Software Architecture Documentation

Picnify – Image Sharing App

Authors:

Ayse Özaltin, Sascha Potesil and Konrad Konlechner

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

Introduction and Goals

The Image Sharing App is made to revolutionize the way photography enthusiasts and professionals engage with visual content. This document outlines the core goals and scope of the app that focuses on delivering a unique image-sharing experience. With a commitment to advanced editing tools, customizable filters, and community building, the app aims to set new standards in the dynamic landscape of visual content platforms.

Requirements Overview

The Image Sharing App aims to create a unique platform for editing, sharing, and discovering high-quality images with a global community. Picnify aims to enable both expression and community interaction.

Key requirements:

- iOS / Android mobile app
- Integrated User Management
- Community Features
- Advanced picture editing & customized filters
- Picture sharing
- Subscription model

Quality Goals

Priority	Quality	Motivation
1	Reliability / Stability	Implement robust backend to handle an increasing number of users, images, and interactions
2	Performance Efficiency	Fast load times despite high quality uploads
3	Operability / Usability	Seamless and responsive user experience across different devices and network conditions.

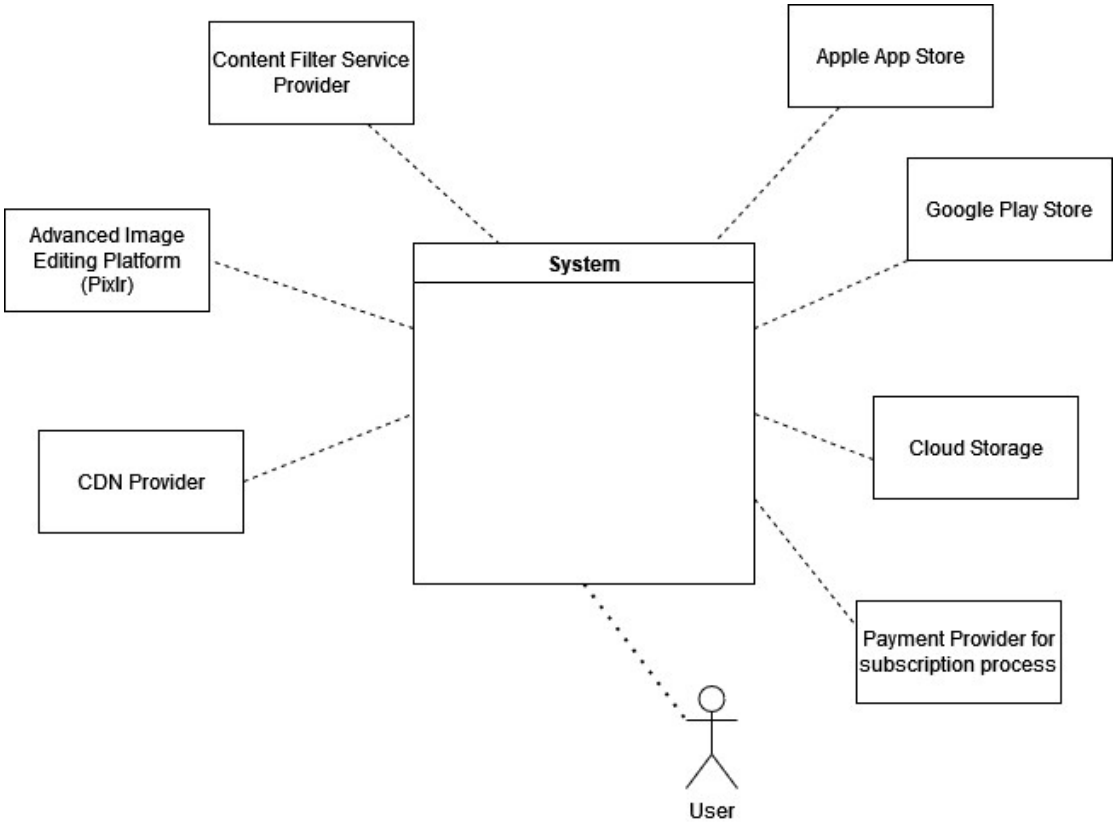
Stakeholders

Role / Name	Expectations
User (photography enthusiasts)	Dynamic storytelling with high quality images and option to interact with family & friends
Marketing Manager	Market a high quality picture sharing app that appeals to a wide & diverse target audience
Project Manager	Perform transparent risk management & quality assurance to stay within budget constraints
Lead Developer	Establish & maintain code quality standards & cross platform integration
UX / UI Designer	Focus on visually consistent but also user-centered and accessible design

