# MCG Assessment Technical Design

Patient Care Management System

## Summary

The purpose of this document is to provide a high level description of the Patient Care Management System API design requested by MCG's Hiring Department as part of Tony Robinson's Staff Software Engineer application. This application is built on the .NET Core Framework v8.0.

The system requested is separated into the following key components:

- **Patient Management API (.NET Core WebApi)**
  - Customer interaction layer. Validates requests for and properly formats responses from PatientManagementService. Ensures proper user authorization and permissions.
- **Patient Management Service**
  - Processes requests and coordinates business rules between data providers. Handles data transformation for current and future data stores/services.
- **Patient Data Provider**
  - Handles security, storage, and retrieval of patient data by interacting with the implementation of the **Patient Repository.**
- **Document Data Provider**
  - Handles the security, storage, and retrieval of uploaded documents. Currently limited to patient documents, but future expansion could allow for other document types to be stored as well. No direct patient data is associated with the document storage.
- **Patient Repository**
  - Queryable database of patient data. Data structure and indexing needs points to a RDBMS for this purpose.
- **Document Repository**
  - Simple key based storage system used to store potentially large files. This could be accomplished by a table in an RDBMS, NoSQL implementation, or Cloud/Local file storage system.

# External Dependencies

The expectation is that the following services will be provided outside of the application, and are not covered in this document.

- **User Authorization Service**
    - For security reasons, it will be important to have the ability to limit access to only those authorized to view/update these records, as well as limit permissions based on the user's role. For prototyping, a simple stand in authorization service relying on a predefined token in the Request header handles this function. A proper service will need to be implemented before releasing to production.
- **Cryptography Service**
    - As data security is an important concern, further discussion is needed on what cryptography algorithm is required and how it should be handled to ensure compliance with all data protection laws, both general and HIPAA related. The prototype has a stand in class to show where this will take place, but does not actually perform encryption/decryption at this time.

# Application Structure

Application will follow proper SOLID programming practices. Layers below the AppRoot (Client API) will be broken into at least 2 projects, one for contract, other for implementation. This will improve portability and extensibility, while limiting risk of direct coupling of layers.
The AppRoot will be the only project to reference all implementations required for the application.
When an implementation project relies on usage of another layer, it will reference only the contract project for that layer.
Layers in this application are defined as the following:

- AppRoot (Client API)
- Domain Service (Patient Management Service)
- Data Access Layer (Patient Data Provider, Document Provider)

# Data Security

All identifiable patient data, beyond first and last name, will need to be encrypted prior to storage in any repository, and decrypted upon retrieval. Separating this from the database will limit bad actors from being able to access patient data if they gain access to the repository directly.
Any queryable data that could be considered identifiable (medical conditions, document types, etc) should be obfuscated in the data provider so that querying can still take place, but will not provide information directly without translation.

Secure data should include a header at the start or end of the encrypted data blob that signals to the cryptography layer what version of encryption was used so that future changes to the cryptography algorithm can be instituted without needing to re-encrypt the entire repository.

# Prototype Details and Limitations

For the purposes of the prototype, an in memory database via Entity Framework was used for the Patient Repository. This uses static data provided by the application bootstrap.
While all services and data methods have been implemented, only a handful of data retrieval methods and one data update method has been implemented at the API level. This shows the patterns expected to be used by the controllers for all implementations. This was a rapidly developed prototype, none of the code is ready for direct release. Automated unit tests should be added for all layers, as well as integration and end to end tests for the overall application.
While EF Core was used for the Patient Repository, and can be used for interacting with a database in the future, proper data design should take place leveraging internal database security, data fingerprinting via hash, and full transactional logging added.
Two static user authorization tokens have been created to test permission validation and allow for interaction with the API. Providing one of these tokens as the “Authorization” header will allow the included methods to be executed. Excluding this header, or providing a different value will result in an Unauthorized response.

- **clerk-valid -** Provides read only access to patient data
- **admin-valid -** Provides read/write access to patient data