DeChat

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.

The requirements are to make a SOLID based on-line chat by ourselves.

1.2. Quality Goals

Nr.	Quality	Motivation
1	Understandability	It needs to be understandable from a programmer that doesn't take part of the development team.
2	Domain based	We focus the application around the model without matters the technologies used.
3	Single Responsability	Due to the nature of the project, it must be easy to make changes on it. It can be reached with the single responsibility principle.
4	Efficiency	As a chat it's important that messages don't take a long time to reach the other side.
5	Security	A decentralized chat wouldn't be useful to the users if everyone could access to other's chats.

Nr.	Quality	Motivation
6	Sturdiness	Since this is an important project it needs to be strong, that's why we test the code and prove it with different users.

1.3. Stakeholders

Role/Name	Contact	Expectations
SOLID team	https://solid.mit.edu/	Their expectations are the same as the teachers' expectations, they want an application that uses SOLID principles. This means to be decentralized.
ARQUISOFT teachers	https://arquisoft.github.io/	Their expectations are the same as the SOLID team's expectations, they want an application that uses SOLID with the added hurdle of having a limited time 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 important to us.
Individual final users		The final users of the application are looking for one easy to use and a good response speed.
Groups of users		Groups of users want to communicate among them to share their experiences.

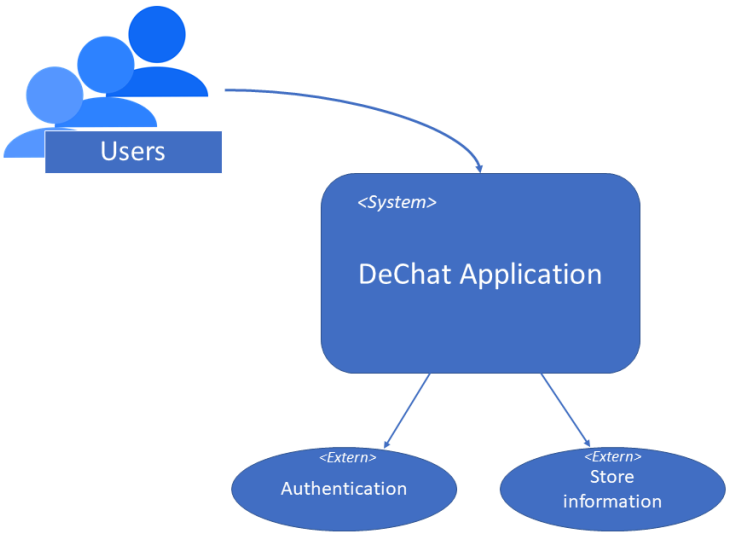
2. Architecture Constraints

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 of the application, so each user will have control over their data and where to place it.
SOLID	The decentralization of the data must be implemented using SOLID, so the data will be stored in personal PODs using 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 need 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.

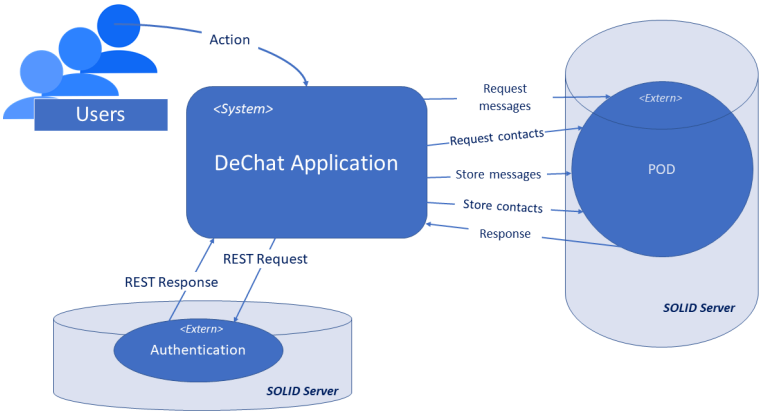
3. System Scope and Context

3.1. Business Context



Each figure marked as "Extern" mean to be an external resource. The label "System" represents the application and the users' label the people who use it.

3.2. Technical Context



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

We use the labels "System", "Extern" and "Users" again as we have described in the section before. Moreover, we use a kind of cylinders which represent the SOLID server. The external resource "POD" is the store of the user information.

4. Solution Strategy

4.1. Contents

To achieve a satisfactory solution we have decided to use Angular, because of the many libraries related to SOLID that exist in javascript and the amount of documentation available to use this framework.

4.2. Motivation

These decisions are the hot spot of our architecture. They are the base of many other decisions and implementation rules.

