**MEETING NOTES**

**Okay, somehow I think we are not properly set up here in a way that people can understand a big picture. Now, since we are all working, ensure that everyone understands the flow while doing it. Add more doesn't mean that only I have to add this task. So make sure that if any specific confusion, create more details around it so that others can understand, okay, all that stuff. So to start with, okay, so the objective is to wrap the client. Let us go from the bottom up. So the client at the end of the day, the top view, the client comes. They say, I want to create a cork. Obviously, we have to go behind the scenes. We get the access token, create something, something, create the API. Okay, so again, so at this point, obviously, I have to add something. I don't know where to add it. So basically, obviously, what are the inputs? The payload, right, the request payload. I expect, who is the client coming? I want to ensure that we do some more authentication around that. So we have to do two authentications. So right now, we'll do a basic thing because of time crunch, right. Let us give a hard-coded value for each client. There's only one client now. So one client will give some hard-coded value. We will come to the details around it later, but let us go ahead quickly. Client comes. He'll give a payload. In the payload, he has something. Who is this client? Then immediately, he will turn around and call this API and give the data, right. And then also, this is very important, create and query cork. So potentially, he may even call cork API, or we will put this together. So we'll see based on the right way. What I'm trying to say is that, see, just to set the context, right, in general, right, why this goes hand-in-hand is that suppose you are doing a maintenance on a device, right. For example, you are fixing your laptop. I can't come and work on your laptop, right. So likewise, when a device is, what's happening is that we are resetting the devices, okay. Someone is not being. They are resetting devices, and they want to create a maintenance request. That is the gist of it because obviously, when you do a task, you create a JIRA, right. Likewise, it's a mandatory compliance also that if somebody is performing some activity on the wireless devices, they have to tell, I am doing activity, and they create a cork request, right. So the task here is that, okay, I'm doing an activity on, say, a device called ABCD, okay, but it doesn't disown my device. The other can also work, right. Alex can also work. So when Alex is working, right, we cannot go and work on the device, right. So that's why first they do a query to ensure that, hey, is the device in maintenance. If not, then they will do their activity, okay. Now the question is that should we hand off the client to first do the query by themselves, and then they do create? We are all the same team, so that's something we can do that. So I'm just telling you. That is the purpose of why they first do a query, and then they do a create. Okay. I'll pause. Any questions you have here?**

**Explanation:  
Objective**

**The goal is to handle client requests to query and create a "cork" object (essentially a maintenance request) for a device, ensuring proper authentication, validation, and seamless interaction between APIs.**

**Key Points in the Workflow**

1. **Client Request:**
   * **The client initiates a request (payload).**
   * **Payload includes basic information (e.g., client details, device identifier).**
2. **Authentication:**
   * **Perform two levels of authentication:**
     + **For now, use hard-coded values for client authentication (time-constrained solution).**
     + **In the future, implement a dynamic authentication mechanism.**
   * **Validate the client identity based on the payload.**
3. **API Interaction:**
   * **Access Token Generation:**
     + **Use the credentials to generate an access token behind the scenes.**
   * **Cork Management APIs:**
     + **Query Cork API: Check if the device is in maintenance.**
     + **Create Cork API: Create a maintenance request for the device if eligible.**
4. **Decision-Making:**
   * **Should the client query the device status before creating the request themselves?**
     + **If yes, provide API documentation for querying.**
     + **If no, automate the query within the system workflow and provide a single interaction point for the client.**

**Real-World Context**

* **Why Query First?**
  + **Maintenance on devices (like laptops or wireless equipment) must be pre-checked to ensure no conflicts with ongoing activities.**
  + **For example:**
    - **If Alex is working on Device ABCD, others cannot perform overlapping tasks. Hence, querying the device ensures it is not under maintenance before creating a new request.**
* **Compliance Requirements:**
  + **Creating a maintenance request is mandatory to:**
    - **Log the activity in a traceable system (similar to creating a JIRA).**
    - **Maintain a record for audits and compliance.**

**Proposed Workflow**

1. **Client Interaction:**
   * **Client submits a request payload with required details (e.g., device, activity type).**
2. **Query Device Status:**
   * **System checks if the device is under maintenance.**
   * **Returns status to the client (e.g., "Available" or "Under Maintenance").**
3. **Create Cork Request:**
   * **If the device is free, proceed to create the cork request automatically or prompt the client to confirm.**
4. **Feedback/Response:**
   * **System provides a response with the cork details (if created) or the maintenance status.**

