**Initial Report of Group InfluxUI-PG02**

**Project of ATSYS**

**No-Code Solution for InfluxDB**

**LeStartUP**

Zilin Song - a1833935

Jen-Hao Liu - a1893169

Dang Quy Duong - a1893592

Baojing Li - a1894836

Shih-Han Lin - a1900715

Feinan Guo - a1903270

Xiaoqing Zhao - a1904344

Hao Jiang - a1907177

Ziqi Zhang - a1909438

**Project Vision**

Our vision is based on three dimensions. First, creating the 'No-Code Solution for InfluxDB' project and delivering value to ATSYS's customers and users. We aim to build a user-friendly system that enhances the user experience for ATSYS's customers.

Secondly, our team's positivity will drive the project to success. Before the kickoff, we thoroughly analyzed and discussed the project requirements, initial architecture, and technology stacks. We are determined to achieve project success and ensure customer satisfaction.

Finally, as a student development team, we aim to overcome the challenges of software development and agile project management. Key success factors include transparent stakeholder engagement, clear communication, efficient teamwork management, and adaptability to any changes in our SCRUM process. We strive to successfully deliver the product while enhancing the agile development abilities of each team member.

**Customer Q&A**

* Questions and the confirmation to the customer representative and the product owner.

Q: Should we integrate Grafana at the beginning of the project, or should we first complete the non-extension requirements and integrate Grafana later?

A: You should integrate Grafana from the beginning, and the exact timing is based on the group sprint plan.

Q: Are there any specific accessibility requirements for the drag-and-drop interface?

A: As a user, I just need a drag and drop interface and how I want to make it right.

Q: How should the interface handle invalid selections or combinations of data sources? How should error handling be implemented, especially in cases where the generated Flux query is invalid or returns no results?

A: It would be better including error messages and ensure a user-friendly interface.

Q: When users log in to the page, should there be authorization steps that check which data sources users have permission to view/query?

A: Yes, there should be authorization checks.

Q: When a user selects a different data source, should the application dynamically update the available measurements and fields?

A: Yes, it should dynamically update.

**Users**

* **User story 1: Drag-and-Drop Interface for Selecting Data Sources**

|  |  |
| --- | --- |
| **Goal** | As a user, I want to use a drag-and-drop interface to select the bucket, measurements, and fields from InfluxDB, so that I can easily choose the data I need without writing code. |
| **Actors** | User |
| **Pre-conditions** | The user is logged into the no-code interface. |
| **Main Flow** | * The user logs into the no-code interface. * The user is presented with a list of available buckets, measurements, and fields. * The user selects the desired data sources by dragging and dropping items into the query builder area. * The interface automatically prepares these selections for the next steps in the data query process. |
| **Post-conditions** | * The selected buckets, measurements, and fields are ready for filtering and querying. * The user successfully prepares the data sources without writing any code. |
| **Acceptance Criteria** | * The interface must allow the user to drag and drop items to select buckets, measurements, and fields. * The selected items must be accurately reflected in the query builder. |

* **User story 2: Filter application via drag-and-drop**

|  |  |
| --- | --- |
| **Goal** | As a user, I want to apply filters to my selected data using a drag-and-drop interface, so that I can refine the data retrieval process without having to write complex queries. |
| **Actors** | User |
| **Pre-conditions** | * The user has selected the bucket, measurements, and fields using the drag-and-drop interface.​ * The data sources are ready for filtering.​ |
| **Main Flow** | * The user accesses the filter options in the no-code interface. * The user drags and drops filter criteria onto the selected data fields. * The user sets parameters for the filters (e.g., date range, value thresholds). * The interface prepares the filtered query based on the user’s inputs. |
| **Post-conditions** | * The user’s filters are applied to the selected data, refining the query.​ * The system is ready to execute the query with the applied filters.​ |
| **Acceptance Criteria** | * The interface must allow the user to drag and drop filters onto the selected data fields.​ * The applied filters should accurately reflect the user's input.​ * The interface should provide clear feedback on how the filters are affecting the data selection. |

