POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Master of Science in Computer Science and Engineering

Design and development of a business dashboard for monitoring an outdoor augmented reality mobile app

Supervisor: Prof. Piero Fraternali
Co-supervisor: Prof.sa Elena Baralis

Master Graduation Thesis by:
Andrea Radaelli
id. 858550

Academic Year 2016/2017

Sommario

Una *business dashboard* (pannello di controllo aziendale) è uno strumento di reportistica che genera viste con indicatori di rendimento che, "a colpo d'occhio", forniscono informazioni analitiche rispetto a particolari obbiettivi e processi aziendali. Le *dashboards* supportano i manager di medio e alto alto livello a monitorare l'andamento aziendale e a prendere decisioni strategiche. PeakLens è una *mobile application* di realtà aumentata, sviluppata da un team di ricerca del Politecnico di Milano per identificare le cime delle montagne utilizzando nuove tecniche di visione artificiale e deep learning. Fin dal suo lancio nel dicembre 2016 nel Google Play Store, PeakLens ha attratto il crescente interesse della comunità degli appassionati di montagna (127.000 download su Android a novembre 2017). Questa tesi documenta la progettazione e lo sviluppo della *PeakLens Dashboard*. Gli obbiettivi della *dashboard* sono: (1) concentrare i dati spazio-temporali provenienti dalle diverse sorgenti "dell'ecosistema" di PeakLens in un unico posto; (2) generare rappresentazioni grafiche degli indicatori di rendimento per supportare l'analisi. Questi obbiettivi sono stati raggiunti creando una *data warehouse* come archivio centrale e un'applicazione web per generare grafici e diagrammi interattivi degli indicatori di rendimento. La progettazione della dashboard è stata ottimizzata per i requisiti di PeakLens, ma è abbastanza generale per supportare i bisogni di qualsiasi *mobile application* caratterizzata da dati di utilizzo geolocalizzati dove un'analisi spazio-temporale è necessaria.

Parole chiave: business intelligence, monitoring, data warehouse, mobile applications, ingegneria del software

Abstract

A business dashboard is a reporting tool that provides at-a-glance views of key performance indicators (KPI) relevant to a particular objective or business process. Dashboards support the mid and higher management to monitor business trends and make strategic decisions. *PeakLens* is an outdoor augmented reality mobile app, developed by a research team of Politecnico di Milano for identifying mountain peaks using novel computer vision and deep learning techniques. Since its public launch in December 2016 on the Google Play Store, PeakLens attracted the interest of a growing community of mountain enthusiasts (127.000 downloads on Android as of November, 2017). This thesis work reports the design and implementation of the *PeakLens Dashboard*. The goals of the dashboard are to: (1) concentrate the spatio-temporal data coming from the different sources of the PeakLens "ecosystem" in one place; (2) generate visual presentations of the key performance indicators to support analysis. These goals have been accomplished by creating a *data warehouse* as central repository and a web application for presenting interactive charts and diagrams of the KPI. The design of the dashboard has been optimized for the requirements of PeakLens, but is general enough to serve the needs of any mobile application that features geo-referenced and timestamped usage data and where the analysis of the spatial and temporal spread of usage is required.

Keywords: business intelligence, monitoring, data warehouse, mobile applications, software engineering

Acknowledgments

I would like to thank Prof. Piero Fraternali for giving me the opportunity of working on this project and Chiara Pasini for her support during the preparation of this thesis. I would also like to thank all my friends and colleagues who have given moral support.

The biggest thanks go to my family who sustained and encouraged me during my formative years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter describes the context and motivations that led to the development of this thesis, the problem that the thesis tackles, and an outline of the thesis structure.

1.1 PeakLens

PeakLens [1] is an augmented reality mobile application for identifying mountain peaks in real time through the device camera of the smartphone or tablet that is running it. Figure 1.1 shows how PeakLens looks like when it is running.The mobile application has been developed by a research team at Politecnico di Milano using a custom deep learning algorithm [2] that identifies mountain peaks with high precision and is resilient to errors from the compass and GPS sensor.

PeakLens has been released on the Google Play Store [3] in December 2016. Since then it has seen great success by reaching 127.000 downloads as of November 2017. The research team behind PeakLens recognizes the opportunity behind a successful application and foresees the possibility to create a startup or an open source crowdfunded initiative.

An essential need for a startup (and any enterprise) is the regular monitoring of the metrics pertaining the business activities. The monitoring activity helps to support the internal decision making process and shows the value of the startup externally to potential investors.

Operatively speaking, the monitoring activity consists of: collecting, storing, and analyzing data over time [4].

Data collection is the process of gathering data from the designated sources in a systematic fashion. The sources of data in the context of PeakLens are: the PeakLens applications itself which logs the user activities,

Figure 1.1: Screenshot of PeakLens

and the digital marketplaces where PeakLens is distributed (now only the Google Play Store but the Apple App Store will also be used in the future). The digital marketplaces already offers monitoring services but their scope is limited to the marketplace itself and do not integrate the other sources.

The storing of data consists in recording the gathered data on an isolated medium that would not be influenced by any change or failure of the sources. This step is needed because we cannot rely on customer devices or the digital marketplaces to maintain the data forever.

The analysis of the data consists in creating *at-a-glance* views which allow the analysts to understand the current performances. The views are composed of key performance indicators (KPIs), value drivers, and other metrics relevant to the specific business of mobile applications. In business terms this kind of views are called *dashboards*.

1.2 Business Dashboard

Technically, a dashboard is computer program (usually a web application) that generates reports. The dashboard is linked to a database containing all the relevant data, updated constantly, and the reports are dynamically generated to show up-to-date measurements. A business dashboard for PeakLens would allow to understand the adoption level of the application, its popular-

ity, the usage; offer visual presentation of performance measures, help in the identification of trends and usage patterns. See Figure 1.2 for a screenshot of how a typical dashboard looks like.

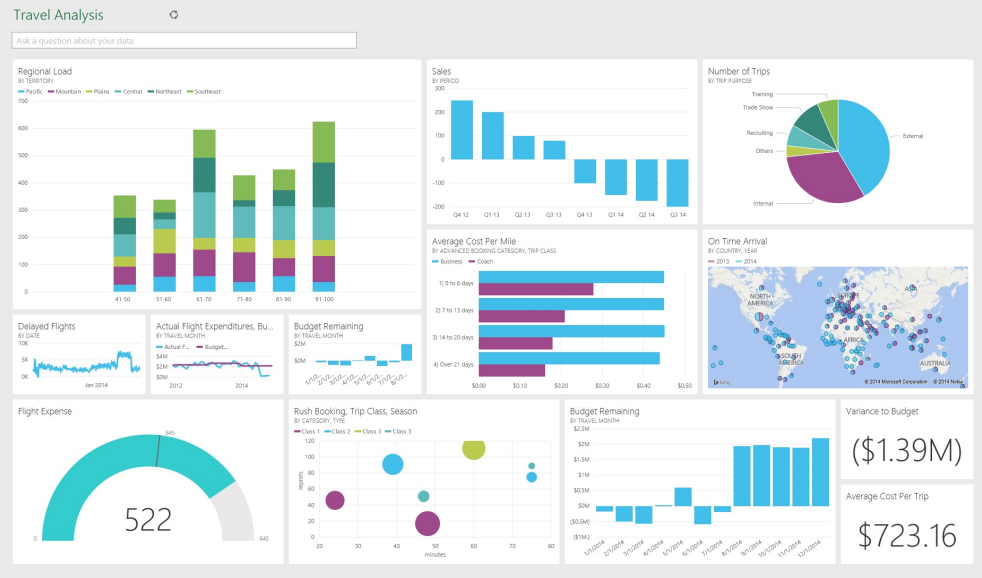


Figure 1.2: Screenshot of a commercial business dashboard

1.3 Application Scenarios

The application scenarios where the PeakLens Dashboard will be used are:

- Monitor the usage of PeakLens.
- Monitor the competitors of PeakLens.

The first scenario is related to the analysis of customer behavior. Consumer behavior is the study of consumers and the processes they use to choose, use (consume), and dispose of products and services to satisfy needs and desires [5]. The goal of the scenario is to allow the research team to understand how the customers are using PeakLens, what their experience is like, and how they are impacted by issues. This knowledge will directly influence the development of PeakLens and the marketing decisions.

The second scenario is related to marketing strategy. Marketing strategy has the fundamental goal of increasing dissemination (e.g., sales and/or

downloads) and achieving a sustainable competitive advantage. The dashboard will support this goal by providing information about the status of competitors of PeakLens in the market.

1.4 Project objectives

The main objective of this project are:

- Concentrate the spatio-temporal data coming from the different sources of the PeakLens ecosystem in one place.
- Generate visual presentations of the key performance indicators to support analysis.

The first objective enables: easy access and querying of the data by storing it in a single place, maintaining data history in the event of change or loss of a part of the ecosystem, improving data quality by providing consistent codes and descriptions, provide a single data model of all interest data regardless of the data source. The goal will be achieved through the construction of a *data warehouse*.

The second objective is related to the field of *visual analytics* which is the science of analytical reasoning facilitated by visual interfaces that amplify human cognitive capabilities to better transmit information between the user and the machine [6]. This goal will be achieved by creating custom interactive visualizations for each metric.

1.5 Thesis Outline

Chapter 2 presents the user groups, then use cases with the functional and non-functional requirements derived from them. Chapter 3 presents the conceptual and logic data model which formalizes the objects and the relationships used in the PeakLens Dashboard. Chapter 4 presents the high level structure of the application by defining the software components and their connections. Chapter 5 presents how the application has been realized, the challenges that has been encountered, and how to deploy it on a server. Chapter 6 presents the conclusions of this thesis.

Chapter 2

Software Requirements

This chapter contains the specifications of the system to be developed. The *use case driven approach* [7] has been used to draft the specifications. The methodology consists in: defining the target user groups of the system, exploring their needs and defining the use cases, deriving the functional and non-functional requirements from each use case.

2.1 Target Groups

The Table 2.1 describes the users who can access the services provided by the PeakLens Dashboard.

User group	**Description**
Admin	Users in charge of managing the system
Marketing Analyst	Users who can access the dashboard to monitor the performances of PeakLens

Table 2.1: List of the user groups

2.2 Use Cases and Derived Requirements

The Table 2.2 gives an overview of the use cases provided by the system. Each use case is described in detail in the following sub-sections along with the derived functional and non-functional requirements.

#	**Use case**
	PeakLens dashboard
1	Visual exploration of active users
2	Visual exploration of area downloads
3	Visual exploration of installs/uninstalls
4	Visual exploration of ratings
5	View engagement metrics
6	View API calls summary
	Competitors dashboard
7	See ranking of PeakLens against competitors
8	See competitors' app changelogs over time
9	Visual exploration of competitor's app reviews
	Admin dashboard
10	Set relevant events
11	Set competitor apps to scrape
12	Enable/disable alerts on PeakLens metrics
13	Enable/disable alerts of ranking changes
14	Enable/disable alerts of new possible competitors
15	Manually query Logger data

Table 2.2: List of the use cases

2.2.1 Use Cases of PeakLens Dashboard

This set of use cases describes the interaction of the user with the PeakLens dashboard to inspect the performance metrics of PeakLens. Table 2.3 presents the high-level functional requirements that explain the core functionality of the PeakLens dashboard. The requirements will be elaborated in the following sub-sections, which describe the separate use cases. Table 2.4 lists the derived high-level non-functional requirements.

#	**High-level functional requirement**
1	The system should be able to query and store the logs of PeakLens daily
2	The system should be able to download and store the statistics from Google Play Store of PeakLens daily
3	The system should be able to download and store the statistics from Apple App Store of PeakLens daily
4	The system should be able to visualize generic time series in a line chart
5	The system should be able to highlight the stored custom events on the line chart
6	The system should be able to highlight the weekends on the line chart
7	The system should allow to zoom the line charts
8	The system should be able to visualize generic values with GPS coordinates associated data in a heat map
9	The system should show the values on the heat map colored in a logarithmic scale
10	The system should allow to zoom the heat maps
11	The system should allow to create time lapse of the heat maps
12	The system should allow to filter generic values by their characteristics

Table 2.3: High-level functional requirements of the PeakLens dashboard

#	**High-level non-functional requirement**
NF1	The system should be usable intuitively from a non-technical audience
NF2	The system should be well documented and described
NF3	The system should have a response time of less than 5 seconds
NF4	The system should not expose any private data about the users of PeakLens

Table 2.4: High-level non-functional requirements of the PeakLens dashboard

2.2.2 Use case: Visual exploration of active users

Purpose	Visual exploration of the active users of PeakLens
Pre-condition	The database is populated with data from the *Logger*
Post-condition	The active users are visualized
Workflow	1. The user opens the website 2. The user enters the "PeakLens" area 3. The user clicks on the "Active Users" tab 4. The user chooses the visualization type: line chart or heat map

Table 2.5: Use case: Visual exploration of active users

#	**Functional requirement**
13	The system should be able to visualize the user activities on a line chart as a time series
14	The system should be able to visualize the user activities on a world map as a heat map
15	The system should visualize the user activities in the heat map grouped as rectangles
16	The system should use the *renders* from the Logger to create the heat map and the line chart
17	The system should allow the user to play a time lapse of the user activities on the heat map
18	The system should allow the user to choose the time interval for the time lapse

Table 2.6: Derived requirements of the use case "Visual exploration of active users"

2.2.3 Use case: Visual exploration of area downloads

Purpose	Visual exploration of the area downloads
Pre-condition	The database is populated with data for the selected time frame
Post-condition	The area downloads are visualized
Workflow	1. The user opens the website 2. The user enters the "PeakLens" area 3. The user clicks on the "Area Downloads" tab

Table 2.7: Use case: Visual exploration of area downloads

#	**Functional requirement**
19	The system should be able to visualize the amount of area downloads on a world map as a heat map
20	The system should be able to visualize the area downloads grouped by the areas
21	The system should be allow the user to toggle the visibility of specific areas

Table 2.8: Derived requirements of the use case "Visual exploration of area downloads"

2.2.4 Use case: Visual exploration of installs/uninstalls

Purpose	Visual exploration of the installs and uninstalls of the PeakLens app
Pre-condition	The installs and uninstalls are visualized
Post-condition	The area downloads are visualized
Workflow	1. The user opens the website 2. The user enters the "PeakLens" area 3. The user clicks on the "Installs/Uninstalls" tab

Table 2.9: Use case: Visual exploration of installs/uninstalls

#	**Functional requirement**
22	The system should be able to visualize the amount of installs and uninstalls on a line chart
23	The system should allow to filter the source data of the series on the line chart

Table 2.10: Derived requirements of the use case "Visual exploration of area downloads"

2.2.5 Use case: View engagement metrics

Purpose	Inform the user about the customer engagement in the PeakLens app
Pre-condition	The database is populated with data for the selected time frame
Post-condition	The installs and uninstalls are visualized
Workflow	1. The user opens the website 2. The user enters the "PeakLens" area 3. The user clicks on the "Engagement Metrics" tab

Table 2.11: Use case: View engagement metrics

#	**Functional requirement**
24	The system should be able to visualize the engagement metrics on different line charts
25	The system should compute the *Customer Retention Rate* metric (installs - uninstalls)
26	The system should compute the *Average active users* metric
27	The system should compute the *Bounce Rate* metric (people who use X days then never)
28	The system should allow the user to set the time window for the Bounce Rate metric
29	The system should compute the *Number of Sessions per User* (a session consists in the usage of the app after X time of inactivity)
30	The system should allow the user to set the time for inactivity for the Number of Sessions per User

Table 2.12: Derived requirements of the use case "View engagement metrics"

2.2.6 Use case: View API calls summary

Purpose	Inform the user about the amount of calls made to the PeakLens back-end
Pre-condition	The database is populated with data for the selected time frame
Post-condition	The number of API calls are visualized
Workflow	1. The user opens the website 2. The user enters the "PeakLens" area 3. The user clicks on the "API Calls" tab

Table 2.13: Use case: View API calls summary

#	**Functional requirement**
31	The system should be able to visualize the amount of different API calls on a line chart with a logarithmic Y axis
32	The system should allow the user to filter for the specific API call

Table 2.14: Derived requirements of the use case "View API calls summary"

2.2.7 Use Cases of Competitors Dashboard

This set of use cases describes the interaction of the user with the competitors dashboard which provides an overview of the status the competitors' applications of PeakLens. Table 2.15 presents the high-level functional requirements that explain the core functionality of the Competitors dashboard. The requirements will be elaborated in the following sub-sections, which describe the separate use cases. Table 2.16 lists the high-level non-functional requirements.

