**CHAPTER 1**

**INTRODUCTION**

* 1. **OVERVIEW**

JWTs represent a set of claims as a JSON object that is encoded in JWS and/or JWE structure.  This JSON object is the JWT Claims Set. The member names within the JWT Claims Set are referred to as Claim Names. The corresponding values are referred to as Claim Values.The contents of the JOSE Header describe the cryptographic operations applied to the JWT Claims   by the JWE.  A JWT may be enclosed in another JWE or JWS structure to create a Nested JWT, enabling nested signing and encryption to be performed. A JWT is represented as a sequence of URL-safe parts separated by period ('.') characters.  Each part contains base64url-encode value.  The number of parts in the JWT is dependent upon the representation of the resulting JWS using the JWS Compact Serialization or JWE using the JWE Compact Serialization.

* 1. **OBJECTIVE**

 The basic objective of JSON web token based authentication is to enhance the transmission of credentials between the client and the server by overcoming the disadvantages of traditional session and cookie creation procedures. It also helps to improve the privacy of the client’s and the server’s personal details that are embedded in it.

* 1. **SCOPE**

To develop JSON Web Token based authentication framework suited for the organization, and to integrate the existing artifacts.

Features of Framework:

* Compact
* Self Contained
* URL Safe
* Highly Secure

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 JSON Web Algorithm(JWA)- M.Jones**

This specification registers cryptographic algorithms and identifier to be used with the JSON Web Signature (JWS) [JWS], JSON Web Encryption (JWE) [JWE], and JSON Web Key (JWK) [JWK] specifications. It defines several IANA registries for these identifiers.  All these specifications utilize JSON-based [RFC7159] data structures. This specification also describes the semantics and operations that are specific to these algorithms and key types. Registering the algorithms and identifiers here, rather than in the JWS, JWE, and JWK specifications, is intended to allow them to remain unchanged in the face of changes in the set of Required, Recommended,Optional, and Deprecated algorithms over time.  This also allows changes to the JWS, JWE, and JWK specifications without changing this document. Names defined by this specification are short because a core goal is  for the resulting representations to be compact.

**2.2 JSON Web Signature(JWS)-M.Jones, N.Sakimura and J.Bradley**

               JSON Web Signature (JWS) represents content secured with digital signatures or Message Authentication Codes (MACs) using JSON-based[RFC7159] data structures.  The JWS cryptographic mechanisms provide integrity protection for an arbitrary sequence of octets. Two closely related serializations for JWSs are defined.  The JWS Compact Serialization is a compact, URL-safe representation intended for space-constrained environments such as HTTP Authorization headers and URI query parameters.  The JWS JSON Serialization represents JWSs as JSON objects and enables multiple signatures and/or MACs to be applied to the same content.  Both share the same cryptographic underpinnings. Cryptographic algorithms and identifiers for use with this specification are described in the separate JSON Web Algorithms (JWA) specification and an IANA registry defined by that specification.  Related encryption capabilities are described in the separate JSON Web Encryption (JWE) [JWE] specification. Names defined by this specification are short because a core goal is for the resulting representations to be compact.

**2.3 JSON Web Emcryption-M.Jones and J.Hilderbrand**

JSON Web Ecryption (JWE) represents encrypted content using JSON-based data structures [RFC7159]. The JWE cryptographic mechanisms encrypt and provide integrity protection for an arbitrary sequence of octets. Two closely related serializations for JWEs are defined. The JWE Compact Serialization is a compact, URL-safe representation intended for space constrained environments such as HTTP Authorization header and URI query parameters. The JWE JSON Serialization represents JWEs as JSON objects and enables the same content to be encrypted to multiple parties. Both share the same cryptographic underpinnings. Cryptographic algorithms and identifiers for use with this specification are described in the separate JSON Web Algorithms (JWA) specification and IANA registries defined by that specification. Related digital signature and MAC capabilities are described in the separate JSON Web Signature (JWS) [JWS] specification. Names defined by this specification are short because a core goal is for the resulting representations to be compact.

