A System to Promote Agenda for a Better Society

14/01/2022

Alon Fliess

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Document Version Number | Author | Change Summary | Reviewer | Date |
| V1.0 | Alon Fliess | First Draft |  | 14/01/2022 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Contents

[3. Definitions, Acronyms and Abbreviations 4](#_Toc93032201)

[4. Introduction 5](#_Toc93032202)

[5. Requirements 5](#_Toc93032203)

[5.1. The system users: 5](#_Toc93032204)

[5.2. User Stories (Functional Requirements) 5](#_Toc93032205)

[5.3. Non-Functional Requirements 6](#_Toc93032206)

[5.4. Assumptions & Constraints: 6](#_Toc93032207)

[5.5. The development process and DevOps 7](#_Toc93032208)

[5.6. Testing 7](#_Toc93032209)

[5.7. DevOps 7](#_Toc93032210)

[6. High Level Design 8](#_Toc93032211)

[6.1. IDesign Method 8](#_Toc93032212)

[6.2. System Components & Cross Cutting Concerns 9](#_Toc93032213)

[7. Conceptual Architecture 10](#_Toc93032214)

[8. Architecture Analysis (Microservices) 11](#_Toc93032215)

[8.1. Service Responsibilities 24](#_Toc93032216)

[9. Implementation Strategy 25](#_Toc93032217)

[10. Cross cutting concerns 25](#_Toc93032218)

[10.1. Hosting 25](#_Toc93032219)

[10.2. Logging 25](#_Toc93032220)

[10.3. Software Analytics 26](#_Toc93032221)

[10.4. Identity (SSO) & Security 26](#_Toc93032222)

[10.5. Manageability, Administration and Telemetry (KPI) 26](#_Toc93032223)

[10.6. Caching 26](#_Toc93032224)

[10.7. Storage (No SQL/SQL/Blob) 26](#_Toc93032225)

[10.8. Queuing 26](#_Toc93032226)

[10.9. Inversion of Control Containers (IoC), Dependency Injection (DI) & Plugin Framework 26](#_Toc93032227)

[11. UI technology 26](#_Toc93032228)

[12. Risk Mitigation 26](#_Toc93032229)

[12.1. Overview 26](#_Toc93032230)

[13. Summary 27](#_Toc93032231)

Table of Diagrams

[Figure 1 Conceptual Architecture 7](#_Toc93030215)

[Figure 2 All Services 8](#_Toc93030216)

[Figure 3 A User registration 9](#_Toc93030217)

[Figure 4 Creating Campaign 10](#_Toc93030218)

[Figure 5 Donate Goods 11](#_Toc93030219)

[Figure 6 Get the list of campaigns 12](#_Toc93030220)

[Figure 7 The users that bought gifts 13](#_Toc93030221)

[Figure 8 An activist buys a product 14](#_Toc93030222)

[Figure 9 An activist donates a gift 15](#_Toc93030223)

[Figure 10 Send a gift 16](#_Toc93030224)

[Figure 11 Process Tweets 17](#_Toc93030225)

[Figure 12 Show Balance 18](#_Toc93030226)

[Figure 13 Update Report 19](#_Toc93030227)

[Figure 14 Generate Report 20](#_Toc93030228)

# Definitions, Acronyms, and Abbreviations

|  |  |  |
| --- | --- | --- |
| Term | Description | Comments |
| ProLobby | The startup company |  |
| PromoIt | The system that we build |  |
| MVP | Minimal Viable Product |  |
| MSA | Micro Service Architecture |  |
| Azure Functions | A serverless component that executes code on trigger |  |
| REST | Representational state transfer |  |
| Reporting | Data Mining & Report generation |  |
| DTO | Data Transfer Object |  |
| IoC | Inversion of Control |  |
| Application Management Service (AKA API Gateway) | A cloud/local API gateway that routes API requests to target services |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Introduction

The ProLobby Company is seeking to build a new system – “PromoIt” that handles the social promotion of the non-profit organization and other related campaigns. The first version of the System – The Minimal Viable Product (MVP), is limited, but it can be extended in the future. For the first version, we only use the Twitter social network.

The system’s goal is to promote social campaigns. The means to do so involves onboarding business organizations that donate products, onboarding non-profit organizations that want to promote campaigns, and onboarding social activists – users of Twitter that can promote those campaigns.

# Requirements

## The system users:

1. ProLobby Owner – the company representative. This user manages the system.
2. Non-Profit Organization Representative – the user that creates a campaign
3. Business Company Representative – the user that represents a company that donates products for campaigns
4. Social Activists – Twitter users that promote campaigns

## User Stories (Functional Requirements)

1. As a non-profit representative, I’d like to register to the system to be able to create a campaign
   1. I’ll provide a name, email, and a link to my organization’s Website
2. As a non-profit organization representative, I’d like to create a campaign
   1. I’ll provide a link (URL) to the campaign landing page and the campaign hashtag
3. As a business company representative, I’d like to register to the system so I can donate products
4. As a business company representative, I’d like to donate goods to a set of chosen campaigns
   1. I’ll provide the amount of product and the value (price in dollars) of each product that I provide to each campaign
5. As a business company representative, I’d like to get the list of users and products that I need to ship so I can supply them
   1. The data includes the product id and the user details
6. As a business company representative, I’d like to inform the system that I sent a product to a user so that the system can finish the transaction
7. As a social activist, I’d like to register to the system to earn money and to be able to use the money to buy products
   1. I’ll provide my Email, Address, and Phone Number
8. As a social activist, I’d like to tweet about a campaign, so I’ll promote it and also earn money
9. As the ProLobby owner, I’d like the system to give a dollar to the social activist for each tweet that promotes a campaign and to each following retweets
   1. The tweet has to have the link and hashtag to the campaign page
10. As a social activist, I’d like to buy products so I can spend the money I earned
11. As the ProLobby owner, I’d like the system to issue a tweet whenever a social activist uses the points to buy a product
    1. The tweet includes the Twitter handle of the social activist and the business company
12. As a social activist, I’d like to know my earning status, so I know my balance
13. As a social activist, I’d like to donate a product to my chosen campaign so that I can promote it
    1. I can earn money, buy a product and donate it to a campaign, so the campaign now has more products attached to it
14. As the ProLobby owner, I’d like the system to provide the report about:
    1. Campaigns
    2. Users
    3. Tweets

## Non-Functional Requirements

1. Scale - The MVP version can serve a few users concurrently running on the same machine. However, in the future, the system must handle thousands of users without changing the system code
2. High Availability (HA) – The MVP version runs on a single PC and does not provide HA. However, using the same code and services in the Cloud should provide HA.
3. Observability – Each operation and each error should issue a log using the standard .NET Microsoft [ILogger](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.logging.ilogger?view=dotnet-plat-ext-6.0)
4. The Cost of Operation when running in the Cloud should be considered. The customer wants to know the cost of operation for a system that handle 20 campaigns, 10 business customers, and 1000 activists
5. Security – No need to secure the system in the MVP.
6. Usability - The user interface can be based on a Console or a simple WinForms app for the MVP.
7. Hosting - The System runs on the PC, but it uses cloud services hosted locally or .NET facilities

## Assumptions & Constraints:

1. Optional local (cloud) services include:
   1. <https://docs.microsoft.com/en-us/azure/azure-functions/functions-develop-local> or <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-6.0>
      1. REST API
   2. <https://github.com/Azure/Azurite> or <https://docs.microsoft.com/en-us/azure/storage/common/storage-use-emulator>
      1. You can use Storage Queue for queuing mechanism. If a better mechanism is required, use <https://www.rabbitmq.com>
   3. <https://docs.microsoft.com/en-us/azure/cosmos-db/local-emulator?tabs=ssl-netstd21>
   4. You can use any SQL DB or MongoDB if required instead of using the CosmosDB emulator
2. Create and use [Twitter developer account](https://developer.twitter.com/en)

## The development process and DevOps

The development team has two developers. The process starts by analyzing the requirements and creating the system architecture, i.e., the list of Azure Function apps or Minimal APIs apps responsible for implementing the system, the data structure, algorithms, databases, queues, and any other foundation resource.

After having the system diagram, create a list of development tasks, sorted by priority. Each time a developer finishes a task, it takes the next development task in the list.

