arc42 Documentation

1. Introduction and Goals

1.1. Requirements Overview

The driving force behind this project is to pass the subject of Software architecture (https://arquisoft.github.io/) and trying to archive a decent qualification deserving of being looked at by professionals.

Its requirements are to make an on-line chat with solid technology by ourselves and a little help from the corrections our teachers give us.

1.2. Quality Goals

Nr.	Quality	Motivation
1	Understandability	Since this is a qualifiable project it needs to be understandable from a programmer that isn’t part of the development team
2	Efficiency	As a chat it’s important that messages don’t take a long time to reach the other side.
3	Security	A decentralized chat wouldn’t be useful to the users if everyone that wanted to steal information could do it.

1.3. Stakeholders

Role/Name	Contact	Expectations
SOLID team	https://solid.mit.edu/	Their expectations are the same as the teachers'

Role/Name	Contact	Expectations
		expectations, they want an application for PC that uses SOLID and that isn’t centralized.
ARQUISOFT teachers	https://arquisoft.github.io/	Their expectations are the same as the SOLID team’s expectations, they want an application for PC that uses SOLID and that isn’t centralized with the added hurdle of a time limit and a rigorous use of GitHub.
The developer team	this GitHub repository	Since we are trying to pass a subject we can consider that every aspect of this project is of most importance for us.
Individual final users		The final users of the application will want it to be easy to use, for it to go at a decent speed and for their data to remain private.
Groups of users		Groups of users like companies may want conversations to be linked to their web page or to make automated messages in addition to the other requirements.

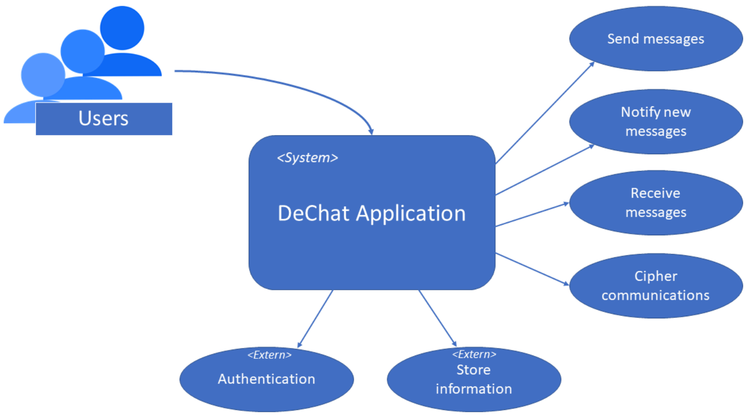
2. Architecture Constraints

Table 1. Arquitecture constraints with explanation

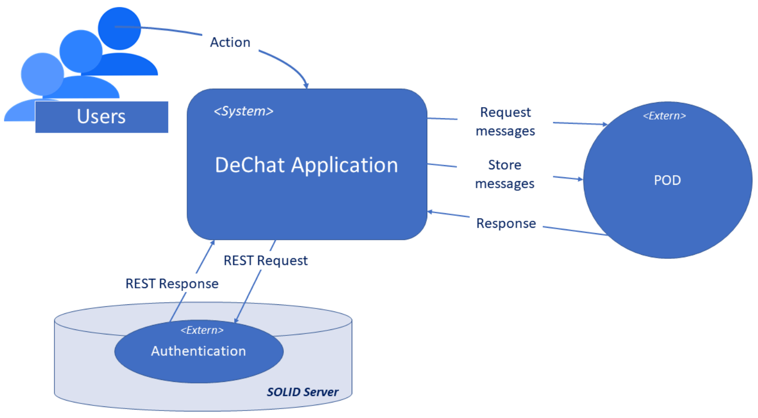
Constraint	Explanation
Decentralization	The first most obvious constraint of the project is the fact that the application has to use decentralized data. That means that the data can not be stored on any server, so each user will have control over their data.
SOLID	The decentralization of the data must be implemented using SOLID, so the data will be stored in personal PODs, and the rest of the people will use a linked data.
Time limit	Another important constraint of the project is the time, the last complete delivery of the project has to be done the week of April 29, 2019 to May 3, 2019. On that date the application must have a state as closing as possible. However, we also have to have a small functional prototype in the week of 12 March, 2019 to 18 March, 2019.
Organization of the project on GitHub	During the development, the project must be monitored using version control. The use of version control is constrained to the existing repository on GitHub, and any type of modification can be made to it.

