17 ноября 2018 г.

Reva P., Volobuieva O., Saukh D., Bahno O.

|  |
| --- |
|  |
| System Architecture Description |
|  |
| ProFinder |

System Architecture Description

ProFinder

**Content**

[**1.** **Introduction** 2](#_Toc532152883)

[1.1. Purpose 2](#_Toc532152884)

[1.2. Scope of application 2](#_Toc532152885)

[**2.** **Architectural Representation** 2](#_Toc532152886)

[**3.** **Logical View** 2](#_Toc532152887)

[3.1. System Structure Model 2](#_Toc532152888)

[3.2. Layered Diagram 4](#_Toc532152889)

**[4.](#_Toc532152890)****[Quality Attributes Scenario](#_Toc532152890)** [5](#_Toc532152890)

[**5.** **Design Rationale** 7](#_Toc532152891)

[**6.** **Tech Stack** 8](#_Toc532152892)

[**7.** **Human Resources** 9](#_Toc532152893)

[**8.** **Architecture Evaluation** 9](#_Toc532152894)

1. **Introduction**
   1. **Purpose**

This document presents a comprehensive architectural overview of the ProFinder application using various architectural representations to describe various aspects of the system. It is designed to collect and transfer important architectural solutions that were adopted in the system.

* 1. **Scope of application**

This description of the system architecture provides an architectural overview of the ProFinder application. The ProFinder application is developed to simplify process of finding well-quality food with sane price. This document contains an overview of the architecture, goals and constraints, system-based representation, size and performance indicators, quality problems based on quality attributes.

1. **Architectural Representation**

This document represents architecture as a series of representations; type of use, type of logic, type of process and type of deployment. This document does not have a separate implementation view. These are views on the model of the Unified Model Language (UML), developed using Diagram Designer.

1. **Logical View**
   1. **System Structure Model**

From the very beginning of the development it was clear that the only possible architectural.

Choice needs to be divided into three parts:

* For Web-based application
* IOS client
* Android client

For Web-based application, mobile clients we chose the Client-Server Architecture.

Importantly, both the Web-Based application and mobile client applications user REST for client-server communication.

This choice allows us to build the system with such features:

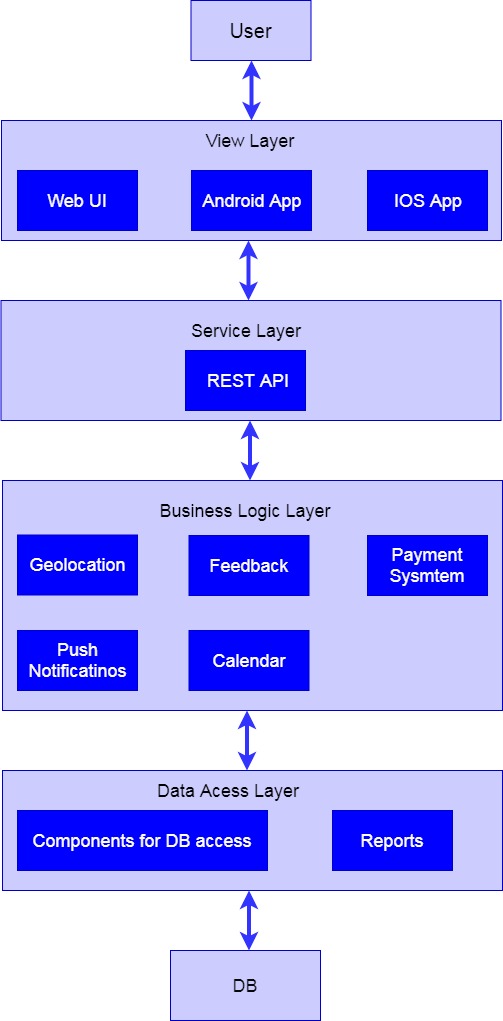
**Flexibility** – system usually will need implementation of new features so it should be flexible enough that’s fully provided by client-server decomposition

**Interoperability** – the system data should usually synchronize that is provided via connecting to centralized server

**Low Dependency** – all modules of the application must be separated from each other, so that one is updated others are not affected.



* 1. **Layered Diagram**



As we said earlier, we have independent layers in our architecture. There are 5 main layers:

**View layer** – user interface layer, all frontend part of the project, different ui controls, consist of 3 parts – web-based client, ios client, android client.

**Service layer –** interlayer between view and business logic layers. It has one main function – to transfer data between client and server.