Use one of your GitHub accounts and create the development environment, i.e., the GitHub repository. Now, clone the project to each developer machine. Each time a developer starts a new task, the developer creates a task branch. Each time the developer finishes a task, the developer creates a pull request. After the second team member approve the task (by doing a code review), the pull request is merged into the master branch.

Read more about the process [here](https://docs.github.com/en/get-started/quickstart/github-flow).

## Testing

Write integration tests by invoking the APIs from a [xUnit.net](https://xunit.net/) project. In addition, you can use [ApprovalTests.net](https://github.com/approvals/ApprovalTests.Net) to simplify the testing process. You may need to mock cloud resources and the Twitter APIs.

## DevOps

Use [GitHub Actions](https://github.com/features/actions) to [build and test](https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-net) the code on each pull-request merge.

# High-Level Design

The analysis process is based on the IDesign method, with which we captured the requirement and created the IDesign flows.

## IDesign Method

The IDesign method dictates that each micro-service belongs to one of three distinguished layers:

1. The Business Logic Layer – The What:
   1. There is the main manager microservice that handles each request business flow.
   2. Strive to asynchronous calls between managers.
   3. Managers use engine microservices to fulfill requests.
   4. Managers implement call-back contracts that engines declare or use pub/sub mechanism to enable communications from the layer below.
   5. Sometimes managers can be implemented as a Workflow or can use a Workflow to orchestrate the requests.
   6. The Manager calls an engine when a need is to process a specific business logic scenario. For example, it can directly call a resource accessor microservice when only CRUD operations are needed.
2. The Engines Layer – The How:
   1. Engine microservices are responsible for the low-level execution of a request.
   2. Engines do not call managers directly.
   3. Engines call lower-level Accessors, such as DB Accessors, File System Accessors, and external system Accessors, to access resources.
   4. Engines do not call each other directly; they do so through a manager.
3. The Accessors Layer – The Whom:
   1. Responsible for the connection, communication, and any other low-level resource-related such as file-system, database, and external systems.
   2. An Accessor decouples the low-level resource detail from the rest of the system. Using the accessor layer makes the system stable. A change in an external systems contract, protocol, or even a total replacement of a resource is handled by adjusting only the Accessor code.

The benefits of using the above-layered design are:

1. The system is built from autonomous micro-services:
   1. Each of them is a management unit.
   2. We can easily get logging, lifetime management, security boundaries, and location transparency using an appropriate infrastructure
   3. No side effects or bugs can leak between microservices
   4. Managers can be developed, tested, and deployed separately.
2. Engines have very small fan-in and fan-out levels:
   1. An Engine knows (refers to) Accessors.
   2. An Engine can use a dependency injection mechanism for cross-cutting concerns and utilities such as logging, error handling, security & identity mechanisms.
   3. One or a few managers know (refers to) each engine.
3. Accessors are stable components
   1. They do not refer to engines or managers.
   2. They have dependencies only with the underlying service contract, database, or external system that they communicate with.

## System Components & Cross-Cutting Concerns

Defining the Managers, Engines, Accessors, and Resources is a task for the high-level design stage. Usually, a few managers use many engines, accessors, and resources.

The design starts by going through the analysis model and deciding which Manager is responsible for each user story. The managers are the user story orchestrators; they activate engine routines to implement the user story tasks. For each sub-task, we need to define the engines that serve as the building block of the task. Each task needs data and communication. For each task, we define the accessors and the resources.

Each microservice in the system needs many cross-cutting services such as logging, security, caching, database, and more. These cross-cutting-concerns utilities are part of the software infrastructure.



