Architecture Document

15/04/2022 Eindhoven
Version: 0.1

CB-S3 Group 1

Members:
Aleksej Borisov: 2776286
Oleksandr Gurianov: 4178092
Mohammad Nazibul Khan: 4263308
Lars Kluijtmans: 4220269
Noelia Rodriguez Morales: 3635988
Esther Wolfs: 3329984

Tutor:
Nicole Zuurbier

Version history

Version	Date	Author(s)	Changes	State
0.1	15/04/2022	Esther Wolfs, Lars Kluijtmans, Noelia Rodriguez Morales	Add C4 diagram and solid	started

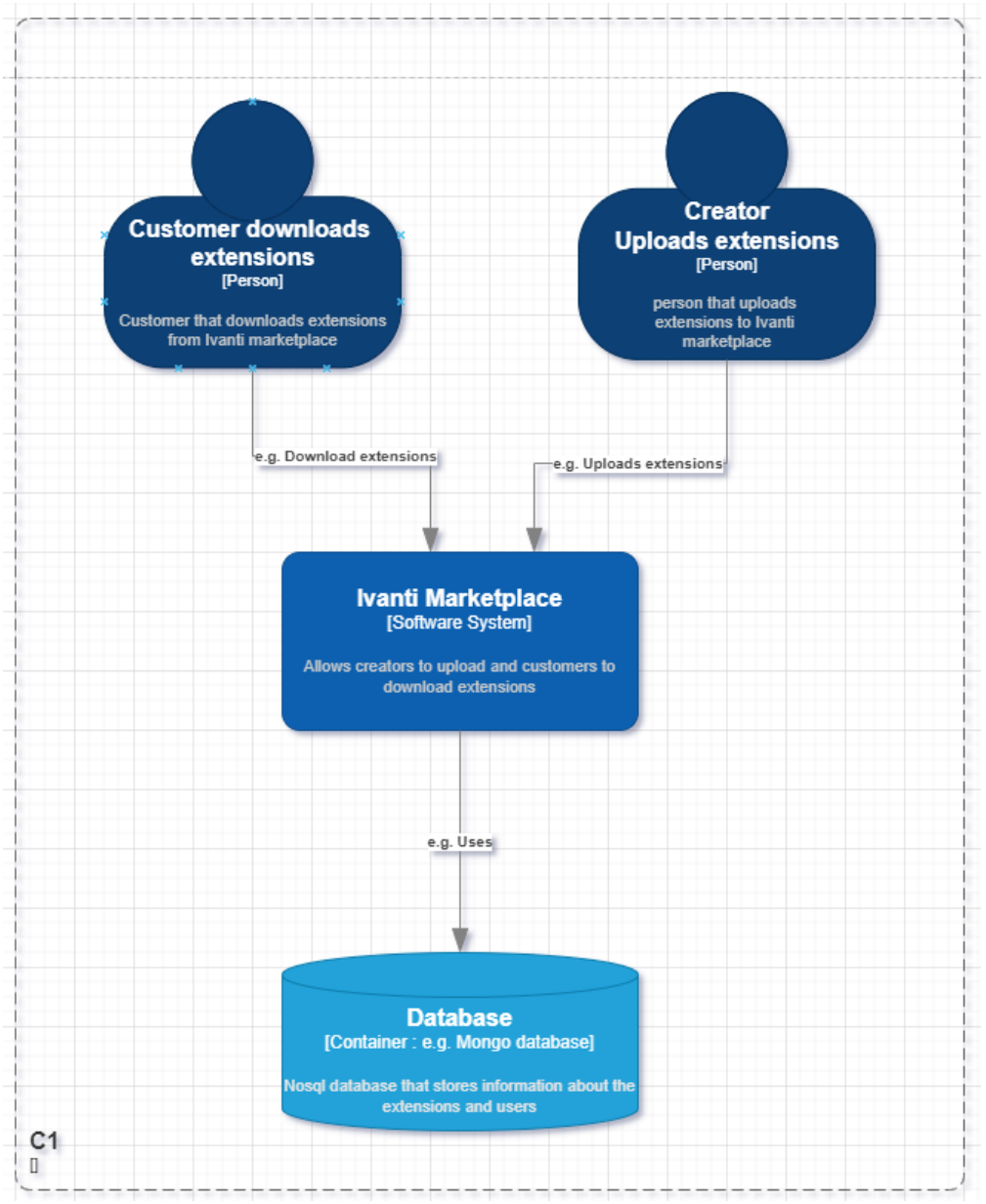
1. Introduction

In this document we will explain our design decisions we have made for this project. This is based on the research we have done. We will start with the C4 diagram and then after that we will explain how we used the SOLID principles.

