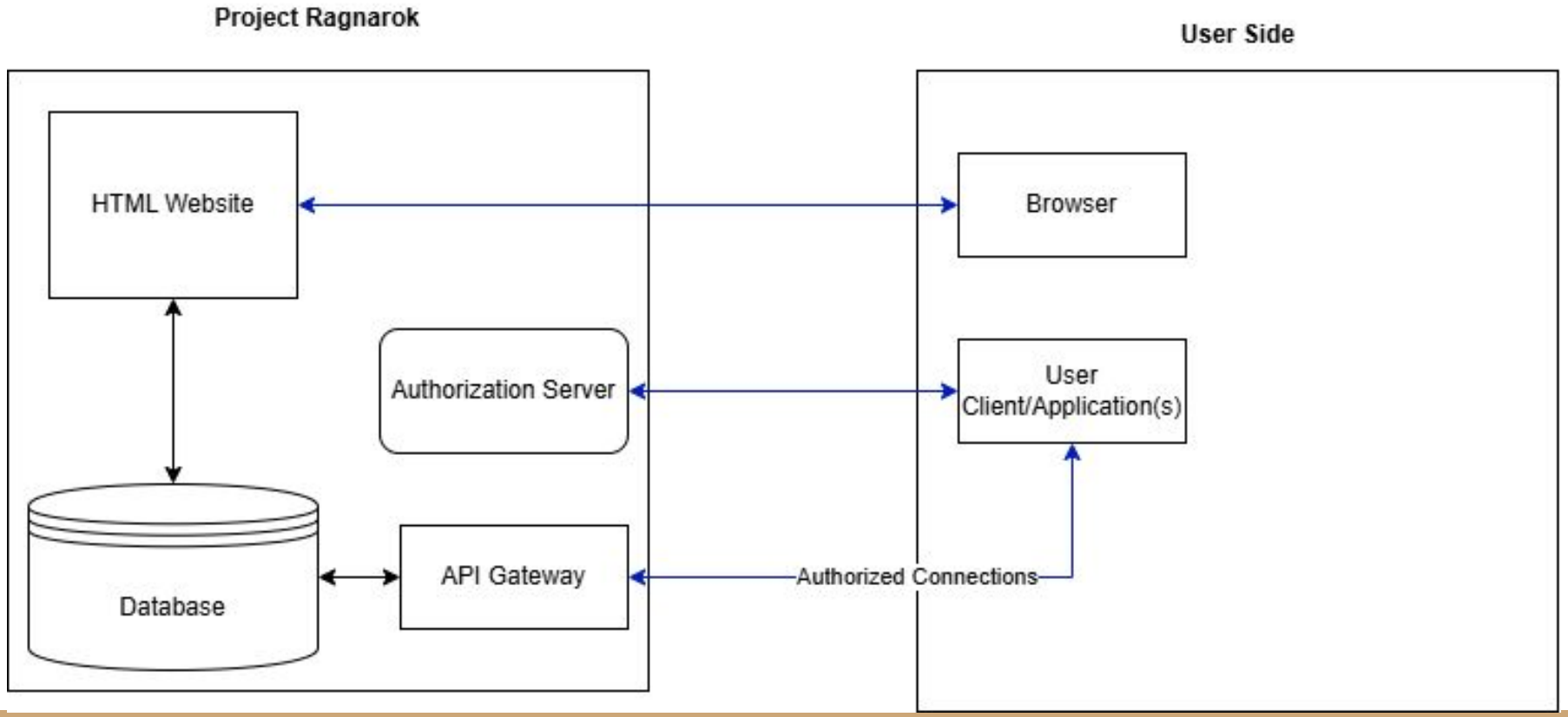
Task 3: Requirements

- Specifies functional and non-functional requirements
- Aligns team, advisor, and client expectations
- Guides design, implementation, and testing
- Prevents scope creep and misunderstandings
- Serves as the foundation of the project

