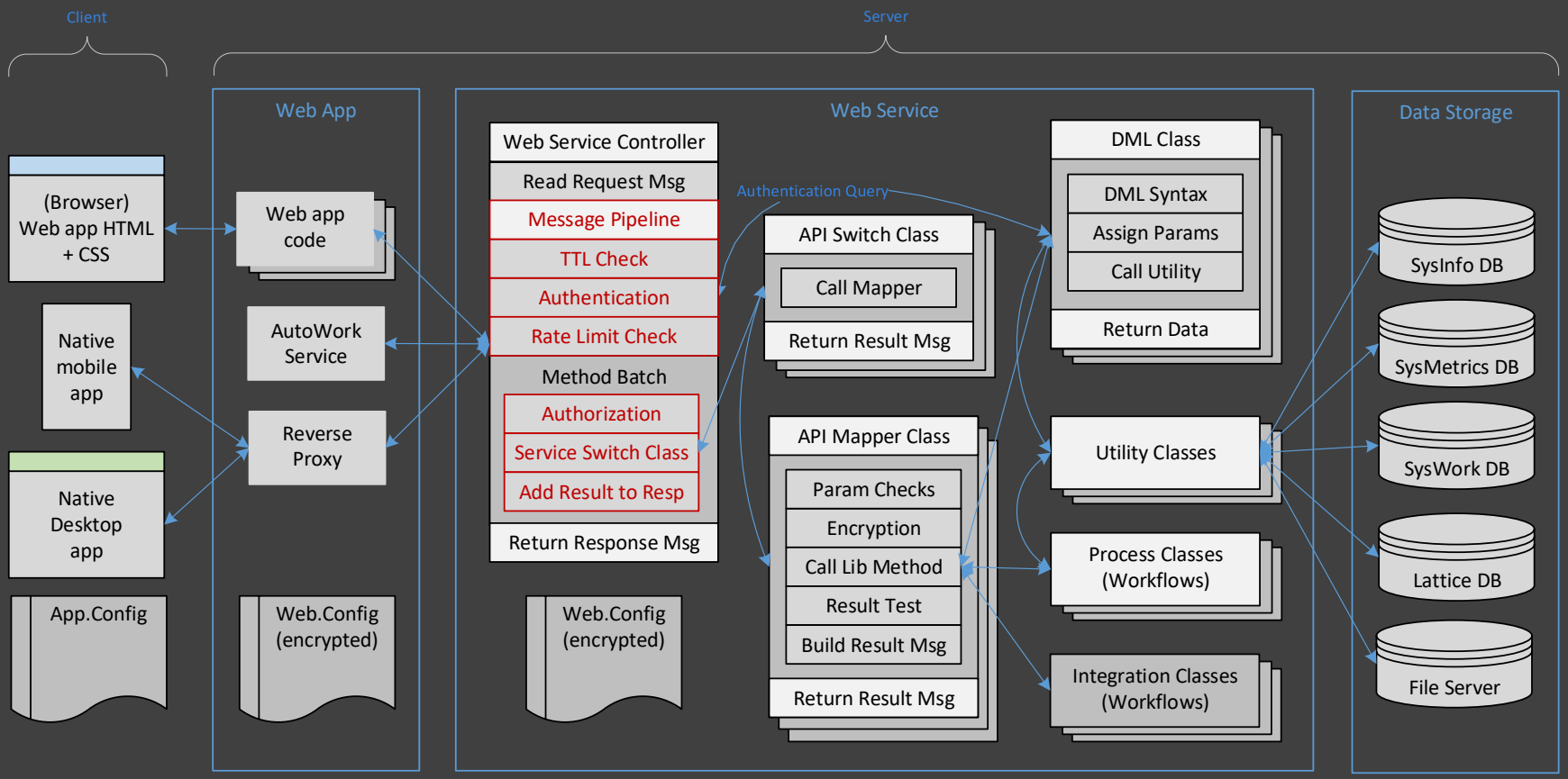

The message pipeline accepts the API request from the controller and performs all of the work to process the API method calls contained in the API request message.

The first steps of the message pipeline are security-related, and check if the message itself has expired, checks the user account authentication, or if the account has made too many API requests within the last minute. If those checks are successful, then each API

method call in the API request batch is individually authorized. If it passes the authorization check, it is passed into the web service switch class, which in turn will instantiate and call the appropriate API Switch class.

Security

1. The message processing pipeline class encapsulates the functionality of an API gateway:
    a. Message TTL check
    The API request message includes a UTC timestamp value that is set in the client app and is used to enforce the maximum Time-To-Live of each message. This is useful to minimize replay attacks, especially when the TTL allowed is fairly short.
    b. User account authentication
    The API request message includes both a UTC timestamp value and an HMAC hash of that timestamp. An HMAC hash is calculated by the web service to authenticate the API request message if they match. The timestamp is the same one used to enforce the message TTL, so the HMAC hash value expires at the same time as the UTC timestamp.
    c. Failed login count check
    The maximum number of failed login attempts allowed before the account is disabled. Each successful login resets the count to zero.
    d. Account rate limit check
    The number of methods called each minute is tracked in the web service, and any method calls over the rate limit are rejected by the message processing pipeline logic.
    e. API method authorization
    Each individual method call contained in the API request message (batch) is checked vs. the user account access control list of authorized API methods. Authorization is checked before the internal method is called.
    f. Data access authorization
    The user account access control lists of authorized data groups (one for read-only access and another for read-write access) are passed as input parameters to the data access method for a given table. The SQL syntax incorporates the ACL into the query to enforce the data authorization restrictions

2. The data-driven RBAC security subsystem provides the foundation for user account authentication, API method authorization, and data group authorization. The authentication query of the SysInfo security database returns all of the API method authorization data and the data group authorization data for the account.

3. The appSettings section of the Web.config file is encrypted using the aspnet_regiis.exe utility, which in turn uses Microsoft's PKI to encrypt/decrypt, manage the encryption keys behind the scenes, etc. This encryption is completely transparent to the code that uses the web.config values.
    a. A list of all the symmetric encryption keys used in a system over time is maintained in the Web.config file, and must be encrypted. These encryption keys are used for the zero-knowledge encryption of sensitive data within the web services, as well as for the remote end-to-end encryption of content files in the Lattice file store.
    b. All ADO.NET connection strings are maintained in the Web.config file, and must be encrypted because of the SQL Server usernames and passwords they contain.

Performance

All of the user account authorization data from the security database is cached into each user account record. As a result, the query to authenticate the account and authorize each method in the API request message is accomplished by retrieving a single record from a single table (APIUser) in the SysInfo database. This cached data is updated when needed by the Login method called by each user account when they connect to one of the locations of a DGP system.

Performance is further improved when the UserInfo object is cached in the memory of the web server using the ASP.NET Cache object, eliminating the need for the query to the SysInfo database plus the decryption of the account password for each API request message.

The use of the ASP.NET cache object for the user data and rate limit data benefit from the use of "sticky sessions" (session affinity) within the load balanced server farms of larger scale locations, but is not mandatory (the web service uses the cache aside pattern).