**CHAPTER 3**

**ANALYSIS**

**3.1 SYSTEM ANALYSIS**

**3.1.1 Problem Definition**

The problem definition of this project is to illustrate the detailed requirements of “**JSON WEB TOKEN BASED AUTHENTICATION**”**.** This will explain the purpose and features of the system, what the system will do, and the constraints under which it must operate.

**3.1.2 Proposed System**

The API model has been used a great amount recently in applications. This has come about because applications can’t just rely on their own data anymore, for a project to fully see its potential, it must be able to have third-party applications, intermingle with other applications, and have its data easily accessible by developers. when we want to build our own APIs, there’s always going to be the topic of  **how to secure our own API** for which JWT is used. JSON Web Tokens (JWT), pronounced “jot”, are a standard since the information they carry is transmitted via JSON. **JSON Web Tokens work across different programming languages** that is, it work in .NET, Python, Node.js, Java, PHP, Ruby, Go, JavaScript, and Haskell. So you can see that these can be used in many different scenarios. **JWTs are self-contained,** they will carry all the information necessary within itself. This means that a JWT will be able to transmit basic information about itself, a payload (usually user information), and a signature. **JWTs can be passed around easily**, Since JWTs are self-contained, they are perfectly used inside an HTTP header when authenticating an API. You can also pass it through the URL.

**ADVANTAGES:**

* Usability
* Easy access
* Version management
* Enhanced Security
* Compact

**3.2 SYSTEM CONFIGURATION**

**Introduction**

The requirement specification is a technical specification of requirements for the software products. It is the first step in the requirement analysis process. It lists the requirements of a particular software system including functional, performance and security requirements. The requirements also provide usage scenarios from a user, an operational and an administrative perspective. The purpose of software requirement specification is to provide a detailed overview of software projects, its parameters and goals. This describes the project target audience and its user interface, hardware and software requirements. It defines how the client, team and audience see the project and its functionalities.

**3.2.1 Hardware Requirements**

Processor : > 2GHz

RAM : 512 MB

Hard Disc : 80 GB and above

**3.2.2 Software Requirements**

Front End : JSP and servlets

Back End : Apache tomcat server

Operating System : Windows 7 Enterprise

Tools used : Eclipse kepler win 32

Language : Java

**CHAPTER4**

**SYSTEM DESIGN**

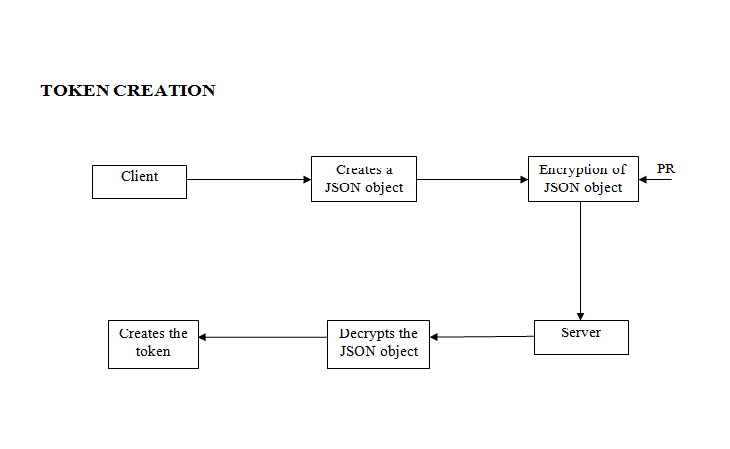
**4.1 INTRODUCTION**

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. This section explains the system design. This provides details about architecture and components that are necessary to implement the system.