#	**High-level functional requirement**
33	The system should be able to scrape the Google Play Store for data of published applications daily
34	The system should be able to scrape the Apple App Store for data of published applications daily

Table 2.15: High-level functional requirements of the competitors dashboard

#	**High-level non-functional requirement**
NF5	The system should be fault-tolerant to changes regarding to changes of the scraped websites

Table 2.16: High-level non-functional requirements of the competitors dashboard

2.2.8 Use case: See ranking of PeakLens against competitors

Purpose	The user can see the ranking of PeakLens against competitors' apps
Pre-condition	The database is populated with data about the competitors' apps
Post-condition	The user has visualized the rankings
Workflow	1. The user opens the website 2. The user enters the "Competitors" area 3. The user clicks on the "Rankings" tab

Table 2.17: Use case: See ranking of PeakLens against competitors

#	**Functional requirement**
35	The system should be able to show the rankings of the tracked apps a table table
36	The system should be able to sort the rankings
37	The system should be able to show the competitors' apps average ratings

Table 2.18: Derived requirements of the use case "See ranking of PeakLens against competitors"

2.2.9 Use case: See competitors' app changelogs over time

Purpose	The user can see the different changelogs of the competitors' app
Pre-condition	The database is populated with data about the competitors' apps
Post-condition	The user has visualized the changelogs
Workflow	1. The user opens the website 2. The user enters the "Competitors" area 3. The user clicks on the "Changelogs" tab

Table 2.19: Use case: See competitors' app changelogs over time

#	**Functional requirement**
38	The system should be able to show when the competitors' changelogs changed in a time line
39	The system should be able to show the content of the changelog when the users click on the time line

Table 2.20: Derived requirements of the use case "See competitors' app changelogs over time"

2.2.10 Use case: Visual exploration of competitor's app reviews

Purpose	The user can view the competitor's app reviews as a table
Pre-condition	The database is populated with data about the competitors' apps
Post-condition	The user has visualized the reviews
Workflow	1. The user opens the website 2. The user enters the "Competitors" area 3. The user clicks on the "Reviews" tab

Table 2.21: Use case: Visual exploration of competitor's app reviews

#	**Functional requirement**
40	The system should be able to show the competitors' apps reviews in a table
41	The system should be able to sort the reviews by most recent
42	The system should be able to highlight the reviews that got an answer by the developer of the app

Table 2.22: Derived requirements of the use case "See competitors' app changelogs over time"

2.2.11 Use Cases of Admin Dashboard

This set of use cases describes the interaction of the user with the admin dashboard which exposes the functionalities to: define relevant events of the development of PeakLens, set the external mobile app defined as competitors, enable or disable system alerts, manually query the PeakLens backed for new data. The requirements will be elaborated in the following sub-sections, which describe the separate use cases.

2.2.12 Use case: Set relevant events

Purpose	The user can set relevant events about the development/history of PeakLens
Pre-condition	-
Post-condition	The user has set some events
Workflow	1. The user opens the website 2. The user enters the "Admin" area 3. The user clicks on the "Events" tab 4. The user creates a custom event

Table 2.23: Use case: Set relevant events

#	**Functional requirement**
43	The system should be able to store the custom events in the database
44	The system should be able to show the custom events as a table
45	The system should allow the user to edit the custom events

Table 2.24: Derived requirements of the use case "Set relevant events"

2.2.13 Use case: Set competitor apps to scrape

Purpose	The user can set the competitors' app names to scrape
Pre-condition	-
Post-condition	The system will regularly scrape the data of the user set app
Workflow	1. The user opens the website 2. The user enters the "Admin" area 3. The user clicks on the "Apps to scrape" tab 4. The user sets the names of the apps to scrape

Table 2.25: Use case: Set competitor apps to scrape

#	**Functional requirement**
46	The system should be able to scrape the app data from the Google Play Store
47	The system should be able to scrape the app data from the Apple App Store
48	The system should scrape data regularly and store it in the database

Table 2.26: Derived requirements of the use case "Set competitor apps to scrape"

2.2.14 Use case: Enable/disable alerts on PeakLens metrics

Purpose	The user can set email alerts for any PeakLens metrics
Pre-condition	The database is updated regularly with log data from PeakLens
Post-condition	If the condition of an alert are meet, an email is sent to the user
Workflow	1. The user opens the website 2. The user enters the "Admin" area 3. The user clicks on the "Alerts" tab 4. The user creates a custom alert 5. The user specifies the email address for receiving the alert

Table 2.27: Use case: Enable/disable alerts on PeakLens metrics

#	**Functional requirement**
49	The system should be able to store the custom alerts in the database
50	The system should support an alerts to be set on all the measures exposed in the PeakLens and Competitors dashboard
51	The system should support an alert about the sudden drop/growth of a specific measure
52	The system should support an alert about the reaching of a given threshold of a specific measure
53	The system should check regularly if the alerts have been triggered
54	The system should send an email to the user if an alert is triggered
55	The system should allow the user to edit the custom alerts on PeakLens metrics

Table 2.28: Derived requirements of the use case "Enable/disable alerts on PeakLens metrics"

2.2.15 Use case: Enable/disable alerts of ranking changes

Purpose	The user can can enable or disable alerts related to the ranking of PeakLens against competitors
Pre-condition	The database is regularly updated with data from the competitors' apps
Post-condition	If the rank of PeakLens changes, an email is sent to the user
Workflow	1. The user opens the website 2. The user enters the "Admin" area 3. The user clicks on the "Alerts" tab 4. The user enables or disables the rank alert

Table 2.29: Use case: Enable/disable alerts of ranking changes

#	**Functional requirement**
56	The system should regularly check the rank of PeakLens and the competitors' apps
57	The system should send and an email if the rank of PeakLens changes

Table 2.30: Derived requirements of the use case "Enable/disable alerts of ranking changes"

2.2.16 Use case: Enable/disable alerts of new possible competitors

Purpose	The user can enable or disable alerts related to the presence of new competitors
Pre-condition	The database is regularly updated with data from the app stores suggestions
Post-condition	The alert of new possible competitors is enabled
Workflow	1. The user opens the website 2. The user enters the "Admin" area 3. The user clicks on the "Alerts" tab 4. The user enables or disables the new possible competitors alert

Table 2.31: Use case: Enable/disable alerts of new possible competitors

#	**Functional requirement**
58	The system should regularly check the Google Play Store suggestions of similar apps of PeakLens
59	The system should regularly check the Apple App Store suggestions of similar apps of PeakLens
60	The system should send and an email if a new suggested similar app of PeakLens appears

Table 2.32: Derived requirements of the use case "Enable/disable alerts of new possible competitors"

2.2.17 Use case: Manually query Logger data

Purpose	The user can trigger a query of the PeakLens logs manually within the dashboard
Pre-condition	The system can import the data from the PeakLens *Logger*
Post-condition	The database is populated with the latest PeakLens logs
Workflow	1. The user opens the website 2. The user enters the "Admin" area 3. The user clicks on the "Logger" tab 4. The user click the "Manually query the log" button

Table 2.33: Use case: Manually query Logger data

#	**Functional requirement**
61	The system should allow the user to trigger the query of the Logger data

Table 2.34: Derived requirements of the use case "Manually query Logger data"

2.3 Highlight: key performance indicators

The Table 2.35 contains a list of the key performance indicators identified in the previous requirements.

KPI	Description
Active users	Absolute number of active users
Area downloads	Absolute number of area downloads
Installs & Uninstalls	Absolute number of installs & uninstalls of PeakLens
API calls	Number of API calls made to the application tier[1]of the dashboard
Average active users	Average number of user that use PeakLens at least one time in the reference period
Customer retention rate	Percentage of new users that keep using PeakLens relative to the number at the start of the reference period
Bounce rate	Percentage of new users that use PeakLens only in the reference period and never again
Average number of sessions	Average number of sessions[2]per user in the reference period
PeakLens ranking	Ranking of PeakLens against competitors

Table 2.35: List of key performance indicators

2.4 Functional Scope

The functional scope represents the set of requirements which are taken in consideration. The requirements defined in this chapter cover all the use case of the dashboard; however because the time required to design and implement them all would exceed the time available for delivering this thesis, it has been decided to reduce the functional scope.

The functional scope assumed from now on is limited to:

[1]See Chapter 4

[2]A session consist in the use of PeakLens in a specific location continually

- The functional requirements 1-23 (num. 3 excluded since PeakLens is not is not available on the Apple App Store at this time).
- The non-functional requirements 1-4.

These requirements corresponds to the use cases 1-4 (see Table 2.2 for the list of the use cases).

Chapter 3

Data warehouse model

This chapter describes the data model of the data warehouse.

3.1 Conceptual model

The conceptual model is modeled as a *multidimensional data model* [8]. A multidimensional data model is composed of three concepts: facts, measurements, and dimensions. Facts are events of the normal business activities on which is interesting to perform some analysis. A measure is an atomic property of a fact. A dimension is group of measures that define the facts.

The model is represented with the entity-relationship (ER) notation. ER models are composed by entities which classifies the things of interest and by the relationships that exists between instances of those entities.

The conceptual model maps entities and relationship around the needs of the business. It is built with the assumption that all the information that are pertinent to the business are available; also the database limitations in terms of structure or complexity of data are not considered.

The facts that make up the conceptual model of the PeakLens Dashboard are grouped in three categories: the Logger, the Google Play Store, and the Dashboard.

3.1.1 Logger

The Logger model represents all the facts that are logged by the PeakLens application itself. These facts are the direct result of the user interacting with the mobile application. The facts recorded are:

Render: represents a situation where the user has opened PeakLens and has used it to identify the peaks around himself. A render is identified

by the latitude and longitude where it has occurred.

Area list: represents a request of the user about the available areas where PeakLens is able to work.

Area download: represents the download action of an user of a PeakLens area which contains a list of patches.

Patch download: represents the download action of an user of a PeakLens patch. A patch is an offline map containing information about the terrain geometry and the peaks. A patch is identified by the latitude and longitude of its position.