3. System Scope and Context

3.1. Business Context



3.2. Technical Context



The authentication is made using a request to a SOLID server. The rest of the information (contacts, messages…) are placed in the POD which is in possession of the user.

4. Solution Strategy

4.1. Contents

To achieve a satisfactory solution we've decided to use Angular, because of the many libraries related to SOLID that Angulas has.

4.2. Motivation

These decisions form the cornerstones for our architecture. They are the basis for many other detailed decisions or implementation rules.

4.3. Form

Quality goal	Scenario	Solution approach	Link to Details
Send/ recieve messages	A user sends a message to another user	The receptor gets a link to the message that the sender wanted him to read and access the message with that	TODO
Have a contacts list	A user has a list of all the peolpe they have had communication with	The other's person POD's link is registered and saved for future use	TODO

5. Building Block View

5.1. Whitebox Overall System

TODO

<Overview Diagram>

Motivation

TODO *<text explanation>*

Contained Building Blocks

TODO *<Description of contained building block (black boxes)>*

Important Interfaces

TODO *<Description of important interfaces>*

<Name black box 1>

<Purpose/Responsibility>

<Interface(s)>

<(Optional) Quality/Performance Characteristics>

<(Optional) Directory/File Location>

<(Optional) Fulfilled Requirements>

<(optional) Open Issues/Problems/Risks>

<Name black box 2>

<black box template>

<Name black box n>

<black box template>

<Name interface 1>

…

<Name interface m>

5.2. Level 2

White Box <building block 1>

<white box template>

White Box <building block 2>

<white box template>

…

White Box <building block m>

<white box template>

5.3. Level 3

White Box <_building block x.1_>

<white box template>

White Box <_building block x.2_>

<white box template>

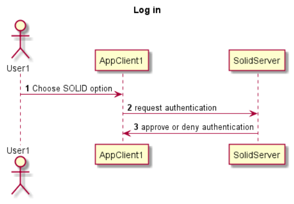
White Box <_building block y.1_>

<white box template>

6. Runtime View

6.1. Log in

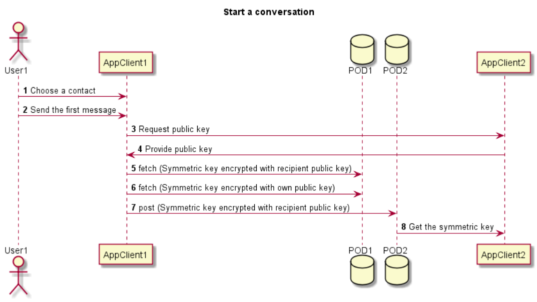
1. The user should choose one of the options provided by SOLID. They are the following: using a local POD or use a POD stored in one of the SOLID servers.
2. Once the user has chosen an option, the SOLID server approve or deny the authentication.
3. Eventually, the user is able to use the chat.



6.2. Start a conversation

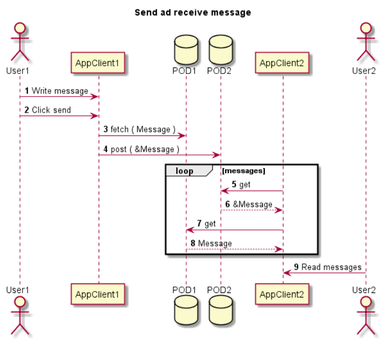
1. The first step is to choose a contact without a started conversation.

2. When it is sent the first message, at first the sender application create a symmetric key.
3. The application of the sender request the public key of the recipient.
4. The symmetric key is encrypted twice using asymmetric keys. It is meant using the public key requested before and the own public key.
5. The symmetric key encrypted with both public keys is stored in the POD of the sender. So finally, the symmetric key is being stored twice with two different public keys.
6. The uri of the key encrypted with the recipient public key is stored in the POD of the recipient who is going to decrypt it using his private key. The sender is going to decrypt the symmetric key using its own private key.
7. Then they are already able to send and receive messages securely.



