****

**Final Report**

**Dependability Analysis of the Solution developed until WA#2**

**Group 6**

**Bruno Vicente Santos (Student no. 44935)**

**Filipe Miguel Santos (Student no. 45411)**

**Rúben André Barreiro (Student no. 42648)**

Abstract

*This summary report is focused on a dependability study (of security, availability and reliability guarantees) as implemented and provided for the final implementation requirements and the observed operations.*

1. **Introduction**

*The* ***NOVA Crypto Banking Service*** *it’s a* ***Banking Service with Cryptocurrencies for New University of Lisbon****. It’s a Byzantine Fault-Tolerant platform, using distributed Consensus Proofs Agreements on all the operations/transactions of the system.*

*This platform it’s implemented in Java, and uses some Frameworks/Libraries such as Node.js and React.js, for Web Front-end, H2 for Back-end Persistent Databases, Spring and RESTful/REST (Representational State Transfer) for Web Services, JSON, JSON Web Tokens (JWT) for Objects’ Serialization, Bft-SMaRt for Byzantine Fault-Tolerant State Machine Replication and Redis’ Cluster Databases, for additional backup mechanisms.*

*The platform also uses HTTPS (HTTP 1.1) using Web connections over TLS Protocol 1.2 (Transport Layer Security Protocol 1.2) for Server-Only Authentication, using self-signed certificates.*

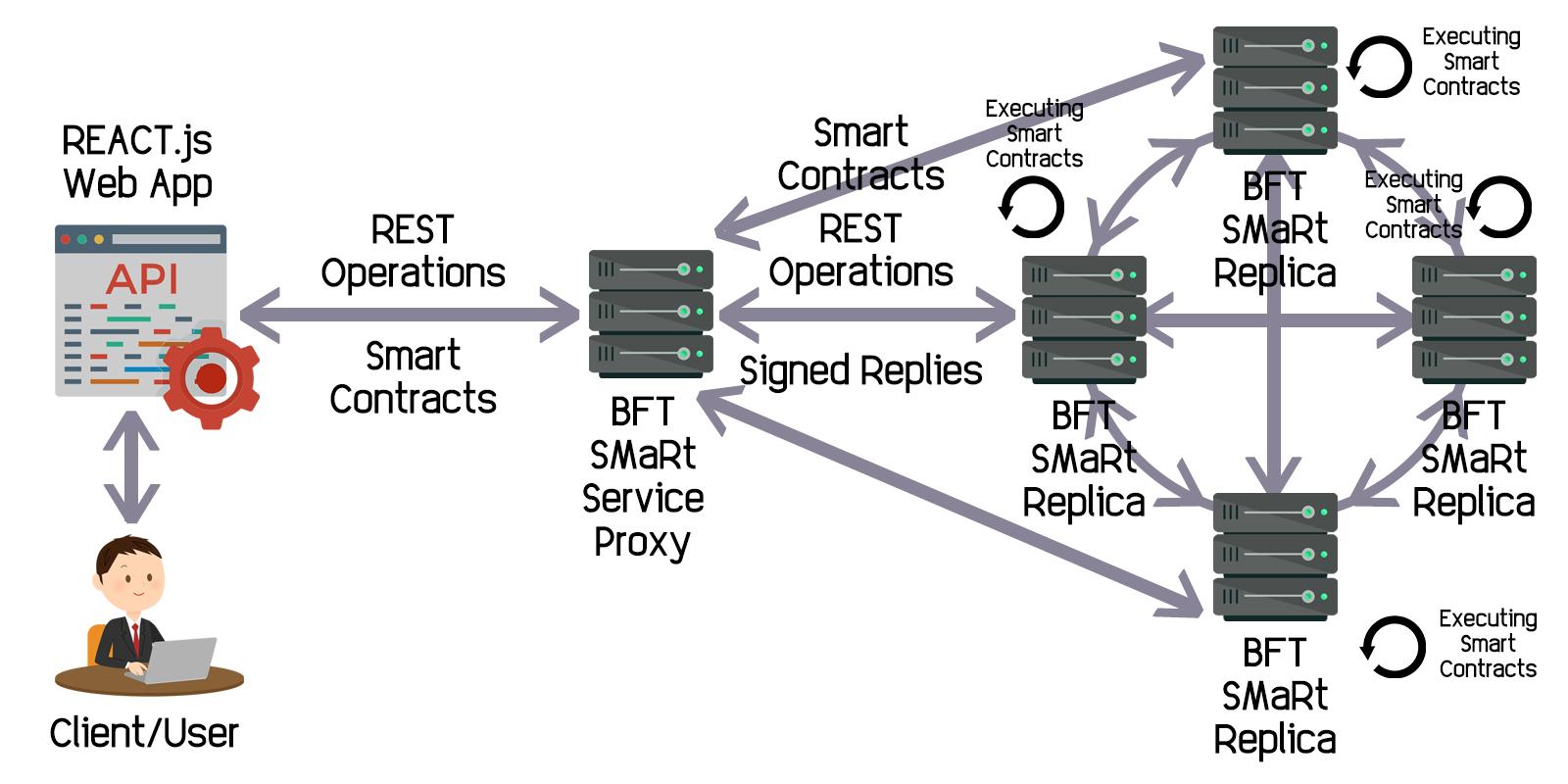
*Our platform it’s intended to support multiple machines (replicas), in a distributed (and decentralized) fashion, even with faulty nodes, guaranteeing the Dependability and Reliability of the System.*

