Absolutely — let's walk through your **full system DFD flow step-by-step**, with both **explanations** and **PlantUML code** for each segment.

## **🔷 Step-by-Step DFD Flows with Capabilities**

### **✅ Step 1: User → GUI**

#### **🔍 Explanation:**

* The user starts the interaction by entering ICD input into the GUI.
* GUI is responsible for displaying the interface and collecting input.

#### **🌱 PlantUML:**

@startuml

left to right direction

actor User

rectangle "GUI" as GUI

User --> GUI : enter ICD input

note right of GUI

- Collect user input

- Display report

- Show errors

end note

@enduml

### **✅ Step 2: GUI → API**

#### **🔍 Explanation:**

* GUI sends the collected ICD JSON data to the API for processing.

#### **🌱 PlantUML:**

@startuml

left to right direction

rectangle "GUI" as GUI

rectangle "API" as API

GUI --> API : send ICD JSON

note right of API

- Route data

- Validate input

- Interface with DB and Backend

end note

@enduml

### **✅ Step 3: API → Validation → API**

#### **🔍 Explanation:**

* API sends data to the validation module.
* Validation checks field completeness and formats, then returns result.

#### **🌱 PlantUML:**

@startuml

left to right direction

rectangle "API" as API

rectangle "Validation" as Validation

API --> Validation : validate\_fields()

Validation --> API : valid?

note right of Validation

- Check fields

- Enforce format rules

end note

@enduml

### **✅ Step 4: API → CppEngine → ResultHandler → API**

#### **🔍 Explanation:**

* After validation, the API calls CppEngine to run the ICD logic.
* Result is parsed and formatted by ResultHandler before returning to API.

#### **🌱 PlantUML:**

@startuml

left to right direction

rectangle "API" as API

rectangle "CppEngine" as Cpp

rectangle "ResultHandler" as RH

API --> Cpp : run\_cpp()

Cpp --> RH : parse result

RH --> API : formatted output

note right of Cpp

- Run compiled logic

- Generate raw output

end note

note right of RH

- Parse and structure results

end note

@enduml

### **✅ Step 5: API → DB**

#### **🔍 Explanation:**

* The API stores the validated and processed ICD entry into the database.

#### **🌱 PlantUML:**

@startuml

left to right direction

rectangle "API" as API

database "Database" as DB

API --> DB : save ICDEntry

note right of DB

- Store ICD records

- Allow retrieval

end note

@enduml

### **✅ Step 6: API → GUI → User**

#### **🔍 Explanation:**

* API sends back the final results to GUI.
* GUI displays the report to the user.

#### **🌱 PlantUML:**

@startuml

left to right direction

rectangle "API" as API

rectangle "GUI" as GUI

actor User

API --> GUI : return result

GUI --> User : show ICD report

@enduml

## **🧩 Summary: Full Flow Steps**

| **Step** | **Flow** | **Description** |
| --- | --- | --- |
| 1 | User → GUI | User inputs ICD data |
| 2 | GUI → API | GUI sends JSON to API |
| 3 | API → Validation → API | API validates fields |
| 4 | API → CppEngine → ResultHandler → API | Backend logic runs and result is formatted |
| 5 | API → DB | Entry is saved to database |
| 6 | API → GUI → User | Final result displayed to user |

Would you like all these combined into a single downloadable PDF or rendered as individual diagrams?

Great question — drawing a **Data Flow Diagram (DFD) for a full system with capabilities** means you want a visual that includes both:

1. **Core system components and data flows**
2. **Descriptions of what each component can do** (i.e., **capabilities**)

## **✅ Step-by-Step to Draw a DFD with Capabilities**

### **🔹 1. Identify Core Elements**

#### **💼 External Entities (users, systems):**

* User

#### **🌀 Processes (functional components):**

* GUI
* API
* Validation
* CppEngine
* ResultHandler

#### **🗃️ Data Stores:**

* Database

#### **🔄 Data Flows:**

* ICD input, validation request, analysis, report generation, storage, result display

### **🔹 2. List Capabilities per Component**

| **Component** | **Capabilities** |
| --- | --- |
| **GUI** | Collect user input, display results, show errors |
| **API** | Route data, validate input, process results, handle DB |
| **Validation** | Check field completeness and structure |
| **CppEngine** | Run compiled analysis logic |
| **ResultHandler** | Format results for display and storage |
| **Database** | Store and retrieve ICD entries |

### **🔹 3. Draw the DFD With Annotations**

To **include capabilities**, you can do this in **two ways**:

### **✅ A. Annotate Capabilities as Notes**

Here's a **PlantUML-based approach** with embedded notes:

@startuml

left to right direction

skinparam packageStyle rectangle

actor User

rectangle "GUI" as GUI

rectangle "API" as API

rectangle "Validation" as Validation

rectangle "CppEngine" as Cpp

rectangle "ResultHandler" as Formatter

database "Database" as DB

' Data flows

User --> GUI : enter ICD input

GUI --> API : send ICD JSON

API --> Validation : validate\_fields()

Validation --> API : valid?

API --> Cpp : run\_cpp()

Cpp --> Formatter : parse result

Formatter --> API : formatted output

API --> DB : save ICDEntry

API --> GUI : return result

GUI --> User : display report

' Capabilities (as notes)

note right of GUI

- Collect user input

- Display report

- Show errors

end note

note right of API

- Route data

- Validate input

- Process results

- Handle DB

end note

note right of Validation

- Field completeness

- Format rules

end note

note right of Cpp

- Run compiled logic

- Generate raw result

end note

note right of Formatter

- Format output

- Structure report

end note

note right of DB

- Store ICD data

- Retrieve past entries

end note

@enduml

### **✅ B. Separate Capability Diagram (Alternative)**

If you want a **cleaner DFD**, you could create **a second diagram or table** just showing:

* Components
* Descriptions
* Capabilities

This is often useful for large systems or documentation deliverables.

## **How This DFD with Capabilities Works**

### **🧱 Structure**

This diagram follows the DFD model:

* **External Entity**: User
* **Processes**: GUI, API, Validation, CppEngine, ResultHandler
* **Data Store**: Database

### 

| **Component** | **Capabilities** |
| --- | --- |
| **User** | Initiates and views ICD report |
| **GUI** | - Collects user input  - Displays reports  - Shows validation errors |
| **API** | - Routes data  - Validates input  - Sends to DB and Backend |
| **Validation** | - Checks fields  - Enforces formatting rules |
| **CppEngine** | - Runs compiled logic  - Produces raw output |
| **ResultHandler** | - Parses backend results  - Formats them |
| **Database** | - Stores ICD entries  - Retrieves saved data |