Task 4: Design

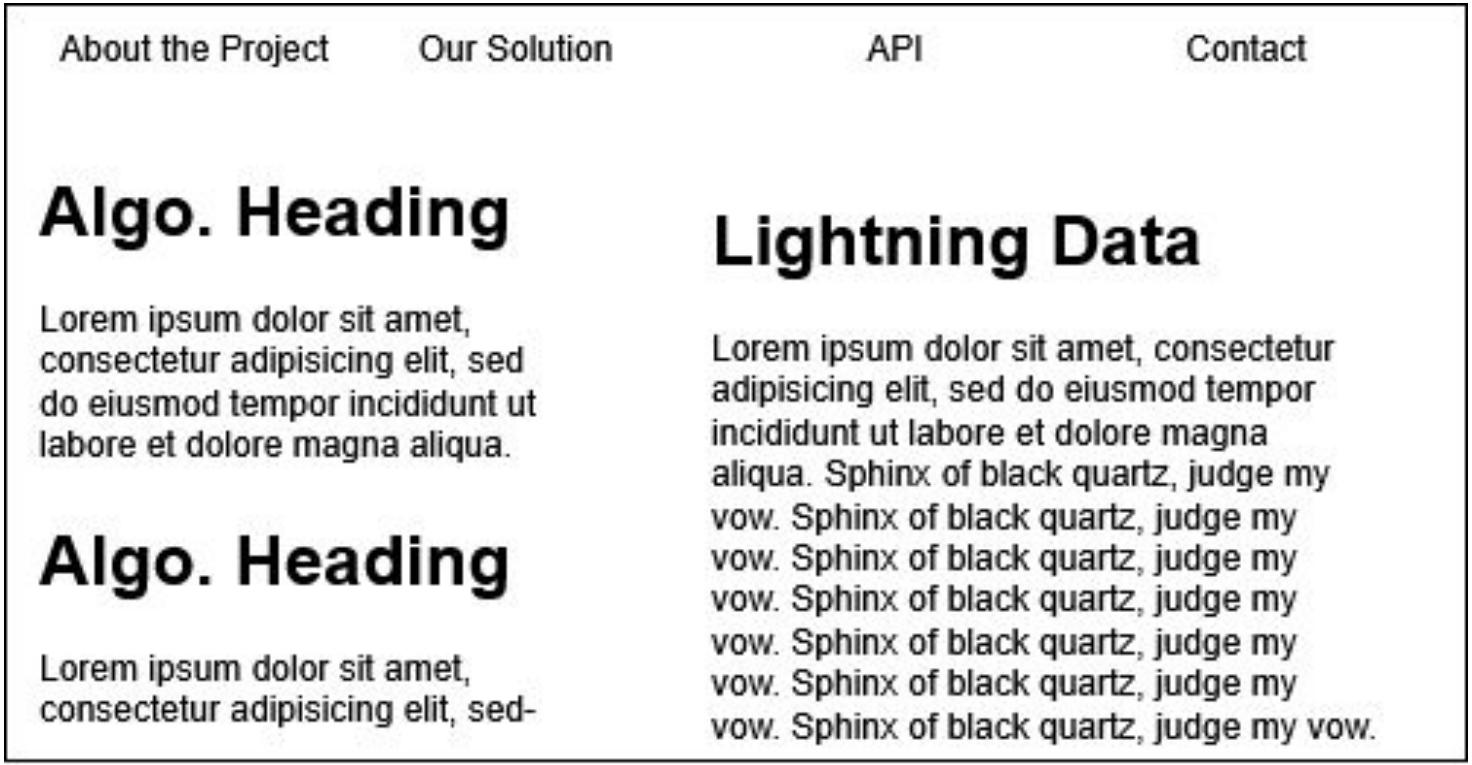
- Describes how the system will be built
- Defines system architecture and components
- Explains data flow, APIs, and database structure
- Details security mechanisms and technology choices
- Serves as a blueprint for implementation

