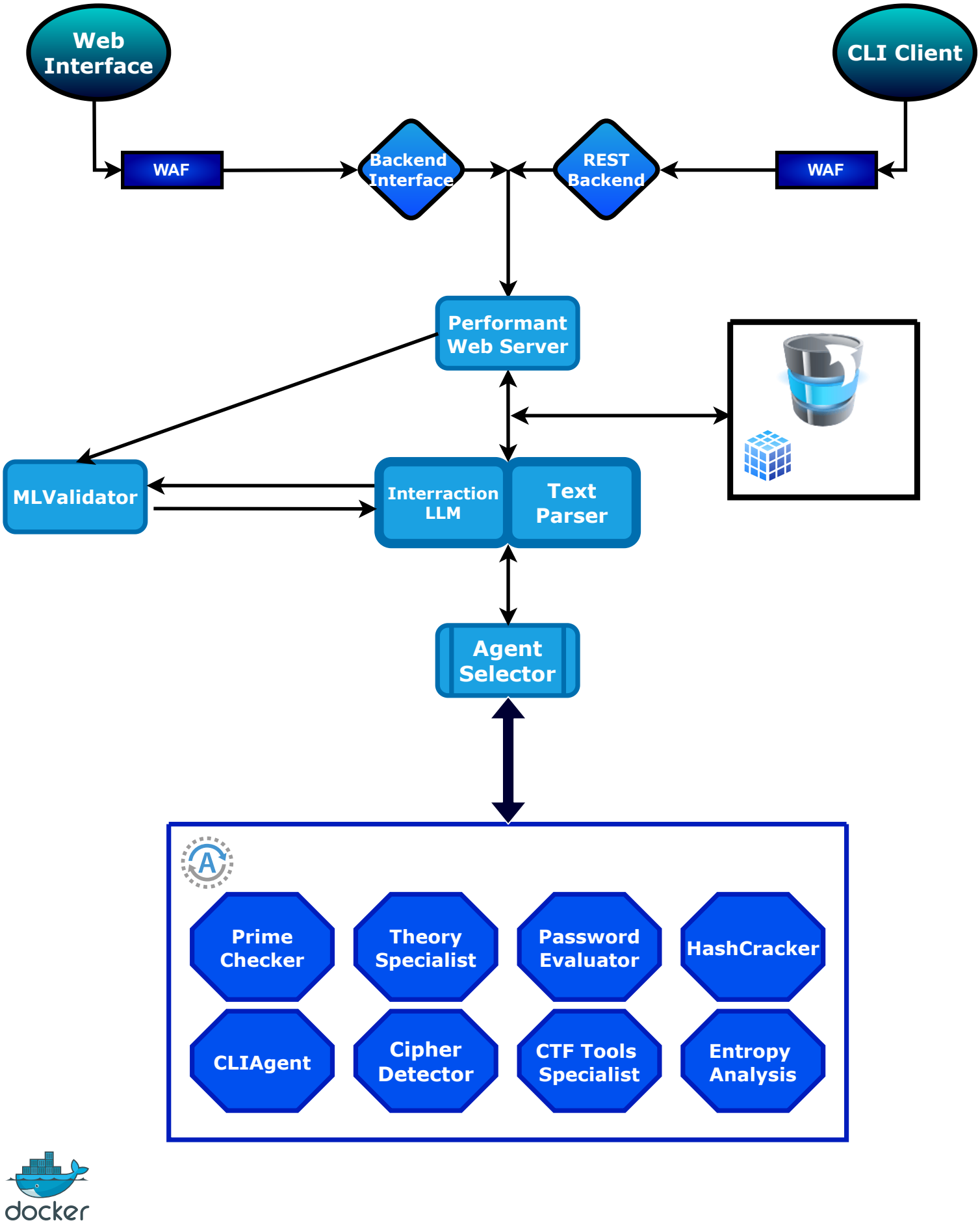
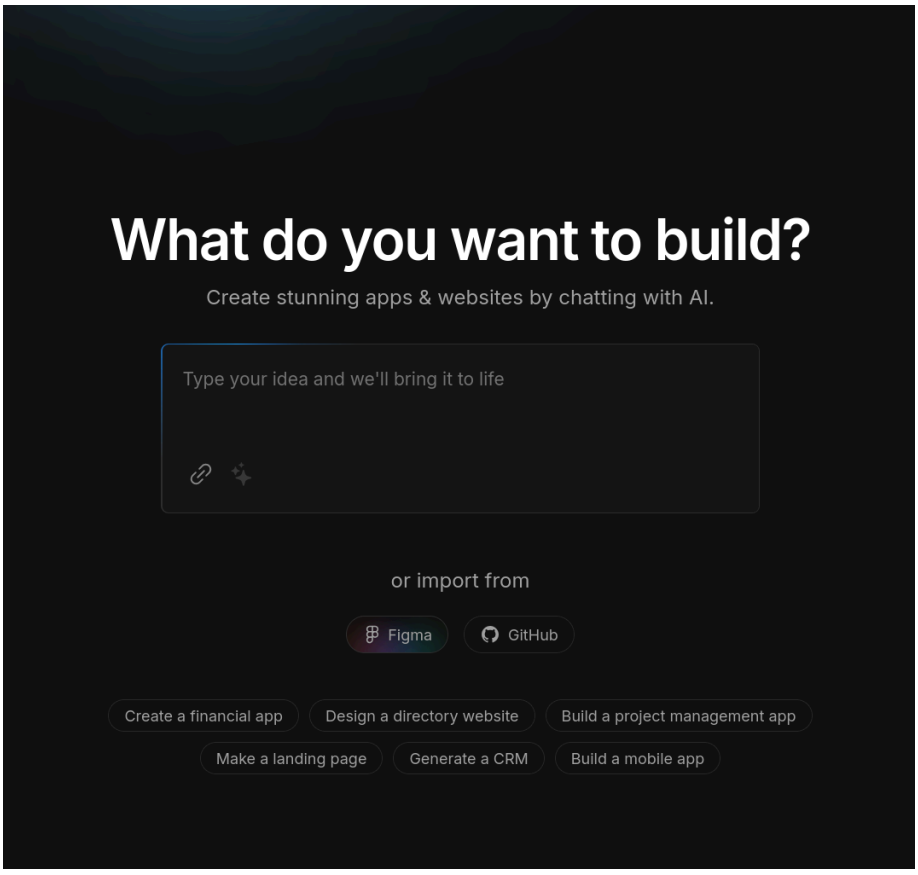