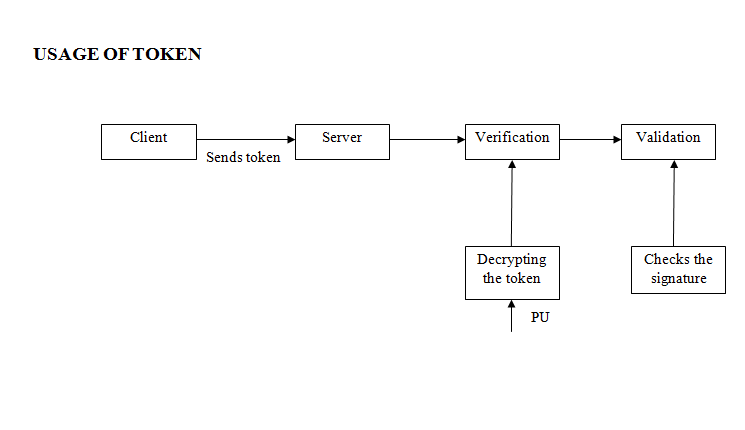
**4.1.1 Purpose**

Systems design will take the requirements and analysis into consideration and come out with a high level and low level design that will form the blue print to the actual solution to the problem in hand. In this dynamic world, analysis and design have to look into making systems that a flexible enough to accommodate changes as they are inevitable in any system. The purpose of this section is to specify the detailed design of the system which provides a smooth way to multiply and accelerate shared enterprise knowledge.

**4.2 ARCHITECTURE DESIGN**



***Fig 4.2.1 Token Creation***

***Fig 4.2.2 Usage of token***

**4.3 FRAMEWORK**

A JSON Web Token, or JWT, is used to send information that can be verified and trusted by means of a digital signature. It comprises a compact and URL-safe JSON object, which is cryptographically signed to verify its authenticity, and which can also be encrypted if the payload contains sensitive information. JWT is mainly used for two purpose and they are authentication and message exchange.

* Authentication: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.
* Information Exchange: JSON Web Tokens are a good way of securely transmitting information between parties, because as they can be signed, for example using public/private key pairs, you can be sure that the senders are who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

STRUCTURE: JSON Web Tokens consist of three parts separated by dots (.), which are:

* Header
* Payload
* Signature

HEADER: The header typically consists of two parts: the type of the token, which is JWT, and the hashing algorithm being used, such as HMAC SHA256 or RSA.

{

"alg": "HS256",

"typ": "JWT"

}

Then, this JSON is Base64Url encoded to form the first part of the JWT.

PAYLOAD: The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional metadata. There are three types of claims: reserved, public, and private claims.

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

1. RESERVED CLAIMS: Claims that are not mandatory whose names are reserved for us. These include:

* iss: The issuer of the token
* sub: The subject of the token
* aud: The audience of the token
* exp: This will probably be the registered claim most often used. This will define the expiration in NumericDate value. The expiration MUST be after the current date/time.
* nbf: Defines the time before which the JWT MUST NOT be accepted for processing
* iat: The time the JWT was issued. Can be used to determine the age of the JWT
* jti: Unique identifier for the JWT. Can be used to prevent the JWT from being replayed. This is helpful for a one time use token.

2. PUBLIC CLAIMS: These are the claims that we create ourselves like user name, information, and other important information.

3. PRIVATE CLAIMS: A producer and consumer may agree to use claim names that are private. These are subject to collision, so use them with caution. The payload is then Base64Url encoded to form the second part of the JSON Web Token.