*The platform also supports Smart Contracts for the operations offered on the system’s API, with verification of ByteCode from Java Class Files and Java Policies, using the Java’s Security Manager.*

1. **Description of the System**

*Our platform is composed, mainly, by 3 Components:*

* *User/Client API (React.JS Web App);*
* *Proxy Server (BFT SMaRt Service Proxy);*
* *Replica Server (BFT SMaRt);*

**

* *User/Client API (React.JS Web App):*
  + *This component it’s responsible for the interaction between the Client/User and the Service Proxy;*
  + *It’s an interactive frontend, to be runned in a Web Browser;*
* *Proxy Server (BFT SMaRt Service Proxy):*
  + *This component “acts” similar to a Proxy Server, being responsible for the interaction between the User/Client API and the multiple Server Replicas, ensuring that the data created, provided and managed by the several Replicas are consistent, coherent and correct;*
  + *This components it’s also responsible for analyse the consensus between the several Server Replicas, acting like a Master Server;*

* *Replica Server (BFT SMaRt):*
  + *This component represents a Server Replica, following the notion of S.M.R. (State Machine Replication), where it’s responsible to replicate the data, ensuring that the same data it’s most coherent and correct possible, in comparison to the other Replicas Servers;*

*The operations supported by our platform are:*

* *CreateAuctions (username-creator, auction-id);*
* *CloseAuction (username-closer, auction-id);*
* *ListCurrentOpenAuctions ( );*
* *ListClosedOpenActions ( );*
* *ListBidsFromAuction (auction-id);*
* *ListBidsFromClient (username);*
* *CheckWinnerBidFromClosedAuction (auction-id);*
* *MakeBid (username-bidder, auction-id, amount);*
* *CreateMoney (username, amount);*
* *TransferMoney (username-from, username-to, amount);*

*Additionally, as Backup Service for the Replicas, it was also used a Jedis Database (Redis for Java), improving even more the dependability of the System, ensuring an additional layer of Primary Backup of the data.*

1. **Security auditing and analysis**
   1. **Security properties in End-Client/Service interaction**

*Our platform uses RESTful calls/interactions for Web Services provided to the clients.*

*The RESTful calls/interactions are protected following the security properties provided by the TLS Protocol v1.2 for Server-Only Authentication over all the connection.*