Figure 2 - IDesign Method Analysis Process

# Conceptual Architecture

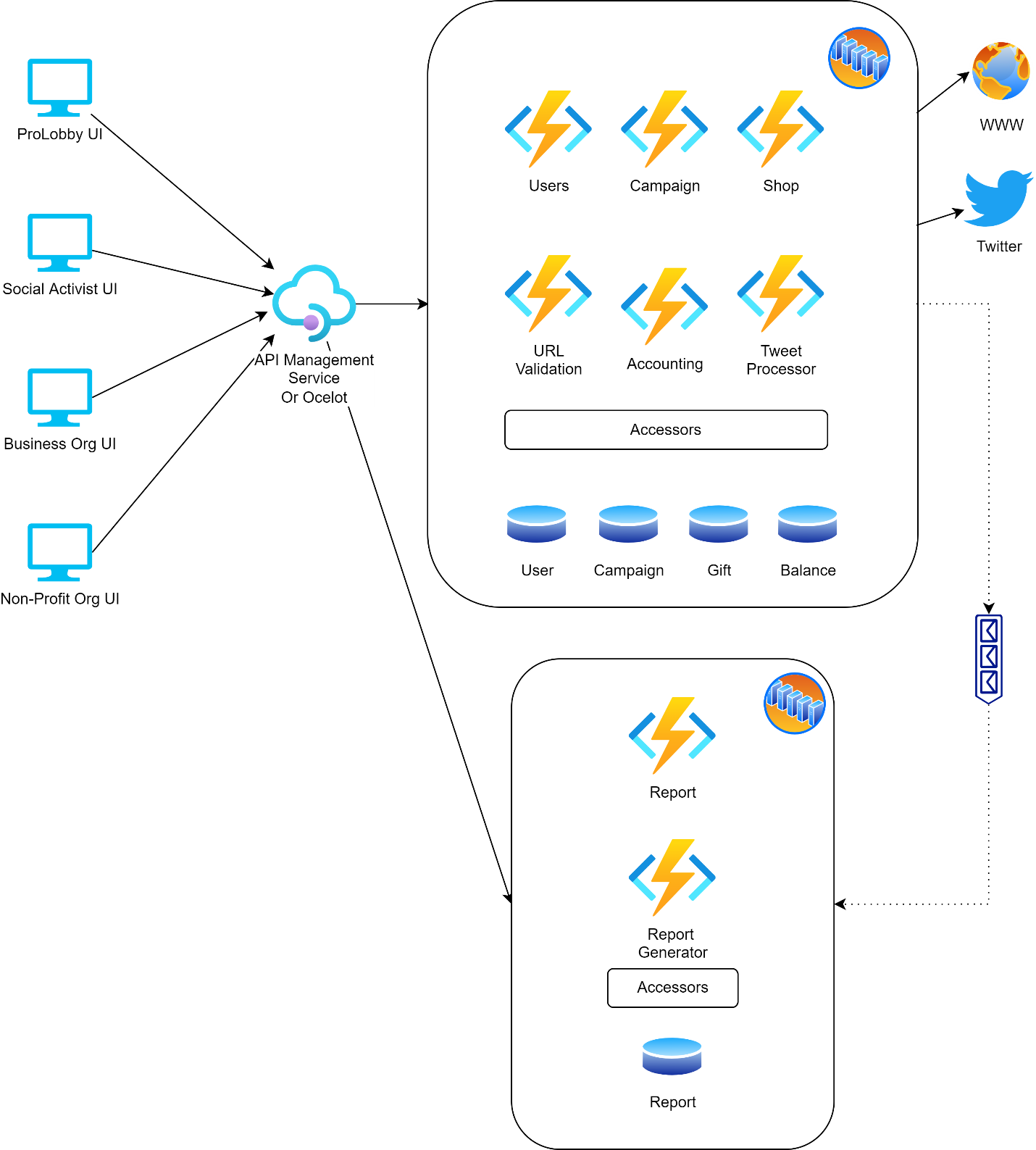


Figure 1 Conceptual Architecture

# Architecture Analysis (Microservices)

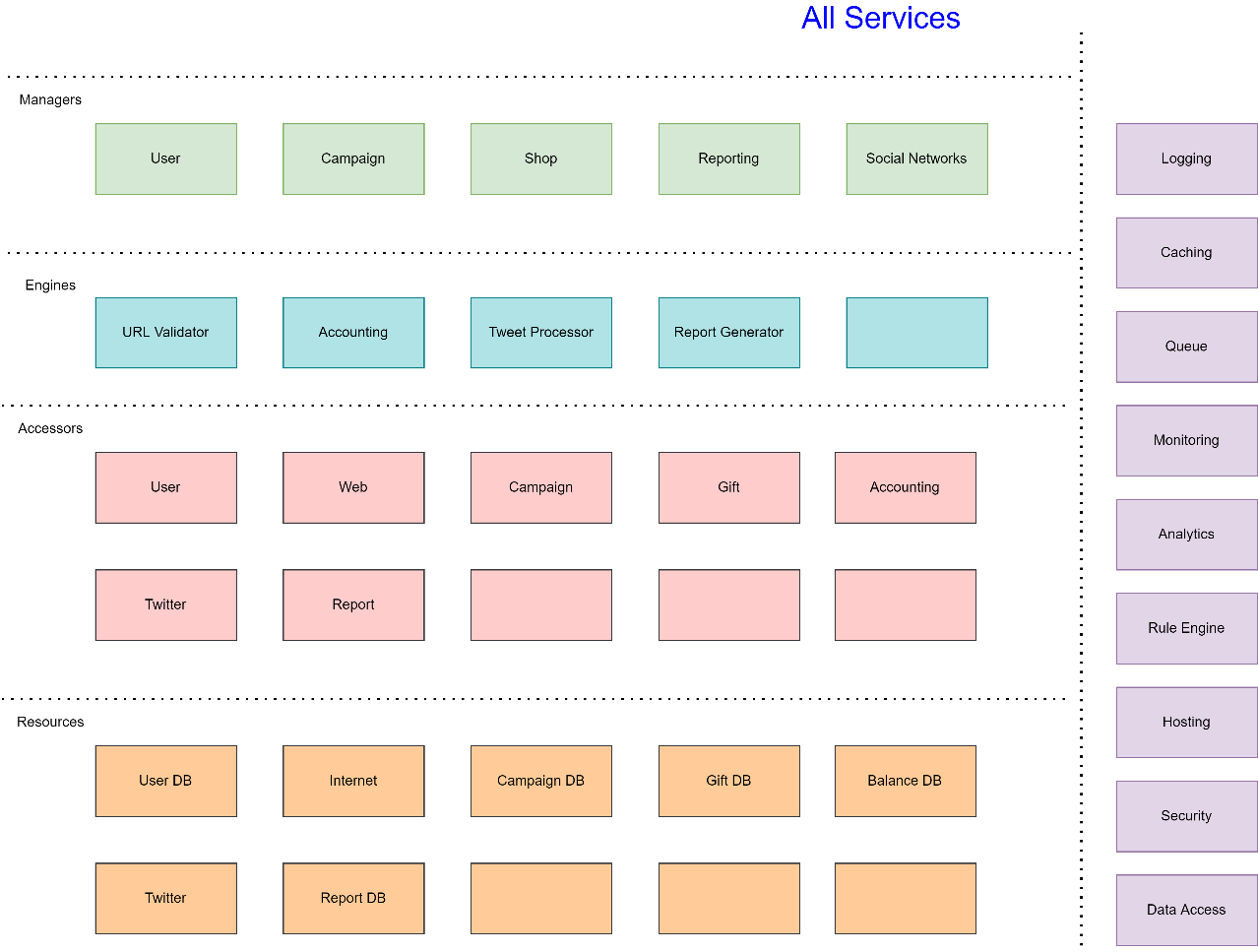


Figure 2 All Services

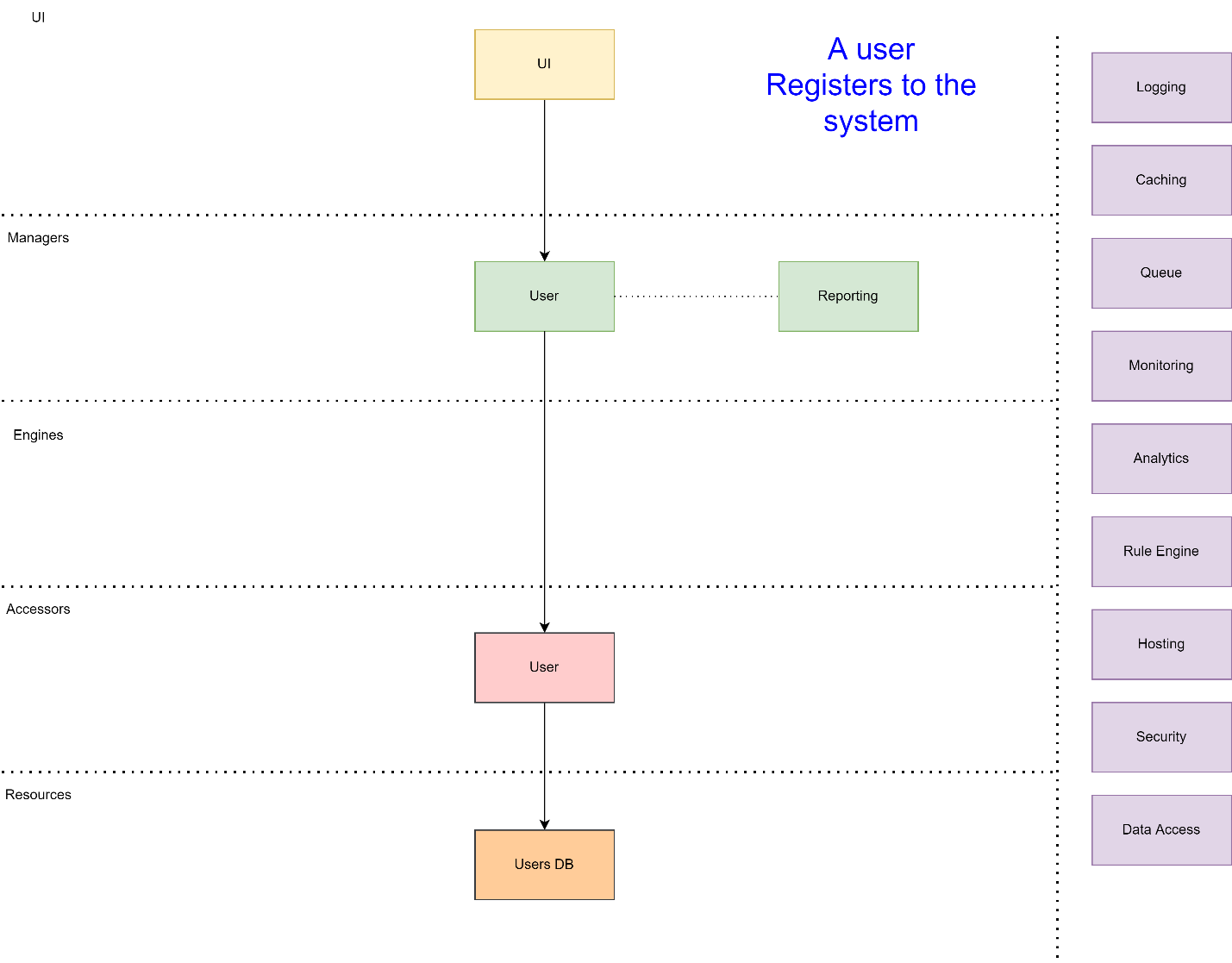


Figure 3 A User registration

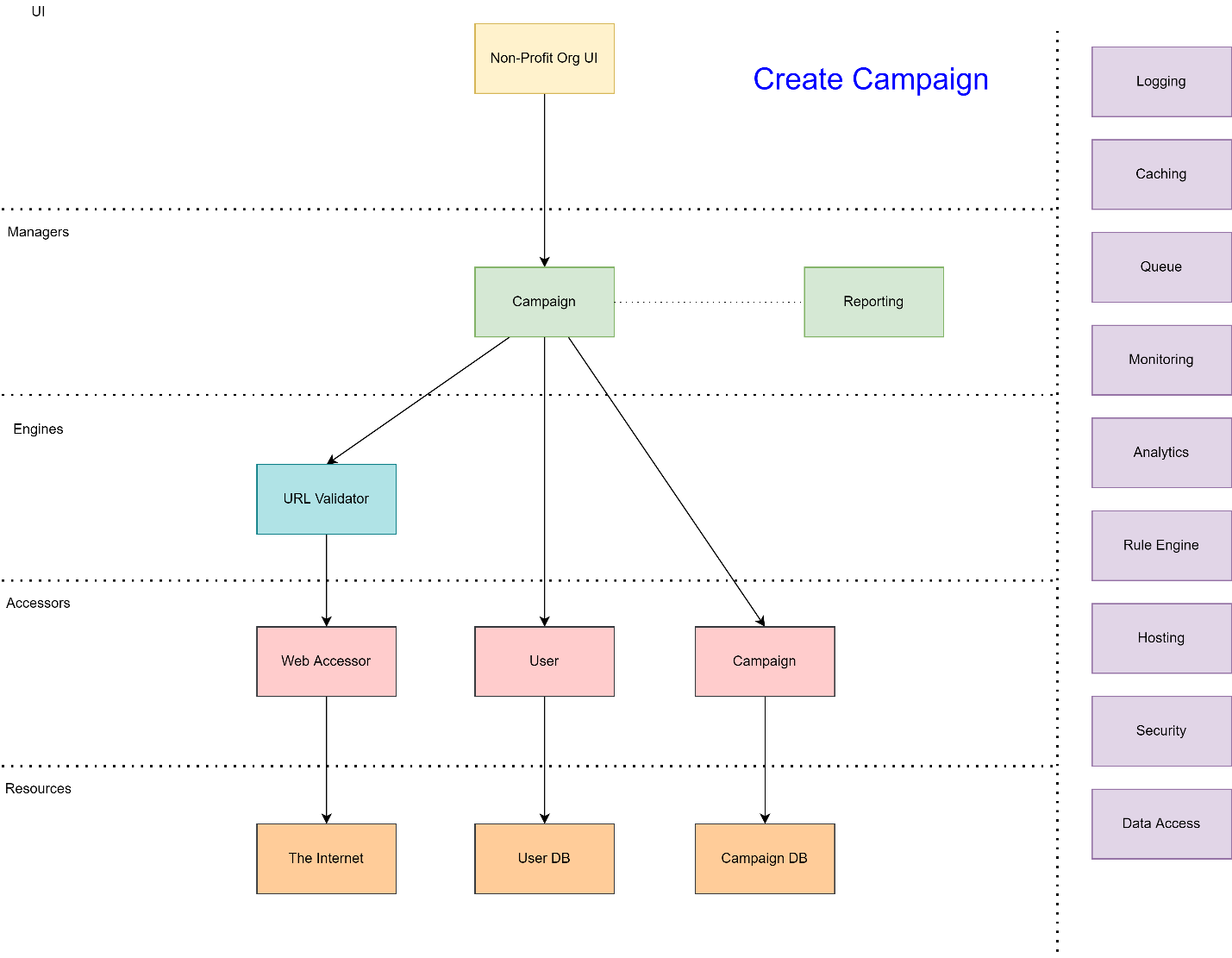


Figure 4 Creating Campaign

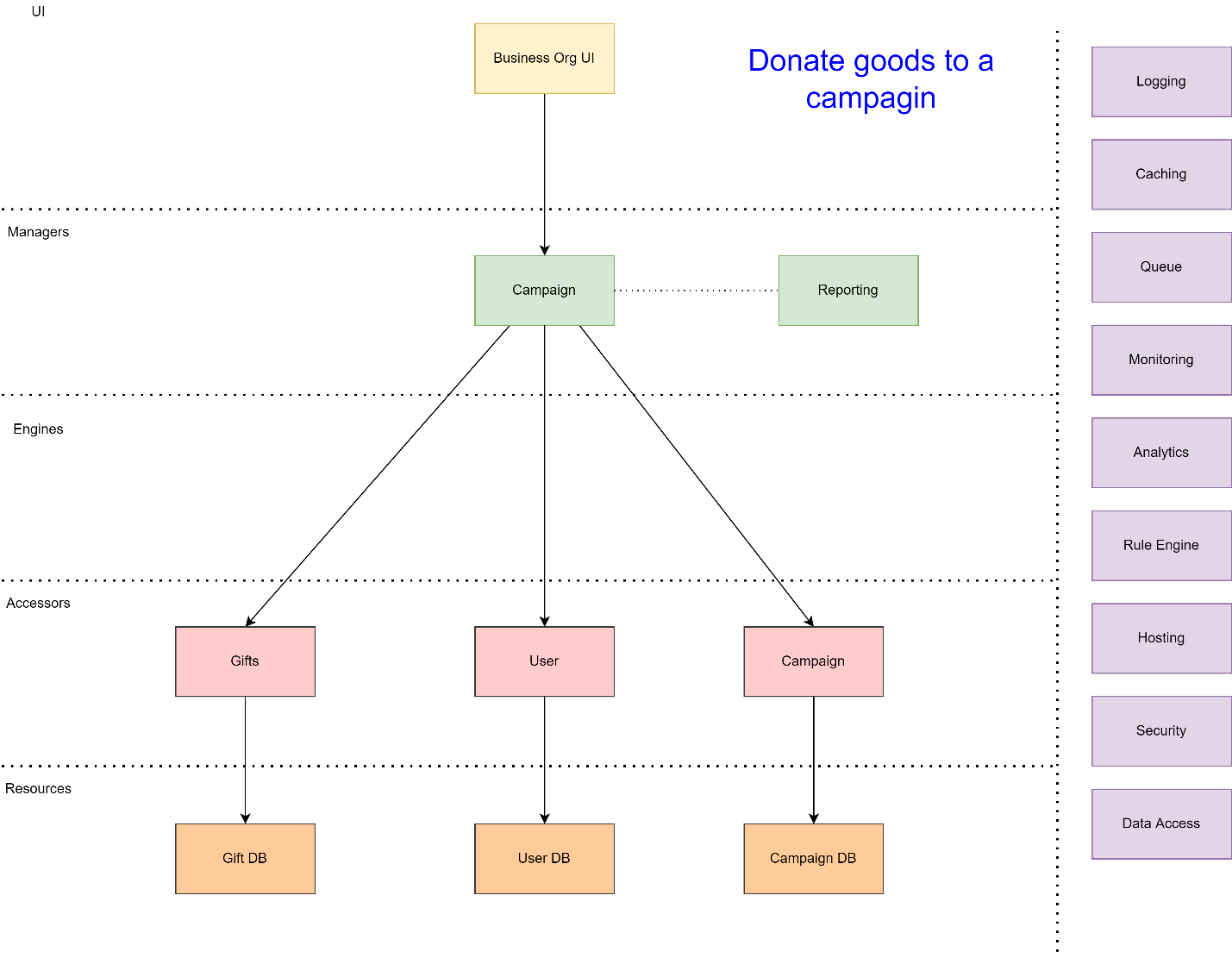


Figure 5 Donate Goods

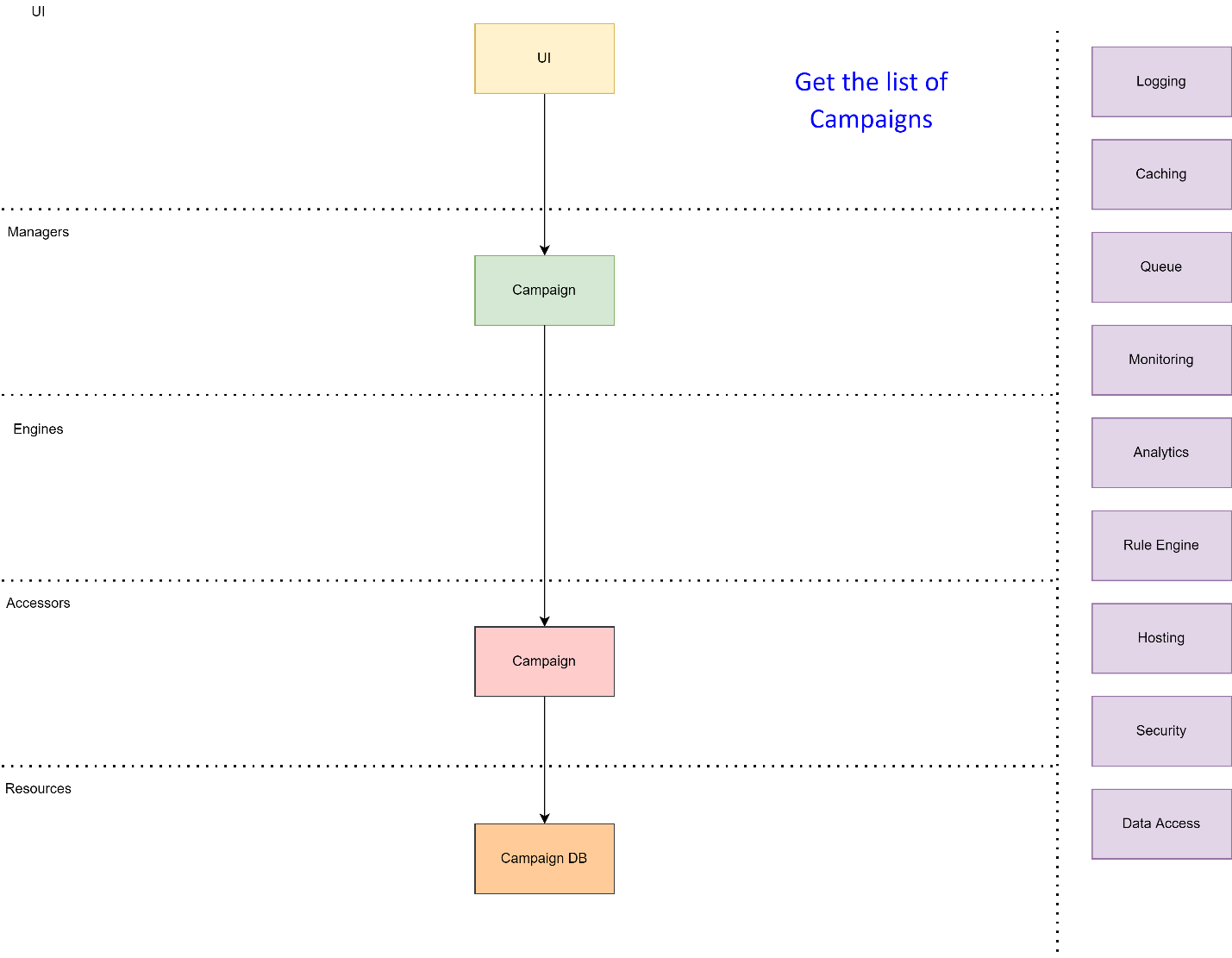


Figure 6 Get the list of campaigns



Figure 7 The users that bought gifts

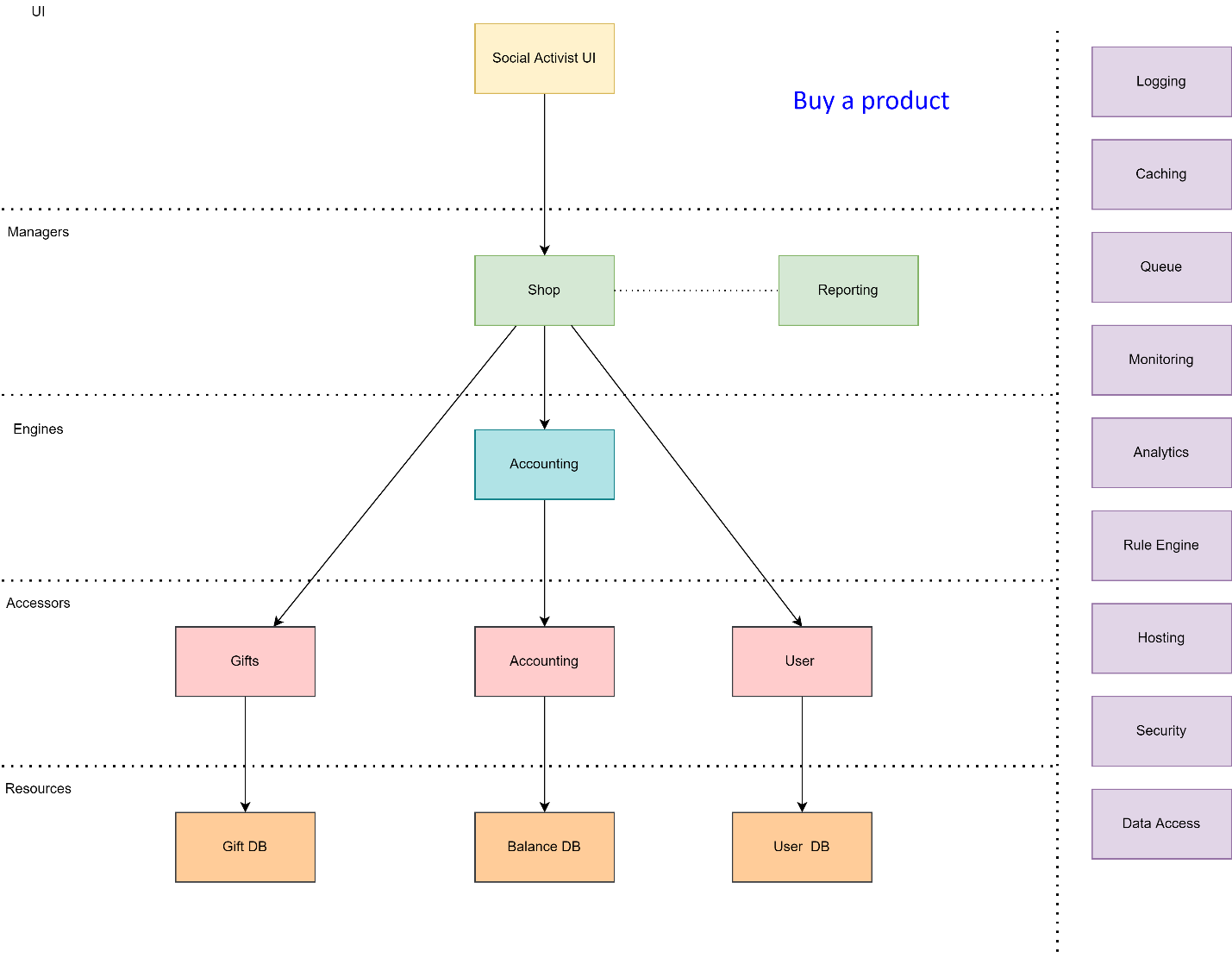


Figure 8 An activist buys a product

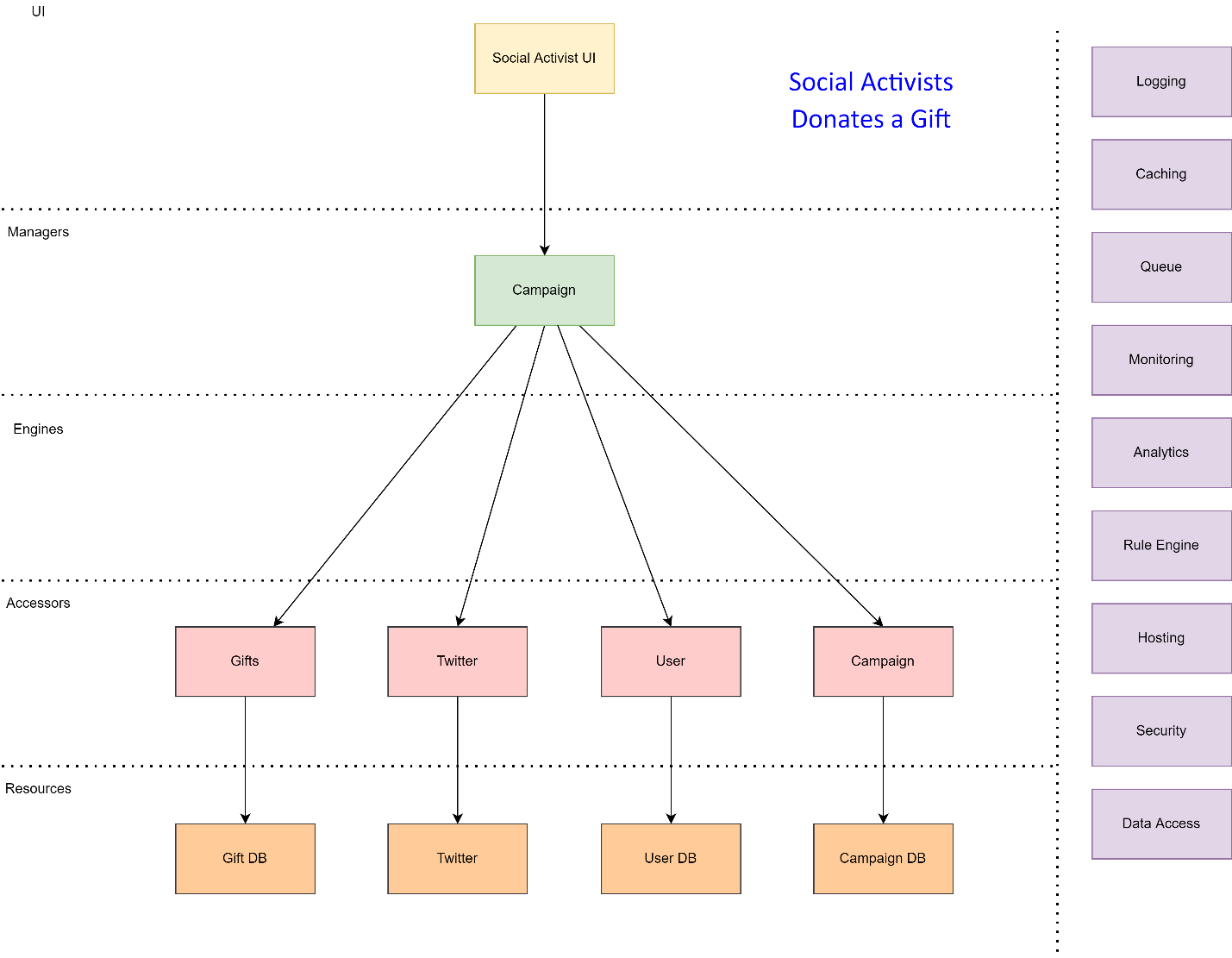


Figure 9 An activist donates a gift

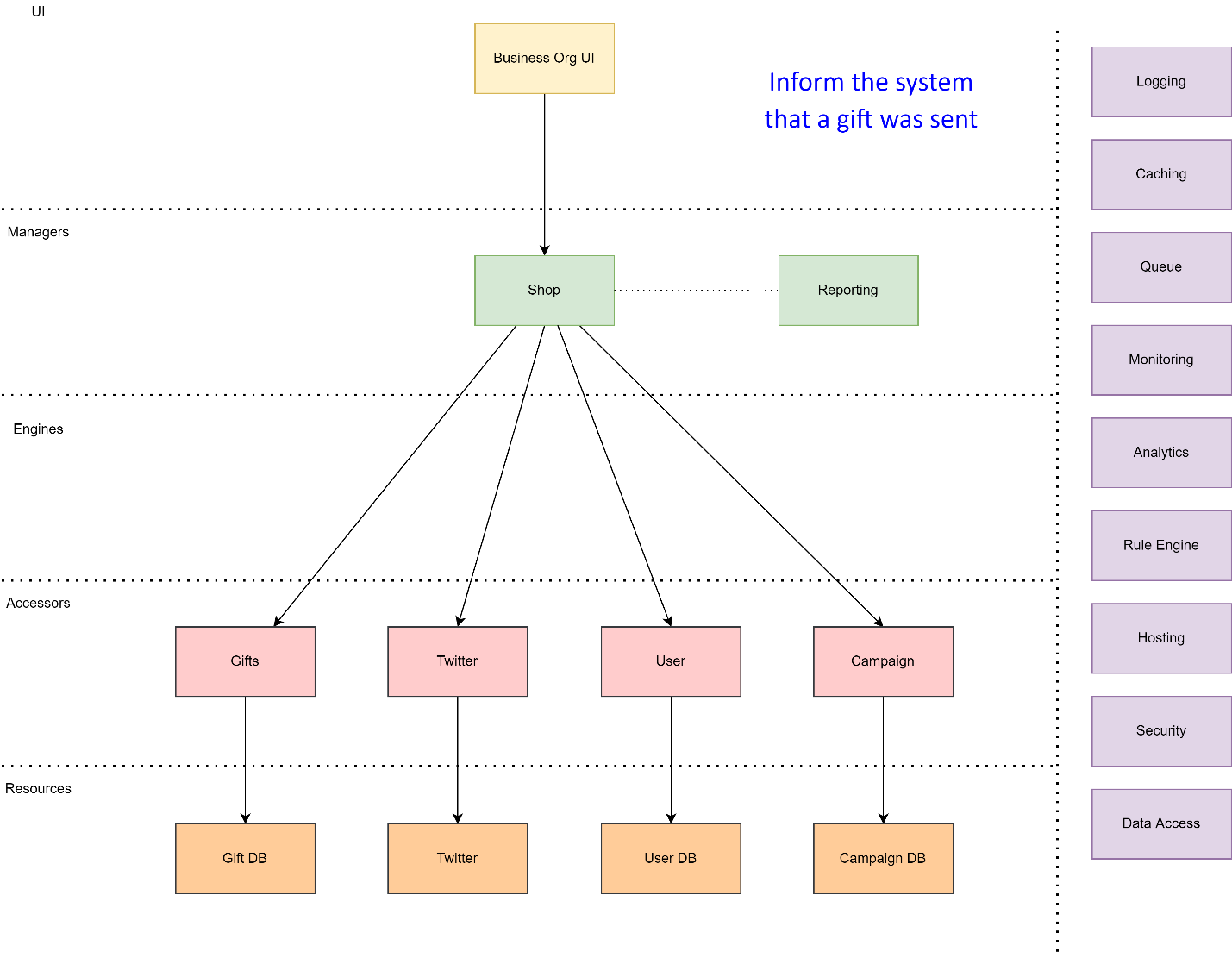


Figure 10 Send a gift

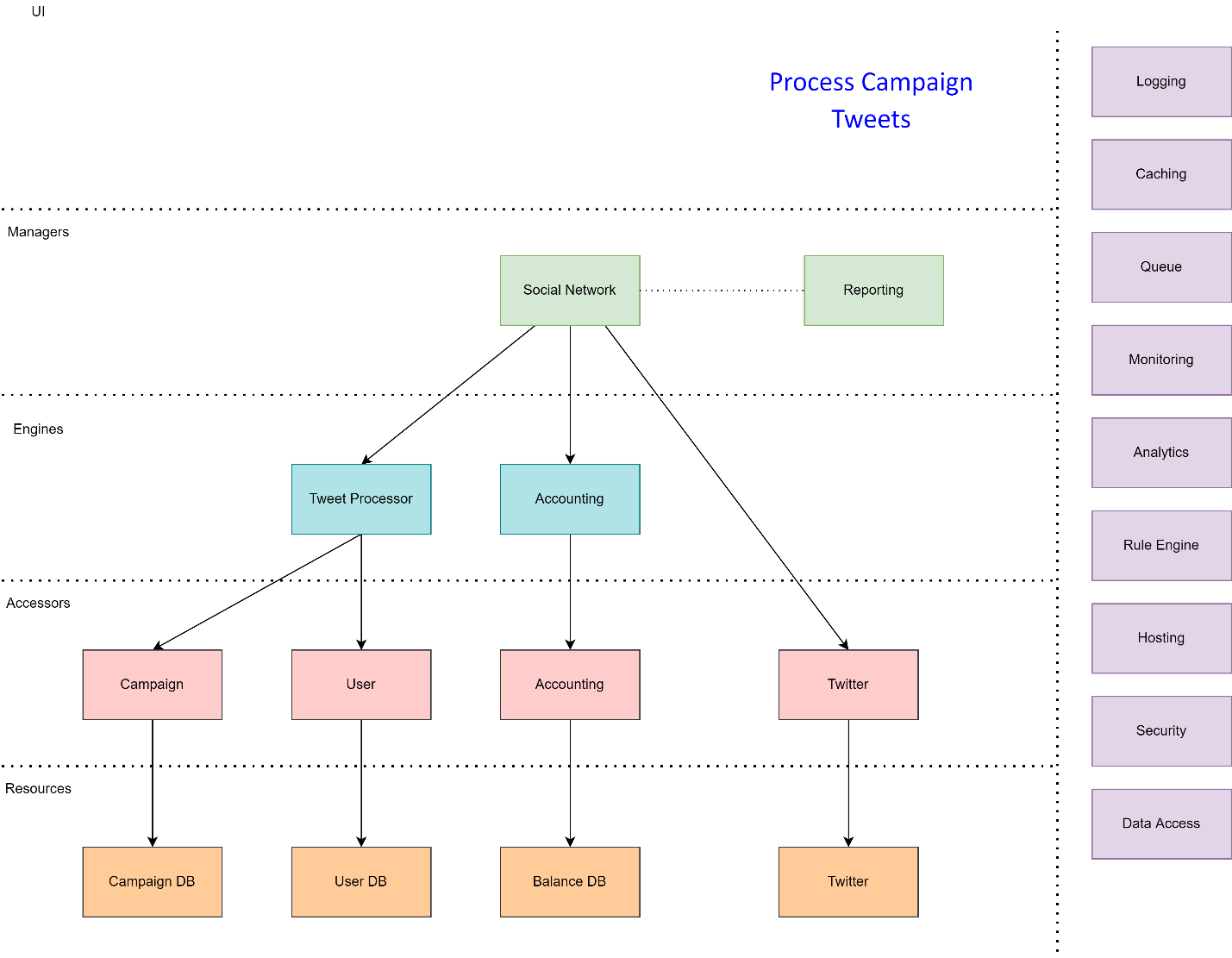


Figure 11 Process Tweets

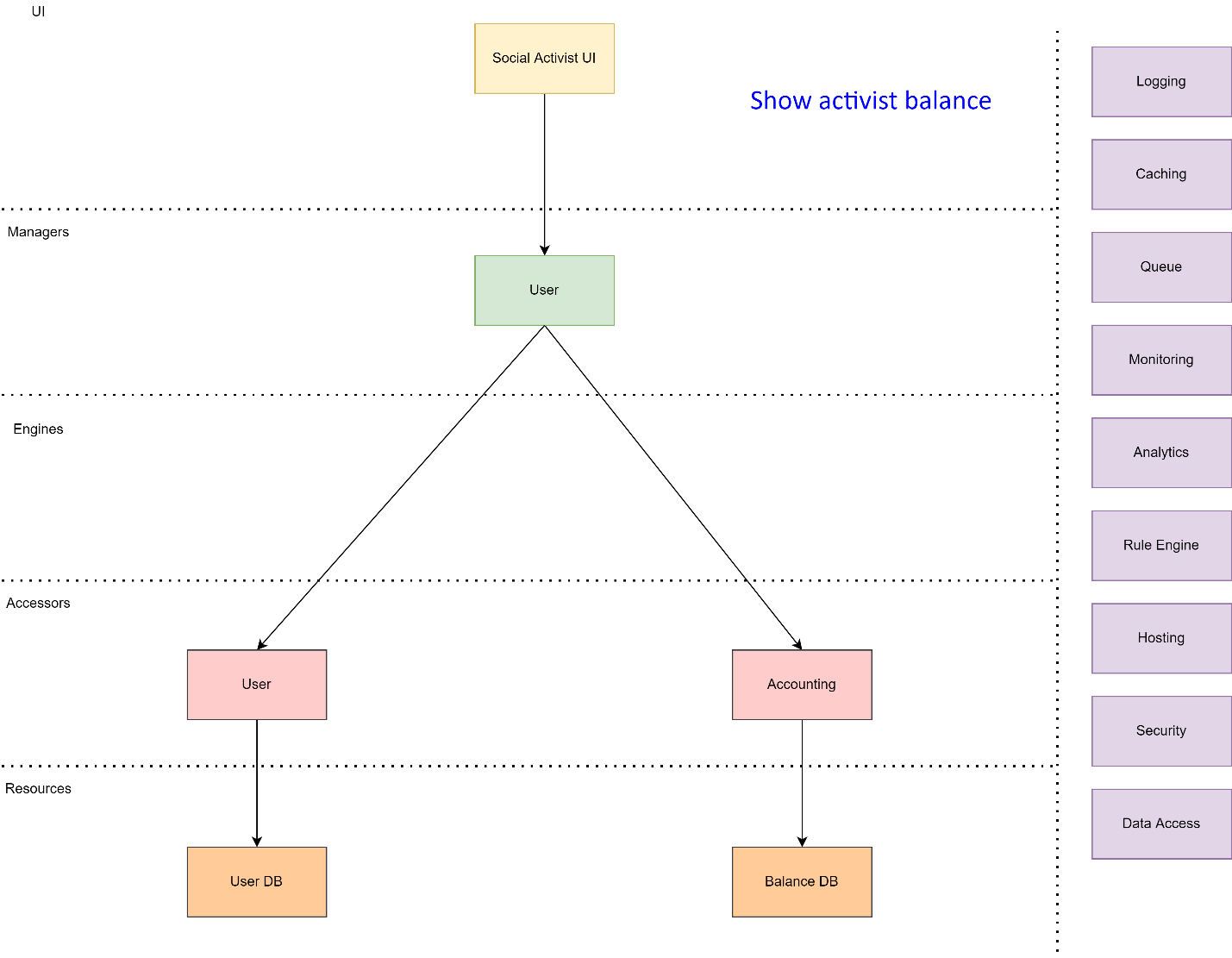


Figure 12 Show Balance

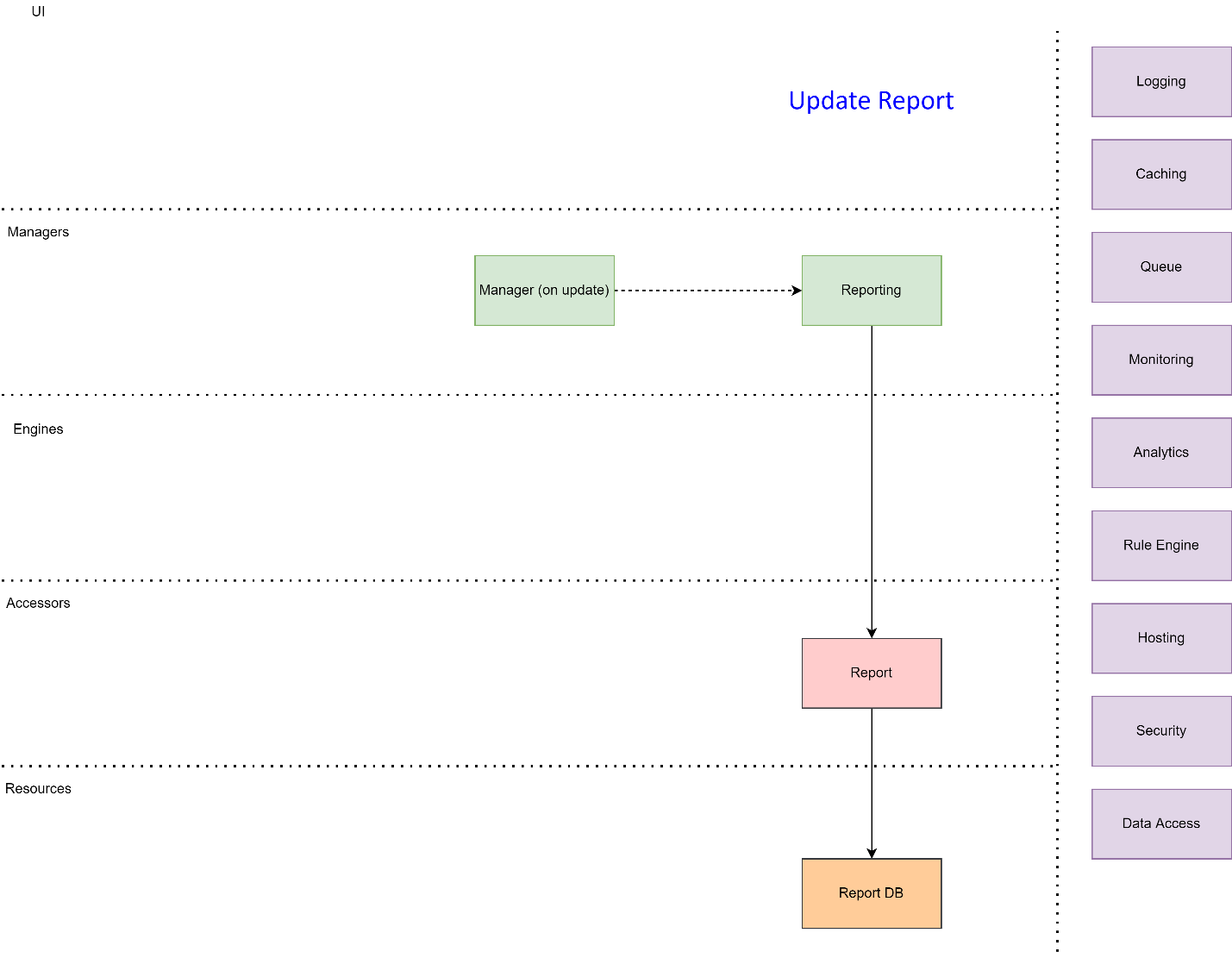


Figure 13 Update Report

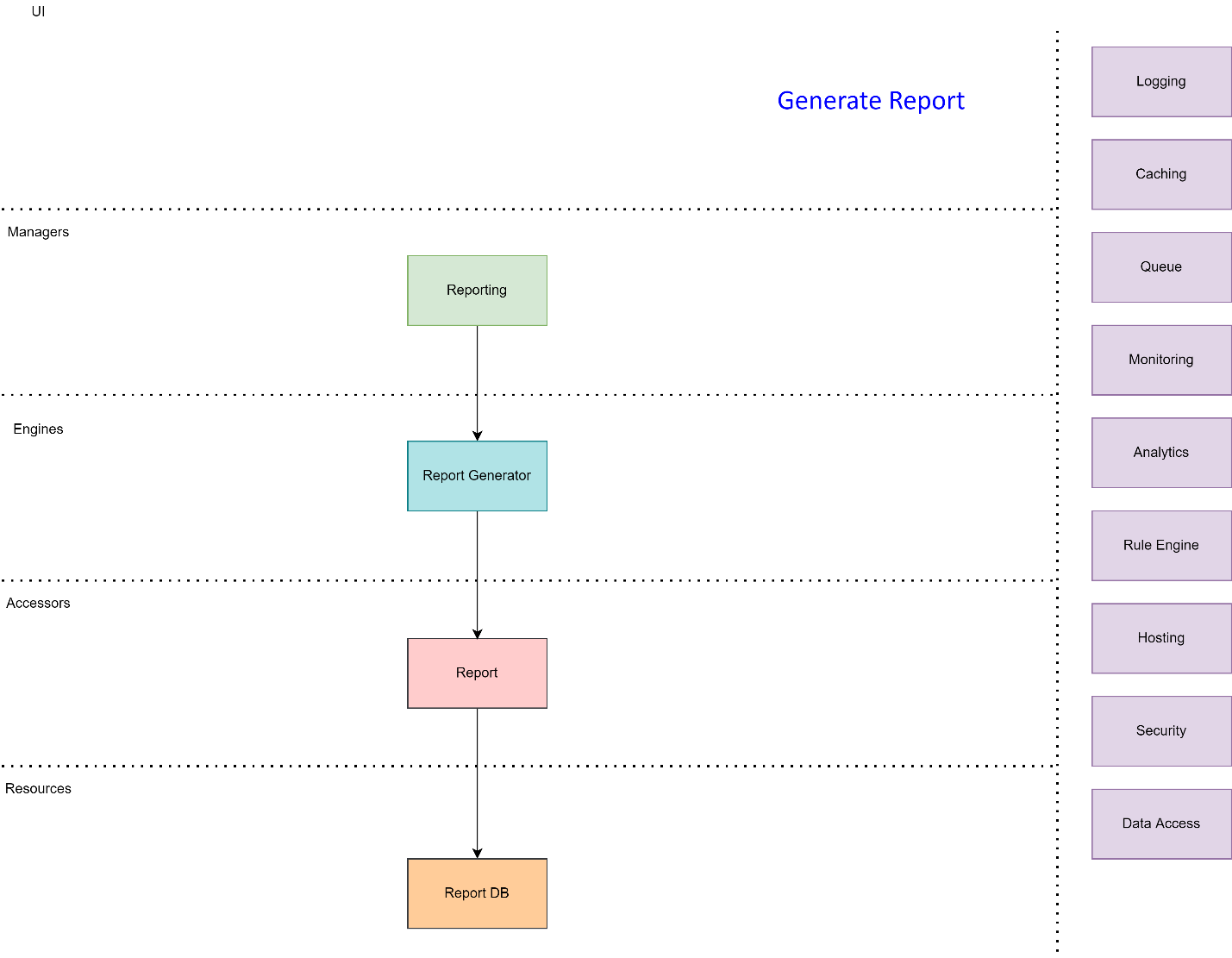