Architecture Constraints

Type	Constraints	Background and / or motivation n
Technical	The app must be implemented in Flutter	Facilitate a consistent development approach
Technical	Third party software/filters must be open-source & installable via a package manager	External dependencies should be available via package manager/installer
Operating system	Mobile OS independent development	Application should run on both iOs and Android to reach a wide audience
Security	Robust security measures (Compliance with Privacy Regulations)	Protect user data, including images and personal information

Business Context

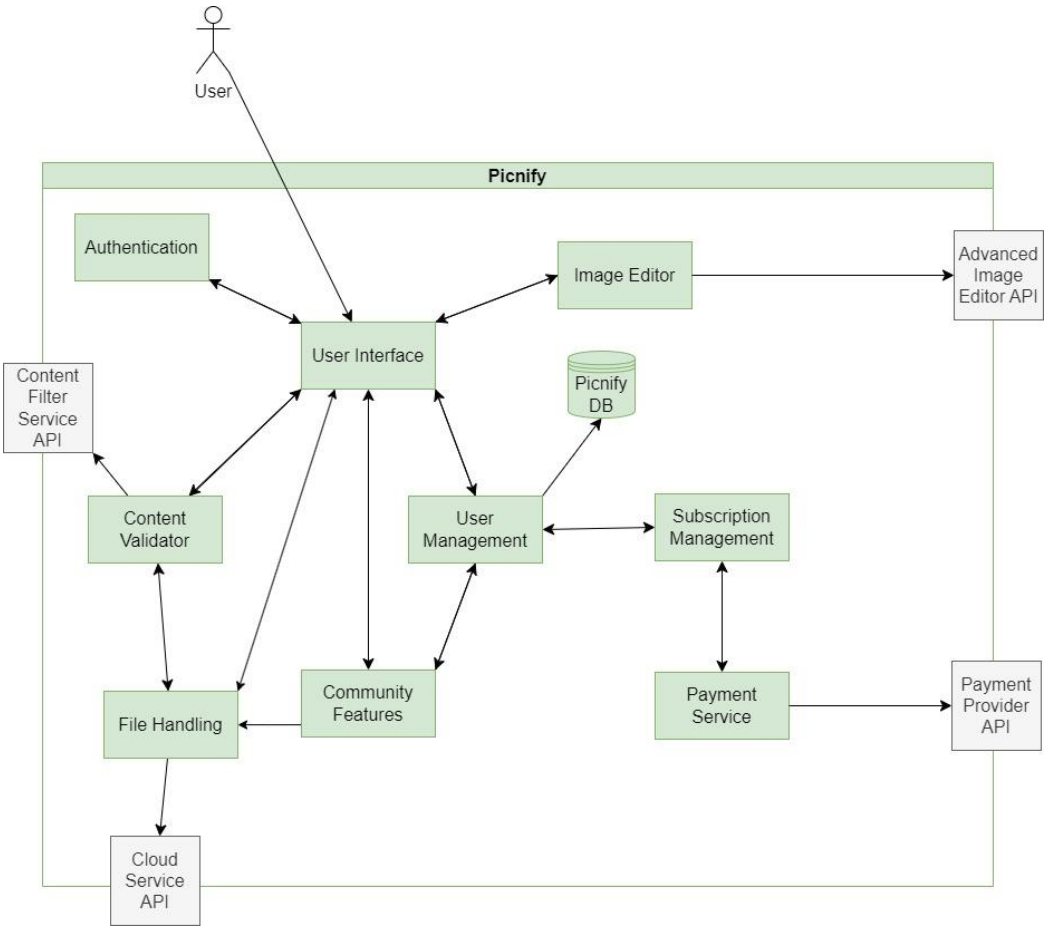


Solution Strategy

Goal/Requirements	Architectural Approach
Reliability / Stability	Microservices
Performance Efficiency	Content Delivery Network (CDN)
Operability / Usability	Single Page Application architecture
Cross-platform	Flutter