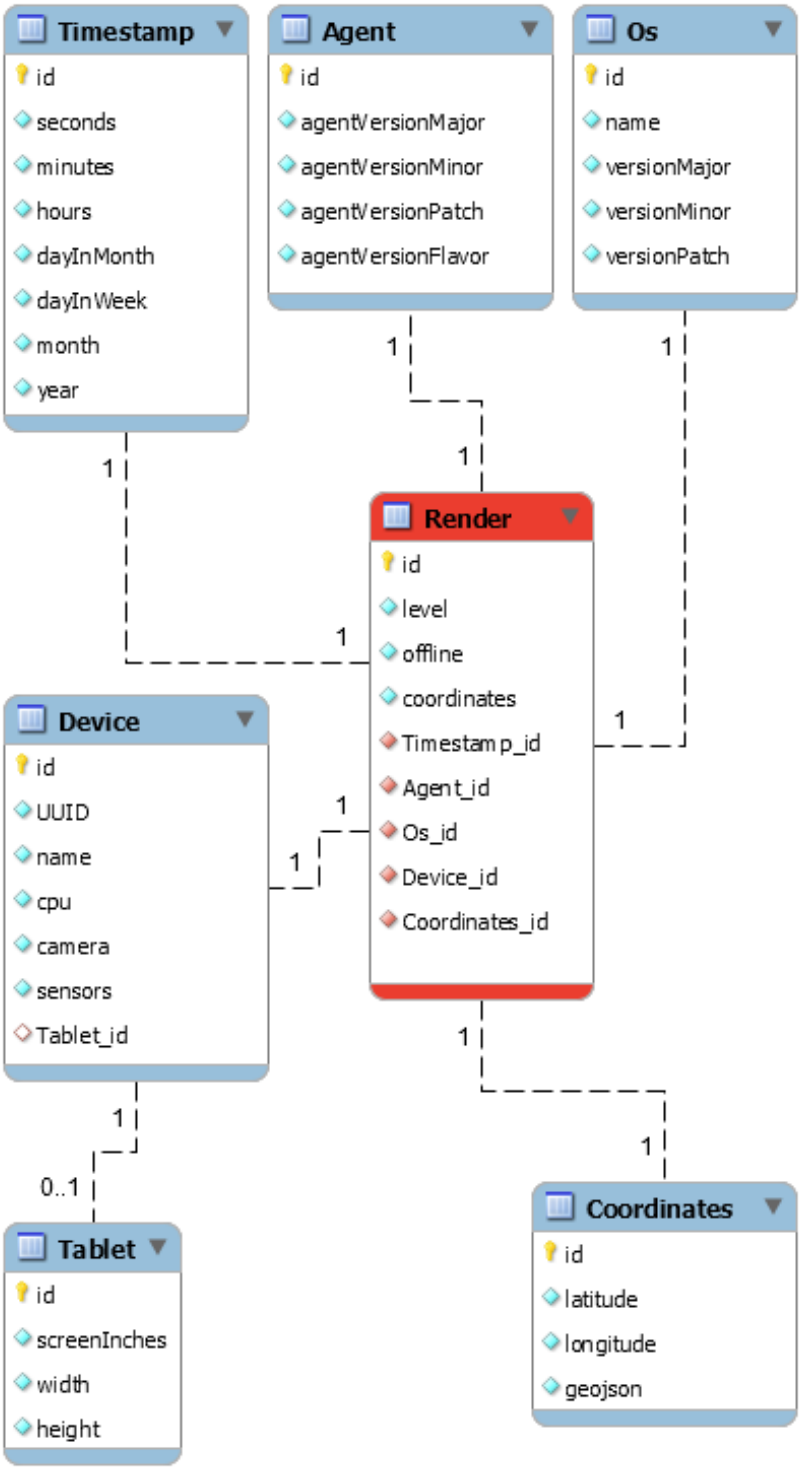


Figure 3.1: Conceptual data model of the "render" fact

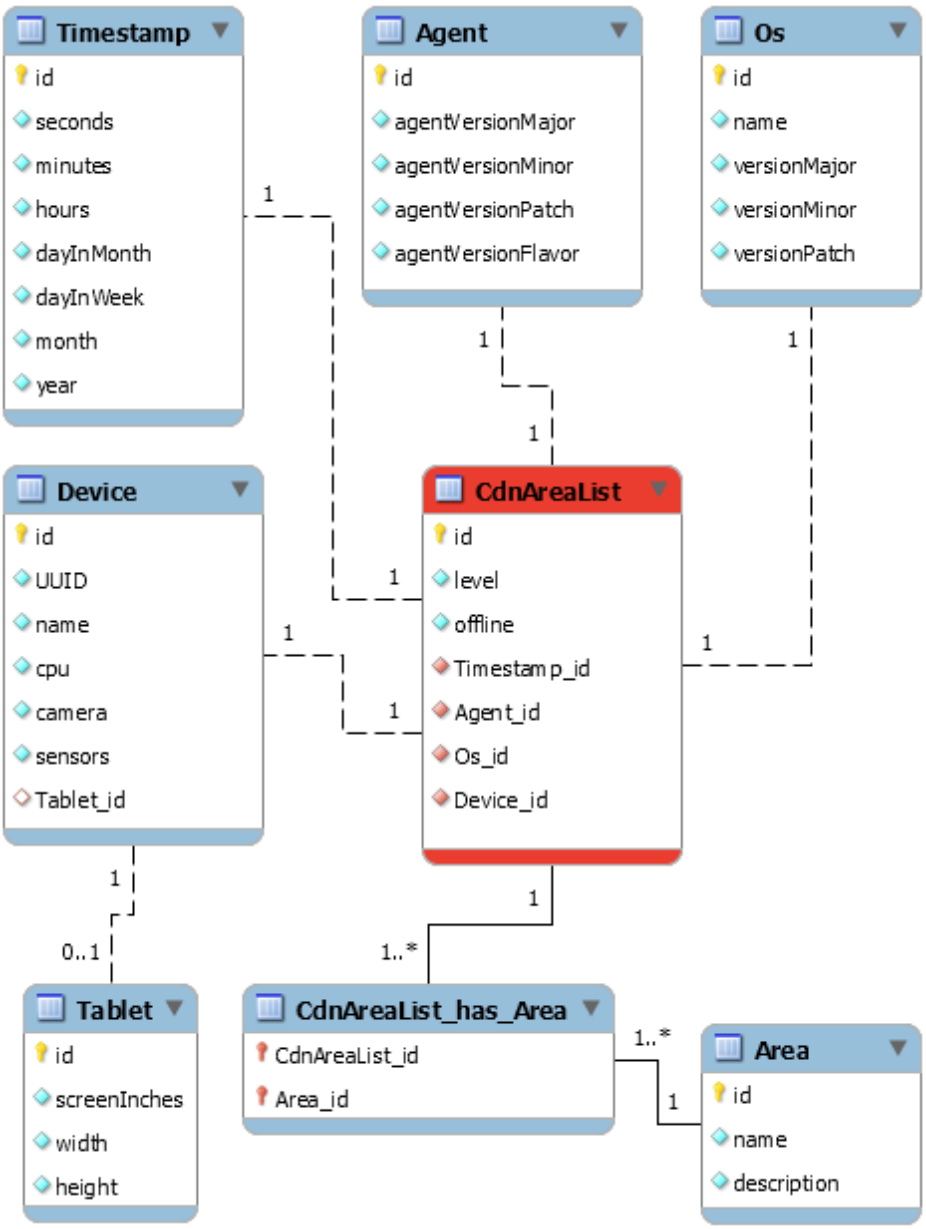


Figure 3.2: Conceptual data model of the "area list" fact

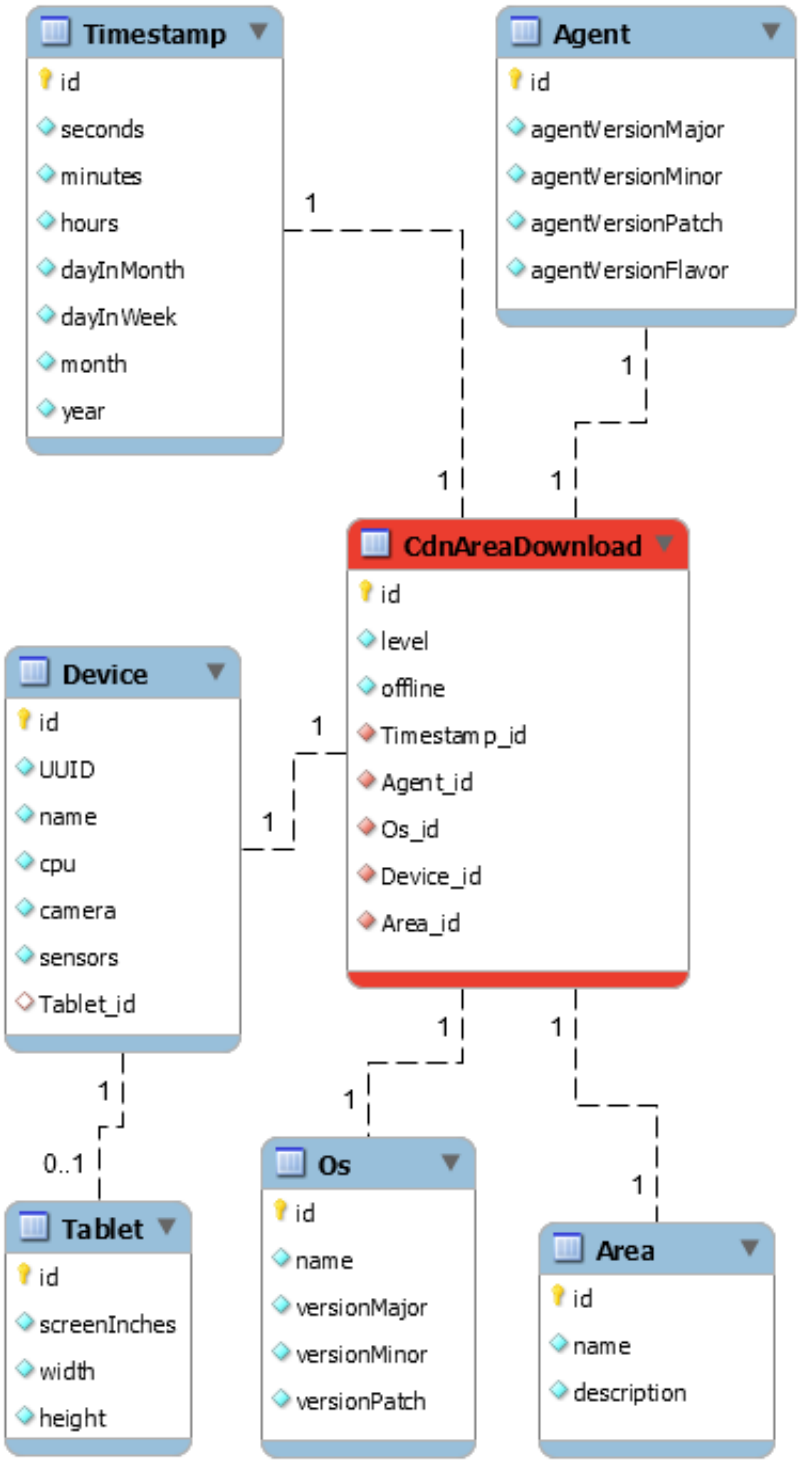


Figure 3.3: Conceptual data model of the "area download" fact

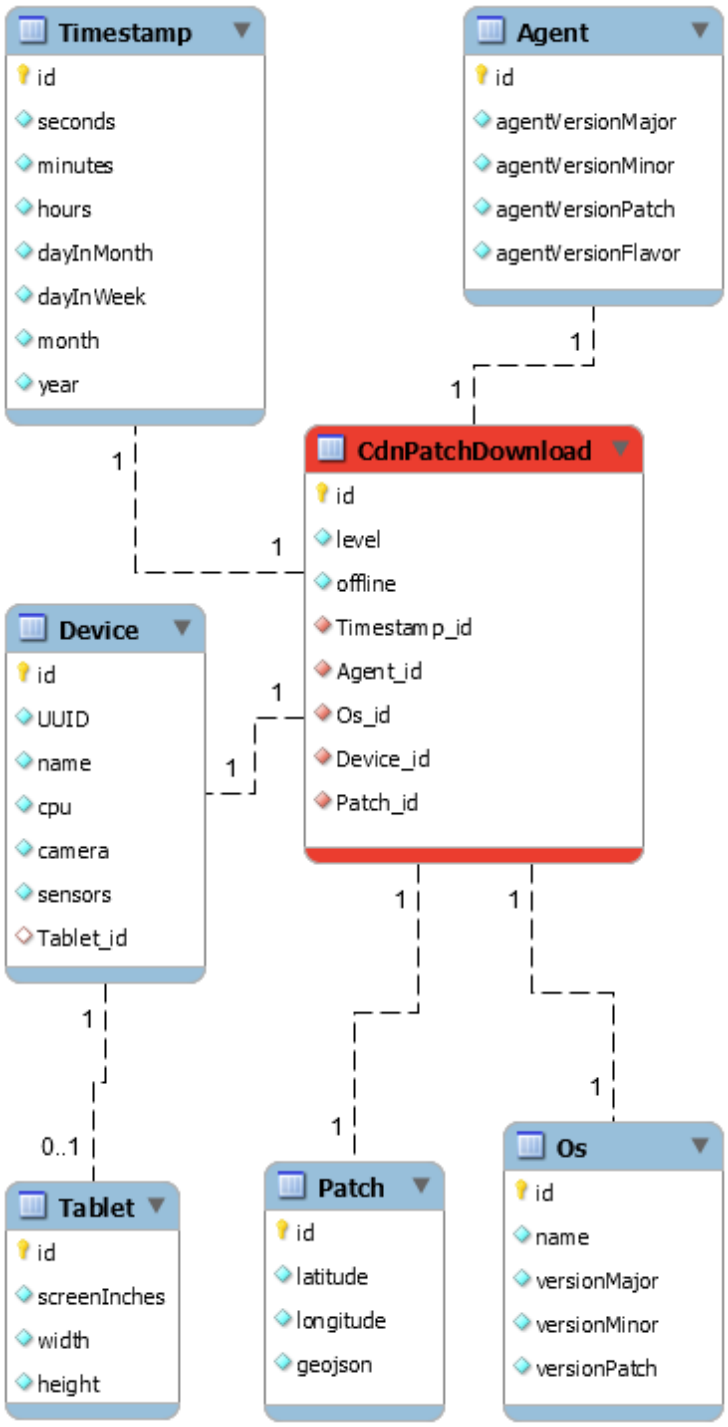


Figure 3.4: Conceptual data model of the "patch download" fact

3.1.2 Google Play Store

The Google Play Store model represents all the facts that are collected by the Play Store marketplace. The facts recorder are:

Review: represents a submitted user review of PeakLens.

Install: represents a installation of PeakLens on a device.

Crash: represents a reported crash of PeakLens.

Gcm: represents a Google Cloud Message (now called Firebase).

Rating: represents a submitted user rating of PeakLens.