Figure 14 Generate Report

## Service Responsibilities

1. Managers
   1. User
      1. Register new users
      2. Provide the social activist balance
   2. Campaign
      1. Creates a new campaign
      2. Manages gift donations
      3. Provides the list of all running campaigns
   3. Shop
      1. Manages the shopping of a gift
      2. Publish a tweet when a product is sent to an activist
   4. Social Network
      1. Manage campaign tweets
   5. Report
      1. Manages reporting related queries
      2. Manage report generations
2. Engines
   1. URL Validation
      1. Validate that a campaign URL is valid
   2. Accounting
      1. Update the user balance per campaign
   3. Tweet Processor
      1. Validate tweet content (valid campaign hashtag and registered social activist)
      2. Extract campaign and activist information
   4. Report Generator
      1. Generates status report
3. Accessor
   1. User
      1. Create a new user in the user D.B.
      2. Provide user-related information
   2. Web Accessor
      1. Surf the web to validate URLs
   3. Campaign
      1. Campaign CRUD operations
   4. Gift Accessor
      1. CRUD for donations (goods type, amount, information, price)
      2. Mark good as shopping pending
      3. Remove the good when the business org reports that the gift was sent
   5. Accounting
      1. Keep the user balance per campaign
   6. Twitter
      1. Read campaign tweets from Twitter
      2. Create a Tweet when a gift is sent to the user or when a user donates a gift
   7. Report
      1. CRUD for reports
4. Resources
   1. User DB.
   2. The Internet
   3. Campaign DB.
   4. Gift DB.
   5. Balance DB.
   6. Twitter
   7. Report DB.

# Implementation Strategy

Since we don’t have a cloud account yet, we intend to develop the application to run on the local PC. However, we ensure that the system can be deployed in the Cloud and benefit from the Cloud-managed services and scale.

Each Get operation is served synchronously (request-reply). Likewise, each insert, update or delete message is served asynchronously.