Web Interface
CLI Client
WAF
Backend Interface
REST Backend
WAF
Performant Web Server
MLValidator
Interraction LLM
Text Parser
Agent Selector
Prime Checker
Theory Specialist
Password Evaluator
HashCracker
CLIAgent
Cipher Detector
CTF Tools Specialist
Entropy Analysis
docker

Web Interface

u

Functionalities

Secure Authentication Using JWT, OAuth..
Interraction with the Chatbot
History Management
User Settings
Custom System Prompts
Command Copy/Paste
MarkDown Format
Browser Independence
Possible Client-Side Mechanisms(Dcode.fr)
React Framework?



CLI Client

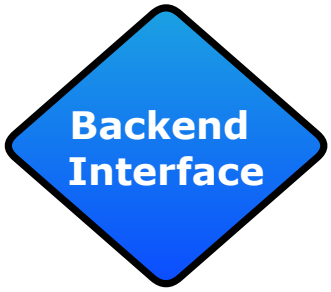
Functionalities

Npx-based CLI tool
Authentication via Web App => Redirection
GeminiCLI-Like behavoir
Per-Session Token Count
Per-Session History
No File Access required
Conversational Flow

Functionalities

Login based on the account from the Web Application
History Match with the Web Interface
Communication with the Database
After Closing, automatic LogOut
Extended functionalities Eliminated (User Settings, etc)
Endpoints:

- **POST /auth/login => login via credentials**
- **POST /auth/token => get access token**
- **POST /auth/logout => logout from application**
- **GET /conversations => get conversations ID's**
- **GET /conversation/ID => get content of a specific conversation**
- **POST /conversation/ID => continue a conversation/start a new conversation**
-



Functionalities

Login based on OAuth / New Account / Credentials / Forgot Password
Hashed Passwords management
History Management
REST architecture
Communication with the Database
Extended functionalities integrated

Endpoints:

- **POST /auth/login => login via credentials**
- **POST /auth/token => get access token**
- **POST /auth/logout => logout from application**
- **GET /conversations => get conversations ID's**
- **POST /conversation/ID => continue a conversation/ start a new conversation**
- **GET /whoami => get all informations about the user**
- **POST /me => modify existing account**
- **POST /delete_account => delete the existing account**
- **POST /recover_paswd => recover password endpoint based on email**

