[arc42] deChat - Decentralized Chat 0

NOTE	This version of the template contains some help and explanations. It is used for familiarization with arc42 and the understanding of the concepts. For documentation of your own system you use better the *plain* version.

1. Introduction and Goals

The development will consist of a WebApp whose objective is to be a decentralized chat, using SOLID technologies. This chat will allow users to exchange messages in different formats (audio, text, video), as well as other functions such as: share screen or video call; It will also allow you to save friends lists, show notifications, among other functions. It will be a web application developed at Angular, and an RDF library will be used to communicate and obtain information on the different PODs

1.1. Requirements Overview

The application will mainly comply with the following:

- The system will be based on a decentralized architecture where data storage is separated from the app
- Users can store their chat data in their own pods
- The app will allow a user to share pictures, videos or other kinds of files with other friends through the chat
- A user can get notifications when some friend wants to chat with him
- Users can have groups of friends with whom they may want to chat
- It will be possible to have group chats where all members receive the messages
- Users can have live calls and videos
- Users can share their screens in the chat

1.2. Quality Goals

The main Stakeholders of this application are Inrupt and users, I will list the most important quality goals for them:

- Usability : Ease of use by non-technical people.
- Responsiveness : The application must send messages in real time.
- Privacy: The application must keep the data private (this is what solid is based on, each one owns their data).
- Security: The application must be secure, users must know who they are talking to and who has access to the chat.

1.3. Stakeholders

Stakeholders of the system:

- Developers
- Professor
- Users
- Inrupt

Role/Name	Contact	Expectations
Developer	Laura Sanchez	Have to work with the architecture or with code, documentation of the architecture,have to come up with decisions about the system or its development
Developer	Enrique Fernandez	Have to work with the architecture or with code, documentation of the architecture,have to come up with decisions about the system or its development
Developer	Jose Luis Bugallo	Have to work with the architecture or with code, documentation of the architecture,have to come up with decisions about the system or its development
Developer	Victor Manuel Chaves	Have to work with the architecture or with code, documentation of the architecture,have to come up with decisions about the system or its development
Developer	Carlos Gomez	Have to work with the architecture or with code, documentation of the architecture,have to come up with decisions about the system or its development
Developer	Daniel Fernandez	Have to work with the architecture or with code, documentation of the architecture,have to come up with decisions about the system or its development
Professor	Jose Emilio Labra	Evaluate and guide the project
Users		Users can chat,they can have live calls and videos and they can share their screens in the chat
Inrupt	Inrupt	They has been proposed as a challenge

Architecture Constraints

Doc Writer <<UO244741@uniovi.es>> v1.0, 2019-08-02

There are a lot of constraints that makes us work harder to make a good application.

1. Technical Constraints

Code	Restricction	Motive
TC001	Program language: JavaScrip or TypeScript	We have chosen this language, but we don't have enough knowledge about this languages.
TC002	Frameworks: Angular	We haven´t worked with this framework before, so we will have to learn how to use it.
TC003	Online independence: no central control of data	Linked-data are completely decentralized and fully under users' control rather than controlled by other entities.

2. Organizational Constraints

Code	Restricction	Motive
OC001	About Version Control: Git, Tool: Github	Keep a version of the project after making changes
OC002	6 person group	We didn't know each other
OC001	Time schedule	We don't have a lot of time for developing the app. Furthermore, we are developing other proyects at the same time.

3. Conventions Constraints

Code	Restricction	Motive
CC001	Language	English. The project and the corresponding blog targets an international audience, so English should be used throughout the whole project.
OC002	6 person group	We didn't know each other

Code	Restricction	Motive
OC001	Time schedule	We don't have a lot of time for developing the app. Furthermore, we are developing other proyects at the same time.

4. System Scope and Context

The project that we are going to develop aims to communicate two people through different mechanisms:

- **Text messages.**

- **Images.**

- **Videos.**

- **Video calls.**

However, the approach that the application will take it´s around permits. A user can not send a file of any kind or speak with a person if they do not have explicit permission from the recipient.

But the most important point of the chat is not the permissions, but the decentralization. We want only users to have access to the content of those conversations, so that they are the owners of their personal information.

To achieve this goal we will use SOLID. SOLID is a project that consists of a decentralized website. It deals with establishing a connection between several users, separating data application. To establish this connection, we will use what is known as POD. A POD is a unique profile of each user, which allows to store information on the web about it.

We will use these PODs to connect both users, but the data they exchange will be stored in the interlocutors' devices. In this way, we will create a private chat that will have the advantage of being isolated, reliable and safe.

