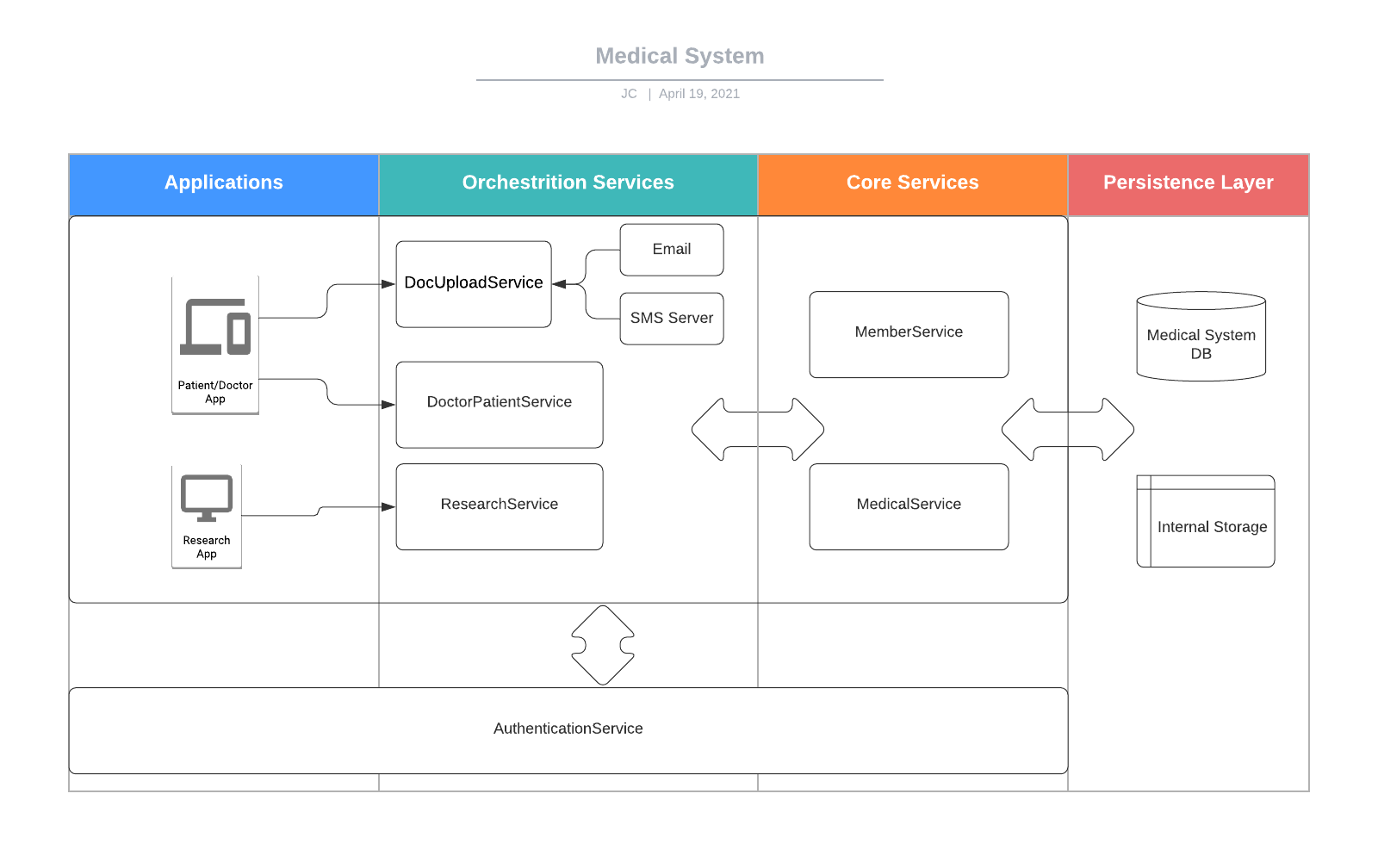
**Medical System**

**Specification** - see original requirement document

**High Level Design**

****

**Design Detail and Implementation Suggestions**

We design the system in four layers, applications, application orchestration service, core service and storage layer, pulse a central AuthenticationService for handling authentication and permission control.

***Application Layer***should not directly access core services. Each application should have its own orchestration service and through it to get services from core services. Applications can be Web UI applications for doctors and patients. Applications can also be server applications such as AI applications for research purposes and ETL applications to generate reports.

***Application Orchestration Service Layer*** combines services from core services to response application specific requirements based applications permission control. By introducing an application orchestration service, we get following benefits

1. Core services don’t need to provide application specific requirements.
2. Decope core services from applications. Changing core services wouldn’t affect applications.
3. Core services don’t need to handle permission control. The application orchestration service handles permission check and data privacy masking.

***Core Service Layer*** provides core business services and performs data persistent functions. Core services doesn’t directly provide services to applications. They only provide services to application orchestration services. They don’t need to check permissions. But they must authenticate client applications for every request. We can use a client certificate or a JWT token for the client authentication.