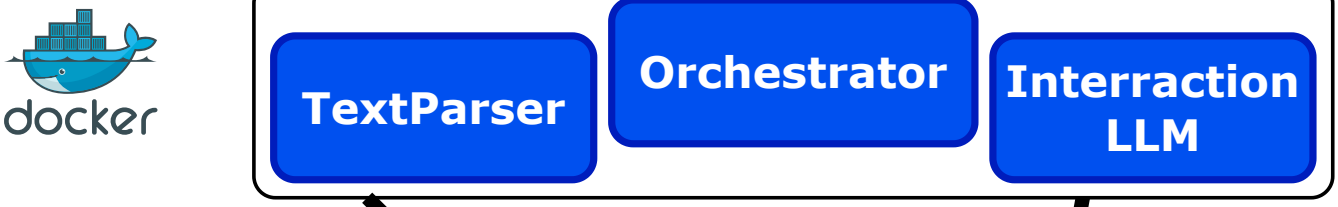
Functionalities

- Enforces HTTPS with TLS 1.2/1.3 and auto-renewing certificates.
- Routes traffic to the correct backend or API endpoint.
- Detects and blocks common web attacks via headers and rules.
- Mitigates DoS/DDoS using request rate limiting.
- Blocks open ports via strict access control and firewall rules.
- Restricts backend access, allowing only through the
- frontend/API gateway.
- Caches static and dynamic content for faster delivery.
- Helps prevent SQL injection when combined with
- secure backend practices.

Database

Functionalities

- **Stores Users with their Data**
- **Stores Users passwords in Hashed form**
- **Stores Per-User Conversations**
- **Can store In Encrypted manner the conversations content**
- **Has mechanisms of health check**
- **PostgresSQL**



It's role is to be able to transform in a normalized form the input text for the agent selector to be able to forward the request in the required format

It's purpose is to be able to talk with the user and to aggregate all the requests and the responses from the agent selecctor for the smooth interraction
It has to be able to answer with "Could not answer" or similar ways

Functionalities

- Talk to the user (via CLI or Web)
- Interpret natural language
- Maintain context (e.g., "How about 103?")
- Generate structured tasks
- Orchestrator should talk with the ML Validator as well for confidentiality assurance in the Agent Choice

Interaction Loop

User:

Is 101 prime?

Orchestrator:

Parsed: {"Agent": "Prime", "args": ["101"]}
→ Sent to Agent Selector → Prime Checker → Result: "Yes"

Orchestrator:

Yes, 101 is a prime number.

User:

What about 104?

Orchestrator Context:

(Still talking about primes) →

→ {"intent": "Prime", "args": ["104"]}

→ "No, 104 is not a prime."