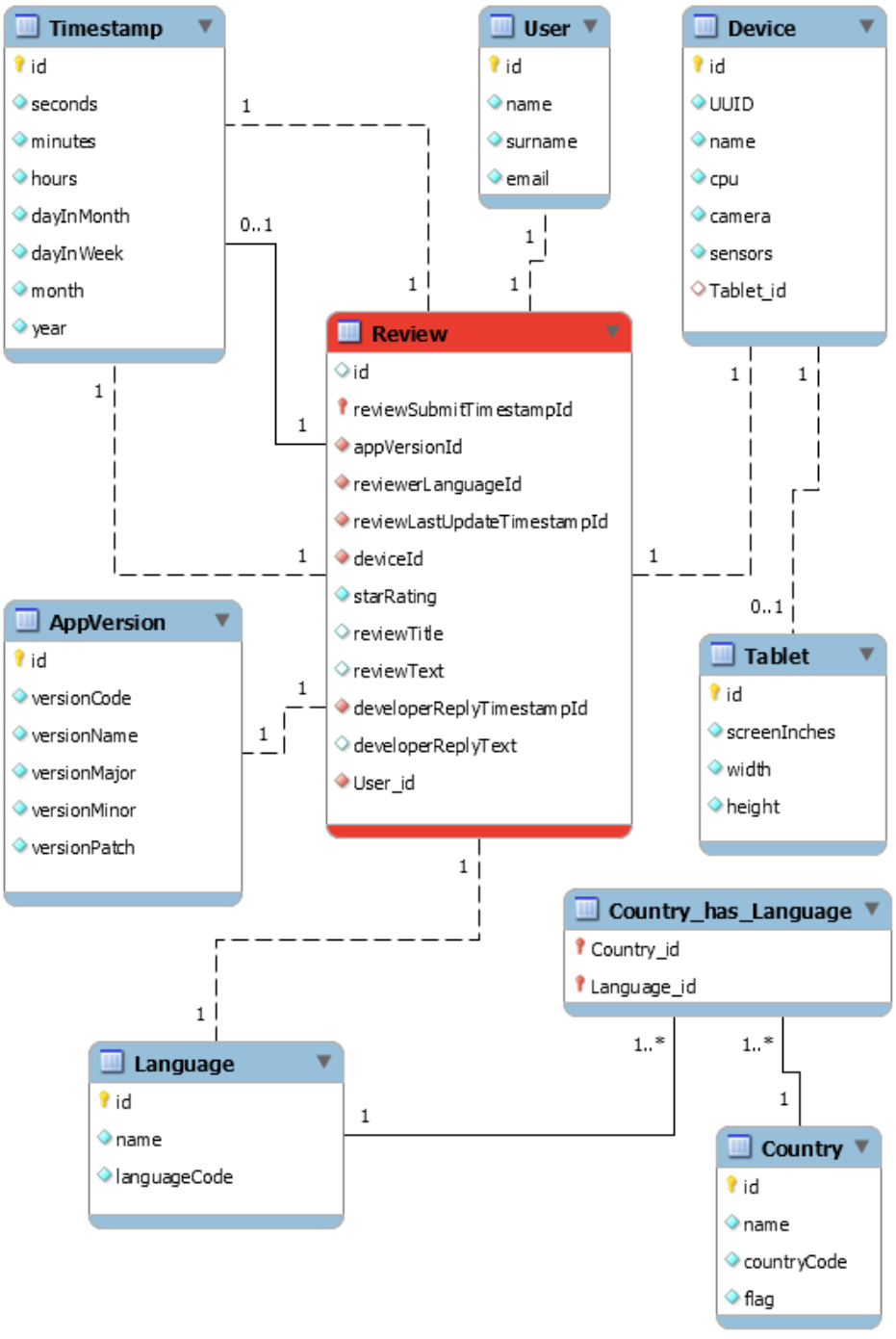


Figure 3.5: Conceptual data model of the "review" fact

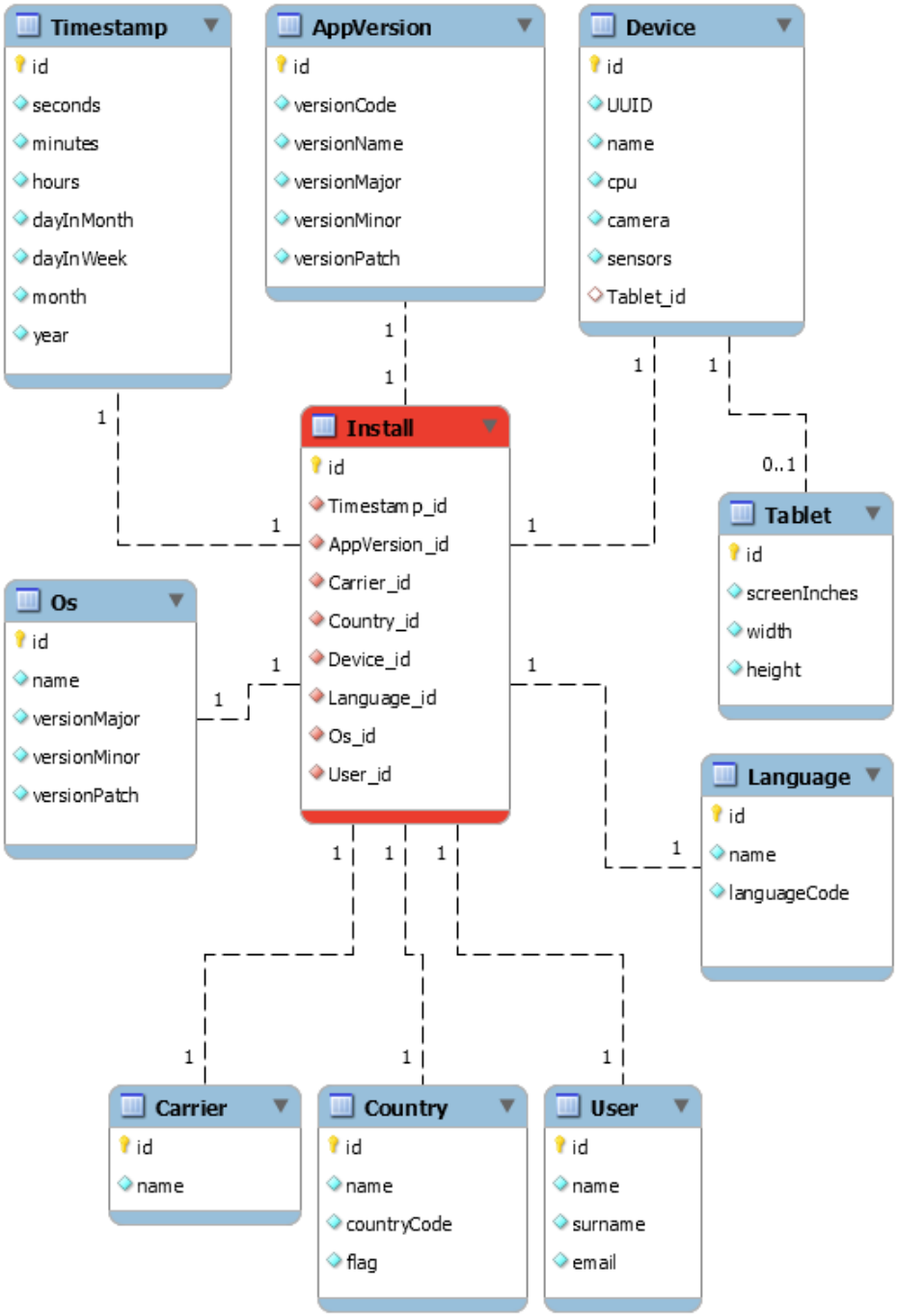
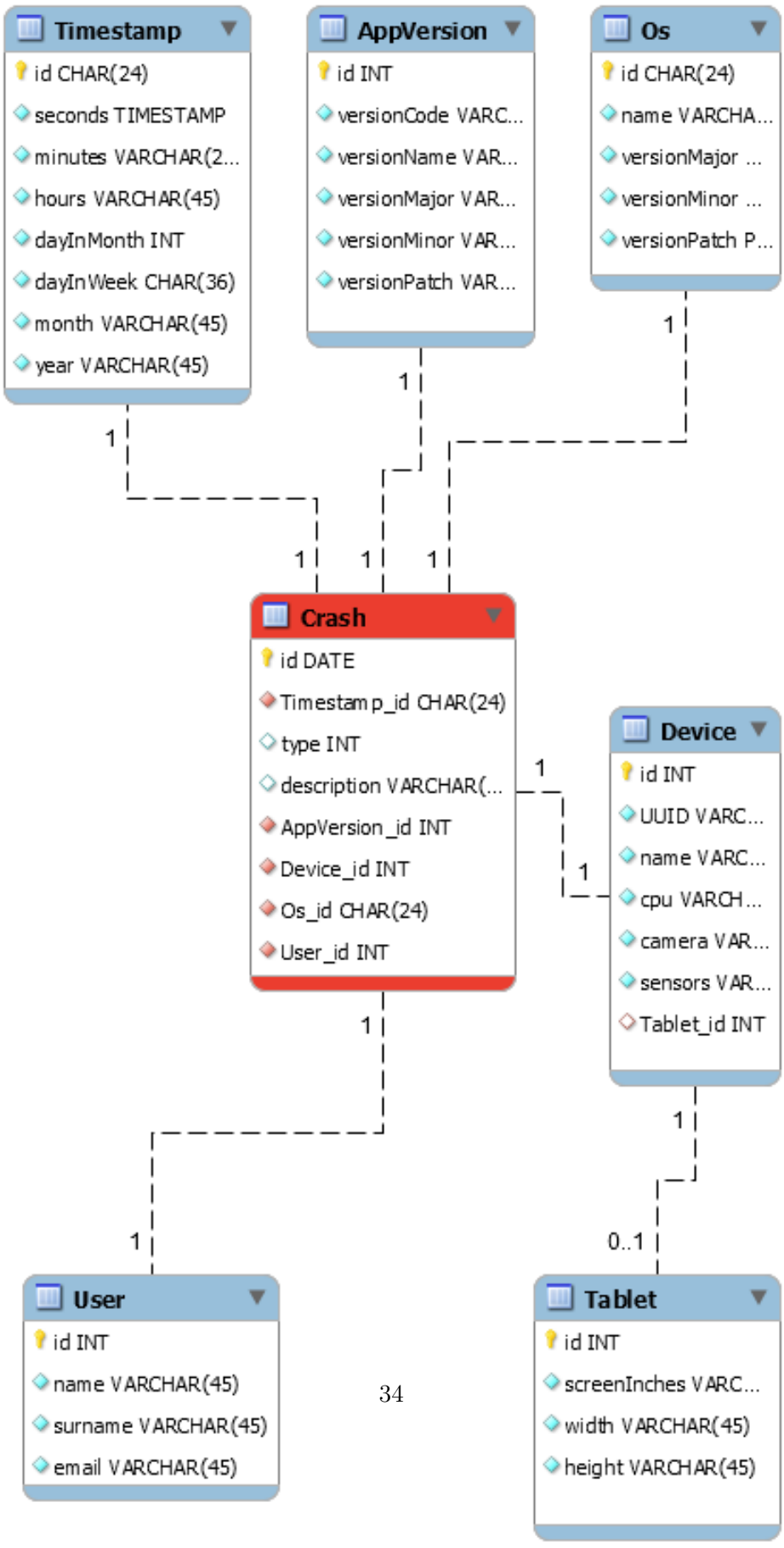


Figure 3.6: Conceptual data model of the "install" fact


Timestamp
id CHAR(24)
seconds TIMESTAMP
minutes VARCHAR(2...
hours VARCHAR(45)
dayInMonth INT
dayInWeek CHAR(36)
month VARCHAR(45)
year VARCHAR(45)
AppVersion
id INT
versionCode VARC...
versionName VAR...
versionMajor VAR...
versionMinor VAR...
versionPatch VAR...
Os
id CHAR(24)
name VARCHA...
versionMajor ...
versionMinor ...
versionPatch P...
Crash
id DATE
Timestamp_id CHAR(24)
type INT
description VARCHAR(...
AppVersion_id INT
Device_id INT
Os_id CHAR(24)
User_id INT
Device
id INT
UUID VARC...
name VARC...
cpu VARCH...
camera VAR...
sensors VAR...
Tablet_id INT
User
id INT
name VARCHAR(45)
surname VARCHAR(45)
email VARCHAR(45)
Tablet
id INT
screenInches VARC...
width VARCHAR(45)
height VARCHAR(45)
1
0..1

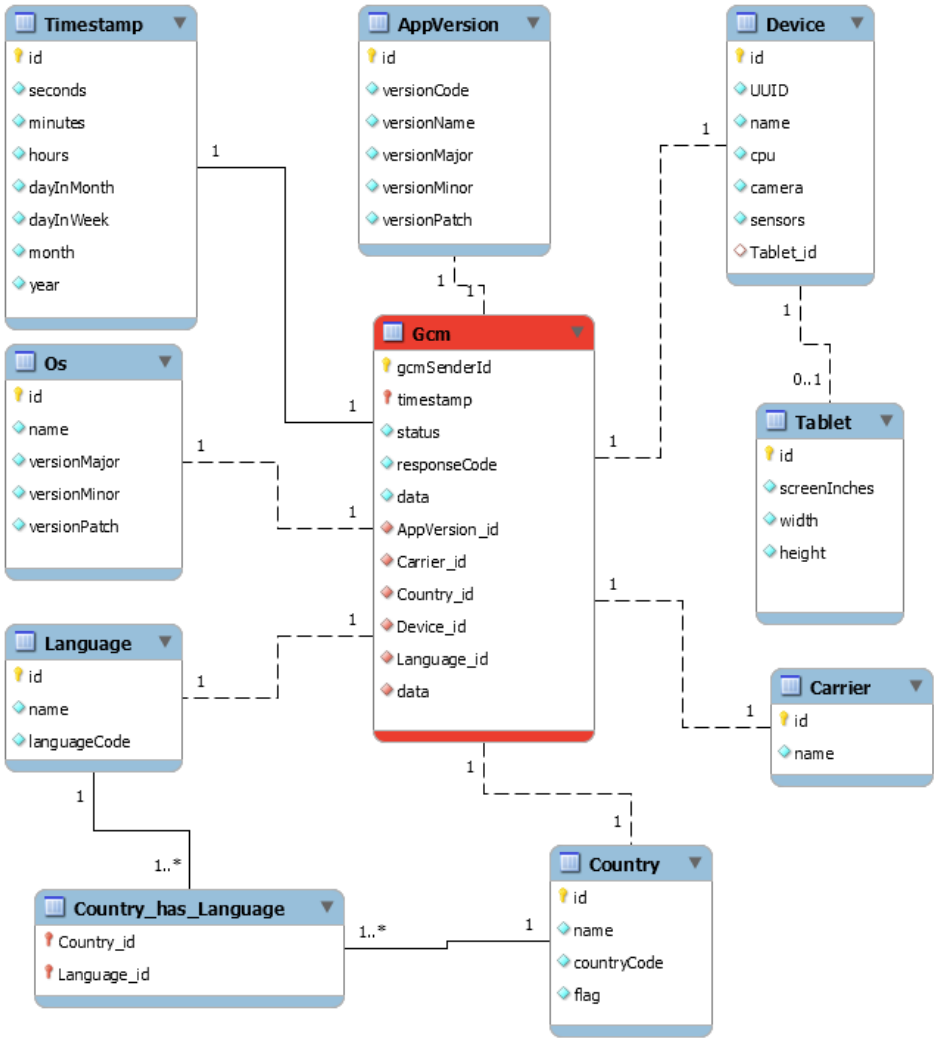


Figure 3.8: Conceptual data model of the "gcm" fact

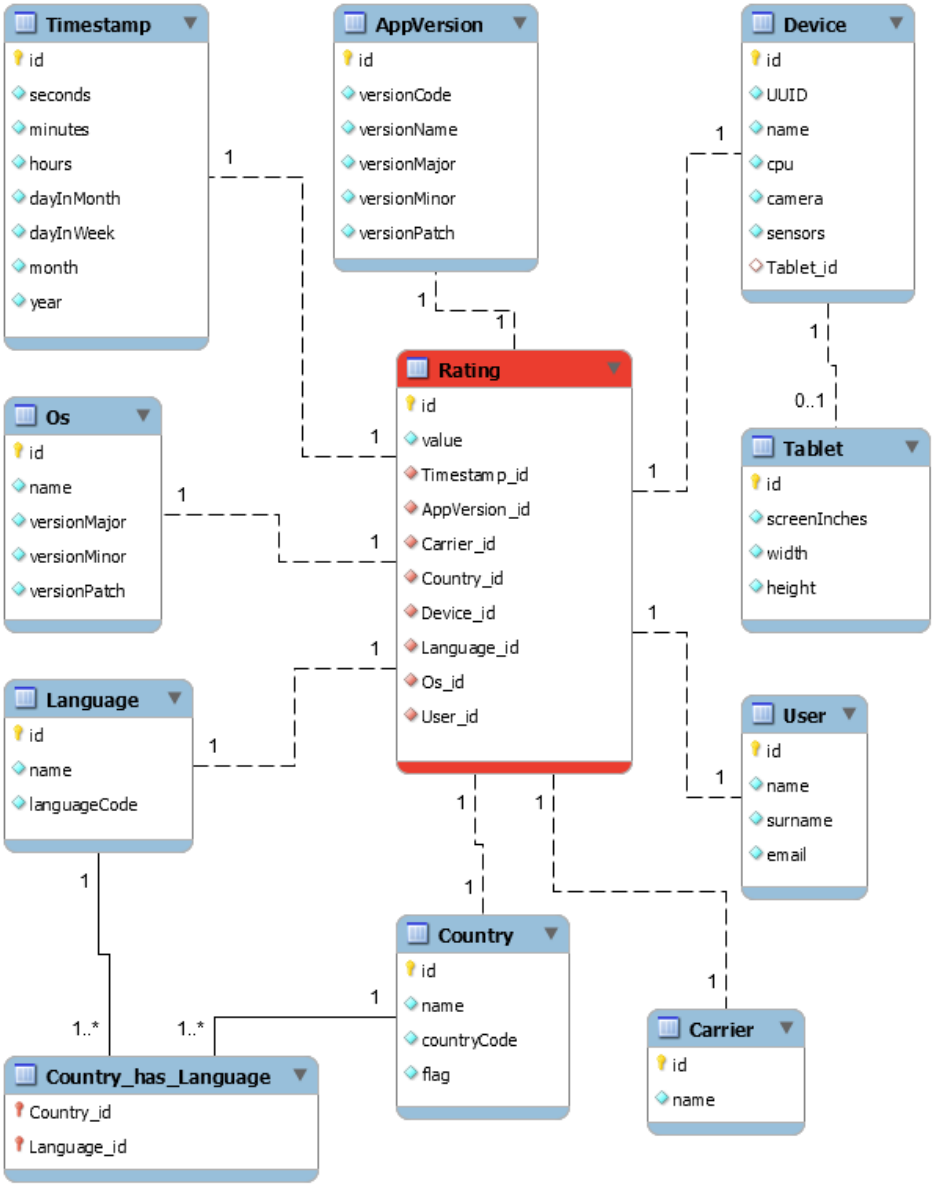


Figure 3.9: Conceptual data model of the "rating" fact

3.1.3 Dashboard

The dashboard model represents all the facts that are collected by PeakLens Dashboard. The facts recorder are:

Event: relevant events that happened during the development of PeakLens.

Alarm: represent the alarm set by the users of the dashboard.

Notification: represent a push notification sent to the users of PeakLens.

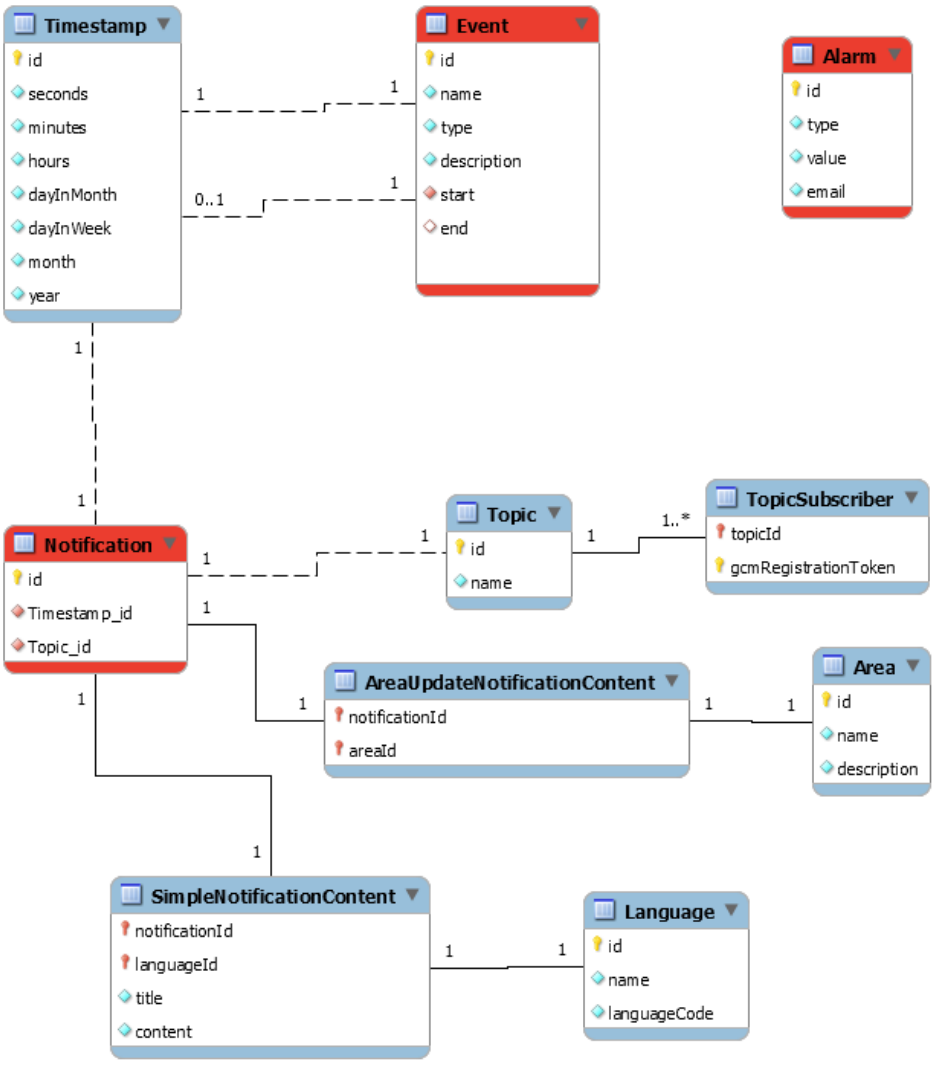


Figure 3.10: Conceptual data model of the "event", "alarm", and "notification" fact

3.2 Logic model

The logical data model is the translation of the conceptual one, bringing it closer to the physical representation. Entities attributes are decorated with type and size, unavailable entities and attributes are removed, hierarchies are denormalized to improve performances.