**Next Steps**

* **For Immediate Implementation:**
  1. **Add a hard-coded authentication check for the single client.**
  2. **Implement basic payload validation.**
  3. **Integrate the Query Cork API to check device status.**
  4. **Provide a workflow where the client queries first, or the system handles it automatically based on requirements.**
  5. **Return meaningful responses to guide the client.**
* **For Future Enhancements:**
  1. **Replace hard-coded authentication with a robust mechanism.**
  2. **Add dynamic support for multiple clients.**
  3. **Develop a more detailed logging system to track activities for compliance.**

**Questions for the Team**

1. **Query vs. Create Integration:**
   * **Should we give the client control over the query step, or integrate it into the system workflow for simplicity?**
2. **API Authentication:**
   * **Should the system handle token management transparently, or require the client to supply tokens?**
3. **Response Expectations:**
   * **What level of detail should we return in responses (e.g., exact device status, cork request details)?**

**as a purpose, but whether we, how to intervene, we'll discuss later. But for now, let us focus on create query to different activities, okay? And then close also, correct also, close, conduct also, same thing. So that's the gist of it. So this is the main thing which has a user's, their own leverages, but, and other things are all how we properly scaffold our project so that it is more friendly, more readable. And that's the reason we put a swagger around it. We want to do unit testing, so that is a very critical piece of the puzzle. And then, yeah, so this is a request payload. Now I'm going to import unit on, so obviously I want to do a proper definition, definition of the request payload, the response payload, so that they understand the client guys, okay? And that's it. So I want to, these are the critical things I have outlined. And also what I did was that I said that I want to use Python classes, so I just created a skeleton thing. I actually did some time back, so I had some things. So I put some cut wrapper, I don't know what to call it. Was it spelling wrong? Oh, yeah, yeah, it's a cut wrapper. I created a project in our app thing, but just feel free to change it, okay? I just created a very basic structure here, which I want you guys to leverage that. So in the December feature, I created some config. I think models have not created anything, but if you go back to the, I think maybe models have created something in the app. So this is what I want you to, what you've been writing. Sorry. What did you say? Can you share your screen? You created some folder, right? Project folder. Yeah, yeah. So in the December, I thought I did this. Where is cut folder? I mean, December feature, yeah, cut wrappers. So the config is there, basic thing, models, and source. In the source, we obviously have a curve and a main API, which this will expose the flags, okay? I just created something, and it's not right, actually. So just you guys have to fix it. I just want to give some structure so that we don't mess up. So this is our class definition, and then we have functions here, okay? So I want to make sure that you go with this approach. And then, so I think, then obviously you have to put a docker. I think I just copied our version, so just fix this docker. Requirements, I just put something that do more. And then obviously we will, what I'm right now doing is that I don't want to store these things, so I'll just log it to our Elk Stack, because any more storing, hearing, we'll see that later part, because they need this so urgent, we have to first get this going. We'll see whether they want to store the request or not unnecessarily adding more time, okay? That's my concern. As long as we just post to our Elk Stack, that would be good. But we'll see what to do next, okay? So that's the basic thing. So I want you to guys take a look deeper into it. So before I, yeah, skeleton is done. I just created one thing. Yeah, so my take would be that I'll be doing some, I think I said this two are done. It was success. I'll do some more things on this one, more and more, okay? And then query API. These are the main things. So I want you to concentrate on the remaining piece of the puzzle. Just structure properly. I know even if this may not work, how do you start from top to bottom, bottom to up? Keep that in mind and start going step by step, okay? Okay, so from my side, if I give you this payload, you guys will be happy, so I'll start giving you this data, okay? So at least I gave you these things. I already gave these two. I'll give you these two, and then at least you can start doing, and then I'll give these two also by today. And then you have your take point. The reason it is so important that I'm hoping that, I know**

**Explanation:**

**Key Objectives**

1. **Focus Areas:**
   * **Create, Query, Close, and Correct/Conduct APIs for managing maintenance activities (e.g., cork requests).**
   * **Ensure the project structure is clean, readable, and scalable.**
2. **Critical Components:**
   * **Payload Definitions:**
     + **Define request and response payloads clearly using Python classes for better understanding by clients and the team.**
   * **Unit Testing:**
     + **Integrate unit tests early to validate functionality and ensure the project remains robust.**
   * **Swagger Documentation:**
     + **Generate Swagger/OpenAPI specs to document the APIs, ensuring clarity for client teams.**
   * **Log Requests:**
     + **Push logs to ELK Stack instead of storing data locally to save time.**
3. **Scaffold and Skeleton:**
   * **You’ve set up a basic structure under the December feature, including:**
     + **Config, Models, and Source folders.**
     + **Skeletons for APIs (with placeholder methods and class definitions).**
     + **A Dockerfile and basic requirements.txt.**

