Ofek Wittenberg owit454

HALO INFECTION SCREENING

325 – Web Service

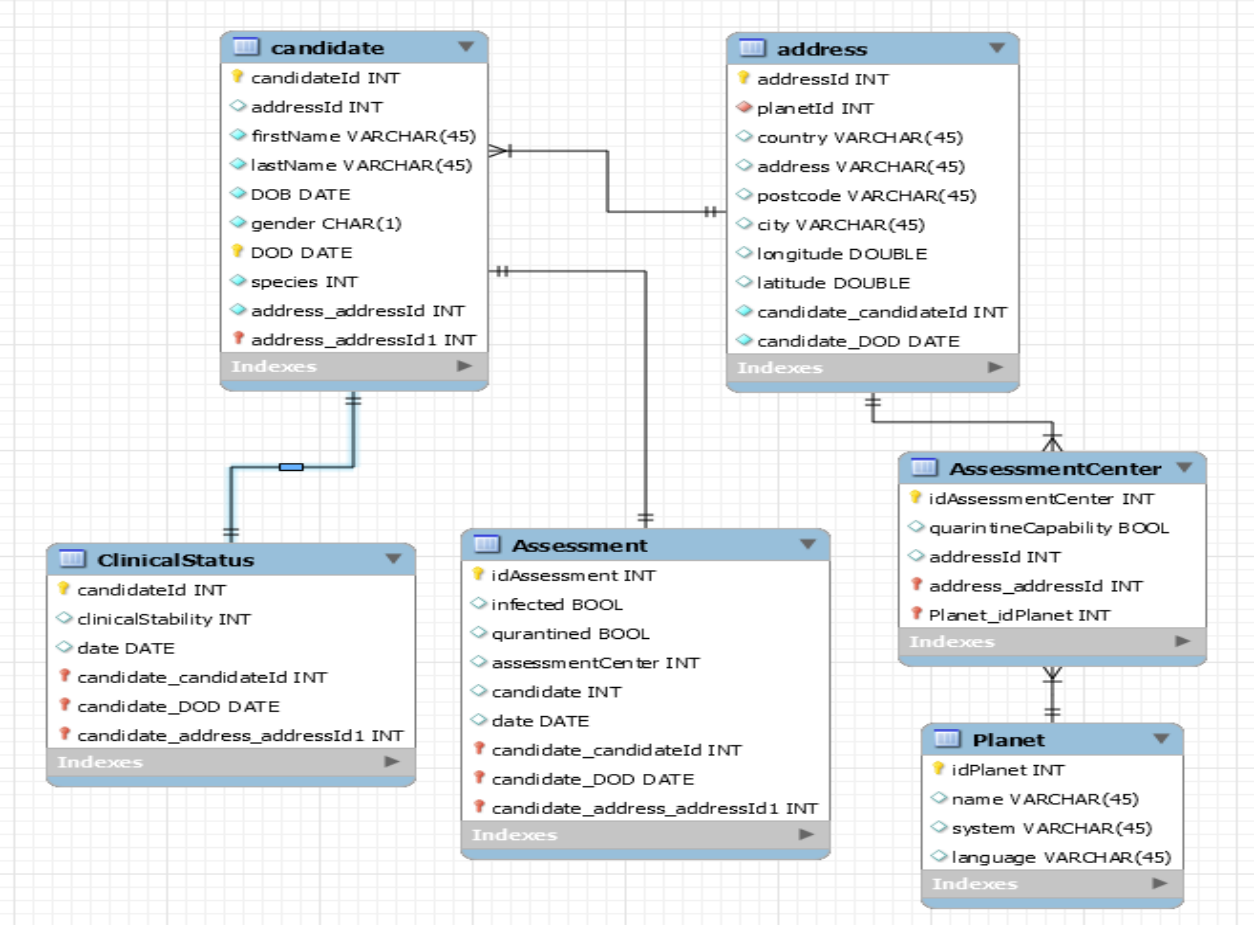
SERVICE OVERVIEW

This web service is set in the Halo universe and is designed around handling and managing the workflow of screening patients for the Flood infection. The Flood infection hacks into the host's nervous system, it releases encapsulated Flood Super Cells into the body. These cells interface with the host's cells, "digest" them and convert their components into new Flood cells. At this point, the infection form burrows into the host body, moving aside the internal organs and taking up residence within the chest cavity. Eventually it will kill the host.

Patients that are entered into the screening program are managed via the web service. The service handles all the data and workflow related to the screening and tracking process. All candidate details are stored such as date of birth, gender, species and address (an address is associated with a planet).

Every candidate has assessments that are associated with that candidate. An assessment is associated with an assessment center which is also linked to its own address.

A candidate also has a clinical log. A clinical log holds unique medical status's of the candidate.
If a candidate chooses to leave the program, all their associated data is deleted apart from their address as candidates can share an address.



SERVICE DESIGN

The service is designed to be efficient, scalable and secure.

Many to one relationships such as assessments are lazy fetched to reduce server load when querying an existing candidate.

Additionally a candidate DTO (data transfer object) is used when sent through the web service. Because the DTO holds less data than the Candidate object, less bandwidth is required to send and receive information.

The web service is designed around a per request model. This means that each incoming request involves creating a service instance whose lifetime is the duration of the request. Although this is slightly more resource intensive the connection to the database meta-data is maintained through singleton class that is instantiated on load. This EntityManagerFactory is expensive to create, because it involves processing all the metadata in the JPA class definition and hence only created once. Using a per-request model also makes the web service more thread safe.

Finally the web service follows the four principles of a Restful application; different request methods are supported within the resource class. These web service follows a uniform, constrained interface as it utilizes all HTTP request methods. It also leverages HTML protocols such as path parameters, query parameters, cookies and caching. HTTP protocols are used because it is a recognized and global web standard that that is compatible with all web browsers.

Resources are addressable through individual URI. The service is representation oriented (a request for getting a candidate can either return JSON or XML formats) and finally the service is stateless (each request contains all information needed for the service request).

QUALITY ATTRIBUTES

ASYNCHRONOUS METHOD

The web service supports an asynchronous method which is used to simulate a resource intensive request that may take some time to process. In the context of the infection screening, the asynchronous looks through all the clinical logs of a candidate calculates the most critical log entry on a separate thread and asynchronously returns the most critical entry.

SECURITY

The web service also supports a form of Cross-Site Request Forgery (CSRF) protection. CSRF is a type of attack that occurs when a malicious Web site, email, blog, instant message, or program causes a user's web browser to perform an unwanted action on a trusted site for which the user is currently authenticated. This is achieved via generating a unique key using a variable seed (in this case the current time). This CSRF token is embedded within a private cookie that is sent in a request. The CSRF token is compared to the resource token and if they are similar (meaning CSRF token in the request was generated at a similar timeframe to the one on the application service and that it was hashed using the same algorithm that is on the server). This dual protection layer is quite simple but very efficient.

Although this security feature is not embedded within all the resource methods (as it was more a proof of concept), it could be quite easily integrated into the current service through a custom interceptor that checks all incoming requests for the CSRF token.