SIGNATURE: To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

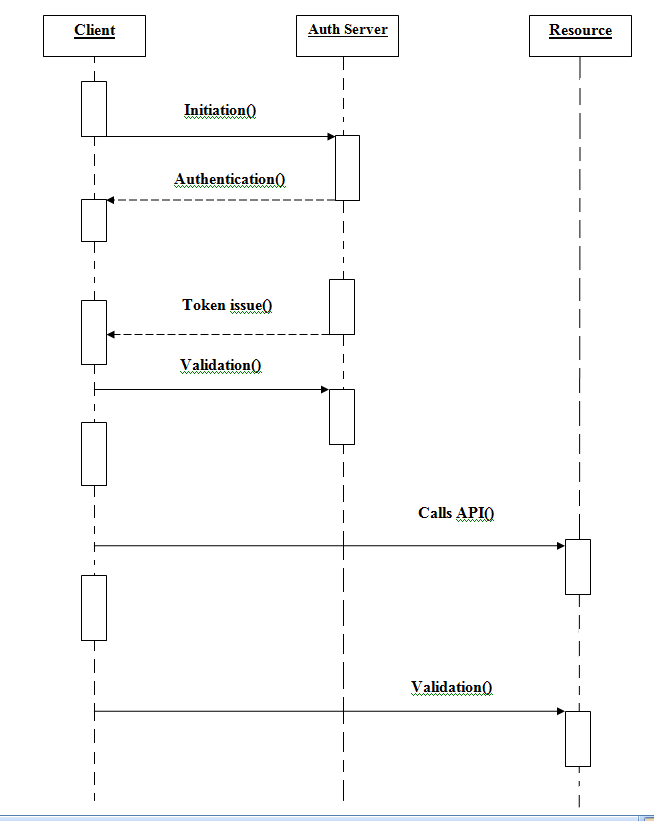
HMACSHA256(base64UrlEncode(header)+"."+base64UrlEncode(payload,secret)The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

PUTTING ALL TOGETHER: The output is three Base64 strings separated by dots that can be easily passed in HTML and HTTP environments, while being more compact when compared to XML-based standards such as SAML. The following shows a JWT that has the previous header and payload encoded, and it is signed with a secret.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9. eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9. TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

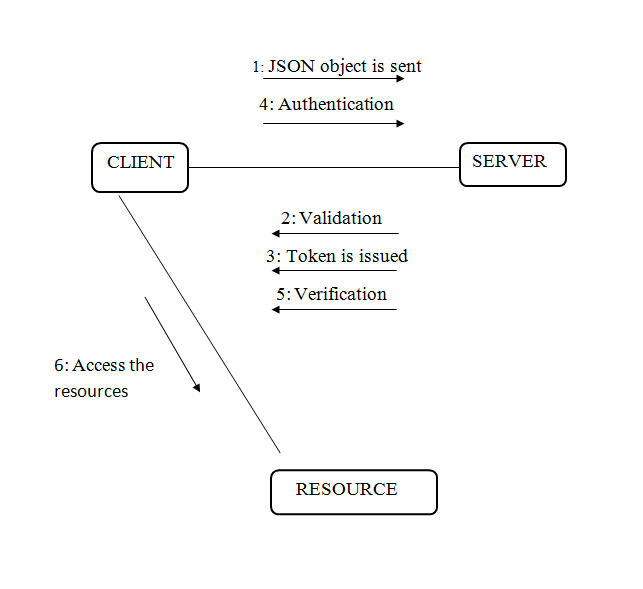
**4.4 DETAILED DESIGN DIAGRAM**

**4.4.1 Sequence Diagram**

A Sequence diagram is an interaction diagram that shows how processes operate with one another and in what order. ****