**Immediate Action Items**

**Here’s how the team should proceed step-by-step:**

**1. Review and Fix the Skeleton**

* **Config: Review the settings and ensure environment-specific configurations are included.**
* **Models: Start defining models based on the payload structure you outlined.**
* **Source:**
  + **Cork API:**
    - **Implement the Create and Query methods as per your payload specifications.**
    - **Ensure modular design (e.g., separate concerns for request validation, processing, and response).**
  + **Main API:**
    - **Set up the entry points for the application (e.g., Flask or FastAPI routing).**

**2. Define Payloads and Validation**

* **Use Python dataclasses or Pydantic models to define:**
  + **Request Payloads:**
    - **Include client ID, device ID, and any activity metadata.**
  + **Response Payloads:**
    - **Define what information will be returned (e.g., cork status, request IDs).**
* **Implement payload validation to handle errors gracefully.**

**3. Logging**

* **Set up the system to log API requests and responses to the ELK Stack.**
* **For now, avoid local storage or database integration unless necessary.**

**4. Docker Setup**

* **Finalize the Dockerfile to ensure the project can run seamlessly in containers.**
* **Include necessary environment variables in the config or .env file.**

**5. Unit Testing**

* **Use a testing framework like pytest to:**
  + **Validate payload definitions.**
  + **Test individual API functions (mock external dependencies if required).**

**6. Swagger/OpenAPI Integration**

* **Use a framework like FastAPI or Flask-Swagger to auto-generate Swagger documentation from the codebase.**
* **This ensures the API is easy to understand and use for external clients.**

**7. Priority Tasks**

* **You will focus on providing:**
  + **The payload definitions (request and response) for Create and Query APIs.**
  + **Skeleton implementations for Close and Conduct APIs.**
* **The team will:**
  + **Complete the implementation of Create and Query APIs.**
  + **Structure the project for scalability and readability.**

**Proposed Project Structure**

**bash**

**Copy code**

**project/**

**├── config/ # Environment-specific configurations**

**├── models/ # Data models (e.g., Cork, Client, Device)**

**├── source/**

**│ ├── cork\_api.py # Core logic for Create, Query, etc.**

**│ ├── main.py # API entry points and routes**

**├── tests/ # Unit tests**

**├── Dockerfile # Docker container setup**

**├── requirements.txt # Dependencies**

**├── README.md # Project documentation**

**Next Steps for the Team**

1. **Review the provided skeleton and payload definitions.**
2. **Begin implementation of the Create and Query APIs.**
3. **Set up Swagger documentation for the exposed APIs.**
4. **Push logs to ELK Stack for all API requests/responses.**

**You can request below. Pre-meeting rights, that will be a thing. Yeah, even is to storing, you mean to say? Yeah, storing... Yeah, that's what I said, that's what I said, right? We don't have to persist now for now, that's what I said. Put that action item. Persistence is not right away. We don't have to worry about it, okay? Because we are a transient system. And the other thing is that, if you put that in a... I don't want that it has a blocker, say hypothetically something happened, you can't store, you can't get, and I don't want to stop the request. Okay? So that's what, right? So, keeping that in mind, yes, I forgot to add that, someone please add that, yes, MongoDB will go. JSON objects, that's a JSON object, we are JSON, get the hell out of it. Why do you have to keep tearing, un-tearing, tearing the damn data and different structures, right? I'm with that, okay? It's simple, I don't want to spend thousands of hours to understand each attribute. If I want something, I'll just get it, so I'm with that. Put a MongoDB attached, but put everything on your wish list. You guys will make that, I'll just unnecessarily say, hey, why is it not working? I don't want you guys to spend hours to tell me what the answer is. Or reporting, a lot of things will come. Keeping everything in mind, put on your wish list, but we'll try it step by step, okay? Perfect. Yeah, sure. Yeah. So, another question, just give me some time. I'll ask you to, I think I need to, I'll ask you to enter somewhere, you guys have to register. Once you do that, I think it will work. Okay, maybe I'll do the registration for you guys. Give me some time. Okay, any questions? No, we are good.**