4.1. Business Context

- Communication elements:
  - **Users:** Users are the main recipient of the application.
  - **Web:** Users will connect to the web through their PODs to establish secure communication.
- Communication process:
  - The partners create a POD.
  - The user adds as friend through the PODs to the second interlocutor.
  - The interlocutors log in with their POD in the chat.
  - Ready!

What data are exchanged with the environment? Through the website, users will enter some small data on the SOLID website to obtain their POD. Once established the connection between both interlocutors, they will be the only owners and holders of the information they exchange.

Communication Partner	Input	Output
User	Chat, video, images, POD information	Chat, video, images, POD information
Web		POD, Server Connection

4.2. Technical Context

As we have said before, we will use the so-called PODs provided by SOLID to connect to the server of the same, and through it, communicate with other people.

For the design of the connection we will use different mechanisms:

- **RDF:** This library allows us to establish the connection. Act as a communicator and thanks to it we can obtain the POD information necessary to establish communication.
- **Authenticator:** This tool receives the POD of a user and allows us to know if it is a valid POD.
- **Angular:** It is a JavaScript framework that will allow us to handle the language in a more comfortable way.

[03 System Scope and Context] | *images/03_System_Scope_and_Context.JPG*

5. Solution Strategy

5.1. Technology and IDE

The team had to choose between betweet new technologies for us and technologies already kown but that dd no have as much suport to SOLID. Based on what we knew, our first option was to choose Java as programming language, but after reading the SOLID documentation, we decided to choose JavaScript, more specifically, the Angular Framework.

About the IDE, we choose WebStorm due to its power and the fact that, as students, we have a free payment license. In addition, some of the members of the group have used it in previous subjects.

5.2. Architectural pattern

We decided to use the architectural pattern model-view-controller (MVC) because it is a pattern that gives us more extensibility and maintainability. Also, its implementation supports rapid and parallel development. For more information visit: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

5.3. Methodology of work

We will use Scrum to develope this project. Scrum can provide us an early software development, and more independence when programming. Our project will start implementing the basic functions of a chat and it will grow in complexity (https://en.wikipedia.org/wiki/Scrum_(software_development))

6. Building Block View

Content

The building block view shows the static decomposition of the system into building blocks (modules, components, subsystems, classes, interfaces, packages, libraries, frameworks, layers, partitions, tiers, functions, macros, operations, datas structures, …) as well as their dependencies (relationships, associations, …)

This view is mandatory for every architecture documentation. In analogy to a house this is the *floor plan*.

Motivation

Maintain an overview of your source code by making its structure understandable through abstraction.

This allows you to communicate with your stakeholder on an abstract level without disclosing implementation details.