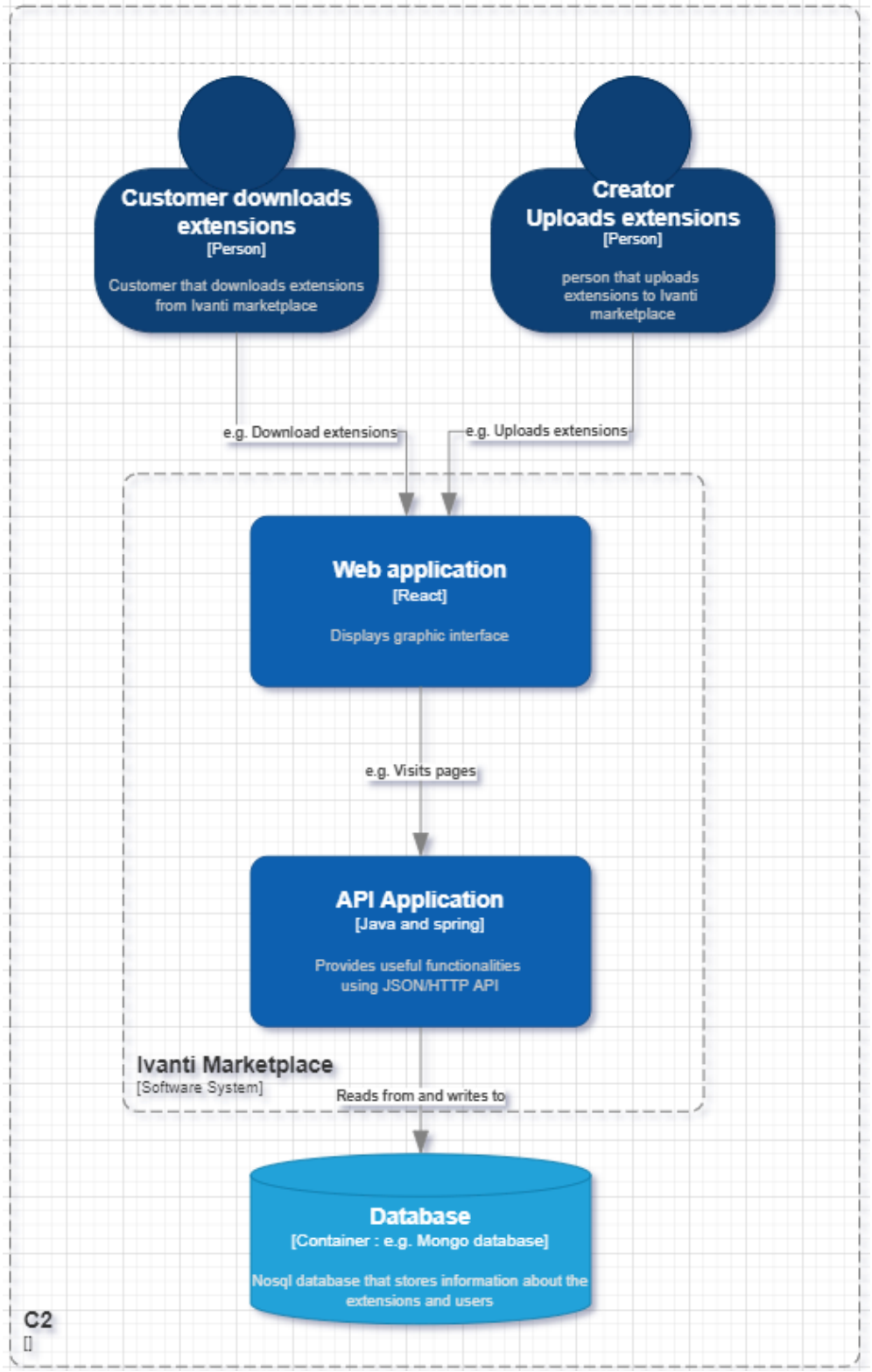
2. Architecture

2.1 C4 Diagram

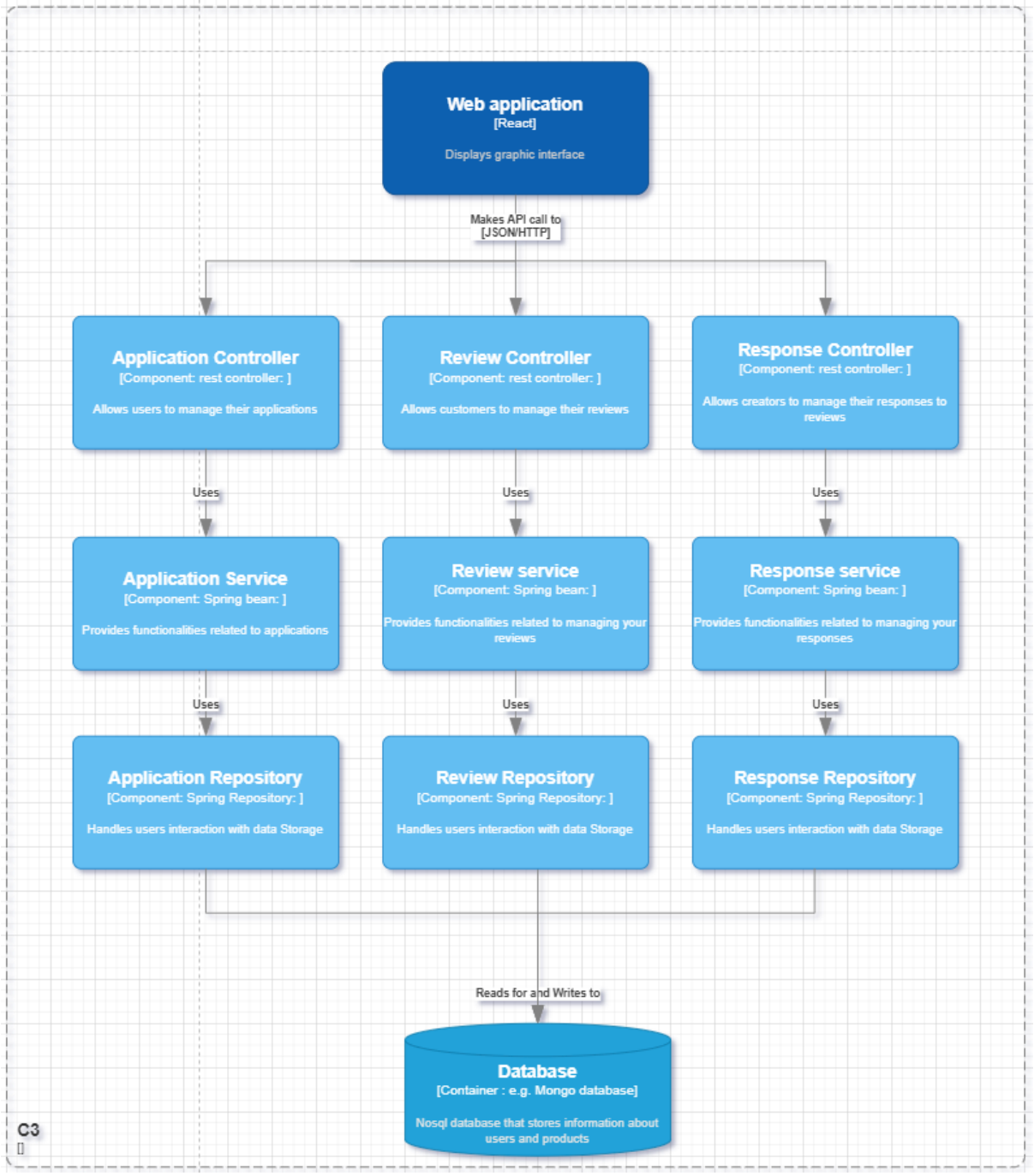
C1: There are two types of users for this application. The customer, who can download applications and leave reviews and the creator, who can upload and manage applications they have made and reply to the reviews.