*The TLS Protocol v1.2 offers the following Security Properties:*

* *Confidentiality (with Symmetric Cryptography and Shared Secret/Keys, during the Start of the Session - TLS Handshake);*
* *Peer-Authenticity (from the Server, using Digital Signatures, signed with a Public Key from its Certificate);*
* *Data Origin-Authenticity (using MACs - Message Authentication Codes with Shared Secret/Keys, during the Start of the Session - TLS Handshake);*
* *Data Integrity (using MACs - Message Authentication Codes with Shared Secret/Keys, during the Start of the Session - TLS Handshake);*
  1. **Security settings in TLS for Rest Operations**

*Everytime, every time a client try/attempt to use some Web Service offered by the System, it will try to establish (or reuse) a connection with a Server (this it’s known as a Session).*

*During the establishment of this connection, the client will request and verify the Server’s certificate.*

*This process it’s performed executing the 4 following steps:*

1. *The Digital Signature is checked;*
2. *The Certificate Chain is checked;*
3. *The Expiry and Activation dates and Validity period are checked;*
4. *The revocation status of the certificate is checked;*

*This process it’s performed through the TLSv1.2 with Server-Only Authentication, through the following Messages Exchange Rounds:*

1. ***Client -> Server:***
   * *Client Hello (Without Authentication from Client);*
2. ***Server -> Client:***
   * *Server Hello, Change Cipher Spec, Encrypted Handshake Message (With Authentication from Server);*
3. ***Client -> Server:***
   * *Change Cipher Spec, Encrypted Handshake Message (Without Authentication from Client);*
   1. **Replicated Service and Dependability guarantees**

*For the Replication Service, our System uses the mechanisms provided by the BFT-SMaRt, taking advantage of the notion of S.M.R. (State Machine Replication).*

*The state of the System it’s replicated by multiple Replicas, with the goal of maintaining a consistent and correct state between them, as also, perform the correct operations, even against byzantine faults and attacks.*

*By default, it was used and launched 4 Machines, in the configuration of the BFT-SMaRt, supporting 1 Byzantine Fault.*

*In this case, it’s necessary to have 3f+1 correct responses, at least, to consider a consensus related to an operation, to be correct, where f it’s the number of byzantine faults that the system will tolerate.*

*As in the configuration of our system, it was established that the system will tolerate 1 byzantine fault (f=1), this ensures that the system needs, at least, to have 4 correct replies (3f+1=3x1+1=3+1=4), for each request, to act correctly and as expected.*

*In addition to that, and considering that, the replies can be attacked because of not having security guarantees, it was added a security support in that replies, using Digital Signatures SHA1 with RSA keys.*

*All the Replies sent from the Replicas to the Proxy Server are signed and verified by the Service Proxy, before the operation be performed, indeed.*

*With this security improvement, it’s ensured that the system needs, at least, to have 3 correct replies (2f+1=2x1+1=2+1=3), for each request, to act correctly and as expected.*

*An operation performed, in a consensus, it’s only considered valid, if and only if, all the replicas agree during the consensus process, and if consensus have, at least, 2f+1 correct and valid replies (i.e., with valid digital signatures verifications).*

*For that goal, it was implemented and provided a customised Extractor and Comparator for the Replies Received by the Proxy Service.*

*Our system also uses the notion of Smart Contracts during the performing of the operations, offering an additional security layer on the System’s behaviour.*

*These Smart Contracts need to be loaded by the Clients/Users, in order to guarantee that the operations are executed correctly in all Replicas, and guarantee also the Peer-Authenticity of the Users, due to the Digital Signatures of the Clients/Users.*

*Additionally, these Smart Contracts also verify the Byte Code signatures of the Smart Contracts and the some Java Policies’ Permissions of it, that must be ensured before being runned/executed.*

* 1. **Benchmarking of the System**

*In order to perform some Experimental Tests, it was developed a Script in Python, that simulates 400 clients, performing Transfers of Money between them and computing the time of this simulation, in an environment with no fault and with byzantine faults., following the models of no-fault replication and crash-fault replication.*