**Storage Layer** storage patient/doctor personal information and patient medical information such as diagnostic, treatment and patient’s documents.

1. Suggest to use a relational database to save patient/doctor personal information and patient visit information.
2. Suggest to use a distributed file system such HDFS or Amazon S3 to keep patient’s document for its high availability and scalability

**AuthenticationService** provides members login process and user permission checks.

1. Authenticates a user by user id and password and returns a security token which contains user rules.
2. Provides permission check. A rule-action mapping can be implemented here.
3. Provides client authentication for services

**MemberService** - a core service to provide services for personal information related actions, view personal information and update personal information.

*MemberService provides following functions:*

1. getMemberInfo(memberId)
2. updateMemberInfo(memberId, MemberInfo)
3. createMember(MemberInfo)

**MedicalService** - a core service to provide services for all medical related actions, view medical information, enter new medical information and upload patient documents. The system doesn’t allow anyone to update existing medical information. The “update” is really done by entering new medical information.

*MedicalService provides following functions:*

1. listAllVisits(patientId, searchCriteria)
2. getPatientMedicalSummary(patientId)
3. enterMeddicalInfo(patientId, doctorId, diagnostic, treatment)
4. uploadMedDoc(patientId, content). Permission: a patient can upload own documents, a doctor can upload his/her patient’s documents
5. listPatientDoc(patientId)
6. getPatientDoc(docId)

**DoctorPatientService -** an application orchestration service toprovide services to an UI application which is used by patients and doctors

**PatientDocUploadService** monitors a special cell number and an email account to download images sent by patients and save them into the system**.** The service uses the sender's number or email address to identify who the patient is. To support mobile SMS messages, we could use a third party service such as Twilio.

**ResearchService** - an application orchestration service to provide services to research systems or personnels. All patient’s personal information should be hidden or masked.

ResearchService provides the following services: to be defined.

**Security Considerations**

Considering the nature of high privacy regarding the medical information, accessing all services has to be done in a very strict security way. No systems/applications/users can access services without a security token.

**Encrypted Transmission**

All data traffic will be encrypted using HTTPS.

**Application Authentication**

1. All interaction between services should pass a client authentication first before services can be provided
2. We can use either a X509 client certificate or JWT token for the client authentication. Suggest to use JWT for the client authentication for two reasons
   1. In general, JWT is much easier to implement than X509. X509 certificates depend more on security infrastructure.
   2. JWT is much more human friendly than X509 certificates. It is much easier to use an on-demand JWT than pre-installing client certificates for each patient and doctor. If we use JWT for patients and doctors, we should use JWT for applications too to keep client authentication consistent.

**Member Authentication and Permission Control**

1. A rule based permission control can be built in AuthenticationService
2. Three rule types: DOCTOR, PATIENT, RESEARCHER.
3. A person might have multiple rules. For example, a doctor can also be a patient or be a researcher.
4. Members need to login first before they can use the system. AuthenticationService performs the user authentication. After a user be authenticated, a JWT token which includes user’s rules will be issued
5. After UI application gets a user’s token, it will use the token to perform request to orchestration services
6. Orchestration services first verify the token and reject any requests with a compromised token.
7. After the token is verified, orchestration services check user permission based on the user's rules in the token. Based on permission check, orchestration services can
   1. Reject the request with a Permission Denied error
   2. Provide the service
   3. Mask or hide the private data

**Permission Control Matrix**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rule** | **Service** | **API** | **Permission** |
| Patient | MemberService | All | Only permitted to read and update the patient’s own information |
| Patient | MedicalService | All except enterMedInfo | Only permitted to read the patient's own information. Also permitted to upload own document |
| Patient | ResearchService | All | No |
| Doctor | MemberService | getMemberInfo | Only permitted to his/her patients |
| Doctor | MedicalService | All | Only permitted to read/enter his/her patients medical information |
| Doctor | ResearchService | All | Yes |
| Researcher | MemberService | All | No |
| Researcher | MedicalService | All | No |
| Researcher | ResearchService | All | Yes |

**Audit Trail**

*Assumption*

1. For updating member information such address, phone number and etc, we only keep an audit trail of who and when update. We don’t keep information history.
2. For updating a patient’s medical information, we not only keep an audit trail, but also keep medical information history. For example, a doctor can “update” a treatment by entering a new treatment. So later, the patient and doctors can see what is original treatment and a new treatment. The original treatment wouldn’t be overwritten.

MemberService and MedicalService should write an audit log into the AuditTrail table for all creation and update operation. Information such as patient id, operation type, time and who did the operation will be persisted.

1. A common audit trail library can be written and used by all services to save an audit trail.
2. For medical information, we only add a new record into the Visit table and wouldn’t ever overwrite a record.

