Authentication

Authentication is used for access control. It is used by web application to determine user's identity. Authentication does not determine what content and actions are available to a given user.

Forms of authentication

1. Username and password *we will use this method for our web applications*
2. Cards
3. Retina scans
4. Voice Recognition
5. Fingerprints

Authentication in Web Applications

1. ***Cookies***
   A small piece of text stored in a browser. Web browser passes cookies to a server via HTTP protocol in the request header. In order to set a cookie, you would use **`Set-Cookie: name=value`** and Browser will send it as **`Cookie: name=value`**. Common examples of cookies are site preferences, shopping card items, and server session identification.

2. ***Sessions***
   A server-side storage used for persistent information. Sessions use session identifier stored in users' browser to associate the data.

3. ***Laravel***
   a. ***Out of the box***
      Laravel provides easy solution for authentication. It will create views, example controller, and update routes for your authentication pages. You will need to call **`make:auth`** command using Artisan command line tool.

   b. ***Third Party (Socialite)***
      If you would like to use other services for authentication such as Facebook, Twitter or GitHub, you could use Socialite package.

      URL: http://socialiteproviders.github.io/

Authorization

Authorization is a process used by web application to determine whether user has permission to access specific content or perform certain actions. Authorization may or may not require authentication. For example, guest users do not need to authenticate, but your application will have set authorization policies for guest users.

Authorization consists of:

1. Defining policies
2. Checking authorization

Encryption [1]

Encryption is a process that transforms data so that it will be unreadable to others while encrypted. Encryption is used to prevent unauthorized access to accounts or data.

// Copy from: What is Data Encryption? Process and Types of Encryption

There are two types of encryption: ***asymmetric*** and ***symmetric***.

Asymmetric Encryption

In public key (asymmetric) encryption, two mathematically-related keys are used: one to encrypt the message and the other to decrypt it. These two keys combine to form a key pair. Asymmetric encryption provides both data encryption and validation of the communicating parties' identities and is considered more secure than symmetric encryption, but is computationally slower.

Components of asymmetric encryption:

1. ***Plaintext*** - This is the text message to which an algorithm is applied.
2. ***Encryption Algorithm*** - It performs mathematical operations to conduct substitutions and transformations to the plaintext.
3. ***Public and Private Keys*** - This is a pair of keys where one is used for encryption and the other for decryption.
4. ***Cipher text*** - This is the encrypted or scrambled message produced by applying the algorithm to the plaintext message using key.
5. ***Decryption Algorithm*** - This algorithm generates the ciphertext and the matching key to produce the plaintext.

The Encryption Process

The asymmetric data encryption process has the following steps:

1. The process of encryption begins by converting the text to a pre-hash code. This code is generated using a mathematical formula.
2. The software using the sender’s private key encrypts this pre-hash code.
3. The private key would be generated using the algorithm used by the software.
4. The encrypted pre-hash code and the message are encrypted again using the sender's private key.
5. The next step is for the sender of the message to retrieve the public key of the person this information is intended for.
6. The sender encrypts the secret key with the recipient's public key, so only the recipient can decrypt it with his/her private key, thus concluding the encryption process.

The Decryption Process

The asymmetric data decryption process has the following steps:

1. The recipient uses his/her private key to decrypt the secret key.
2. The recipient uses their private key along with the secret key to decipher the encrypted pre-hash code and the encrypted message.

3. The recipient then retrieves the sender's public key. This public key is used to decrypt the pre-hash code and to verify the sender's identity.
4. The recipient generates a post-hash code from the message. If the pos~-hash code equals the pre-hash code, then this verifies that the message has not been changed enroute.

Symmetric Encryption
Private Key encryption (Symmetric) also referred to as conventional or single-key encryption is based on secret key that is shared by both communicating parties. It enquires all parties that are communicating to share a common key. The sending party uses the secret key as part of the mathematical operation to encrypt (or encipher) plain text to cipher text. The receiving party uses the same secret key to decrypt (or decipher) the cipher text to plain text.

Examples of symmetric encryption schemes are the RSA RC4 algorithm (which provides the basis for Microsoft Point-to-Point Encryption (MPPE), Data Encryption Standard (DES), the International Data Encryption Algorithm (IDEA), and the Skipjack encryption technology proposed by the United S12; tesgovernment (and implemented in the Clipper chip).