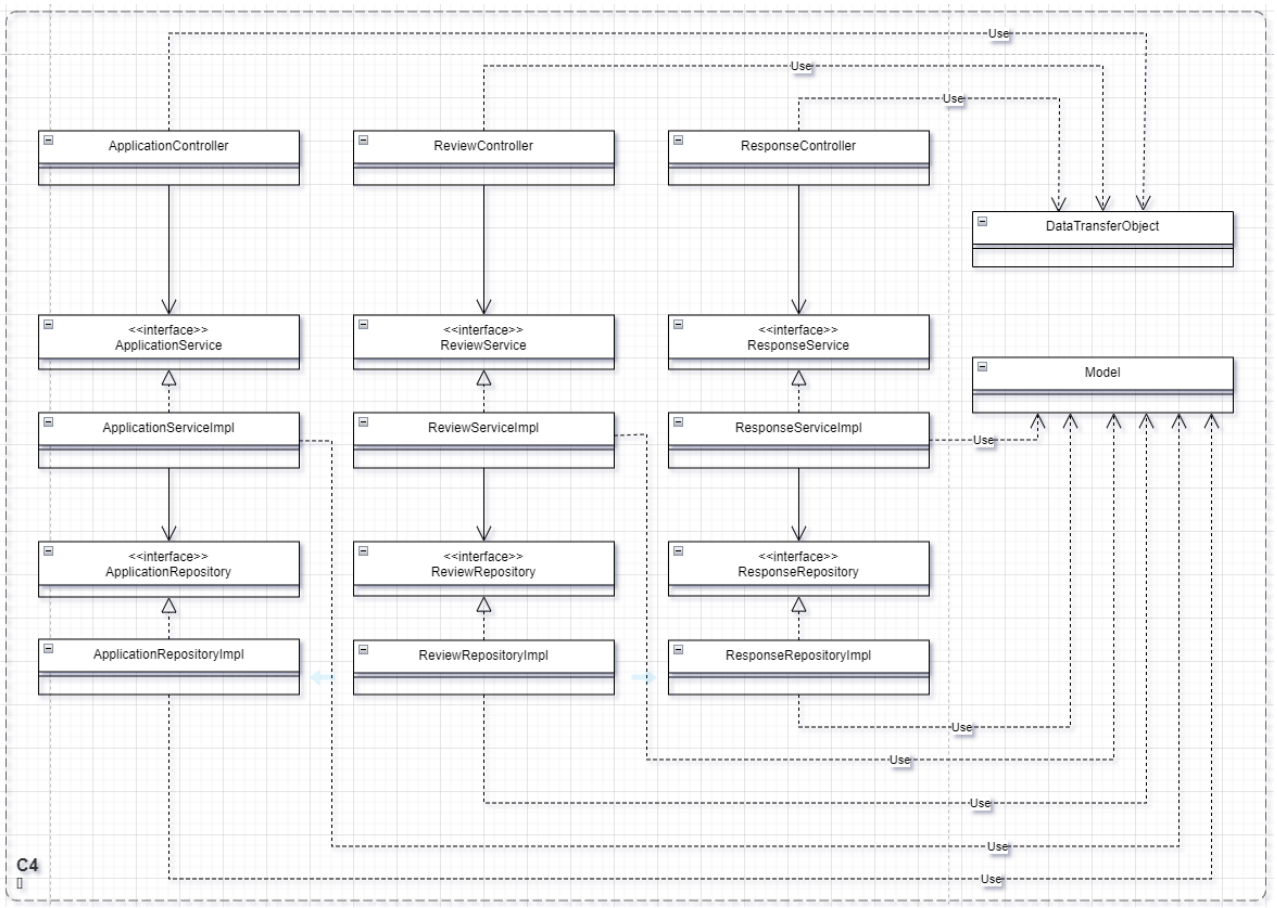
C2: The system is divided in two parts, the web application that is built using the front end JavaScript library React and the backend Java APIs.



C3: For now, the API is divided into 3 sections, a section for applications, reviews and responses. These 3 sections are separated into 3 layers, the Controller, Service and Repository layer, to guarantee the three layer design.



C4: This is the first version of the UML. All three layers are connected to each other and separated by interfaces.



2.2 How is SOLID guaranteed

The solid principles are used in Object Oriented Design, to help create a design that is easily maintainable and extendable.

- **S**ingle Responsibility:
  Every single class created is responsible for only one task. This is possible by dividing the controllers by feature, so there are no controllers responsible for example both the customer reviews and the creator responses at the same time.
- **O**pen/Closed Principle:
  To keep the code extendable, the open/closed principle is used. Using interfaces will help with this, because this means you don't have to change the existing code if you want to add a new feature. The objects are open for extension but closed for modification.
- **L**iskov substitution:
  The liskov substitution means that every subclass should be able to be substituted by the base class. For this application inheritance is used in the User class. Both the creator and customer inherit from this user class.
- **I**nterface Segregation Principle:

Right now we are not using the interface segregation principle. We will add this when we implement authentication/authorisation.

- **D**ependency Inversion Principle:
  The layers are separated by interfaces, the classes depend on the interfaces and are given them when created. For example if you want to switch between using a fake database and a real database you can give the class another instance of the database, without getting any errors or having to modify the existing code.