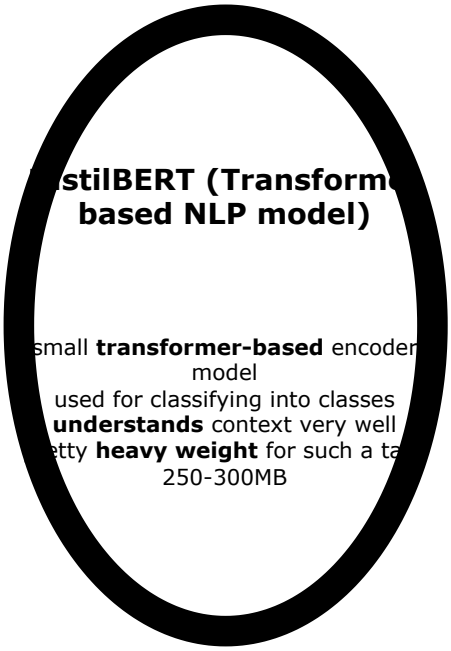
MLValidator

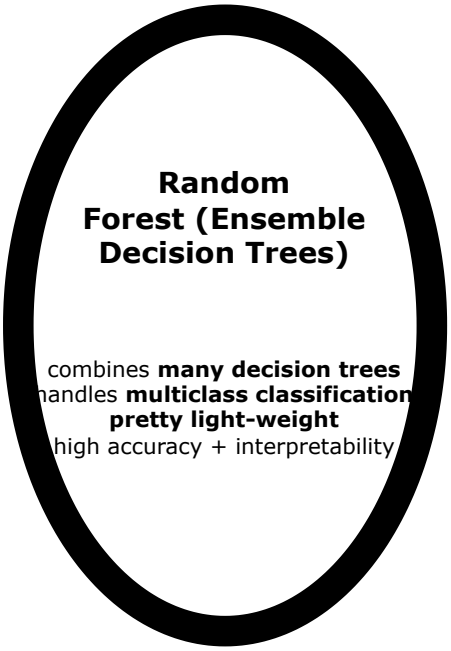
Purpose

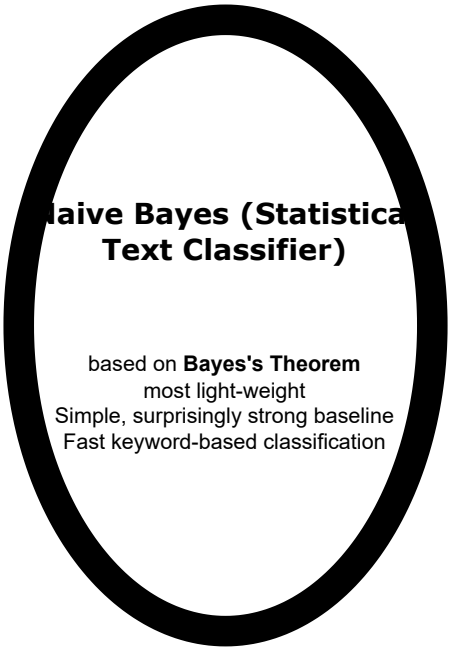
Prevent Hallucinations / Incorrect Choice of the Orchestrator when it comes to choosing the Agent which to process the request

Provide an Extra Layer of decision-making robustness by using a Specific Machine Learning Technique for Classification.

Choices







Functionality