***Fig 4.4.1 Sequence Diagram***

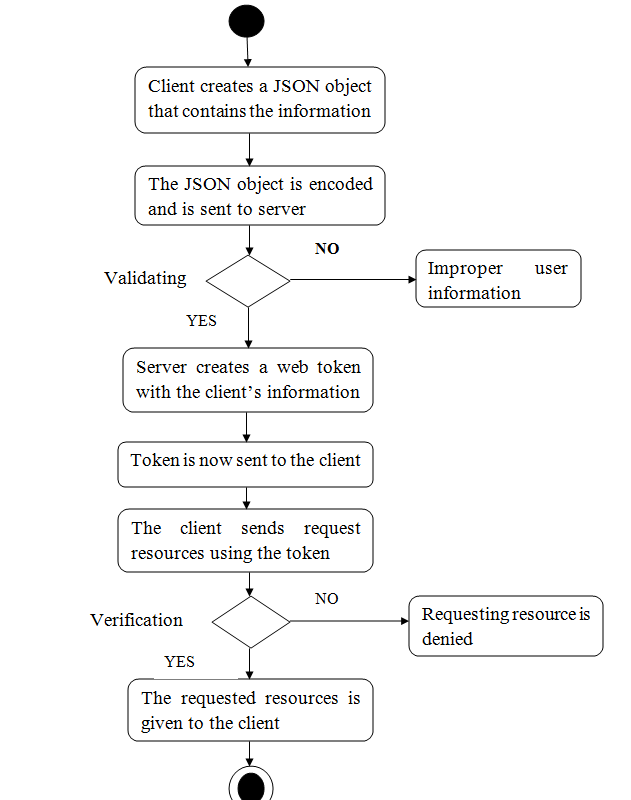
**4.4.2 Collaboration Diagram**

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). ****

***Fig 4.4.2 Collaboration Diagram***

**4.4.3 Activity Diagram**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.

****

***Fig 4.4.3 Activity Diagram***

**CHAPTER5**

**MODULES**

**5.1 INTRODUCTION**

The following are the modules :

* Creation of token
* Verification and Validation

**5.2** **CREATION OF TOKEN**

To create a JWT, the following steps are performed. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps.

1. Create a JWT Claims Set containing the desired claims. Note that whitespace is explicitly allowed in the representation and no canonicalization need be performed before encoding.

2. Let the Message be the octets of the UTF-8 representation of the JWT Claims Set.

1. Create a JOSE Header containing the desired set of Header Parameters. The JWT MUST conform to either the [JWS] or [JWE] specification. Note that whitespace is explicitly allowed in the representation and no canonicalization need be performed before encoding.

4. Depending upon whether the JWT is a JWS or JWE, there are two cases:

* If the JWT is a JWS, create a JWS using the Message as the JWT Payload; all steps specified in [JWS] for creating a JWS must be followed.
* Else, if the JWT is a JWE, create a JWE using the Message as the plaintext for the JWE; all steps specified in [JWE] for creating a JWE MUST be followed.

1. If a nested signing or encryption operation will be performed, let the Message be the JWS or JWE, and return to Step 3, using a content type value of "JWT" in the new JOSE Header created in that step.
2. Otherwise, let the resulting JWT be the JWS or JWE.

**5.3 VERIFICATION AND VALIDATION**

When validating a JWT, the following steps are performed. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps. If any of the listed steps fail, then the JWT MUST be rejected -- that is, treated by the application as an invalid input.

1. Verify that the JWT contains at least one period ('.') character.
2. Let the Encoded JOSE Header be the portion of the JWT before character.

3. Base64url decode the Encoded JOSE Header following the restriction that no line breaks, whitespace, or other additional characters have been used.

1. Verify that the resulting octet sequence is a UTF-8-encoded representation of a completely valid JSON object conforming to RFC 7159 [RFC7159]; let the JOSE Header be this JSON object.
2. Verify that the resulting JOSE Header includes only parameters and values whose syntax and semantics are both understood and supported or that are specified as being ignored when not understood.
3. Determine whether the JWT is a JWS or a JWE using any of the methods described in Section 9 of [JWE].
4. Depending upon whether the JWT is a JWS or JWE, there are two cases:

* If the JWT is a JWS, follow the steps specified in [JWS] for validating a JWS. Let the Message be the result of base64url decoding the JWS Payload.
* Else, if the JWT is a JWE, follow the steps specified in [JWE] for validating a JWE. Let the Message be the resulting plaintext.

1. If the JOSE Header contains a "cty" (content type) value of "JWT", then the Message is a JWT that was the subject of nested signing or encryption operations. In this case, return to Step1, using the Message as the JWT.
2. Otherwise, base64url decode the Message following the restriction that no line breaks, whitespace, or other additional characters have been used.
3. Verify that the resulting octet sequence is a UTF-8-encoded representation of a completely valid JSON object conforming RFC 7159 [RFC7159]. Let the JWT Claims Set be this JSON object.

**CHAPTER6**

**CODING AND TESTING**

**6.1 CODING**

Once the design aspect of the system is finalizes the system enters into the coding and testing phase. The coding phase brings the actual system into action by converting the design of the system into the code in a given programming language. Therefore, a good coding style has to be taken whenever changes are required it easily screwed into the system.