The reporting system gets the different change events and builds the report database. Then, the user can query the report database for status requests.

The system is built for extensibility. New social networks can be added without affecting most of the system services.

The system is implemented using C# with .NET 6.0.

# Cross-cutting concerns

## Hosting

We use Azure Function Apps for our primary hosting facility.

## Logging

We use the Microsoft .NET ILogging mechanism. When the system is deployed to the Cloud, Azure Application insights gathers the ILogging information.

## Software Analytics

When the system is deployed to the Cloud, we use Azure Application Insights for all analytics aspects.

## Identity (SSO) & Security

As mentioned above, a straightforward local user name/password mechanism is implemented for the MVP.

## Manageability, Administration, and Telemetry (KPI)

For the MVP, the only management ability is to view the current system status. Then, later other administration needs can be fulfilled.

## Caching

For the MVP, caching is not needed. In case of high-performance requirements, a Redis cache can be used

## Storage (No SQL/SQL/Blob)

Most of the databases are based on CosmosDB collections. The reporting DB is based on MySql.

## Queuing

Azure Storage Queue is the mechanism for the first release.

## Inversion of Control Containers (IoC), Dependency Injection (DI) & Plugin Framework

We use the .NET Dependency Injection library.

# UI technology

For the MVP, simple Console applications serve the various users.

# Risk Mitigation

## Overview

The proposed solution is based on new technology and concepts. These new concepts answer the main functional and non-functional requirements, such as changing and updating the application functionality in an agile, fast, and controlled way. To achieve these goals, a modern development process is necessary. This comes with risks that need to be discovered, analyzed, and mitigated.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk | Probability | Severity(1-5) | Mitigation Plan | Comments |
| Dev Team new technology knowledge | 80% | 3 | Mentoring & Training. |  |
|  |  |  |  |  |
| Based on Cloud Services Simulation | 70% | 5 | Test each cloud simulation before use. Replace cloud mechanisms with on-prem if we can’t use the simulation. For example, use RabitMQ instead of Azure Storage Queue. | Since we are not deploying to Azure but want to exercise cloud services, we plan to use Cloud Services Simulators, which may behave differently than the cloud counterparts. |
| Project Duration | 30% | 3 | It is a short project. We must focus on simple solutions and make sure that we implement the MVP. |  |
|  |  |  |  |  |

Table 4 - Risk Analysis

# Summary

The ProLobby Company is seeking to build a new system – “PromoIt” that handles the social promotion of the non-profit organization and other related campaigns. We are intended to build the first version of the System – The Minimal Viable Product (MVP), following this architecture and high-level design document.

With the system that we build, users can promote social campaigns. We are proud to be part of such a project and help achieve such a great goal!