* **User story 3: Automatic Query Generation and Execution​**

|  |  |
| --- | --- |
| **Goal** | As a user, I want the interface to automatically generate and execute the Flux query based on my drag-and-drop selections, so that I can retrieve the data I need without writing any code. |
| **Actors** | User |
| **Pre-conditions** | The user has selected the relevant data sources and applied filters via the drag-and-drop interface. |
| **Main Flow** | * The user completes the data selection and filtering process using drag-and-drop. * The interface automatically generates the corresponding Flux query in the background. * The user initiates the query execution by clicking a 'Run Query' button. * The system processes the query and retrieves the data. |
| **Post-conditions** | * The user retrieves the data without manually writing or modifying any code. * The system displays the results for further analysis or visualization. |
| **Acceptance Criteria** | * The system must accurately generate the Flux query based on the user’s drag-and-drop inputs. * The query execution must return the correct data based on the applied filters and selections. * The interface should provide clear feedback on the query execution status and display the results promptly. |

* **User Definition**

|  |  |
| --- | --- |
| **User Definition** | |
| Business Analyst | * Background: e-commerce company * Responsibility: monitoring key performance indicators (KPIs) and generating reports. * Objective: * Quickly access relevant data from InfluxDB to track website traffic, sales trends, and customer behaviour. * Apply filter and visualise the data in Grafana dashboards in a user-friendly version to generate reports for management team. |
| Operation manager | * Background: manufacturing company * Responsibility: mitigating resourcing and identify any issue or anomaly in case of shortage or overuse * Objective: * Easily access and analyse production data stored in InfluxDB, with direct comparison to industry level. * segment the data into the query builder by production line, filter by equipment type, and analyse metrics over specific time periods. |

* **Assumption based on user stories**
* Non-programmer: The user is assumed to lack programming expertise, particularly in writing Flux queries for InfluxDB.
* Data explorer: The user needs to query and analyse time-series data stored in InfluxDB.
* Visualiser: The user aims to create visual representations of the queried data, potentially using Grafana.
* Interface navigator: The user interacts with a drag-and-drop interface to select data sources, apply filters, and initiate queries.
* **Characteristics of the User**
* *Non-Programmer*: The user lacks the skills to write or understand complex queries, particularly in InfluxDB’s internal language, Flux.
* *Data-Driven*: Despite not being a programmer, the user is focused on obtaining, filtering, and analysing data from InfluxDB for purposes such as monitoring.
* *Visual Thinker*: The user prefers graphical interfaces that allow drag-and-drop operations to simplify the data selection and filtering process.
* *Efficiency-Seeking*: The user values tools that automate the generation of queries and seamless integration between InfluxDB and Granfana, enabling them to retrieve and manipulate data quickly without needing to write any code

**Software Architecture**

Our software architecture outlines a no-code solution for interacting with InfluxDB, providing users with an intuitive interface for data querying and visualization. The system consists of four main components: User Interface, Frontend, Backend, and InfluxDB, with optional integration to Grafana for advanced dashboarding

* **User Authentication:**
* Users access the application through the Frontend.
* Credentials are securely transmitted to the Backend.
* The Backend authenticates with InfluxDB.
* Upon successful authentication, a session token is provided to the Frontend
* **Query Construction:**
* Users construct queries using a drag-and-drop interface in the Frontend.
* **Query Construction:**
* The Frontend sends the constructed Flux query to the Backend.
* The Backend validates and executes the query against InfluxDB.
* InfluxDB processes the query and returns time series data.
* **Data Visualization:**
* The Backend processes the raw data from InfluxDB.
* Processed data is sent to the Frontend for visualization.
* The Frontend renders the data in user-friendly charts and graphs.
* **Grafana Integration (Extension):**
* Users can interact with Grafana dashboards for more advanced visualizations.
* Changes made in the Frontend are sent to the Backend.
* The Backend updates Grafana using its API.
* Grafana confirms the update, which is then relayed to the user.