Form

The building block view is a hierarchical collection of black boxes and white boxes (see figure below) and their descriptions.
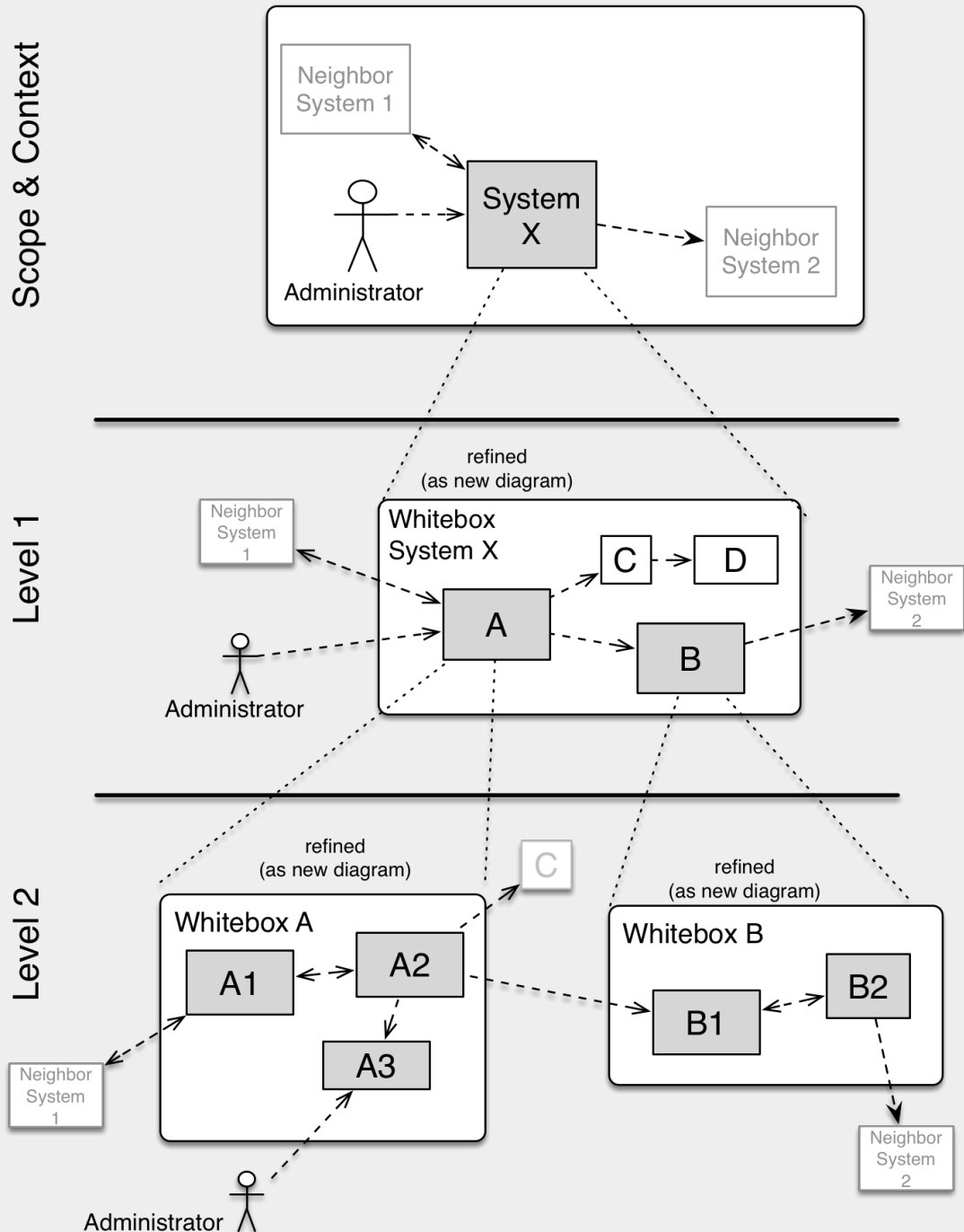
Level 1 is the white box description of the overall system together with black box descriptions of all contained building blocks.

Level 2 zooms into some building blocks of level 1. Thus it contains the white box description of selected building blocks of level 1, together with black box descriptions of their internal building blocks.

Level 3 zooms into selected building blocks of level 2, and so on.

6.1. Whitebox Overall System

\<Overview Diagram\>

Motivation

\<text explanation\>

Contained Building Blocks

\<Description of contained building block (black boxes)\>

Important Interfaces

\<Description of important interfaces\>

6.1.1. <Name black box 1>

Here you describe <black box 1> according the the following black box template:

- Purpose/Responsibility

- Interface(s), when they are not extracted as separate paragraphs. This interfaces may include qualities and performance characteristics.

- (Optional) Quality-/Performance characteristics of the black box, e.g.availability, run time behavior, ...

- (Optional) directory/file location

- (Optional) Fulfilled requirements (if you need traceability to requirements).

- (Optional) Open issues/problems/risks

<Purpose/Responsibility>

<Interface(s)>

<(Optional) Quality/Performance Characteristics>

<(Optional) Directory/File Location>

<(Optional) Fulfilled Requirements>

<(optional) Open Issues/Problems/Risks>

6.1.2. <Name black box 2>

<black box template>

6.1.3. <Name black box n>

<black box template>

6.1.4. <Name interface 1>

...

6.1.5. <Name interface m>

6.2. Level 2

> Here you can specify the inner structure of (some) building blocks from level 1 as white boxes.
>
> You have to decide which building blocks of your system are important enough to justify such a detailed description. Please prefer relevance over completeness. Specify important, surprising, risky, complex or volatile building blocks. Leave out normal, simple, boring or standardized parts of your system

6.2.1. White Box *<building block 1>*

> ...describes the internal structure of *building block 1*.

<white box template>

6.2.2. White Box *<building block 2>*

<white box template>

...

6.2.3. White Box *<building block m>*

<white box template>

6.3. Level 3

> Here you can specify the inner structure of (some) building blocks from level 2 as white boxes.
>
> When you need more detailed levels of your architecture please copy this part of arc42 for additional levels.

6.3.1. White Box <_building block x.1_>

> Specifies the internal structure of *building block x.1*.