Building Block View

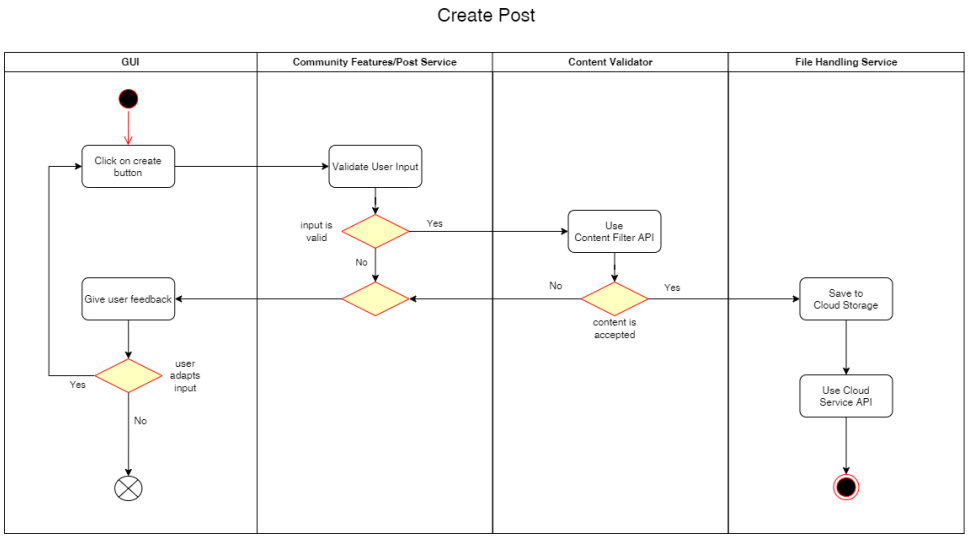
Level 1



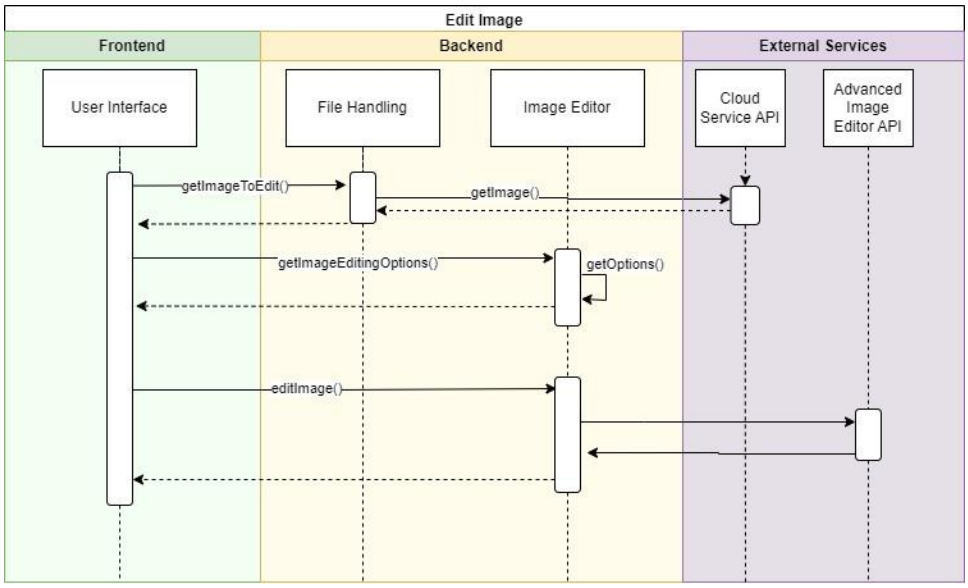
Component	Responsibility
User Interface	UI and UX for Picnify.
File Handling	Service for handling and packing images and other content files.
Content Validator	First-level filter of harmful or undesired images and text.
Authentication	Service for user authentication and authorization
Community Features	Service for handling community features such as commenting, liking and posting.
User Management	Service for features regarding user management, including user-to-user (friend requests, tagging) and admin-to-user interactions (blocking, unblocking, editing user data)
Image Editor	Service for first level image editor.
Subscription Management	Service for user subscription features.
Payment Service	Service and adapter for payment services and providers
Content Filter Service API	API(s) of external content filter service.
Cloud Service API	API(s) of external cloud storage provider.
Payment Provider API	API(s) of external payment providers.
Advanced Image Editor API	API(s) of external image editor.