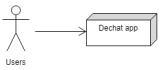
4.3. Form

Quality goal	**Scenario**	**Solution approach**
Send messages	A user sends a message to another user	The sender write the message in its POD and inform the receiver using the inbox of the receiver’s POD.
Receive messages	A user receive messages from another user	The receptor access to the sender’s POD and gets the list of messages that the sender want him to read.
Add contacts	A user add a new contact	The user write the uri of the other user’s POD and a nickname to store it.
Read contacts	A user wants to see its contacts to chat with	The user has a list of all the people it has added to its POD. This list of

Quality goal	Scenario	Solution approach
		people is stored in the own POD as URIs and nicknames.

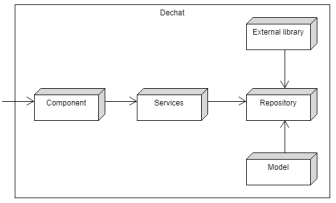
5. Building Block View

5.1. Scope & Context



- As the app is independent from an external server (since the only one it uses is the one that the app is deployed in, the contact's notwithstanding) we can present the app this way. The users will interact with other users without a server in the middle so they only have to worry about the connection among themselves

5.2. Level 1



- Repository is responsible of:

```
+ Adding messages to the POD.
```

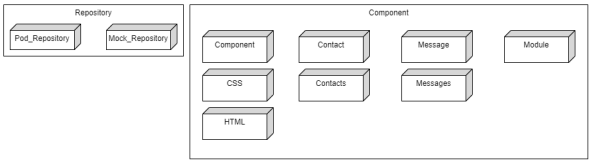
```
+ Gets the Contacts of the user.
```

```
+ Gets the messages between the user and their contacts.
```

```
+ Login out the user.
```

- Services can be divided into: account, contact, message and login.
- From the external libraries that are used the most remarkable among them are N3 for RDF and Solid-file-client to manage the data.

5.3. Level 2

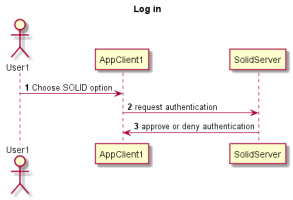


- Mock_Repository is a Repository spacifically made to test the aplication in a local enviroment while Pod_Repository is for online purposes.
- Within Component we find Component.ts (Component), Component.html (HTML), Component.css (CSS) along with Message and Messages, Contact and Contacts and Module.

6. Runtime View

6.1. Log in

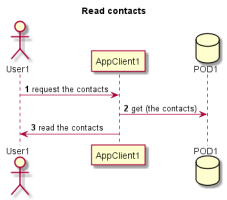
1. The user should choose one of the options provided by SOLID. They are the different SOLID's servers available.
2. Once the user has chosen an option, the SOLID server approve or deny the authentication.
3. Eventually, the user is able to use the chat.



6.2. Read contacts

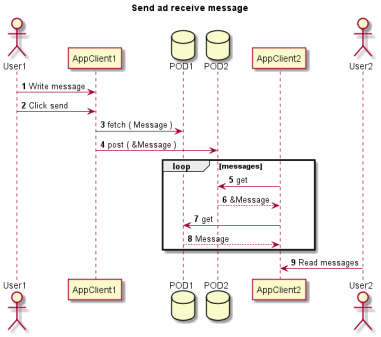
1. The user request to load the contacts stored.

2. The application access to the own POD and it loads the contacts.



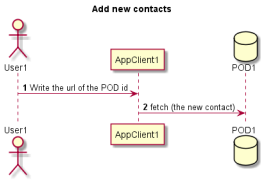
6.3. Send and receive messages

1. The sender write a new message and click send.
2. The sender's application stored it in the POD of the user who sends the message.
3. The sender's application write a notification in the inbox of the receiver's POD.
4. The receiver's application realise that it has new messages.
5. The receiver is able to see the messages and answer to them if it's wanted.



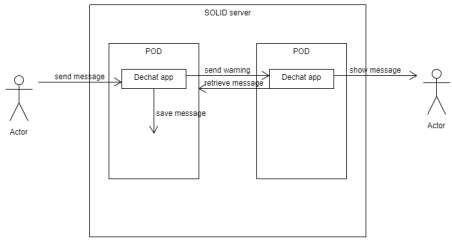
6.4. Add new contacts

1. The user has to introduce the uri of the POD of the person who wants to chat with and a nickname if it's wanted to identify the contact. If it's not specified the identification of the contact will be the uri of the POD. Now it's not able to change the nickname, you need to remove and add the contact again.
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



The infrastructure only has 2 hardware per user: the user's computer with which it access his POD and said POD that rests on SOLID's servers; SOLID's servers are a kind of server that store the PODs which can be configured by anyone, based on certain specifications, so everyone can decide where to store their POD.

While the development of the application is runnign the deployment is easy, it only needs to be served on the user's computer and to reach the user's POD. As long as SOLID's servers are online and the POD isn't broken then the application will start to work.

On the other side, while testing, the application will stay local.