This architecture enables users to leverage InfluxDB's powerful time series capabilities without needing to write complex queries, while also providing the option for advanced visualizations through Grafana integration. The separation of Frontend and Backend concerns allows for scalability and easier maintenance of the system.

* **UML diagram:**

|  |
| --- |
|  |

**Tech Stack and Standards**

* **Front-end stacks:**

For our InfluxDB no-code solution, we have selected a front-end tech stack that balances performance, maintainability, and developer productivity. We have chosen technologies with strong community support and rich ecosystems to ensure access to resources and integration. This stack aims to support an intuitive drag-and-drop interface for query building while providing robust state management for complex application logic. We have incorporated industry-standard tools for styling and testing to blend innovation with proven practices. The table below details our specific technology choices and the reasoning behind each.

| **Categories** | **Proposal** | **Proposal explanation** |
| --- | --- | --- |
| Language | TypeScript | Type Safety  Easier Error Handling  Good Developer Experience |
| UI Library | ReactJS | Best Community Support  Huge Ecosystem  Good Developer Experience |
| Framework | NextJS | Comprehensive Features  Good community Support  Good Developer Experience |
| Styling | TailwindCSS + shadcn/ui | Large Community  Robustness  Good Developer Experience |
| State Management (Optional) | Zustand or *XState (for complex state)* | Large Community  Comprehensive Features  Good Documentation  Good Developer Experience |
| Drag & Drop Flowchart Library | React Flow | Best Community Support  Comprehensive Features & APIs  Good Documentation  Works Well with Other Technologies (Framework, State management, Styling)  Good Developer Experience |
| Code Editor Component for Flux Code | @monaco-editor/react | Easy Integration with NextJS  Good Community Support  Good Developer Experience |
| Testing | * Vitest (unit tests) * React Testing Library (component tests) * Playwright (end-to-end tests) | Industry Standard Testing Suite for React Apps  Good Community Support & Documentation  Good Developer Experience |
| Linting + Formatting | ESLint + Prettier | Industry Standard  Good Community Support & Documentation  Good Developer Experience |

* **Back-end stacks:**

The microservices architecture was selected because it supports scalability and flexibility during project development. Therefore, the tech stacks at the server side are proposed based on the microservice concepts and the analyses of client requirements.

|  |  |  |
| --- | --- | --- |
| **Categories** | **Proposal** | **Proposal explanation** |
| Language | Python (v3.10) | Flexibility |
| Framework | Django | Support MVC |
| API documentation | Swagger | Centralize API docs |
| Primary Database | InfluxDB OSS (v2.7) | Required by the client |

* **Containerization and Orchestration stacks:**

|  |  |  |
| --- | --- | --- |
| **Categories** | **Proposal** | **Proposal explanation** |
| Containerization | Docker | Flexibility and reliability |

* **Monitoring and Logging stacks**

|  |  |  |
| --- | --- | --- |
| **Categories** | **Proposal** | **Proposal explanation** |
| Monitoring | Grafana (v9.5.3) | Flexibility and reliability |

* **CI/CD stacks**

|  |  |  |
| --- | --- | --- |
| **Categories** | **Proposal** | **Proposal explanation** |
| Platform | GitHub & GitHub Action | GitHub ecosystem  Flexibility and reliability |

* **Coding standards**
* Front-end standards
  + Linting: ESlint + NextJS default style config
  + Format: Prettier
  + Commit Message: Conventional Commit
  + Branch Name
* Back-end standards
  + Naming convention
  + Response bodies
  + API versioning
  + Response status codes
  + API documents
* **Version control standards**
* Based on GitHub, CI/CD ... concept
* **Communication platform**
* Microsoft Teams: for flexibility
* **Documentation platform**
* University of Adelaide: for security reasons. Explain as this is secured.