Runtime View

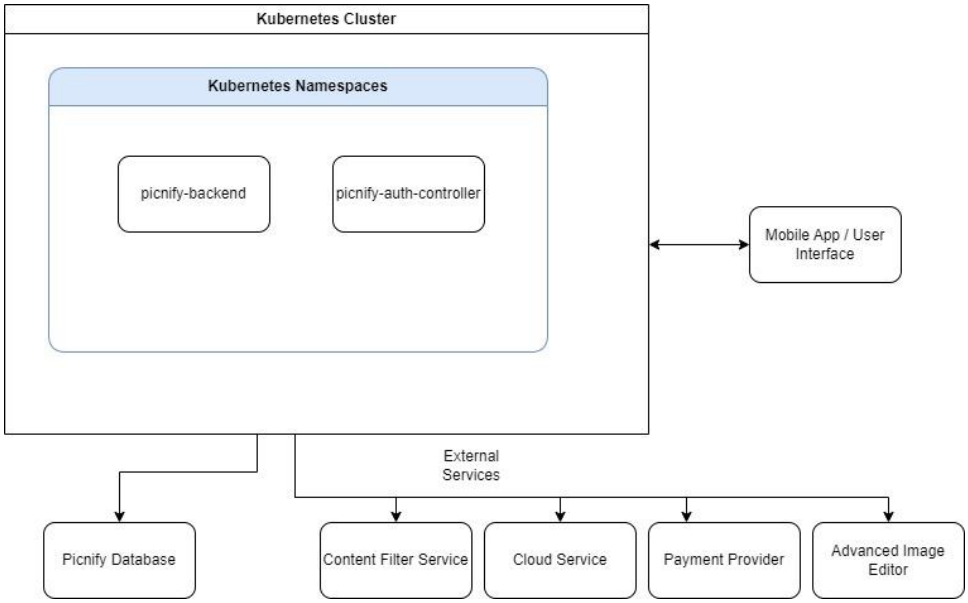
Create Post



Edit Image



Deployment View



Crosscutting Concepts

Development concepts

- Continuous integration and continuous deployment (CI/CD) for rapid delivery of features.
- Automated tests succeeding deployment to improve quality assurance.

Architecture and design patterns

- Usage of Microservices offers a modular design approach while being easy to scale based on demand. Moreover, if one microservice fails, it doesn't affect the entire system since the services are widely decoupled.
- Usage of the "Saga" pattern for implementation of service-spanning business transactions.
- CAP-Theorem: Focus on availability and partition tolerance while accepting only eventual consistency will allow the system to be highly available to the customers.

- Separation of concerns: Do not implement business logic within the presentation layer or the data source (--> MVC pattern)! This enables exchangeability and therefore increases future flexibility by avoiding getting stuck to a certain technology or vendor.

Safety and security concepts

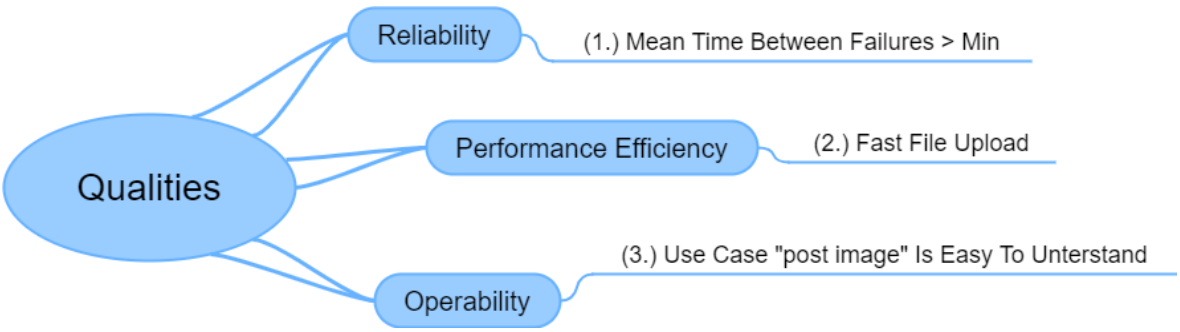
- Biometric Authentication via fingerprint recognition for identity verification.
- Data Encryption to protect sensitive data such as user credentials and images.