Requirments and OS

It's mandatory, in order to run the application, that Node.js and angular/cli are installed on the computer, at least at this point in time. As for the OS, it's irrelevant, the application is made to work in every OS, that's to say, it's multiplataform.

7.2. Infrastructure Level 2

It must be said that even if PODs could be in the same server, they could also be in totally diferent SOLID servers. Even so, that doesn't change the way the

application works since it doesn’t affect the deployment nor the conexion between PODs. The app gets deployed in a computer by using "ng serve" on the cmd while on the directory of the app, and form there the user logs in and connects to other PODs regardless of their location.

8. Cross-cutting Concepts

8.1. Maintain an independent architecture

One of the goals and rules of the development team is to keep the application within the confines of a MVC architecture with the intent of eliminating dependencies; to begin with, it’s a matter of cleanliness, since the code will be easier to understand the less meshed together it is. Another thought that prompted us to take this stand is the "what if" of changing some part of the application (like Angular to another framework or the POD for a database).

8.2. Using Angular and TypeScript

As we studied the project we came to the conclusion that Angular was the most complete framework for what we meant to do, for the most part because it had libraries compatible with SOLID and most apps for SOLID were made with Angular. As Angular uses nodejs it’s mandatory that we use TypeScript, and since TypeScript can be compiled into Javascript, it was made possible to use libraries written for both TypeScript and JavaScript.

8.3. Why SOLID

From the very beginning of this project, SOLID was a mandatory part of what this application had to be, so thinking about why became trivial.

9. Design Decisions

Table 2. Architectural Decisions

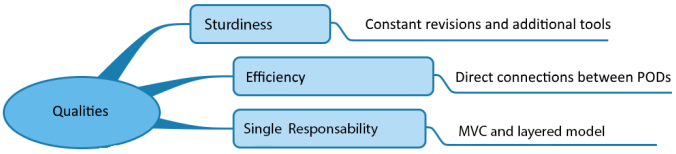
Aspect	Description	Choice	Explanation
Programming language	The language used to develop the app is a critical decision due to the fact that it’s not possible to change it	TypeScript	The main reason why we have chosen this language is the large quantity of documentation and libraries

Aspect	Description	Choice	Explanation
	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.		available. While we were researching about SOLID, we realized that the vast majority of the information is related to TypeScript and JavaScript, and the first one compile to javascript, so we can use javascript's libraries too.
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, that's why at the beginning of the development we consider whether to use a database that stores the messages, which are stored in the POD through the database, or to use the POD directly.	Use just the POD	The fact of use a database to store the messages and then store the database in the SOLID POD can overload the system doing the same operation (just store data) twice, that's why we decided to write them into the POD directly.
RDF library	The data stored in the PODs is in RDF format. To facilitate obtaining and	N3	We considered N3 and simpleRDF among others, but finally we decided to use N3 because is the base

Aspect	Description	Choice	Explanation
	writing information from them we can use a library.		library used by another libraries and we considered simpler to use than the others apart from the fact that N3 allow more operations than simpleRDF for example.
Architectural model	We have thought about different architectural models that allow us to achieve the principles of single responsibility, open-closed principle and dependency inversion principle mainly.	Layered Architecture and MVC	We finally decided to use the layered architecture and the model view controller (MVC) because we are familiarized with them and they provide us what we were looking for.

10. Quality Scenarios

10.1. Quality Tree



10.2. Quality Scenarios

- As a message from a contact arrives:

```
+ The app should show it if the user is on the conversation.
```

```
+ The app should alert the user that a new message has arrived if
 they are not on the conversation.
```

- As the user adds another user as a contact:

```
+ The app should open a conversation with said contact to begin to
 send messages.
```

11. Risks and Technical Debts

Description	Solution
Since we use the POD technology we are subject to its changes and bugs, wich can end up making the user lose information due to a bug unrelated to the app.	Unfortunately there is little we can do without interceding in Solid's development.
Because we have to depend on Solid's confusing interface it may lead to authentication problems due to the different authentication ways Solid has which can confuse an amateur user of Solid.	This point is easily supressed once the user gets used to interacting with solid, however, to ensure that it doesn't cause problems, we've made our own interface where the user can choose its provider of PODs and then access.
As every application on SOLID, once the user has verified its identity, this application, if something goes wrong, can make undesired changes, corrupt the system or touch data that other application may be using, causing it to malfunction	As this is an inherent problem of the POD architecture it isn't possible for us to solve this issue, at most we can try to limit the data that the application may try to touch in hopes that it doesn't damage anything.
Related to the previous point, privacy is at risk as well. Since the POD's data can be read by every application you use it with then your conversations can be read by every other app.	Unfortunately we couldn't come up with a solution to this one, since using a key to encrypt the data wouldn't work because the place where you save the key is also public to the other applications.
A notable deficiency is that, as SOLID is implemented right now, gathering	No solution has been found as of now.