**Business logic layer –** part of system, which encapsulate all hidden business logic, it is responsible for all calculations; prepare different statistic data and other hard work, which require large computing power. This function executes remote server.

**Data access layer –** interlayer between business logic layer and directly DB. It transfers data from DB to business logic layer to execute some calculations and operations.

**DB –** storage of user’s data and information about markets.

1. **Quality Attributes Scenario**

Portability scenario:

|  |  |
| --- | --- |
| Source of stimulus | End user |
| Stimulus | Tries to use different platforms |
| Artifact | System |
| Environment | Normal mode; overload mode |
| Response | Correct access to system data |
| Response measure | Data loss, correct data is provided |

Availability scenario:

|  |  |
| --- | --- |
| Source of stimulus | Internal for system |
| Stimulus | Crash |
| Artifact | Storages, processors |
| Environment | Normal operation |
| Response | System should detect event and do one or more of the following:   1. Record it 2. Notify appropriate parties, including the user and other systems 3. Disable sources of events that cause fault or failure according to defined rules 4. continue to operate in normal or degraded mode |
| Response measure | Repair time  Availability time |

Usability scenario:

|  |  |
| --- | --- |
| Source of stimulus | End user |
| Stimulus | Minimize impact of errors |
| Artifact | System |
| Environment | Runtime |
| Response | Wishes to cancel current operation |
| Response measure | Cancellation takes less than one second |

Security scenario:

|  |  |
| --- | --- |
| Source of stimulus | Correctly identified individual |
| Stimulus | Tries to modify information |
| Artifact | Data within system |
| Environment | Normal operation |
| Response | System maintains audit trail |
| Response measure | Correct data is restored |

Modifiability scenario:

|  |  |
| --- | --- |
| Source of stimulus | developer |
| Stimulus | Wishes to add functionality |
| Artifact | UI, platform, environment |
| Environment | Runtime, compile time |
| Response | Locate places in architecture to be modified |
| Response measure | Cost in terms of numbers of elements affected, effort, money |

Performance scenario:

|  |  |
| --- | --- |
| Source of stimulus | One of a number of independent sources |
| Stimulus | Periodic events arrive |
| Artifact | System |
| Environment | Normal mode |
| Response | Processes stimuli |
| Response measure | Latency, deadline |

1. **Design Rationale**

We adopted a 3-tiered architecture because our client, who is highly technical, wanted a very flexible design. Specifically, the users required (a) a clear separation between the user interface and the logic; and (b) a clear separation of the logic from the data storage and management. In addition, as described below, we have chosen to use an off-the-shelf DBMS, which does not does not provide custom logic.

The following list shows the 3-tiers (commonly called “layers”) of the architecture and the specific components in each tier.

* User Interface Layer
* User Interface component
* Business Logic Layer
* User Management component
* Database Management Layer
* DBMS

The three-tiered architecture clearly shows the separation between user interface and business logic and between business logic and data storage. The Business Logic layer components are broken down in such way that each component performs specific functions that do not overlap with the functions assigned to any other component.

Although the User Management and Authentication components may appear to be highly coupled, they serve different purposes and their separation allows for better integration with the other systems. As the client required that the ProFinder have a modular design, the User Management will provide services to manage user’s personal information to other systems requesting them.

1. **Tech Stack**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Server | Client | Web | CI | Git |
| Php  MySql | **IOS**  MVVM architecture pattern  Swift 4.2  Alamofire  CoreData  Kingsfisher  RxSwift  RxCocoa  Moya | React JS  Vue JS  Flexbox  CSS  HTML | Jenkins | SourceTree  Gitlab |
|  | **Android**  MVVM architecture pattern  Kotlin  Retrofit  Room  LiveData  Glide  RxKotlin |  |  |  |

1. **Human Resources**

Basing on estimated 506 hours of work, chosen architecture, technical stack, 8 hours working day and 25% risk our team can consist of the following specialists:

* 1 Backend developer (Middle+/Senior)
* 1 Android Developer (Middle)
* 1 IOS Developer (Middle)
* 3 Manual Quality Assurance Specialists (Junior+/Middle), one for each platform
* 2 UI/UX Designer(Middle)

Due to agile process and size of developer team, we also should have:

* 1 Project manager

The document describes responsibilities and skills of team members mentioned above.

1. **Architecture Evaluation**

Evaluating of the architecture relative to quality attribute goals was done with The Architecture Tradeoff Analysis Method (ATAM). The results can be found in appendix – ATAM.