**6.2 CODING STANDARDS**

Coding standards are guidelines to programming that focuses on the physical structure and appearance of the program. They make the code easier to read, understand and maintain. This phase of the system actually implements the blueprint developed during the design phase. The coding specification should be in such a way that any programmer must be able to understand the code and can bring about changes whenever felt necessary.

**6.3 TESTING**

6.3.1 System Testing

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product it is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**6.4 TEST DATA AND OUTPUT**

**6.4.1 Unit Testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly and input produces valid output.

**6.4.2 Integration Testing**

Software integration testing is the incremental integration testing. Integration testing involves building the whole system using the final set of completely-tested individual program components and testing the resultant system for problems that may arise from interactions between the components. To minimize errors and to find the source of error quickly, the incremental approach of adding and testing components is usually followed. The benefits of the incremental integration approach are:

• Errors can be found easily when the number of integrated components is small. This helps in locating and resolving the source of error quickly.

• A new component will be added only if the system with the existing components is completely error-free. In case of a new error, it can be easily attributed to the last added component.

The disadvantage of integration testing is that testing a system feature may require more than one component at a time to be integrated. Testing may find errors between individual components and other parts of the system. Due to this fact, fixing errors can be difficult since it may affect the system functionality as all the components may change.

**6.4.3 White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**6.4.4 Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**CHAPTER7**

**IMPLEMENTATION AND MAINTAINANCE**

**7.1 IMPLEMENTATION PLATFORM**

This implements JSON Web Token based authorization in an organization to ensure secure and safe transmission of the credentials between the client and the server.

**7.2 MAINTENANCE**

System maintenance is that phase of software engineering that helps the user and the developer to maintain the software in an environment required by the user. It is this phase in which any sort of post implementation modifications are made to incorporate the changes that the user needs.

**CHAPTER 8**

**CONCLUSIONS AND FUTURE ENHANCEMENTS**

**8.1 CONCLUSION**

In this project we have proposed a technique to enhance the security of the information traversing in a network between a client and server using a web token and using HMAC algorithm. It ensures safe transfer of the credentials by generating a secret key, that signs the token and checks for the same every time the details are sent to the server.

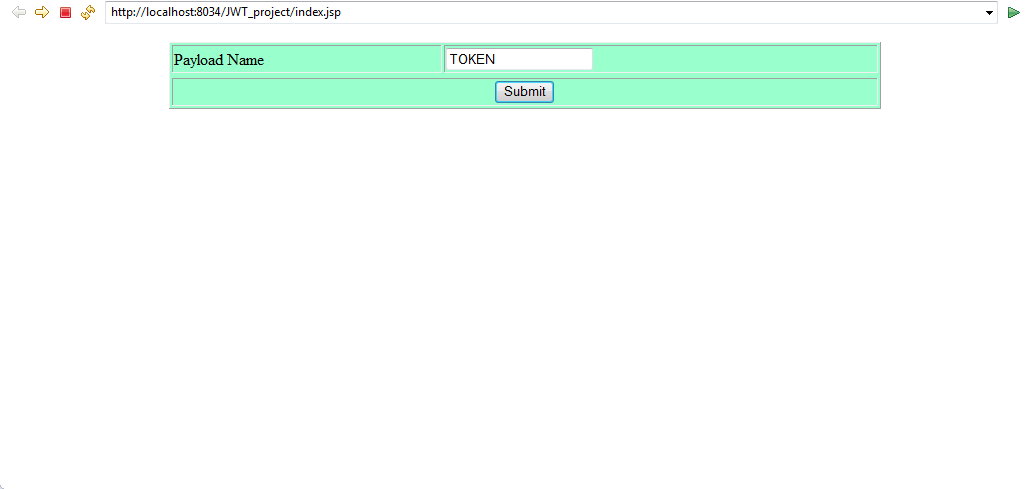
**8.2 FUTURE ENHANCEMENTS**

Future work would be focused on a change to the standard, that could help in preventing the future vulnerabilities, as many libraries are prone to attacks. Libraries treat tokens that holds ‘none’ value in the algorithm field as valid token that results in arbitrary account access on any systems, that can be enhanced by providing a secret key, that fails for the token using ‘none’. Attackers can modify the algorithm field where the server does not have standard algorithm. In order to standardize the algorithm, the server specifies the algorithm that is to be incorporated statically into the verification function.