Based on the input format received form the orchestrator, translate the input to the necessary Agent from the Agent Pool and sends back to the orchestrator the message in the same format

Workflow Example

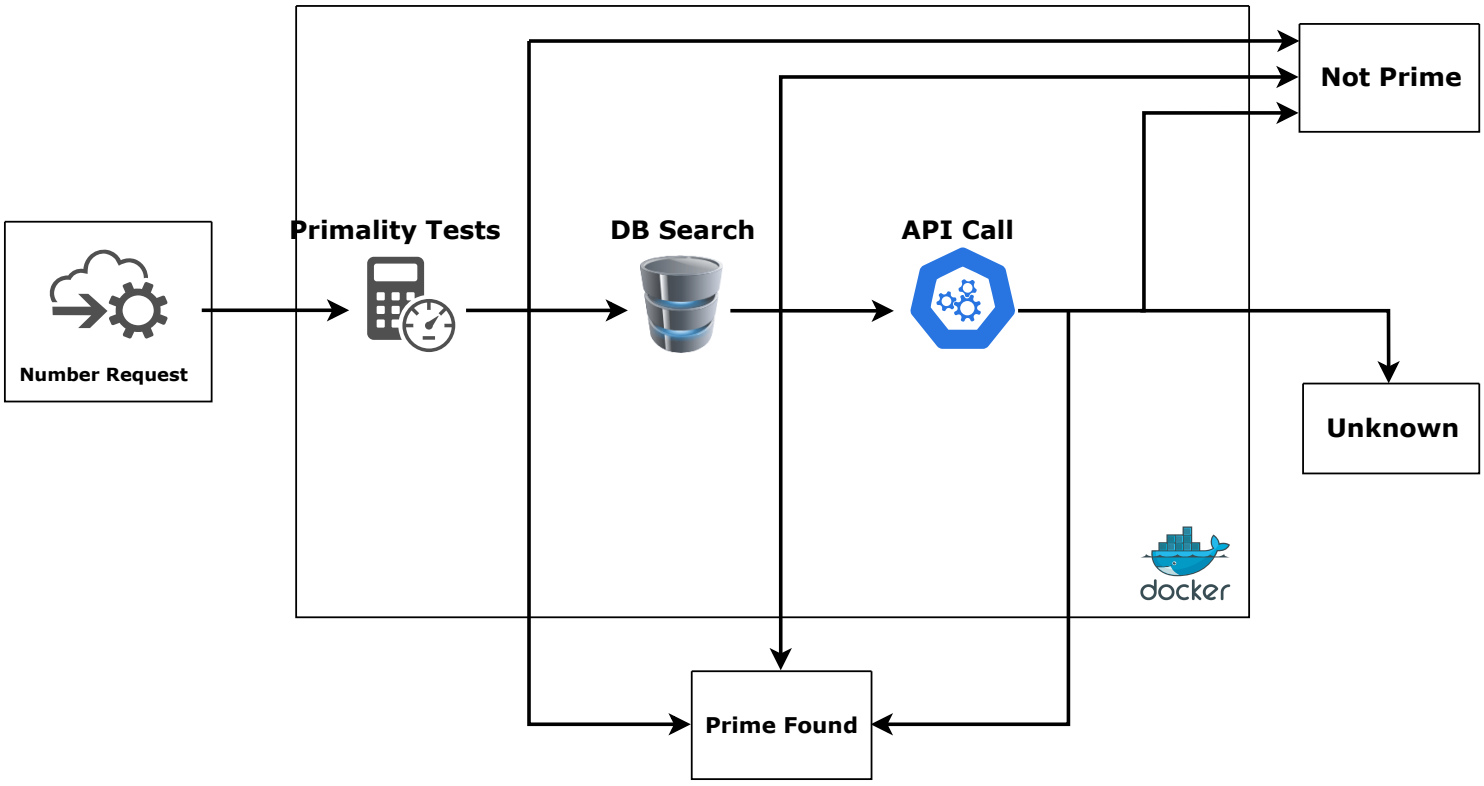
Orchestrator:
<"Prime", "12343102341">
Agent Selector:
Additional Sanitization Mechanisms
Chooses Prime Agent based on <ID, Name> Association list
Sends the Prime Agent the Request
Receives the Response from the Prime Agent
Sends the Response back to the Orchestrator