3.2.1 Logger

The changes from the conceptual data model consists in:

- Merge of the dimensions of each table in the fact table.
- Loss of the device characteristics which are not actually logged for privacy reasons.

The model contains the following entities:

Render: stores "render" events

id: UUID of the event

timestamp: time when the event was received

level: message importance

deviceId: UUID of the device

agentVersionMajor: PeakLens major version number (example: “1”)

agentVersionMinor: PeakLens minor version number (example: “0”)

agentVersionPatch: PeakLens patch version number (example: “0”)

agentVersionFlavor: PeakLens release type (example: “beta”)

deviceName: name of the device (example: “samsung SM-G800F”)

osVerson: operating system version (example: “Android 6.0”)

offline: flag that indicates if the event logged while the device was offline

coordinates: latitude and longitude of the event

CdnAreaList: stores "area list" events

(same properties of "render")

CdnAreaDownload: stores "area list" events

(same properties of "render")

area: UUID of the area requested

CdnPatchDownload: stores "area list" events

(same properties of "render")

patch: name of the patch requested (example: “N39E019”)

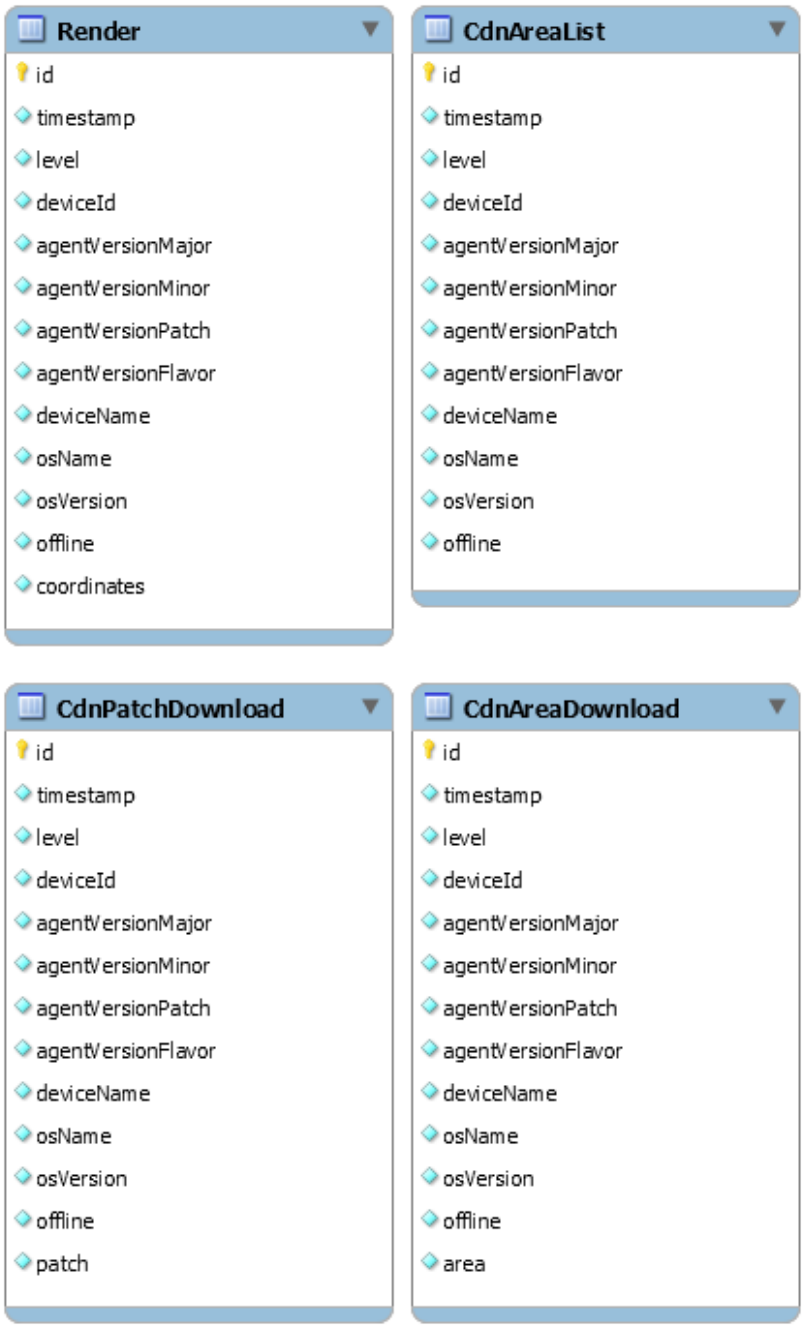


Figure 3.11: Logical data model of the "Logger" schema

3.2.2 Google Play Store

The changes from the conceptual data model consists in:

- Loss of almost all identifying information.
- Creation of tables of aggregated data because unit data is not available.

The model contains the following entities:

Review: stores all the reviews received by the Google Play Store

reviewSubmitTimestamp: time when the review has been submitted

appVersionCode: version code of the application (example: "15")

appVersionName: version name of the application (example: "1.0.0")

reviewerLanguage: language of the reviewer (example: "it")

device: device of the reviewer (example: "OnePlus")

reviewLastUpdateTimestamp: time when the review was last updated

starRating: integer between 1 and 5

reviewTitle: title of the review

reviewText: text of the review

developerReplyTimestamp: time when the developer replied

developerReplyText: text of the developer reply

reviewLink: link to the review

InstallsOverview: stores the total install related metrics for each day

date: day to which the metrics are related

dailyDeviceInstalls: number of new devices that install the app each day (re-installs included)

dailyDeviceUninstalls: number of devices each day that uninstall the app

dailyDeviceUpgrade: number of devices each day that install an update for the app

totalUserInstalls: total number of unique users who have ever installed this app on one or more of their devices. Only one install is counted per user, regardless of how many different devices they installed it on. Includes users who uninstalled the app later.

dailyUserInstalls: unique users who installed the app on one or more of their devices for the first time

dailyUserUninstalls: unique users who uninstalled the app from all of their devices

activeDeviceInstalls: number of devices that have been online at least once that have your app installed.

InstallsByAppVersion: install metrics aggregated by the application version

appVersionCode: version code of the application (example: "15")

(same properties of InstallsOverview)

InstallsByCarrier: install metrics grouped by carrier

carrier: name of the mobile network operator (example: "AT&T")

(same properties of InstallsOverview)

InstallsByCountry: install metrics grouped by country

country: country code (example: "IT")

(same properties of InstallsOverview)

InstallsByDevice: install metrics grouped by device

device: device name (example: "OnePlus")

(same properties of InstallsOverview)

InstallsByLanguage: install metrics grouped by language

language: language code (example: "it")

(same properties of InstallsOverview)

InstallsByOsVersion: install metrics grouped by OS version

androidOsVersion: version of android (example: "Android 7.1")

(same properties of InstallsOverview)

InstallsByTablets: install metrics grouped by tablets

tablets: tablet type (example: "7_up_to_10_inch_tablets")

(same properties of InstallsOverview)

CrashesOverview: total crashes and application not responding (ANR) errors for each day

date: day to which the metrics are related

dailyCrashes: number of crashes per day

dailyAnrs: number of application not responding (ANR) errors

CrashesByAppVersion: crashes aggregated by the application version

appVersionCode: version code of the application (example: "15")

(same properties of CrashesOverview)

CrashesByDevice: crashes aggregated by the device

device: device name (example: "OnePlus")

(same properties of CrashesOverview)

CrashesByOsVersion: crashes aggregated by the OS version

androidOsVersion: version of android (example: "Android 7.1")

(same properties of CrashesOverview)

CrashesByTablets: crashes aggregated by the tablets

tablets: tablet type (example: "7_up_to_10_inch_tablets")

(same properties of CrashesOverview)

GcmOverview date: day to which the metrics are related

gcmSenderId: identifier of the GCM app server

gcmMessages: messages sent to the app

gcmRegistrations: registrations requests received from the app

GcmByAppVersion: messages aggregated by the application version

appVersionCode: version code of the application (example: "15")

(same properties of GcmOverview)

GcmByCarrier: messages aggregated by the carrier

carrier: name of the mobile network operator (example: "AT&T")

(same properties of GcmOverview)

GcmByCountry: messages aggregated by the country

country: country code (example: "IT")

(same properties of GcmOverview)

GcmByDevice: messages aggregated by the device

device: device name (example: "OnePlus")

(same properties of GcmOverview)

GcmByLanguage: messages aggregated by the language

language: language code (example: "it")

(same properties of GcmOverview)

GcmByMessageStatus: messages aggregated by their status

gcmMessageStatus: status of the messages processed

(same properties of GcmOverview)

GcmByOsVersion: messages aggregated by the OS version

androidOsVersion: version of android (example: "Android 7.1")

(same properties of GcmOverview)

GcmByResponseCode: messages aggregated by their response code

gcmMessageStatus: response code of the messages processed

(same properties of GcmOverview)

GcmByTablets: messages aggregated by the tablet types

tablets: tablet type (example: "7_up_to_10_inch_tablets")

(same properties of GcmOverview)

RatingsOverview date: day to which the metrics are related

dailyAverageRating: average of the ratings received during the day

totalAverageRating: average of the ratings received since the app has been released

RatingsByAppVersion appVersionCode: version code of the application (example: "15")

(same properties of RatingsOverview)

RatingsByCarrier carrier: name of the mobile network operator (example: "AT&T")

(same properties of RatingsOverview)

RatingsByCountry country: country code (example: "IT")

(same properties of RatingsOverview)

RatingsByDevice device: device name (example: "OnePlus")

(same properties of RatingsOverview)

RatingsByLanguage language: language code (example: "it")

(same properties of RatingsOverview)

RatingsByOsVersion androidOsVersion: version of android (example: "Android 7.1")

(same properties of RatingsOverview)