<white box template>

6.3.2. White Box <_building block x.2_>

<white box template>

6.3.3. White Box <_building block y.1_>

<white box template>

7. Runtime View

7.1. <Runtime Scenario 1>

- *<insert runtime diagram or textual description of the scenario>*
- *<insert description of the notable aspects of the interactions between the building block instances depicted in this diagram.>*

7.2. <Runtime Scenario 2>

7.3. ...

7.4. \<Runtime Scenario n\>

8. Deployment View

8.1. Infrastructure Level 1

Describe (usually in a combination of diagrams, tables, and text):

- the distribution of your system to multiple locations, environments, computers, processors, .. as well as the physical connections between them
- important justification or motivation for this deployment structure
- Quality and/or performance features of the infrastructure
- the mapping of software artifacts to elements of the infrastructure

For multiple environments or alternative deployments please copy that section of arc42 for all relevant environments.

<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>

8.2. Infrastructure Level 2

Here you can include the internal structure of (some) infrastructure elements from level 1.

Please copy the structure from level 1 for each selected element.

8.2.1. *<Infrastructure Element 1>*

<diagram + explanation>

8.2.2. *<Infrastructure Element 2>*

<diagram + explanation>

...

8.2.3. *<Infrastructure Element n>*

<diagram + explanation>

9. Cross-cutting Concepts

Content

This section describes overall, principal regulations and solution ideas that are relevant in multiple parts (= cross-cutting) of your system. Such concepts are often related to multiple building blocks. They can include many different topics, such as

- domain models

- architecture patterns or design patterns

- rules for using specific technology

- principal, often technical decisions of overall decisions

- implementation rules

Motivation

Concepts form the basis for *conceptual integrity* (consistency, homogeneity) of the architecture. Thus, they are an important contribution to achieve inner qualities of your system.

Some of these concepts cannot be assigned to individual building blocks (e.g. security or safety). This is the place in the template that we provided for a cohesive specification of such concepts.

Form

The form can be varied:

- concept papers with any kind of structure

- cross-cutting model excerpts or scenarios using notations of the architecture views

- sample implementations, especially for technical concepts

- reference to typical usage of standard frameworks (e.g. using Hibernate for object/relational mapping)

Structure

A potential (but not mandatory) structure for this section could be:

- Domain concepts

- User Experience concepts (UX)

- Safety and security concepts

- Architecture and design patterns

- "Under-the-hood"

- development concepts

- operational concepts

Note: it might be difficult to assign individual concepts to one specific topic on this list.
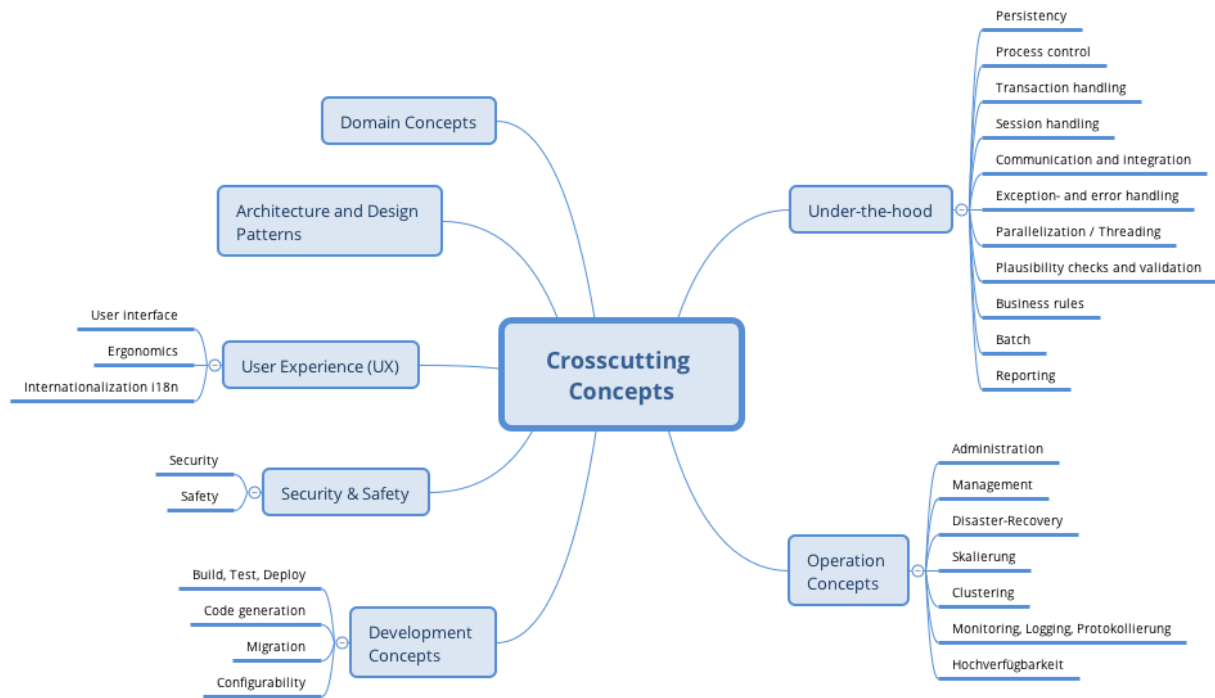
9.1. <Concept 1>