<"Prime", "12343102341"> => <"12343102341"> => <"No">
=> <"12343102341", "No">

Prime Checker

Functionality

Checks if a number is prime
Uses Mathematical tests
Uses Fast Search with an extended Database with prime large numbers
Checks FactorDB API for verification if the
Miller Rabin Test + Database is not found as being a prime number

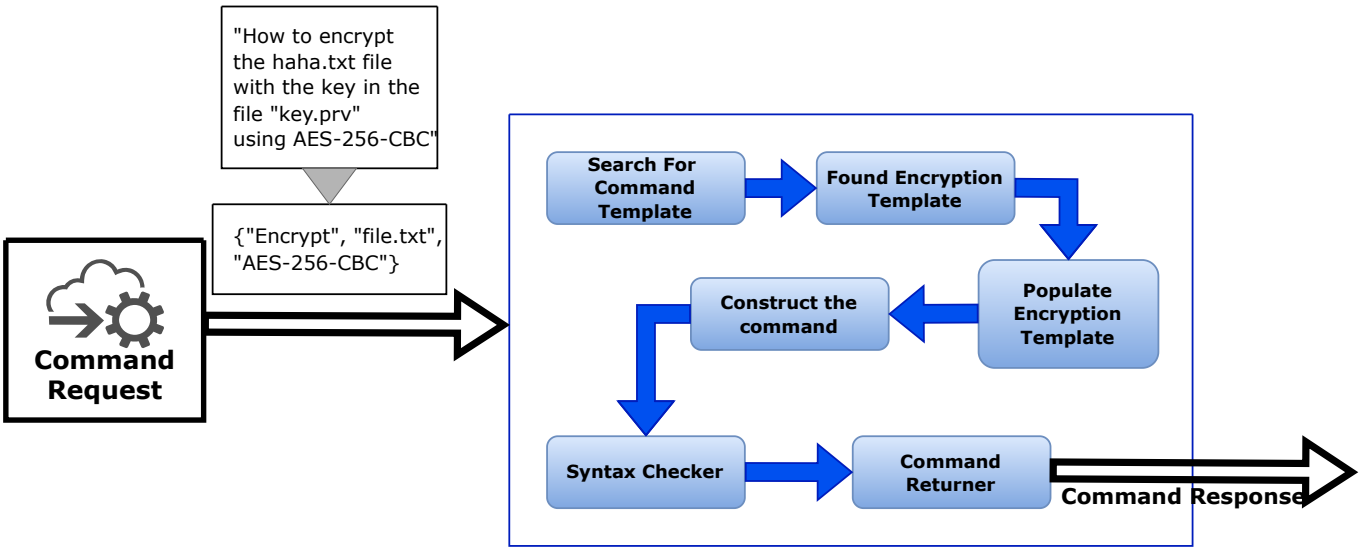


CLIAgent

Functionality

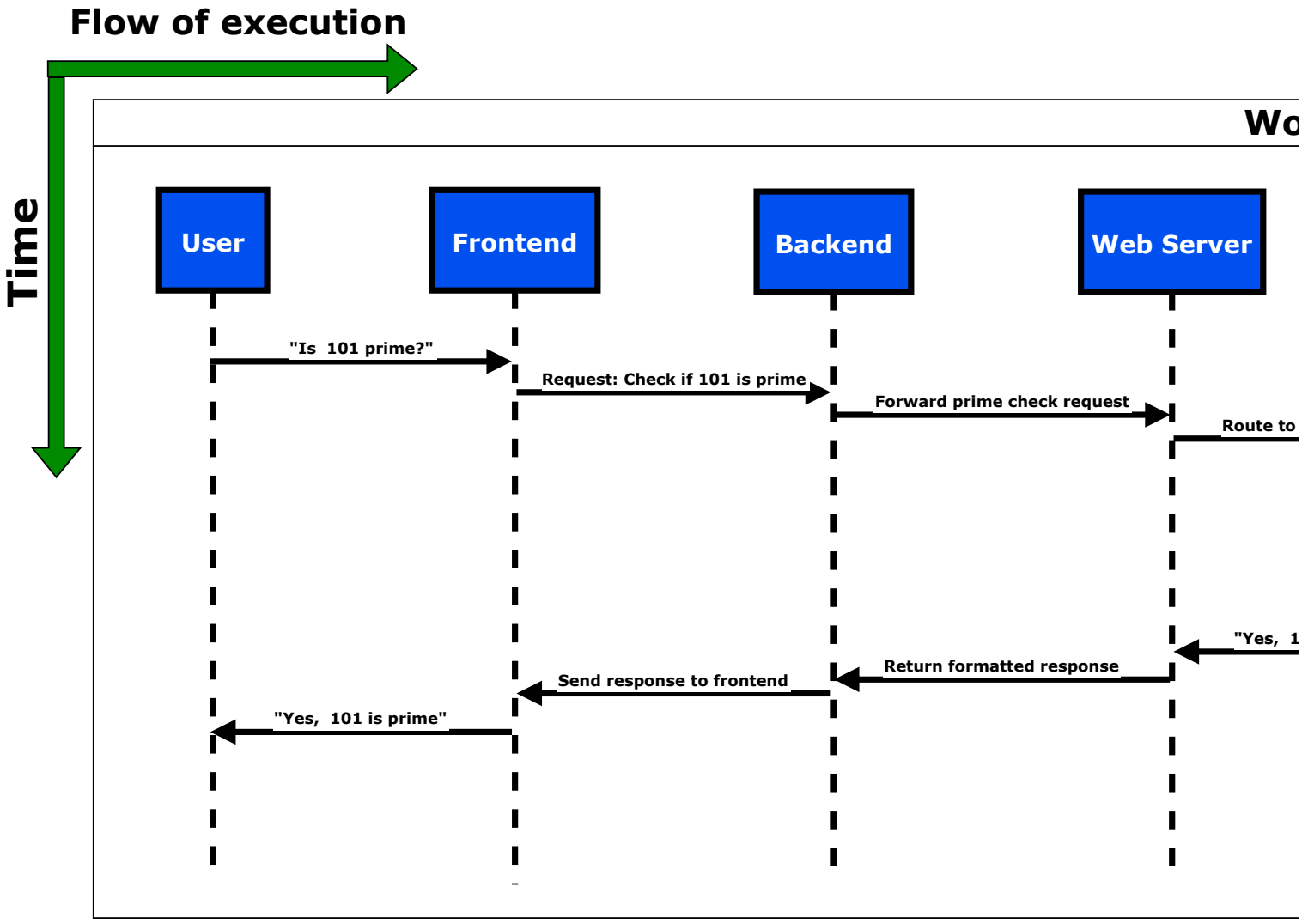
Provide roboust responses using different Utilitaries Commands like OpenSSL, for traditional cryptography and not only, also PQC

The command responses has to be ready to copy in the exact terminal format
It has to provide some API REST interface for being able to provide very precise command, example: encrypt, decrypt, random_generate, certificate_create, etc.



Workflow Example

Create User => Authenticate => "Is the number 101 prime?" => Fronte
Prime Checker ("101") => Agent Selector ("Yes") => Orchestrator ("Yes")



end => Backend => Web Server => Orchestrator => Agent Selector =>
) => Web Server ("Yes, the number 101 is prime") => Backend => Frontend