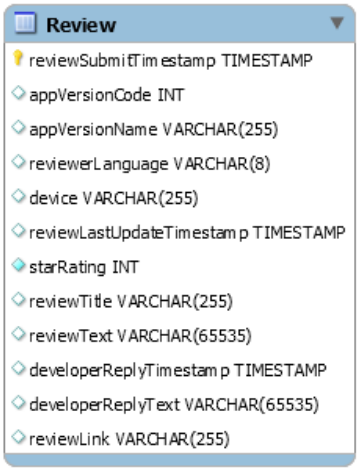


Figure 3.12: Logical data model of the "review" fact

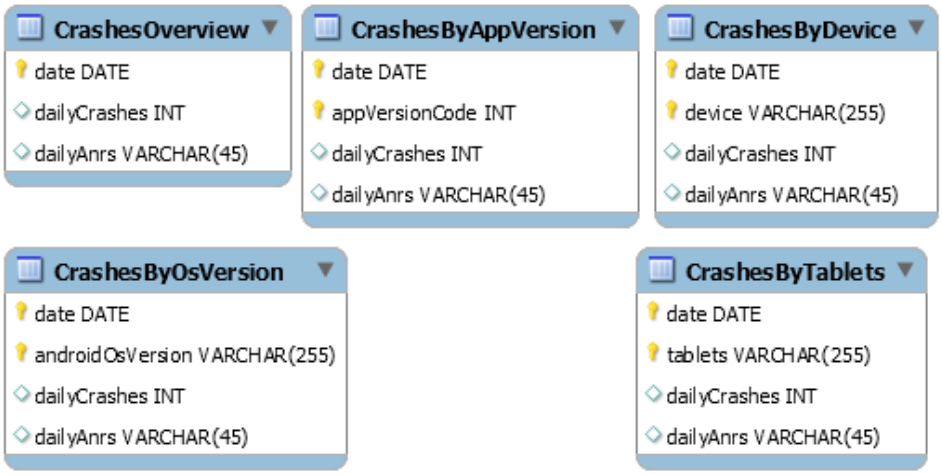


Figure 3.13: Logical data model of the "crash" fact

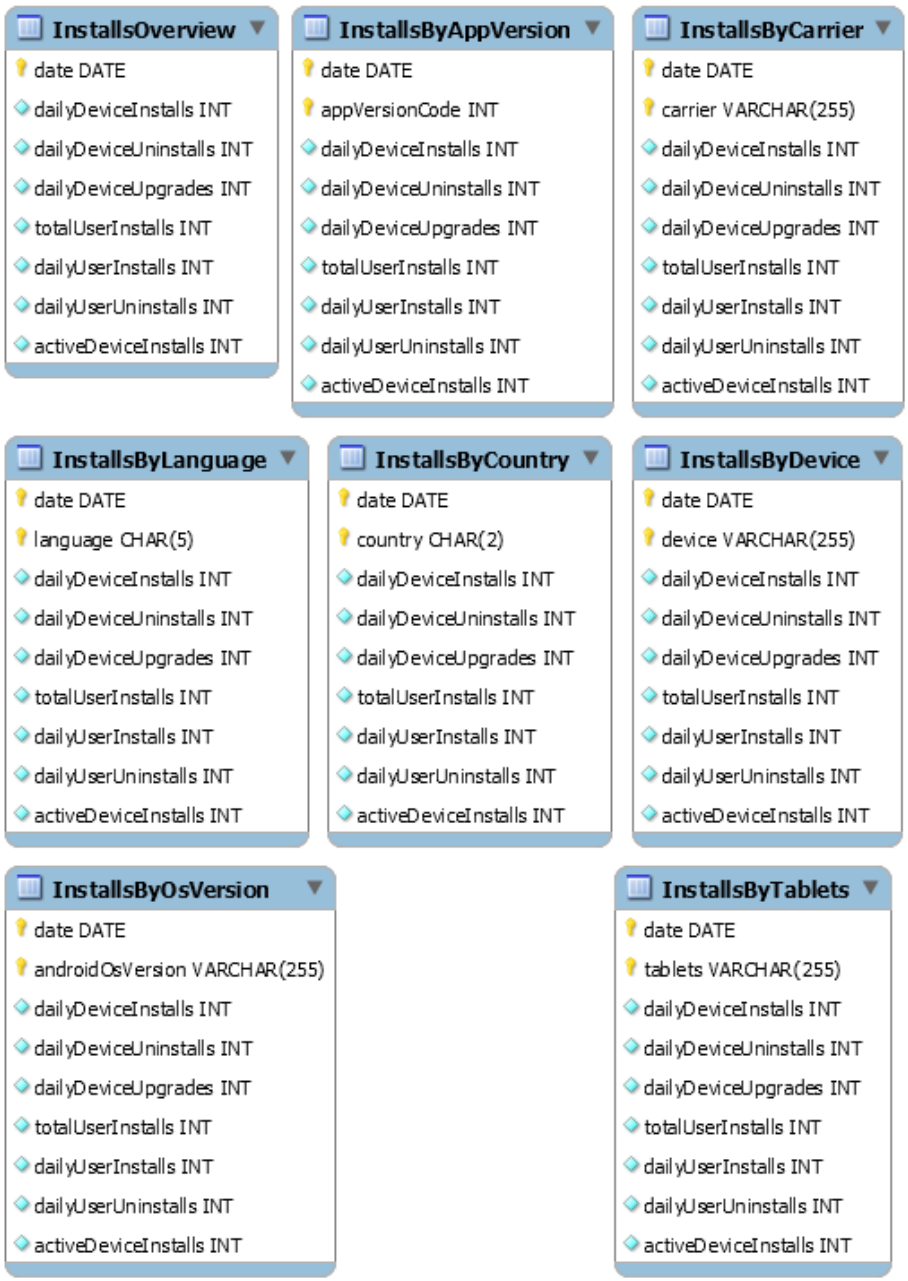


Figure 3.14: Logical data model of the "install" fact

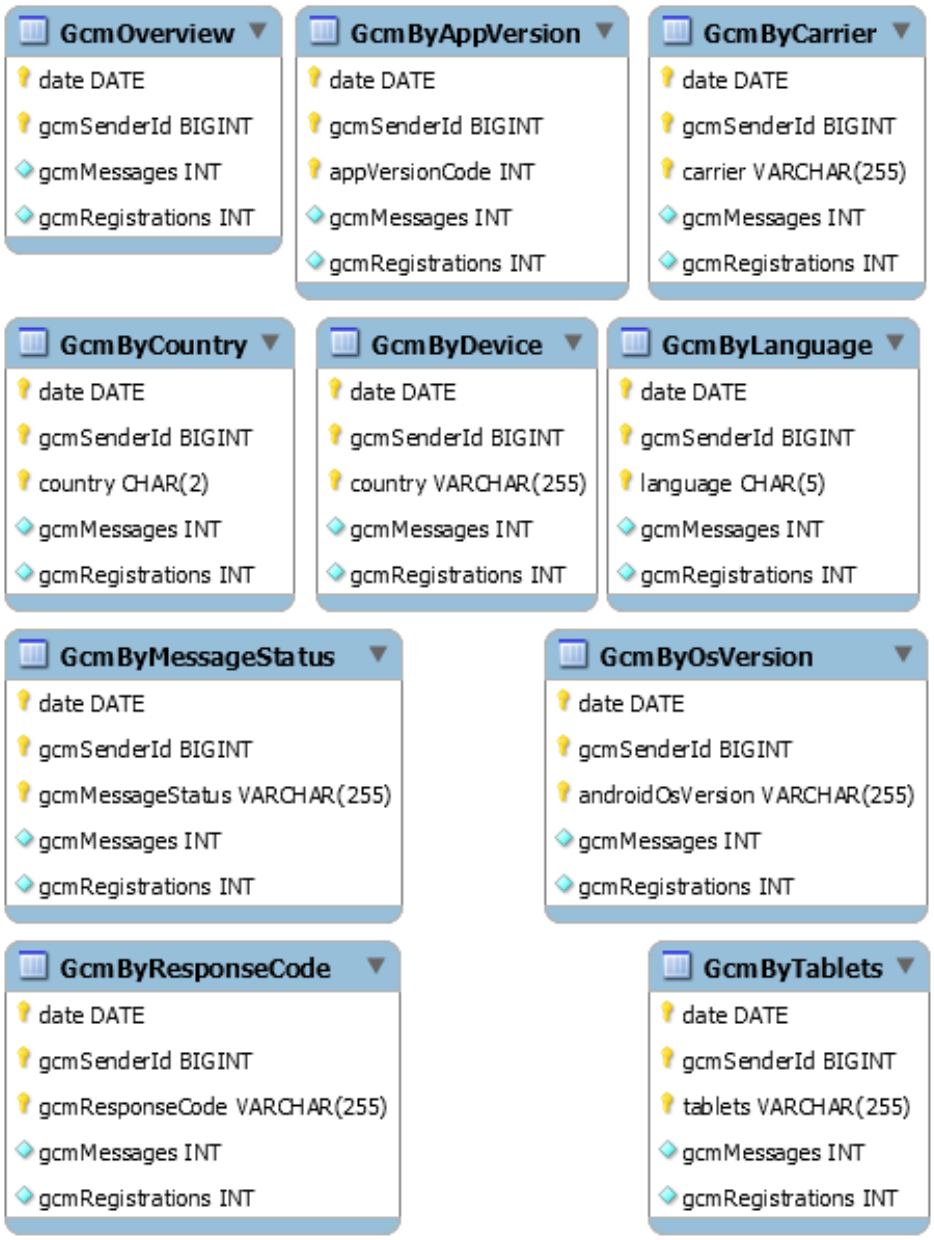


Figure 3.15: Logical data model of the "gcm" fact

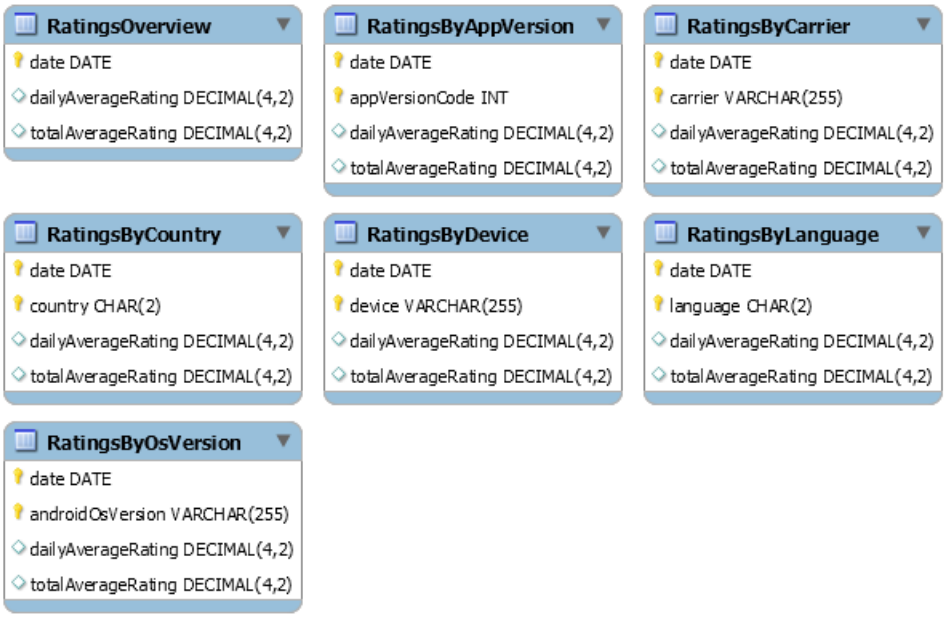


Figure 3.16: Logical data model of the "rating" fact

3.2.3 Dashboard

The changes from the conceptual data model consists in:

- Removal of the Timestamp table.
- Removal of the Language table.

The model contains the following entities:

Event: generic events that are relevant and should be tracked

id: event identifier
name: event name
type: event type
description: event description
start: occurrence of the event
end: termination of the event

Alarm: alarms set by the user

id: alarm identifier
type: alarm type
value: threshold
email: e-mail to send notice

Topic: Notification topics

id: topic identifier
name: topic name

TopicSubscriber: users that are subscribed to a topic

topicId: foreign key to a topic
gcmRegistrationToken: registration token of the device

SimpleNotification: notifications containing only a message

id: notification id
topicId: foreign key to a topic
timestamp: date and time of creation

SimpleNotificationContent: content of a simple notification in different languages

notificationId: foreign key to a simple notification

language: language code

title: title in the specific language

content: text in the specific language

AreaUpdateNotification: notifications about the update of an area

id: notification id

topicId: foreign key to a topic

timestamp: date and time of creation

AreaUpdateNotificationContent: areas that have been updated in a notification

notificationId: foreign key to an area notification

areaId: UUID of the area

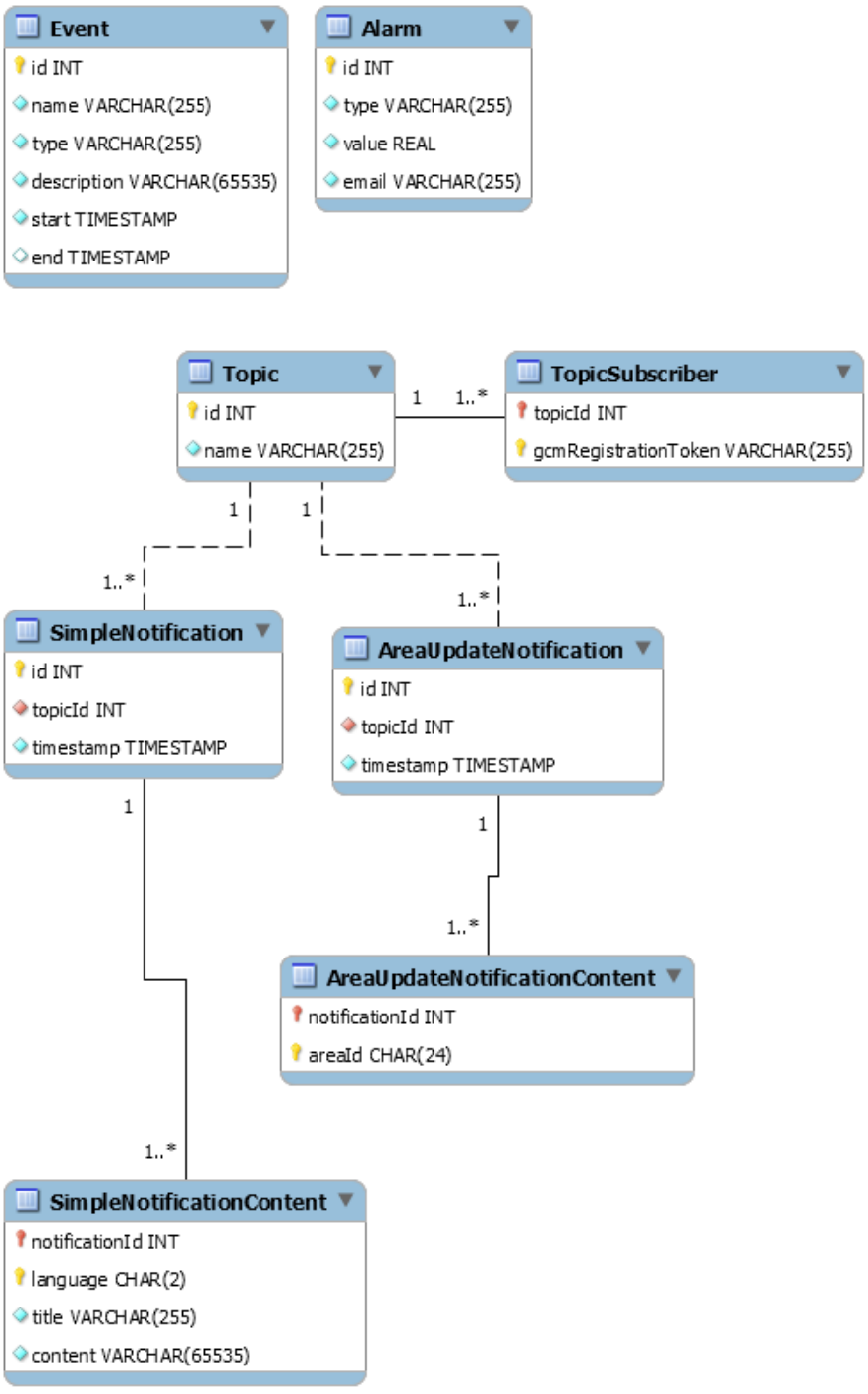


Figure 3.17: Logical data model of the "dashboard" schema

Chapter 4

Software design

This chapter presents the high level software architecture and the graphical user interface (GUI) of the dashboard.

4.1 Architecture

The PeakLens Dashboard is a web application. A web application is a client-server architecture where the client is a web browser which communicates to a web server through the HTTP protocol and displays HTML documents.

The architecture of the server follows the three-tier model, which means it is divided in three parts: a presentation tier which implements the user interface of the application, an application tier which implements the business logic, and a data tier which manages the persistence of the application information.

4.2 Components

The architecture consists of the following software components:

Web Browser : the client that connects to the web server to retrieve the presentation tier and renders it. The presentation layer has a minimum logic to interact with the application tier through a REST interface based on HTTP.

Dashboard Web Server : the server offers the user interface to the browser coded using the World Wide Web (WWW) standards HTML, CSS and JavaScript.

Dashboard Application Server : it is part of the application tier and implements the business logic of the dashboard. It communicates with

the Dashboard Database through the SQL language and compute the metrics.

Dashboard Database : implements the data model described in the Chapter 3.

Importer : it is part of the application tier and it is the software component responsible to collect, process, and clean the data regarding PeakLens retrieved from the Google Play Store and the PeakLens Logger database.

Google Play Store : represents the marketplace where PeakLens is distributed.

PeakLens MongoDB : this component represents the Logger database where the logs of PeakLens are stored.

See Figure 4.1 for a graphical representation of the components in the different tiers of the architecture. See Figure 4.2 for understanding how the components communicate between them.

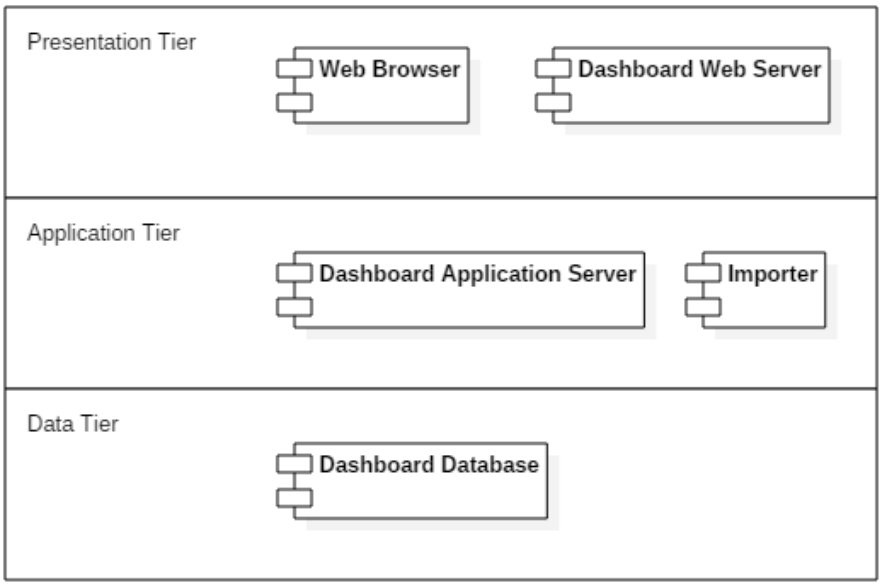


Figure 4.1: Component view of the architecture

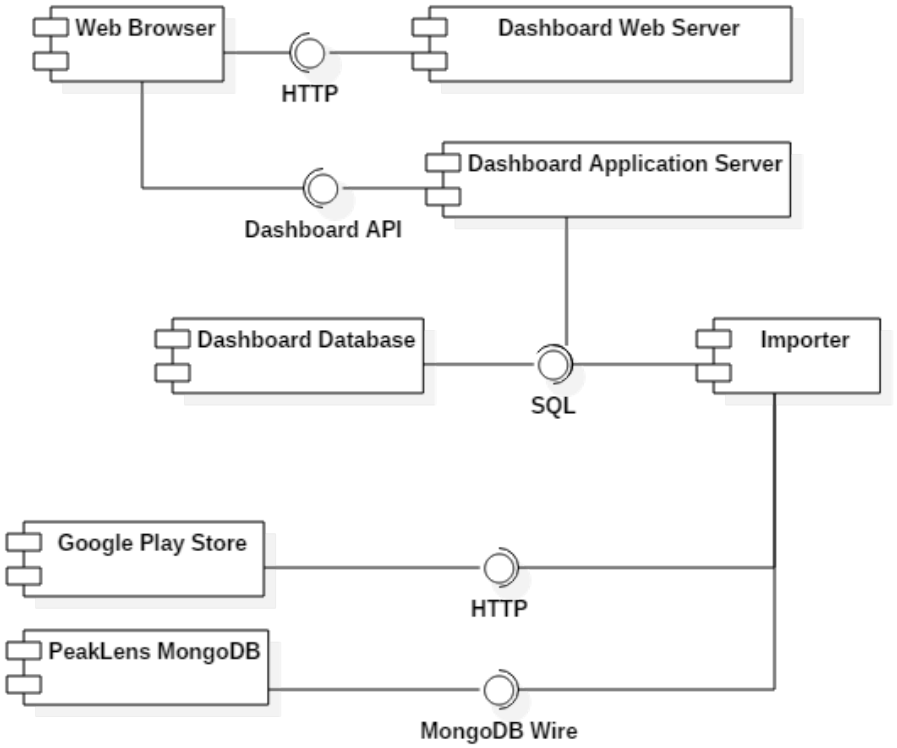


Figure 4.2: Interfaces between the components of the architecture

4.2.1 Web API

The application tier of the PeakLens Dashboard offers a *REpresentational State Transfer* (REST) interface over HTTP (hence *web* API) to interact with the services programmatically. REST allows web clients to request and manuipulate textual representations of web resources using a uniform and predefined set of stateless operations. RESTful APIs follows a set of principles that lead to the development of simple and scalable interfaces [9].