**APPENDIX 1 – SCREEN SHOTS**

**A.1 INPUT PAGE**

The input page is about entering the payload information of the token that has to be transmitted over the two parties.

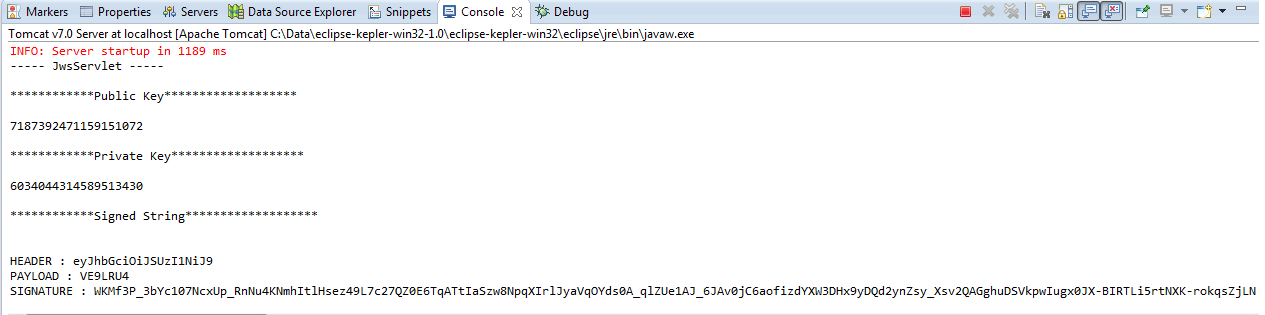
****

***Fig A.1 Input Page***

***Fig A.1 Input Page***

**A.2 TOKEN GENERATION**

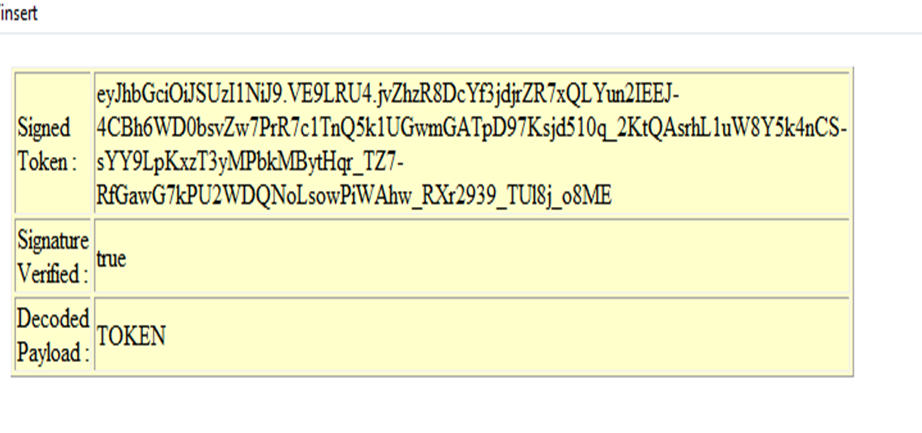
This is the process in the server side where the entire token is generated along with its required contents that include private and primary key.

****

***Fig A.2 Token Generation***

**A.3 OUTPUT PAGE**

The output page therefore displays the token, the decoded payload information that has been sent and also verifies the signature.

****

***Fig A.3 Output Page***

**REFERENCES**

1. Bradley, J. and N.Sakimura (editor), "JSON Simple Sign", September 2010
2. Panzer, J., Ed., Laurie, B., and D. Balfanz, "Magic Signatures", January 2011.
3. Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, May 2015.
4. Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)",RFC 7516, May 2015 MSDN – the Microsoft Developer Network.
5. https://jwt.io/introduction/