Design: System Architecture



Website Layout: Landing Page

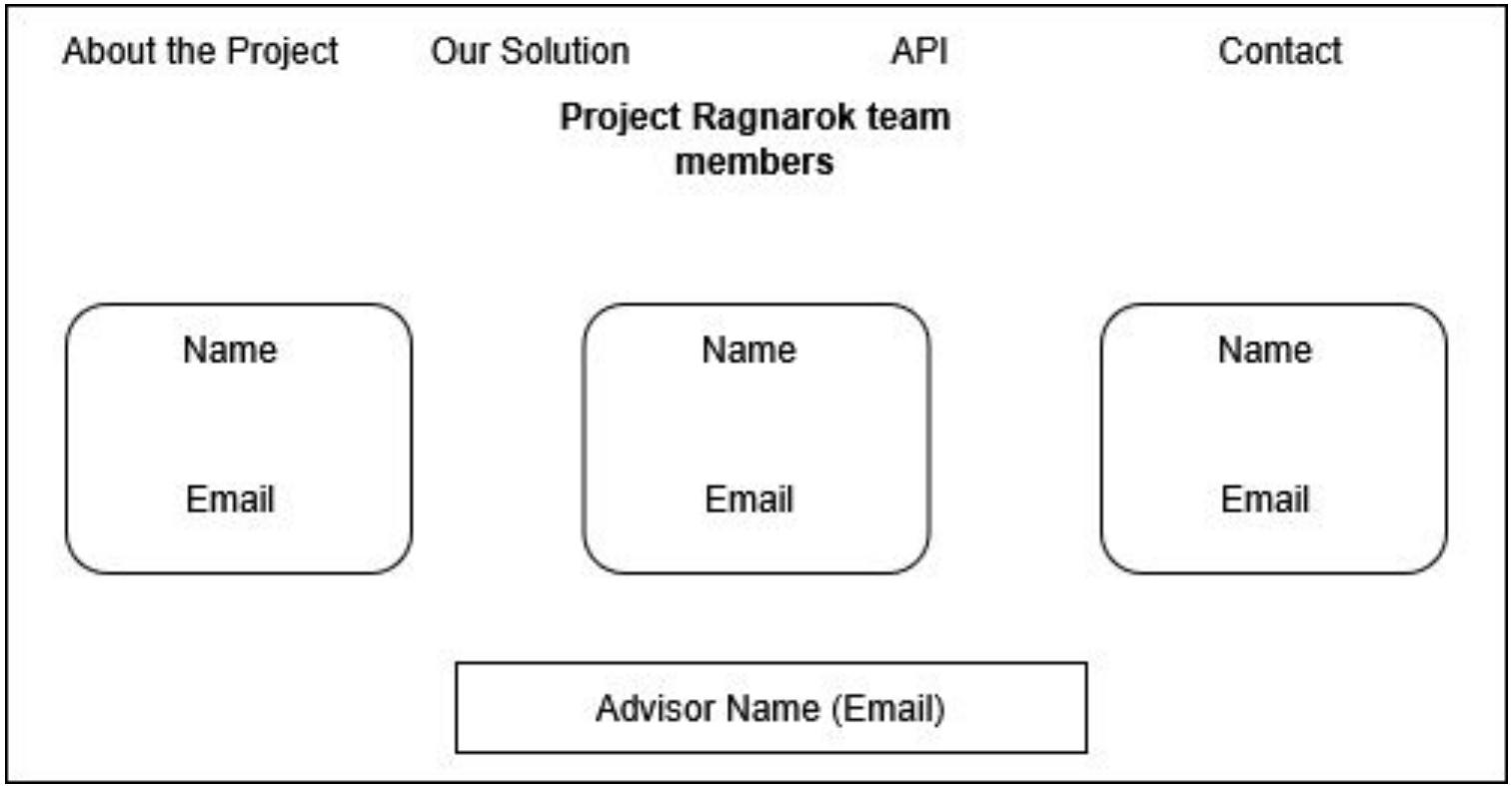
Website Layout: Solution Page



Website Layout: API sign in

Website Layout: API token generation

Website Layout: Contact Page



Task 5: Test

- Defines how system requirements will be verified
- Specifies test cases, inputs, and expected outputs
- Ensures functionality, security, and performance validation
- Identifies success criteria for each feature
- Supports debugging and issue tracking
- Confirms the system meets project requirements

Milestone 1 Task Matrix

Task	Task Description	Completion %	Joanna	Gianni	Aidan	To Do / Issues
1	Investigate tools	90%	10%	40%	40%	Look into Project Thor
2	Hello World demos	30%	0%	15%	15%	Adjust as the project is refined
3	Requirements Document	100%	33%	33%	33%	Adjust as the project is refined
4	Design Document	100%	33%	33%	33%	Adjust as the project is refined
5	Test Plan	100%	50%	50%	0%	Adjust as the project is refined
6	Implement, test & demo feature/module	30%	10%	10%	10%	Adjust as the project is refined

Milestone 2 Tasks Matrix

Task	Aidan	Gianni	Joanna
Implement basic Python API	Assist with backend integration	Lead API endpoint development and routing	Document API structure and update design/test docs
Implement basic Post-Quantum (PQ) algorithms	Assist integration with entropy module	Lead implementation of selected PQ algorithms (NIST-aligned)	Research standards and document security rationale
Database security remediation & fixes	Document remediation steps and update security design	Assist with backend security integration	Implement security controls and hardening

Thank you for your attention.