In the context of the PeakLens Dashboard, the API offered by the application tiers allows the presentation tier to read the processed data from the data tier and edit the events. The full specification of the REST API is available in the Annex A.

4.3 Graphical User Interface

This section defines the views needed for accomplishing the requirements of the PeakLens Dashboard.

4.3.1 Active Users View

The active users view (Figure 4.3) displays the activity of the users on a heatmap. The colors of the heatmap are influenced by the amount of activity on a logarithmic scale. The scale is shown in a legend at the bottom right of the map. The user can move and zoom the heatmap. On the bottom of the view, there is a timeline that indicates what time the heatmap is visualizing and the relevant event of PeakLens. On the right side of the view there is the *Timelapse Player* which allow the user to animate the heatmap to show the evolution of the user activities. Also on the right side there are the filters controls which can be toggled to refine the source data of the heatmap.

4.3.2 Area Downloads View

The area downloads view (Figure 4.4) displays the areas which PeakLens allows to download. These downloaded areas enable PeakLens to work offline without the need of being connected to the Internet. If the mouse is moved over an area, the amount of downloads of that area are shown in the top right of the map. On the right side of the view there are filter controls which can be used to enable or disable the visibility of the different areas and refine the source data of the map.

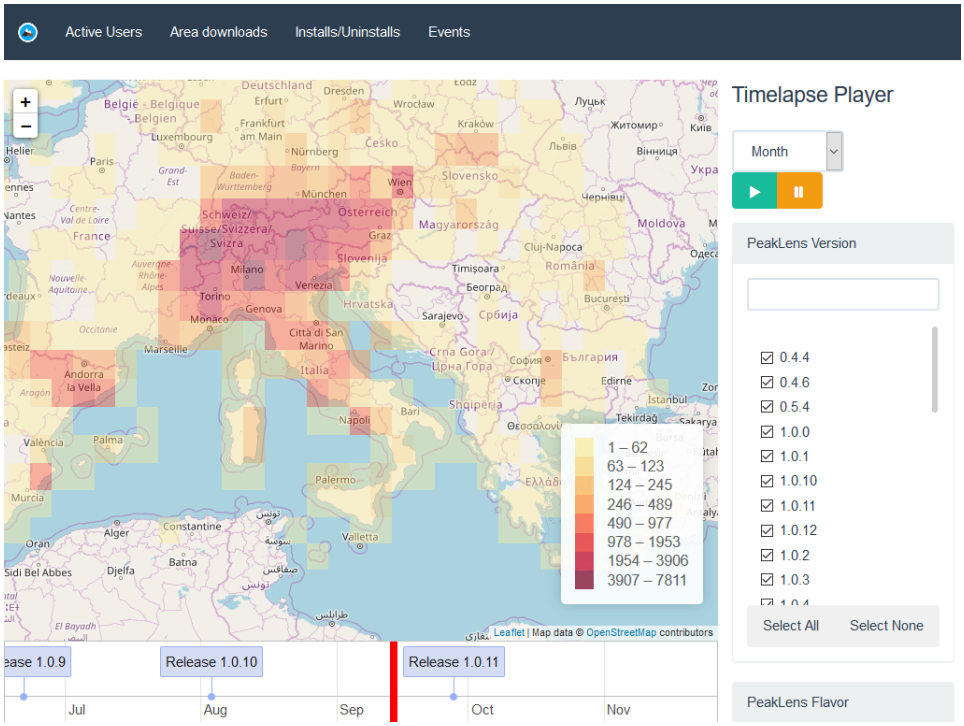


Figure 4.3: Screenshot of the active users view

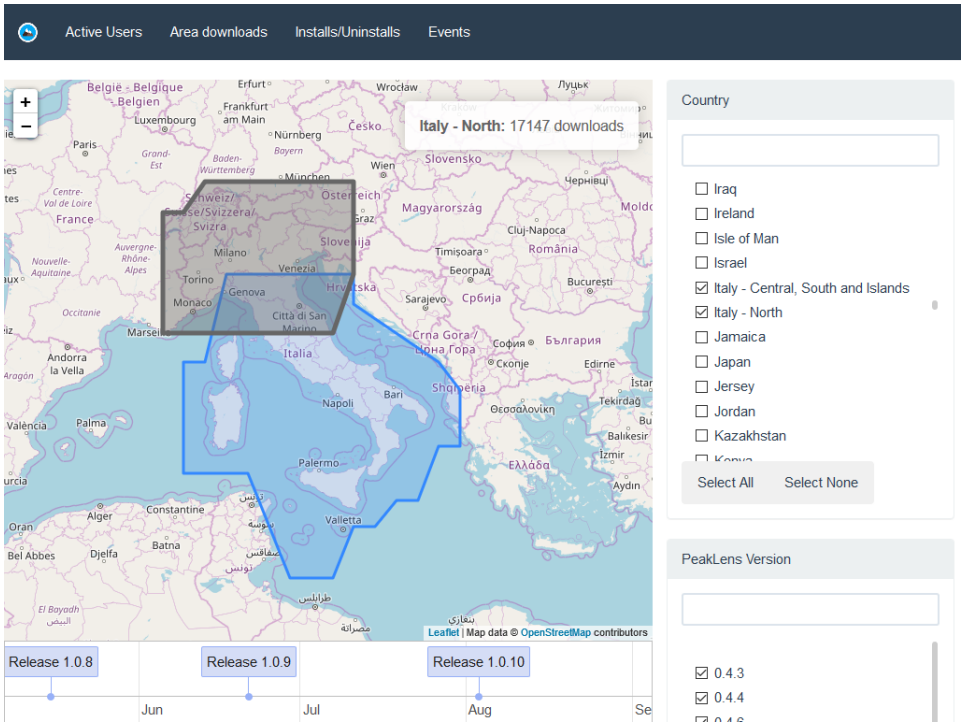


Figure 4.4: Screenshot of the area downloads view

4.3.3 Installs View

The installs view (Figure 4.5) allows the user to plot the data relative to the amount of PeakLens installs and uninstall. On the left side a list shows the active data series. Each series can be customized (Figure 4.6) for displaying the installs or uninstalls, and filtered by the characteristics of each device/user. The chart can be moved and zoomed.

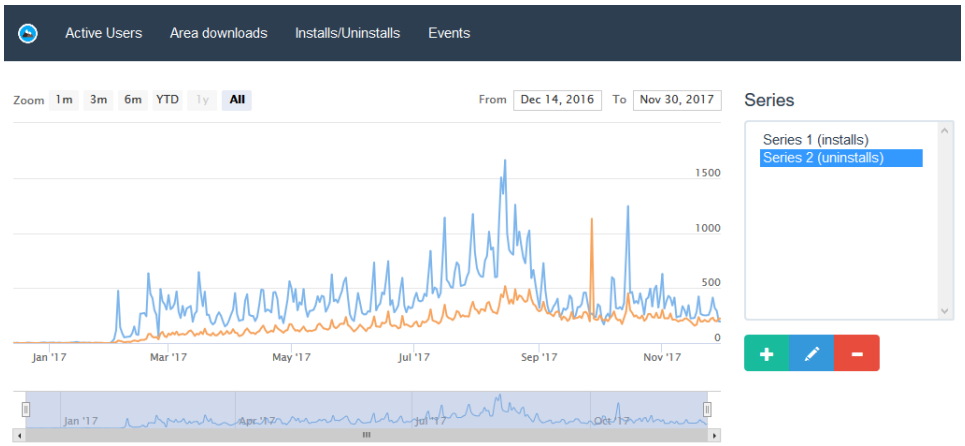


Figure 4.5: Screenshot of the installs view

4.3.4 Events View

The events view (Figure 4.7) allows the user to create, edit, and delete the relevant events of PeakLens. These events are shown on all the timelines of the dashboard.

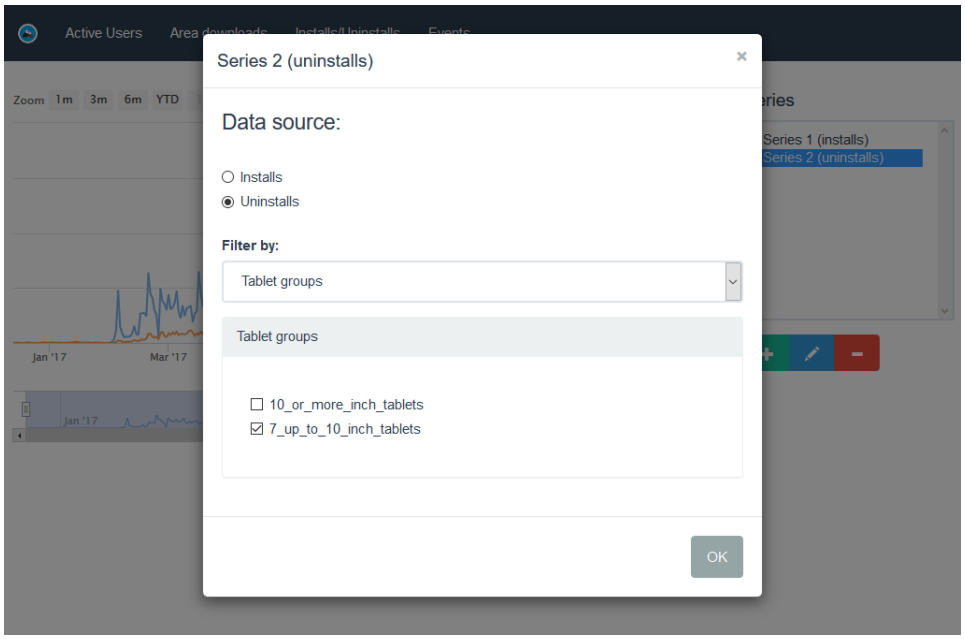


Figure 4.6: Screenshot of modal window to editing a data series

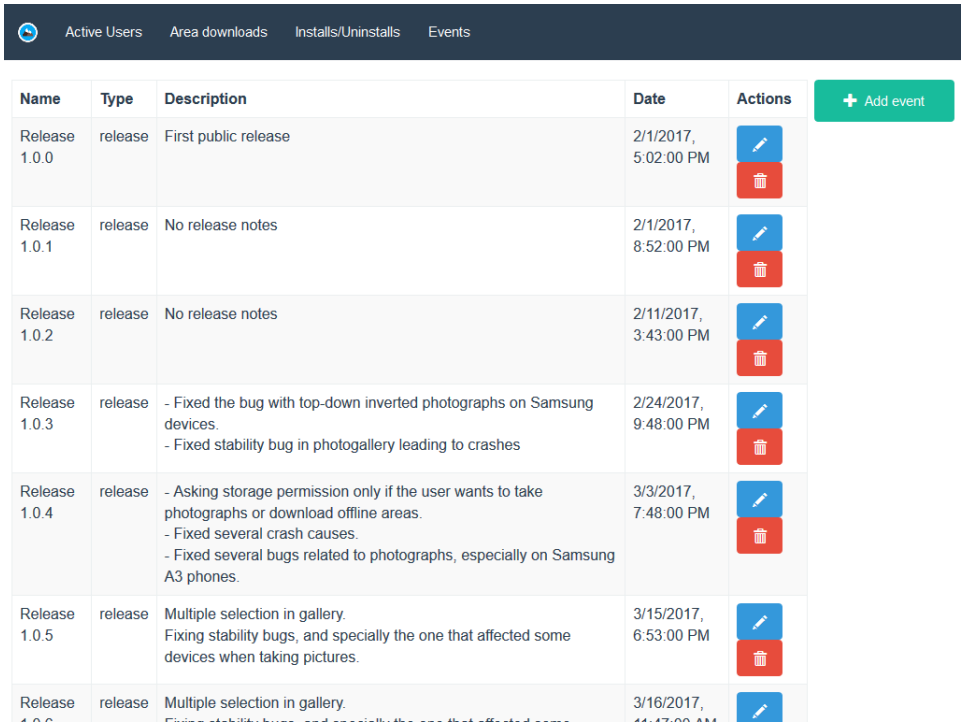


Figure 4.7: Events screen

Chapter 5

Software implementation

This chapter describes the softwares, frameworks, and technologies used to develop each component of the architecture.

5.1 Database

The implementation of the data tier uses the relational database management system PostgreSQL [10]. PostgreSQL offers geometric functions to perform efficiently spatial searches which were useful to create the heat maps. The geometric functions do not follow the SQL/MM Spatial features standard but use a custom syntax. It would be possible to use the standard through the installation the PostGIS [11] extension, but full compatibility with the SQL standard wasn't required for this project.

5.1.1 Heatmaps Creation

Heatmaps are generated on the database by computing the amount of renders performed in each patch. For PeakLens, a patch is a rectangle identified by its latitude and longitude (Figure 5.1) where it is able to identify mountain peaks. Each patch has a size of 1×1 degrees (°).

During the development several queries to compute the heatmaps were tested to find the one with the best performances. The first solution was to naively count the renders in each patch:

```
SELECT COUNT(*) FROM Render
WHERE timestamp BETWEEN $1 AND $2 AND $3::box @>[1] coordinates
```

`$1` is the start time, `$2` is the end time, `$3` is a box geometric type representing the patch location.

[1] @> is the "contains" geometric operator

The creation of a heatmap has a complexity of $O(k \cdot n \cdot \log n)$ where n is the number of renders and k is the number of patches. Note that the `coordinates` attribute has a R-tree index that speeds up the search to $O(\log n)$.

The amount of overhead generated for executing the query independently for each patch ($k \approx 16000$) could be avoided by storing the patches directly in the database and performing a join operation:

```
SELECT Patch.patch, COUNT(*)
FROM Patch
INNER JOIN Render
ON Patch.patch @> Render.coordinates
WHERE Render.timestamp BETWEEN $1 AND $2
GROUP BY Patch.patch
```

The complexity is still the same as before, the join makes uses of the index on `coordinates`, but the overall execution time for generating a heatmap is lower. Note that the complexity of the **`GROUP BY`** operation is negligible since it is linear.