Description	Solution
information is not as fast as it would be desirable	
Another deficiency could be the small number of libraries of RDF, and their resistance to being used, making the code messy	With time other libraries will, probably, appear, and this deficiency will be solved.
If a user takes down its POD then it would leave any user that had a conversation with the first with only half the conversation.	It’s the other side of the decentralization, as information remains in each POD it makes sense that, since a POD is no longer there then the information isn’t either.
If a POD is temporarily inaccessible, due to a failure in the server or faulty connection, then users would only be able to see their part of the conversation with the faulty POD’s user	A downside of decentralization.
Since the communications are done via API REST they could be inercepted and modified as the POD doesn’t have a verification system	A current error of SOLID.

12. Glossary

Term	Definition
POD	It’s a personal online data stores hosted wherever the user desires, as long as it’s a SOLID server.
SOLID server	A server built on the Solid platform following SOLID specifications.
SOLID	Solid (derived from "social linked data") is a proposed set of conventions and tools for building

Term	Definition
	decentralized social applications based on Linked Data principles. Solid is modular and extensible. It relies as much as possible on existing W3C standards and protocols.
ARQUISOFT	It refers to the organization made by the teachers of Software architecture to group together all the teams trying to make a SOLID chat.
Version control	It’s the management of changes to documents and code of the application by identifying changes with names and codes.
DeChat	It’s the shortened version of "Decentralized chat"
Linked Data	Linked Data is a method of publishing structured data so that it can be interlinked and become more useful through semantic queries.
Semantic Queries	Semantic queries are queries that enable the retrieval of both explicitly and implicitly derived information based on syntactic, semantic and structural information contained in data.
RDF	The RDF data model is similar to classical conceptual modeling approaches (such as entity-relationship or class diagrams). It is based on the idea of making statements about resources (in particular web resources) in expressions of the form subject-

Term	Definition
	predicate-object. The subject denotes the resource, and the predicate denotes traits or aspects of the resource, and expresses a relationship between the subject and the object.
Decentralization	Decentralization is the process by which the activities of an application, particularly those regarding planning, decision making and data, are distributed or delegated away from a central, authoritative location or group.
Layered architecture	It’s an architecture that is organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic).
MVC	Model View Controller (MVC) is a style of architecture that separates the data from an application (Model), the user interface (View) and the control logic (Controller) within 3 separate components.
Angular	It’s a TypeScript-based open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations.
TypeScript	TypeScript is an open-source programming language developed and maintained by Microsoft and a superset of ECMAScript 6 (ES6),

Term	Definition
	which is also backwards compatible with ECMAScript 5 (JavaScript).
Node	Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. Since TypeScript is backwards compatible with JavaScript, it also works with Node.
API	An API (application programming interface) is a set of subroutine definitions, communication protocols, and tools for building software.
REST	REST (Representational State Transfer) is a software architectural style that defines a set of constraints to be used for creating Web services. There is also the API Rest which is an API that implements the REST architecture.
Web Petition	It’s, in a client-server computing model (sort of what we are using), a request from the client side (the app) to the server (the POD).
Mock up	mockups are a way of designing user interfaces on paper or in computer images. A software mockup will thus look like the real thing, but will not do useful work beyond what the user sees.
Uri	A Uniform Resource Identifier (URI) is a string of characters that

Term	Definition
	unambiguously identifies a particular resource.
Cross-cutting Concepts	They are rules, principles or other decisions, guidelines, processes that influence one or more elements of the architecture.
Principle of single responsibility	it’s a computer programming principle that states that every module, class, or function should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class.
Open-closed principle	it’s a computer programming principle that states that classes should be open for extension but closed to modification.
Dependency inversion principle	it’s a computer programming principle that states that high-level modules should not depend on low-level modules, both should depend on abstractions (interfaces), an that abstractions should not depend on details. Details (classes) should depend on abstractions.
Bug	An error that can, and normally will, cause damage to your data.
Card	It’s the location inside the POD where the user’s information is stored.
Inbox	It’s the folder in the POD where everyone else can write but can’t read or modify; this is used for notifications.

Term	Definition
Karma	It’s a tool which spawns a web server that executes source code against test code for each of the browsers connected.
Cucumber	It’s a tool used to make end-to-end tests; this kind of tests are used test the entire application in one go.
Selenium	It’s a portable framework for testing web applications. In our case we need it as a driver to use Cucumber.
Protractor	It’s an end-to-end test framework for Angular and AngularJS applications. Same as Selenium, we need it as a driver for Cucumber to work.
Testing	It’s a practice which consists on trying to run the application to confirm that it works when it has to work, and that it doesn’t when it shouldn’t.