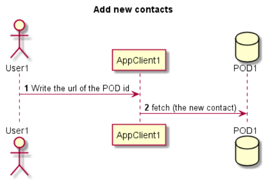
6.3. Send and receive messages

1. The message is encrypted using the symmetric key shared in the first contact.
2. When the message is encrypted, it is stored in the POD of the user who sends the message.
3. The uri is stored in the POD of the recipient using the SOLID api.
4. The recipient is notified about the new message because of the changes produced in his POD.
5. When the recipient application try to access to the content, the message is decrypted using the symmetric key.
6. Finally, the content of the message appears in the recipient application.



6.4. Add new contacts

1. The user has to introduce the url of the POD id of the person who wants to chat with.
2. Finally the contact is stored in the POD of the user who add the other one.



7. Deployment View

7.1. Infrastructure Level 1

TODO

<Overview Diagram>

Motivation

<explanation in text form>

Quality and/or Performance Features

<explanation in text form>

Mapping of Building Blocks to Infrastructure

<description of the mapping>

7.2. Infrastructure Level 2

TODO

<Infrastructure Element 1>

<diagram + explanation>

<Infrastructure Element 2>

<diagram + explanation>

…

<Infrastructure Element n>

<diagram + explanation>

8. Cross-cutting Concepts

8.1. <Concept 1>

TODO *<explanation>*

8.2. <Concept 2>

TODO *<explanation>*

…

8.3. <Concept n>

TODO *<explanation>*

9. Design Decisions

Table 2. Architectural Decisions

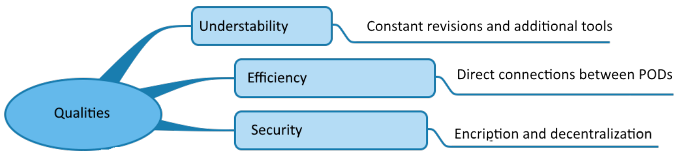
Aspecto	Descripción	Elección	Razonamiento
Programming Language	The language used to develop the app is a critical decision due to the fact that	TypeScript	The main reason why we have chosen this language is the large quantity of

Aspecto	Descripción	Elección	Razonamiento
	it's not possible to change it in the future without causing too much problems. As experience tells us, completely changing the programming language once started is expensive and can cause a project to fail		documentation and libraries available. While we were researching about SOLID, we realized that the vast majority of the information is related to TypeScript and JavaScript
Framework	Another strength point of a project is the capability of being able to use a framework. A framework used in the good way reduces the time dedicated to code and design. The worst part is the fact of learning how to use it and to know its scope	Angular	We decided to use Angular because of it's a framework with a high compatibility with the SOLID project, it's oriented to agile methodologies and it's easy to develop graphic interfaces using just 1 window with differents fragments which change in the time
Database	The databases are an important utility to save the data, but at the beginning of the development we consider whether to use a database that stores the messages, and this is stored in the POD, or to use the POD for storage	Do not use a database	You could use a database to store the messages, and then store the database in the SOLID POD, but we believe that this is done twice the work that is not necessary, and also could cause problems to read messages when the other user is not connected
Query library	The data stored in the pod is in RDF format. To facilitate obtaining	RDflex	We considered RDflex and Comunica alternatives, but finally we opted for RDflex. The reasons are

Aspecto	Descripción	Elección	Razonamiento
	information from them we can use a library		that RDflex has a simpler syntax to understand, and it is also developed by the team that develops SOLID

10. Quality Scenarios

10.1. Quality Tree


Qualities
Understability
Constant revisions and additional tools
Efficiency
Direct connections between PODs
Security
Encription and decentralization

10.2. Evaluation Scenarios

TODO since we haven’t decided what to use to tet the code

11. Risks and Technical Debts

TODO

12. Glossary

TODO

Term	Definition
<Term-1>	<definition-1>
<Term-2>	<definition-2>