Components of symmetric encryption:

1. ***Plaintext*** - This is the text message to which an algorithm is applied.
2. ***Encryption Algorithm*** - Iperforms mathematical operations to conduct substitutions and transformations to the plaintext.
3. ***Secret Key*** - This is the input for the algorithm as the key dictates the encrypted outcome.
4. ***Cipher text*** -This is the encrypted or scrambled message produced by applying the algorithm to the plaintext message using the secret key.
5. ***Decryption Algorithm*** - This is the encryption algorithm in reverse. It uses the ciphertext, and the secret key to derive the plaintext message.

When using this form of encryption, it is essential that the sender and receiver have a way to exchange secret keys in a secure manner. If someone knows the secret key and can figure out the algorithm, communications will be insecure. There is also the need for a strong encryption algorithm. What this means is that if someone were to have a ciphertext and a corresponding plaintext message, they would be unable to determine the encryption algorithm. There are two methods of attacking conventional encryption - brute force and cryptanalysis. Brute force is just as it sounds; using a method (computer) to find all possible combinations and eventually determine the plaintext message. Cryptanalysis is a form of attack that attacks the characteristics of the algorithm to deduce a specific plaintext or the key used. One would then be able to figure out the plaintext for all past and future messages that continue to use this compromised setup.

// End copy

Laravel uses OpenSSL with AES-256-CBC crypt cypher. All messages are signed with message authentication code (MAC) to detect any modifications to the encrypted string. This is a symmetric encryption.

Command line

Command line scripts are executed through your terminal session on a server. Artisan is an example of command line tool.

Cache

Caching is a technique used to speed up data lookups in your application. Instead of processing data that does not get updated often, you might want to create a cached version of that data, which will store the output and provide that output to users instead.

Methods for caching:

1. *Upfront population*
   Upfront population requires caching all your data ahead of time. The primary advantage of caching upfront is removing any potential processing delays while accessing data. However, it will require more time to initially cache all of the data.

2. *Lazy population*
   Lazy population involves caching data upon access so that any future requests would be provided with the cached version of data and cut down on processing time. Lazy population does not have any upfront processing overhead, but there may be potential user experience issues with data that is not cached and requires to be processed.

Cache expiration:

In order to provide relevant up to date information, your application will need to manage cache expiration.

1. *Time based expiration*
   One way to handle cache expiration is to set up time intervals for its expiration, so that your system could keep up to date information.

2. *Active expiration*
   Another way to handle cache expiration is to provide a mechanism that would notify your caching services to remove and re-cache specific data that was modified.

What can you cache?

1. Files
   a. Backend code, like controllers
   b. Frontend code, like CSS, JS and HTML
2. Database query results, like blog posts or user profiles

Caching applications:

1. Memcached
2. Redis

Which one do I choose?

In order to answer this question, you will need to look at the features each of these applications provides, amount of resources it would take to run this application and how active is support for issues and software updates.

Great article that discusses differences between Memcached and Redis in detail:
http://www.infoworld.com/article/2825890/application-development/why-redis-beats-memcached-for-caching.html

Error Handling

Error handling is one of the most important aspects of web application. While some errors may cause unsatisfactory user experience, others may lead to security issues.

1. ***Exceptions***
    a. HTTP Responses
    b. Try.. Catch..
2. ***Logs***
    a. Single
    b. Daily
    c. Syslog
    d. Errorlog
3. ***Log Delivery methods:***
    a. File
    b. Email
    c. Database

Events

Events in the web applications are implemented to provide developers with ability to detect and react using event listeners. In simple terms, you might have a user registration form in your application. The event broadcasts within your application when user creates an account. You may have another services listening for such events and send out a “welcome” email, or cache user profile.

1. Queued event listeners (Could run on different server)
2. Event broadcasting to different services: Pusher, Redis

Working with files

1. Storing, retrieving and deleting files and directories
2. Native
3. Third Party (AWS, dropbox, ftp, stfp)

Footnotes:

1. http://ecomputernotes.com/computernetworkingnotes/security/types-of-encryption