<explanation>

9.2. <Concept 2>

<explanation>

...

9.3. <Concept n>

<explanation>

10. Design Decisions

- First decision is that we are using Git as version control system instead of using mercurial, vsn, etc. and GitHub as source code management instead of using Bitbucket, this decision was imposed by the teachers of the subject.

- We are using javascript family language(javascript, typescript, node.js, angular, etc) to build the application as the main programming language. As we understand, javascript is the easiest way to make a SOLID application.

- We are also using basic languages for the web such as html, css, etc.

- Social linked data(SOLID) based application. A must decision we have to follow if we want to make a decentralized application. This means following several important constraints to build the application. One of them is storing date in PODS, so the user is the owner of their data.

- Following MVC architecture as much as we can. It is the main architecture model we know something about, so we are going to try to follow it as much as we can.

- Following Angular components structure, so we can build a high cohesion and low coupling app.

- To be continued.

TODO Improve Mind-Map

TODO Full decisions table?

11. Quality Requirements

11.1. Quality Tree

11.2. Quality Scenarios

Contents

Concretization of (sometimes vague or implicit) quality requirements using (quality) scenarios.

These scenarios describe what should happen when a stimulus arrives at the system.

For architects, two kinds of scenarios are important:

- Usage scenarios (also called application scenarios or use case scenarios) describe the system's runtime reaction to a certain stimulus. This also includes scenarios that describe the system's efficiency or performance. Example: The system reacts to a user's request within one second.
- Change scenarios describe a modification of the system or of its immediate environment. Example: Additional functionality is implemented or requirements for a quality attribute change.

Motivation

Scenarios make quality requirements concrete and allow to more easily measure or decide whether they are fulfilled.

Especially when you want to assess your architecture using methods like ATAM you need to describe your quality goals (from section 1.2) more precisely down to a level of scenarios that can be discussed and evaluated.

Form

Tabular or free form text.

12. Risks and Technical Debts

Contents

A list of identified technical risks or technical debts, ordered by priority

Motivation

"Risk management is project management for grown-ups" (Tim Lister, Atlantic Systems Guild.)

This should be your motto for systematic detection and evaluation of risks and technical debts in the architecture, which will be needed by management stakeholders (e.g. project managers, product owners) as part of the overall risk analysis and measurement planning.

Form

List of risks and/or technical debts, probably including suggested measures to minimize, mitigate or avoid risks or reduce technical debts.

Risks	Solutions
Lack of knowledge and use of angularjs	• Consult forums • Look at the documentation • take courses
Lack of knowledge of solid	• Look at the solid forums • Ask other classmates
Abandonment / lack of work of a member	• Know the status of the entire project • Hold periodic meetings
Lack of time to complete the project	• Do it stepwise and continue
Misunderstood the requirements of the application	• Consult with the professor about the requirements
Problems with git	• Use a specific methodology to do merges and commits

13. Glossary

Definition of most important terms every stakeholder should know to understand the documentation.

Order by rarety.

Only in english at the moment.

Term	Definition
Decentralized	User is the owner of their data and not the application
Git	Version control system
Inrupt	SOLID enterprise
MVC	Model-View-Controler, software architecture
PODS	Personal Online Data Stores, used to save data
RDF	Resource Description Framework, standard model for data interchange in the web
Scrum	Agile methodology for working in group
SOLID	Social Linked Data
Stakeholders	People interested in a proyect

About arc42

arc42, the Template for documentation of software and system architecture.

By Dr. Gernot Starke, Dr. Peter Hruschka and contributors.

Template Revision: 7.0 EN (based on asciidoc), January 2017

© We acknowledge that this document uses material from the arc 42 architecture template, http://www.arc42.de. Created by Dr. Peter Hruschka & Dr. Gernot Starke.