**Anonymous Access**

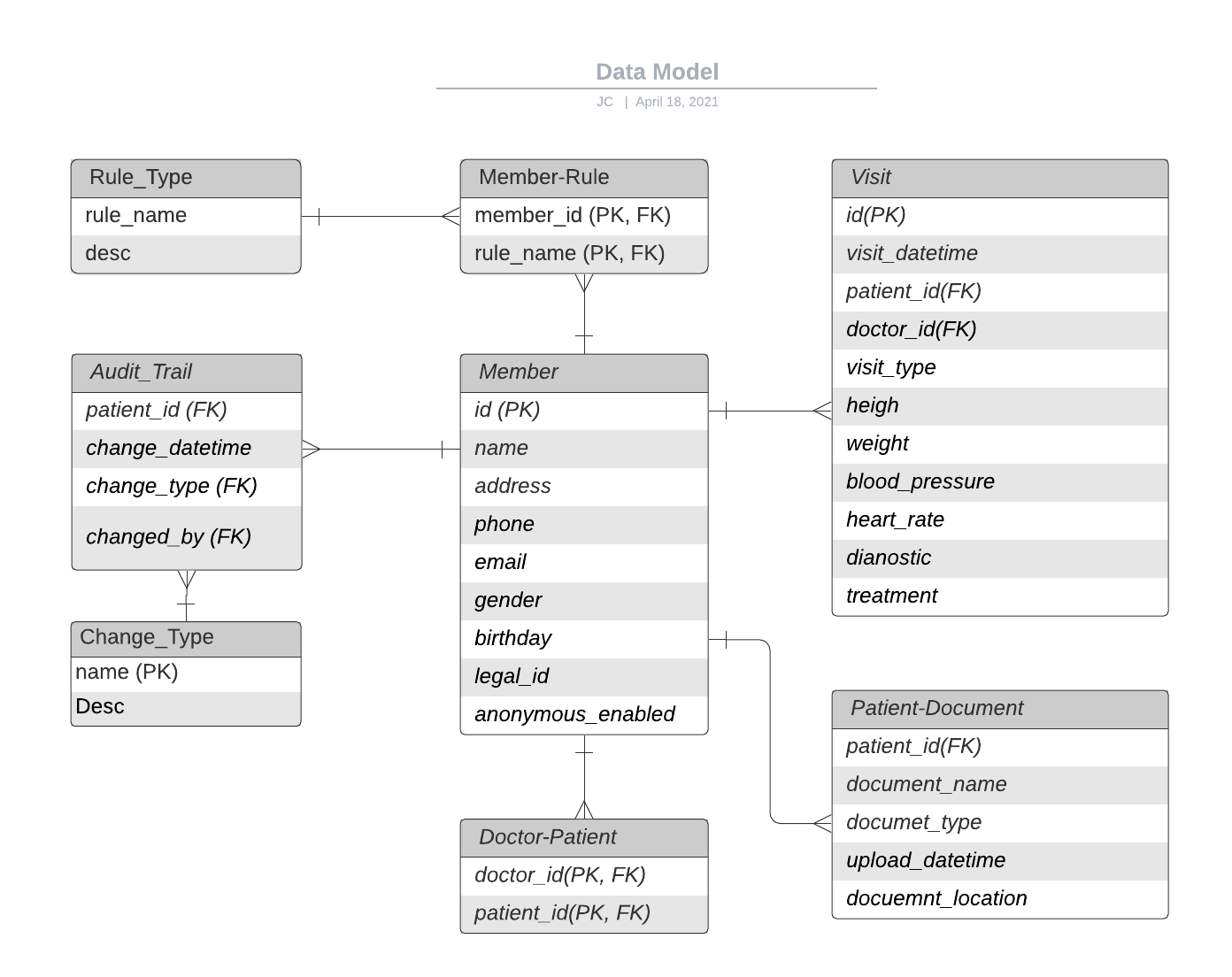
Applications that need to access the system anonymously should only through an application orchestration service. For example, an AI application should only access system data through ResearchService. As described before, the application orchestration service should

1. strip out or mask a patient's private data.
2. only use the a patient’s data if the patient’s enable anonymous access

**Deployment**

All components can be containerized and deployed into Kubernetes enabled cloud or a similar cloud. By using a Kubernetes cloud, we can much easier to scale the system for the growing usages. We also can use multiple replications for each components to improve the system availability

**Data Model Design**



**A sample implementation of patient medical summary service**

1. Java + Spring Boot
2. H2 in memory database is used for persisting the data
3. JPA is used for the data access
4. All sample data is pre-populated at the start time using the script at ./src/main/resources/data.sql
5. Using following commands to see some data
6. For the Medical Summary, I didn’t implement the time range requirement. It should be very easy to implement it at the JPA level.

To run the application

>cd <project directory>

>./mvnw spring-boot:run

See all members in the database

> curl <http://localhost:8080/member/list>

List all visits of a patient

>curl <http://localhost:8080/member/visits?patientId=3>

Get the medical summary of a patient

>curl <http://localhost:8080/patient/3>

Create a new member

>curl -i -X POST http://localhost:8080/member/create -H "Content-Type: application/json" -d '{"phone":"212-222-6666","name":"Eric", "address":"Albany", "legalId": "6666", "birthday":"1990-01-01", "gender":"M"}'