**Group Meetings and Team Member Roles**

* **Group meeting roles:**

|  |  |  |
| --- | --- | --- |
| Types | Purpose | Time |
| Daily meeting | * Daily communication * Quick updates information | * On WhatsApp treat as 15 min stand-up. * Face-to-face in the Uni If necessary. |
| Weekly meeting | * To synchronize each working stage and situation * Weekly retrospective * Work together | * 17:00 -18:00 on Monday * 16:00-17:00 on Wednesday * 17:00-18:00 on Friday |
| Sprint meeting | * Customer requirements updated from Product owner Sanchi Verma. * Sprint work present. * Sprint retrospective. | * 17:00-17:30 on Wednesday biweekly |

* **Communication with Product Owner (Sanchi Verma):**
* Teams chat for real time communication
* Outlook calendar for official meetings
* **Team Member:**
* Scrum Mater:

|  |  |
| --- | --- |
| Sprint | Name |
| 1st | Shih-Han Lin (Peter) |
| 2nd | Baojing Li (Elias) |
| 3rd | Ziqi Zhang (Kelvin) |
| 4th | Jen-Hao Liu |
| 5th | Feinan Guo |

* Development Team:
* Front end:

|  |  |
| --- | --- |
| Role | Name |
| Leader | Hao Jiang (Johnny) |
| Developer | Ziqi Zhang (Kelvin) |
| Developer | Baojing Li (Elias) |
| Developer | Xiaoqing Zhao |
| Developer | Zilin Song (Harry) |

* Back end:

|  |  |
| --- | --- |
| Role | Name |
| Leader | Dang Quy Duong (Tom) |
| Developer | Jen-Hao Liu |
| Developer | Feinan Guo |
| Developer | Shih-han Lin (Peter) |

* Cross function team:

Dynamically allocate human resources based on each sprint.

**Snapshot**

**Snapshot Week 05 of Group PG02**

**Project of ATSYS**

**No-Code Solution for InfluxDB**

**LeStartUP**

Zilin Song - a1833935

Jen-Hao Liu - a1893169

Dang Quy Duong - a1893592

Baojing Li - a1894836

Shih-Han Lin - a1900715

Feinan Guo - a1903270

Xiaoqing Zhao - a1904344

Hao Jiang - a1907177

Ziqi Zhang - a1909438

**Product Backlog and Task Board**

* The product backlog (continuous changes)

|  |  |  |
| --- | --- | --- |
| **Category** | **Features** | **note** |
| Front end | A single-page application using NextJS |  |
| Front end | A login interface for user authentication at the same level of InfluxDB |  |
| Front end | An intuitive drag-and-drop query builder for the Flux language |  |
| Front end | Real-time Flux query generation |  |
| Front end | Option to view the generated Flux query code |  |
| Front end | Data visualization through native implemented charts and graphs |  |
| Front end | Optional integration with Grafana dashboards and panels |  |
|  |  |  |
|  |  |  |
| Back end | User authentication against InfluxDB |  |
| Back end | Query validation and processing |  |
| Back end | Data retrieval with InfluxDB |  |
| Back end | Data processing for visualization |  |
| Back end | Optional integration with Grafana dashboards and panels |  |
|  |  |  |
|  |  |  |
| InfluxDB | Time-series database that powers the authentication of the web app and serves as the data source |  |
|  |  |  |
|  |  |  |
| Grafana | Optional integration for saving and editing data queries and visualization dashboards. |  |
|  |  |  |
|  |  |  |

* The task board

|  |  |  |
| --- | --- | --- |
| **Items** | **Tasks** | **Status** |
| 1 | Software architecture | Version 1 |
| 2 | Infrastructure for dev/staging/prod stages | On-going |
| 3 | Infrastructure for local InfluxDB, Grafana, Server and FE | On-going |
| 4 | From user story 1, form features + API of the app | On-going |
| 5 | Keep forming features and APIs | On-going |
| 6 | Develop BE using Django and APIs defined (Specifically query schema of IDB, query IDB, get/update/create Grafana panel | On-going |
|  |  |  |