Design Decisions

Problem	Considered Alternatives	Decision	Date
Cross-platform development	Flutter, React + Swift	Flutter; avoid need of additional resources for development	Nov 2023
Security issues	validation of uploads, encryption, access controls, rate limiting	validation of uploads using AI and upload limitations	Nov 2023
Insufficient data storage	NoSQL databases, SQL databases	NoSQL databases to handle large-scale data	Nov 2023

Quality Requirements

Quality Tree



Quality Scenarios

Quality Scenario	Description	Date
1. The Mean time between failures is higher than acceptable minimum	The average error rate for a hundred million transactions processed per year is acceptable if lower than 0.00032 per second. This leads to a minimum mean time elapsed between two reported failures of 52 minutes. For a detailed description of the calculation and assumptions see below.	Jan 2024
2. Usecase “post image” is easy to understand	A user can publish a self-made photograph in a post to share it with the community in less than 5 minutes.	Nov 2023
3. Fast file upload	The upload of an image with size of 20 MB at least with 10 Mbit/s upload speed (tested with speedtest.net by Ookla) does not take longer than 30 sec.	Nov 2023

Calculation of the mean time between failures (MTBF):

Assumptions:
It seems to be acceptable to have 0.01% of the transactions failing within a year due to an error. For 100 million transactions per year this would result in 0.01 million errors within a time period of 31.536 million seconds (3600 s/h · 24 h/d · 365 d/y = 31 536 000 seconds per year).

⇒ The average error rate $\lambda = 0.01 / 31.536 \approx 0.00032/\text{s} = 3.2 \cdot 10^{-4}\ \text{s}^{-1}$

Note: Errors are automatically reported to a monitoring server. Based on this data the error rate for an observed time period can be obtained.

The MTBF is given by the expectancy value of the time till an error occurs.
For the probability density function f(t) at the time t of an error event the MTBF can be calculated with the following formula:

$$MTBF = \int_0^\infty t \cdot \mathrm{f(t)} dt$$

Assuming an exponential distribution the density function is given by the following expression:

$$\mathrm{f(t)} = \lambda \cdot e^{-\lambda t} \Rightarrow MTBF = \frac{1}{\lambda}$$

Therefore, the MTBF of the whole bulk of transactions can be easily determined as the inverse value of the error rate:

MTBF = $1 / (3.2 \cdot 10^{-4}\ \text{s}^{-1})$ = 3 125 s ≈ <u>52 min</u>

If the number of transactions is divided by a certain factor, then the MTBF will be multiplied by the same factor.

For 1 million transactions per year (divided by 100) this would result in a MTBF 100 times higher:

MTBF = 3 125 s · 100 = 312 500 s ≈ <u>86,8 h</u>

Risks and Technical Debts

Risk/Technical Debt	Description
dependency on external services	For both AI integration and use of filters for image editing we use external services – possible bottleneck when the future maintenance is unclear.
high costs for CDN servers	High costs & uncertainty for providing fast services all over the world
lack of qualified developers	Difficulty to find Flutter developers that cover both iOS and Android development

Glossary

Term	Definition
API	*Application Programming Interface*
CAP-Theorem	*The CAP-Theorem is about the interaction between **C**onsistency, **A**vailability and **P**artition Tolerance. It states that only two of those can be reached.*
CDN	*Content Delivery Network* is a geographically distributed network of proxy servers and their data centers. The goal is to provide high availability and performance by distributing the service spatially relative to end users.
Community Features	*All kind of features which interact with other users like publishing posts, chat, comments and likes*
Flutter	*Open-source UI software development toolkit, allowing developers to build natively compiled applications for mobile, web, and desktop from a single codebase.*
Kubernetes Cluster	*A Kubernetes cluster is a set of interconnected computers (nodes) that work together to orchestrate containerized applications, providing scalability, resilience, and automation for deployment and maintenance.*
MTBF	*Mean Time Between Failures* *Is a measure used for product quality. Usually this refers to a single device and describes the operation time between two failures.*
MVC pattern	*Model View Controller – design pattern, architectural pattern* *This concept is motivated by the idea of separation of concerns and assures flexibility in future development*
UI	*User Interface*
GUI	*Graphical User Interface*
UX	*User Experience*