**Explanation:  
Key Updates & Decisions**

1. **Persistence (MongoDB):**
   * **No Immediate Persistence:**
     + **The system is transient for now, so storing data is not a priority.**
     + **This ensures the API functions without blockers even if storage mechanisms fail.**
   * **Future Integration:**
     + **MongoDB will be used to persist JSON objects in the long term.**
     + **The design will prioritize simplicity by avoiding unnecessary transformations of the data structure (JSON in, JSON stored).**
     + **This approach minimizes time spent understanding or converting attributes.**
2. **Error Handling:**
   * **The system should handle failures in storage gracefully (e.g., log the issue but continue processing requests).**
   * **Storage-related issues must not become blockers for the main flow.**
3. **Wish List for Future Enhancements:**
   * **Persisting data (MongoDB integration).**
   * **Reporting and analytics functionalities.**
   * **A robust error-handling and notification mechanism for failures.**
4. **Registration Process:**
   * **A registration process needs to be completed (possibly related to client or API access).**
   * **You will handle this registration, but the team should remain prepared for instructions.**

**Action Items**

1. **Immediate Tasks:**
   * **Finalize and implement the Create and Query APIs without persistence.**
   * **Ensure the system is transient and does not break due to storage unavailability.**
   * **Continue logging requests and responses to the ELK Stack.**
2. **Future Planning:**
   * **Add MongoDB integration to the wish list with clear goals for:**
     + **Storing JSON objects for easier access and future scalability.**
     + **Simplifying reporting and analytics by leveraging the stored data.**
3. **Follow-Up:**
   * **You will provide details on the registration process and any related dependencies.**
   * **Once registered, ensure the APIs are tested in the registered environment.**

**Next Steps for the Team**

1. **Focus on API Implementation:**
   * **Continue with the transient design.**
   * **Use logging for traceability in place of persistence.**
2. **Prepare for MongoDB:**
   * **Plan how to structure the data (JSON object-based).**
   * **Keep integration modular to add persistence later without disrupting the flow.**
3. **Action Wishlist:**
   * **Maintain a prioritized list of enhancements for reporting, error handling, and persistence.**

**Tools and Technologies:**

**Development Tools**

1. **Python Frameworks:**
   * **FastAPI or Flask: For creating lightweight and scalable REST APIs.**
   * **Pydantic: For payload validation and schema definitions.**
2. **Unit Testing:**
   * **pytest: For creating and running test cases.**
   * **mock: For mocking external dependencies during testing.**
3. **Code Quality:**
   * **Black: For automatic code formatting.**
   * **flake8: For linting and ensuring PEP8 compliance.**
   * **mypy: For optional static typing checks.**

**Logging and Monitoring**

1. **ELK Stack:**
   * **Elasticsearch, Logstash, Kibana: For centralized logging, analysis, and visualization of logs.**
2. **Application Monitoring:**
   * **Prometheus and Grafana: For monitoring system health, API performance, and resource utilization.**

**Database and Storage**

1. **Primary Database:**
   * **MongoDB: For persisting JSON objects and enabling seamless querying of unstructured data.**
2. **Future Considerations:**
   * **PostgreSQL or MySQL: If relational database support becomes necessary for structured data.**
   * **Redis: For caching frequently accessed data to improve API response times.**

**Containerization and Deployment**

1. **Docker:**
   * **For containerizing the application and ensuring consistency across environments.**
2. **Kubernetes:**
   * **For managing and orchestrating containers at scale (if required later).**
3. **CI/CD Tools:**
   * **GitLab CI/CD or Jenkins: For automating build, test, and deployment pipelines.**

**API Documentation**

1. **Swagger / OpenAPI:**
   * **To auto-generate and maintain API documentation for ease of use by client teams.**
2. **Postman:**
   * **For testing APIs and sharing test collections with the team.**

**Authentication and Security**

1. **OAuth2 / JWT:**
   * **For secure client authentication and token-based authorization.**
2. **HashiCorp Vault:**
   * **For managing sensitive credentials like API keys and environment secrets.**

**Future Enhancements**

1. **Reporting and Analytics:**
   * **Tableau or Power BI: For advanced data visualization and reporting.**
2. **Error Tracking:**
   * **Sentry: For real-time error tracking and debugging.**
3. **Event Streaming:**
   * **Apache Kafka: For handling real-time data pipelines and event-driven systems.**

**Collaboration and Communication**

1. **GitLab:**
   * **For source control, code review, and collaboration.**
2. **JIRA:**
   * **For task tracking and sprint planning.**
3. **Confluence:**
   * **For documenting processes, workflows, and decisions.**