* The GitHub repository we are working on

|  |
| --- |
|  |
|  |

**Sprint Backlog and User Stories**

* The screenshot of the sprint backlog

| **Items** | **Tasks** | **Status** |
| --- | --- | --- |
| 1 | Software architecture | Version 1 |
| 2 | Infrastructure for dev/staging/prod stages | Done |
| 3 | Infrastructure for local InfluxDB, Grafana, Server and FE | Done |
| 4 | From user story 1, form features + API of the app | On-going |
| 5 | Keep forming features and APIs | On-going |
| 6 | Develop BE using Django and APIs defined (Specifically query schema of IDB, query IDB, get/update/create Grafana panel | On-going |
|  |  |  |

|  |
| --- |
|  |
|  |
|  |
| 一張含有 文字, 螢幕擷取畫面, 軟體, 電腦圖示 的圖片  自動產生的描述 |
| 一張含有 文字, 螢幕擷取畫面, 設計 的圖片  自動產生的描述 |

* The user stories in the Sprint.
  + **User story 1: Drag-and-Drop Interface for Selecting Data Sources**

|  |  |
| --- | --- |
| **Goal** | As a user, I want to use a drag-and-drop interface to select the bucket, measurements, and fields from InfluxDB, so that I can easily choose the data I need without writing code. |
| **Actors** | User |
| **Pre-conditions** | The user is logged into the no-code interface. |
| **Main Flow** | * The user logs into the no-code interface. * The user is presented with a list of available buckets, measurements, and fields. * The user selects the desired data sources by dragging and dropping items into the query builder area. * The interface automatically prepares these selections for the next steps in the data query process. |
| **Post-conditions** | * The selected buckets, measurements, and fields are ready for filtering and querying. * The user successfully prepares the data sources without writing any code. |
| **Acceptance Criteria** | * The interface must allow the user to drag and drop items to select buckets, measurements, and fields. * The selected items must be accurately reflected in the query builder. |

|  |
| --- |
|  |

**Definition of Done**

* Our current "definition of done":
* Unit test passed.
* End-to-end test passed.
* Code reviewed in process: individual and group reviewed.
* Non-functional requirements met. (If there is one)

**Completed items**

* In the 1st Sprint, our team had completed:
* The team rules including hierarchy of periodic meetings and communication platform.
* The team roles: Division of work including Scrum Master, front-end sub team and back-end sub team.
* The initial tech stack.
* Group development rules.
* Define the tasks of user story 1 on GitHub.
* The initial report which will be delivered to the client (Submission).

**Meeting Minutes (in GitHub and Teams Files)**

|  |
| --- |
| The 1st group meeting / The kick-off meeting  15:00-16:00, 2nd Aug 2024 |
| The kickoff Sprint meeting / Q&A session with PO Sanchi Verma  15:00-16:00, 9th Aug 2024 |
| The 1st Sprint meeting / Q&A session with PO Sanchi Verma  17:00-17:30, 14th Aug 2024 |
| Meeting type: The 2nd group meeting  16:00-17:00, 15th Aug 2024 |
| Meeting type: The 3rd group meeting  15:00-18:00, 23rd Aug 2024 |

**Summary of Changes**

In the first sprint, our team focused on establishing team rules, allocating roles, and laying the foundation for the development environment in accordance with the client's requirements. We successfully set up the development environment, including the front-end and back-end frameworks. The team was organized into specialized roles to enhance productivity, and responsibilities were clearly defined. Initial user stories were broken down into tasks, and we began work on implementing the core functionalities. We initiated the development process by creating the basic structure of the user interface, which will allow users to log in to the application.

This sprint primarily involved setting up the technical infrastructure and aligning the team to ensure a smooth development process in subsequent sprints. We will continue to work on ensuring the integration with InfluxDB and Grafana for data visualization in future sprints.