However the general use case of the heatmap generation is not the creation of a single heatmap, but the creation of a timelapse (many sequential heatmaps). Therefore the query should be optimized for this use case:

```
SELECT Range.start, Range.end, Patch.patch, COUNT(*)
FROM Patch
INNER JOIN Render
ON patch @> coordinates
INNER JOIN Range
ON Render.timestamp BETWEEN Range.start AND Range.end
GROUP BY Range.start, Range.end, Patch.patch
```

`Range` is a temporary table with the attributes `start` and **`end`** which represents the time intervals of the timelapse to be created. The complexity $O(k \cdot n \cdot m \cdot \log n)$ of the new query is increased by a constant multiplication (the number of ranges m) but in the end the result is another reduction of overhead because of the lower number of calls to perform to the database to generate a timelapse.

A final optimization is the pre-processing of the coordinates of the renders to avoid the join on the `Patch` table altogether. This is possible because the patches are regular boxes of 1×1 degrees, therefore rounding the renders coordinates automatically defines the patch where the render belongs to. By adding the string attribute `patch`, to the `Render` table computed as:

```
patch = to_char(floor(coordinates[0])) ||[2] to_char(floor(
```

[2] || is the string concatenation operator

```
    coordinates[1]))
```

with a B-tree index on it, is possible to use the following query:

```
SELECT Range.start, Range.end, Render.patch, COUNT(*)
FROM Render
INNER JOIN Range
ON Render.timestamp BETWEEN Range.start AND Range.end
GROUP BY Range.start, Range.end, Patch.patch
```

The complexity for this final query is $O(m \cdot n \cdot \log n)$. The **timestamp** attribute has a B-tree index that speeds up the join to $O(\log n)$.

The improvement in respect to the first query, which had complexity $O(k \cdot n \cdot \log n)$ is twofold:

- $m << k$
- The query outputs a timelapse instead of a single heatmap.

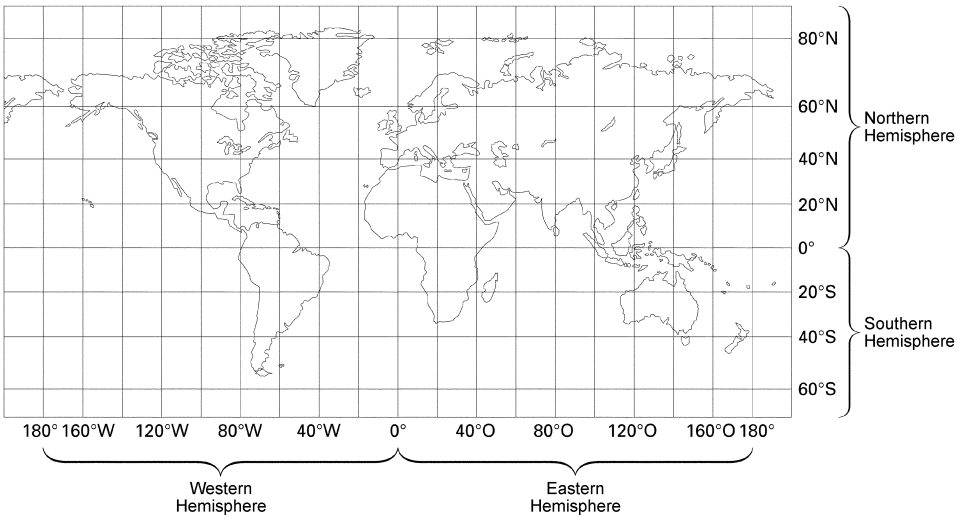


Figure 5.1: Latitude and longitude projected on a flat map

5.2 Web Server

The web server has been implemented using the TypeScript language with the Node.js [12] run-time environment. The function of the web server is to make available the user interface to the web browser of the clients. The web server uses the following frameworks/libraries:

Express: a framework for creating web applications in Node [13].

Pug: a template engine for Node [14].

The user interface has been implemented in HTML, CSS, and JavaScript using the following frameworks/libraries:

Knockout: a web framework for implementing the Model-View-ViewModel (MVVM) pattern [15].

Highstock: a library for creating interactive charts [16].

Leaflet: a library for creating interactive maps [17].

Vis.js: a library for browser visualizations, used for creating the timeline controls [18].

The coding standards used to create the user interface are supported by any major browser released after 2013. To be more specific the minimum requirements are:

- Chrome 29
- Internet explorer 11
- Firefox 22
- Safari 6.1
- Opera 17

5.3 Application Server

The application server has been implemented using the TypeScript language with the Node.js run-time environment. The function of application server is to implement the business logic and make it available to the presentation tier through a set of REST API. The application server uses the following frameworks/libraries:

Express: a framework for creating web applications in Node.

Node-postgres: a Node module that provides functionality to connect to a PostgreSQL database [19].

5.4 Importer

The importer has been implemented as a set of bash scripts that uses the Google SDK tools to download the stats of PeakLens, and the MongoDB command-line utilities to export the Logger database.

The Play Store statistics and the log of PeakLens are temporarily kept as *comma separated values* (CSV) files that are processed to remove malformed record and uniform the data format. The CSV files are then imported into the dashboard database using the command-line utilities of PostgreSQL.

5.4.1 Code organization

The project source is contained in a single Git repository. The following is an explanation of the content of each directory

database: SQL files to create the schema of the database.

public: files that are distributed by the server directly statically (exactly as stored). Images, fonts, CSS styles, and JavaScript libraries to be used in the web browser for rendering the interface.

views: Pug templates used to generate the HTML files for the interface.

- **components:** Knockout components used to implement reusable GUI elements.
- **widgets:** Pug templates used to implement reusable GUI elements.

src: TypeScript source code of the web server and application server.

- **api:** implementation of the REST API.

5.5 Deployment

The following section explain the step necessary to deploy the PeakLens Dashboard on a Linux server running Ubuntu 14, such as the virtual machines available on Policloud. Ubuntu 14 does not contain in its repository the updated dependencies necessary for the dashboard, therefore it is necessary to install them from the official repository.

- PostgreSQL 10
- Node.js 8
- PM2 process manager [20]

- Google Cloud SDK

```
# PostgreSQL 10
wget -q https://www.postgresql.org/media/keys/ACCC4CF8.asc -O - |
    sudo apt-key add -
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/
    trusty-pgdg main" >> /etc/apt/sources.list.d/pgdg.list'
apt-get install postgresql postgresql-contrib

# Node.js 8
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs+

# PM2 process manager
npm install -g pm2
pm2 startup

# Google Cloud SDK
echo "deb http://packages.cloud.google.com/apt cloud-sdk-trusty
    main" | sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.
    list
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo
    apt-key add -
sudo apt-get update && sudo apt-get install google-cloud-sdk
```

After installing the dependencies, it is necessary prepare the environment for the PeakLens Dashboard. Enter the Project directory and execute the following commands:

```
# Create the dashboard database
createdb -U postgres peaklens
psql -U postgres -d peaklens -f database/create_peaklens_db.sql
psql -U postgres -d peaklens -f database/default_events.sql
# Initialize the Google Cloud SDK
gcloud init

# Import PeakLens log in the database
vim import/config.sh # put MongoDB and PostgreSQL credentials
cd import/logger/
./import.sh
cd ../..

# Import Google Play Store statistics in the database
cd import/googleplay/
./import.sh
cd ../..

# Download the modules required by the dashboard
npm install
vim .env # configure the server port and the database credentials
```

```
# Start the dashboard
npm run start-pm2
# Check the status
pm2 status
# Save the PM2 configuration to persist through reboots
pm2 dump
```

Chapter 6

Conclusions and future work

This thesis work has led to the development of a business dashboard that allows a marketing analyst to monitor the performances and trends of PeakLens through visual presentations. The functionalities of the dashboard are enabled by a data warehouse that collect all data coming from the PeakLens ecosystem in one data model simplifying the operations of querying.

6.1 Future Enhancements

A business dashboard is more useful the more aspects of a business it is able to monitor. The PeakLens dashboard could be improved in several ways to be more comprehensive. Some work could be done to gather engagement data from the social media where PeakLens is present.

The visual interaction plays a central role in the usability and effectiveness of the dashboard, therefore some work could be done in this direction to extend the type of visualizations available.

6.1.1 Big Data

The amount of raw data managed by the dashboard, at the moment of writing, can be handled by a modern workstation. However, considering the exponential growth trend of the user activities and the foresight of new data sources (such as social media activity) it is a good practice to plan for expansion. In the medium-short term the practical way would be to give more hardware resources (memory and computation capacity) to the database. For the long term scenario, the whole architecture of the dashboard would need to be redesigned to scale. The critical point of the current architecture is the data warehouse which would need to be engineered to make use of partitioning and clustering [21].

6.1.2 User Experience

The dashboard offers a fixed set of visualizations to the user. A different approach would be to have many available visualizations which can be chosen by the user according to the characteristics of the source data. This would give to the user more freedom to explore the data in new ways which may not be envisioned during the design phase.

Bibliography

[1] PeakLens. http://peaklens.com/, November 2017.

[2] Darian Frajberg, Piero Fraternali, and Rocio Nahime Torres. *Convolutional Neural Network for Pixel-Wise Skyline Detection*, pages 12–20. Springer International Publishing, Cham, 2017.

[3] PeakLens - Android Apps on Google Play. https://play.google.com/store/apps/details?id=com.peaklens.ar/, November 2017.

[4] Stephen Few. *Information Dashboard Design: Displaying Data for At-a-Glance Monitoring*. Analytics Press, 2013.

[5] M Solomon, G J Bamossy, S Askegaard, and M K Hogg. *Consumer Behaviour: A European Perspective (3rd Edition)*. Prentice Hall, 2006.

[6] Brian Fisher. *Illuminating the Path: An R&D Agenda for Visual Analytics*. 01 2005.

[7] Ivar Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[8] Paolo Atzeni, Stefano Ceri, Piero Fraternali, Stefano Paraboschi, and Riccardo Torlone. *Basi di dati*. McGraw-Hill, 4 edition, February 2014.

[9] Leonard Richardson, Sam Ruby, and Mike Amundsen. *RESTful Web APIs*. O'Reilly Media, 2013.

[10] PostgreSQL. https://www.postgresql.org/, November 2017.

[11] PostGIS. http://postgis.net/, November 2017.

[12] Node.js. https://nodejs.org/, November 2017.

[13] Express - Node.js web application framework. http://expressjs.com/, November 2017.

[14] Pug. https://pugjs.org/, November 2017.

[15] Knockout. http://knockoutjs.com/, November 2017.

[16] Highstock Financial Javascript Library. https://www.highcharts.com/blog/products/highstock/, November 2017.

[17] Leaflet - a javascript library for interactive maps. http://leafletjs.com/, November 2017.

[18] vis.js - a dynamic, browser based visualization library. http://visjs.org/, November 2017.

[19] node-postgres. https://node-postgres.com/, November 2017.

[20] Pm2 - Advanced Node.js process manager. http://pm2.keymetrics.io/, November 2017.

[21] Database server scaling strategies. http://realscale.cloud66.com/database-server-scaling-strategies/, November 2017.

Appendix A

PeakLens Dashboard REST API

All api URIs are prefixed by `/api/`

A.1 Get renders as GeoJson

Returns the count of the renders performed in the time period specified aggregated in boxes. The response consists of a Json object with the request parameters plus the renders encoded in GeoJson. The encoded renders consist of a series of boxes covering the world, with a parameter `value` containing the number of renders performed in the corresponding box. The response object contains the following members:

- `range: date[]`, the ranges of date computed
- `osVersion: string[]`, the os versions selected
- `peaklensVersion: string[]`, the PeakLens versions selected
- `peaklensFlavor: string[]`, the PeakLens flavors selected
- `data: GeoJson object[]`, the geometry representing the renders of the range of dates

Method: GET
URI: `/renders/geojson`
Parameters:

- `range=[date, ]`, the range of date, optional, default=earliest render to latest render

- `peaklensVersion=[string,...]`, the PeakLens versions selected, optional, default=all
- `peaklensFlavor=[string,...]`, the PeakLens flavors selected, optional, default=all
- `osVersion=[string,...]`, the os versions selected, optional, default=all

Response status: 200, 400, 500
Example request:

```
GET /renders/geojson?range=2017-02-01T00:00:00Z,2017-02-02T12
    :00:00Z&peaklensVersion=1.0.8,1.0.9
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{   "range": ["2017-02-01T00:00:00Z", "2017-02-02T12:00:00Z"],
    "peaklensVersion": ["1.0.8", "1.0.9"],
    "peaklensFlavor": ["store", "beta"],
    "osVersion": ["Android 4", "Android 5", "Android 6", "Android
        7", "Android 8"],
    "data": [
    {   "type": "FeatureCollection",
        "features": [
        {   "type": "Feature",
            "geometry":
            {   "type": "Polygon",
                "coordinates": [
                    [ [0.0, 0.0], [1.0, 0.0], [1.0, 1.0], [0.0,
                        1.0], [0.0, 0.0] ]
                ]
            },
            "properties": {
                "value": 10
            }
        },
        {   "type": "Feature",
            "geometry":
            {   "type": "Polygon",
                "coordinates": [
                    [ [1.0, 0.0], [2.0, 0.0], [2.0, 1.0], [1.0,
                        1.0], [1.0, 0.0] ]
                ]
            },
            "properties": {
                "value": 20
            }
```

```
        }]
    }]
}
```

A.2 Get renders as time series

Returns the count of the renders performed in the time period specified as a time series. The renders are aggregated in time slices of lenght = time period specified / 100. The response object contains the following members:

- `start: date`, the start time
- `end: date`, the end time
- `osVersion: string[]`, the os versions selected
- `peaklensVersion: string[]`, the PeakLens versions selected
- `peaklensFlavor: string[]`, the PeakLens flavors selected
- `data: number[][]`, an array of data points [x, y], where x is the time in ms since the epoch and y is the number of renders

Method: GET
URI: `/renders/series`
Parameters:

- `start=[date]`, starting date, optional, default=earliest render
- `end=[date]`, ending date, optional, default=latest render
- `peaklensVersion=[string,...]`, the PeakLens versions selected, optional, default=all
- `peaklensFlavor=[string,...]`, the PeakLens flavors selected, optional, default=all
- `osVersion=[string,...]`, the os versions selected, optional, default=all

Response status: 200, 400, 500
Example request:

```
GET /renders/series?from=2017-02-01T00:00:00Z&to=2017-02-02T12
    :00:00Z
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{   "start": "2017-02-01T00:00:00Z",
    "end": "2017-02-02T12:00:00Z",
    "peaklensVersion": ["1.0.7", "1.0.8", "1.0.9"],
    "peaklensFlavor": ["store", "beta"],
    "osVersion": ["Android 4", "Android 5", "Android 6", "Android
        7", "Android 8"],
    "data": [
        [1486121190631, 709],
        [1486273868693, 1418],
        [1486426546755, 657],
        [1486579224817, 41]
    ]
}
```

A.3 Get area downloads

Returns the count of downloads of each area occurred in the time period specified. The response consists in a Json object containing the request parameters and a GeoJson object containing the shape of the areas along with the corresponding count of downloads.

The response object contains the following members:

- `start`: date, the start time
- `end`: date, the end time
- `osVersion: string[]`, the os versions selected
- `peaklensVersion: string[]`, the PeakLens versions selected
- `peaklensFlavor: string[]`, the PeakLens flavors selected
- `countries: string[]`, the countries selected Furthermore the GeoJson features have the following parameters:
- `name: string`, the name of the country
- `downloads: number`, the count of downloads for the area
- `description: string`, optional, a description of the area

Method: GET
URI: `/areadownloads`
Parameters:

- `start=date`, starting date, optional, default=earliest area download
- `end=date`, ending date, optional, default=latest area download
- `peaklensVersion=[string, ...]`, the PeakLens versions selected, optional, default=all
- `peaklensFlavor=[string, ...]`, the PeakLens flavors selected, optional, default=all
- `osVersion=[string, ...]`, the os versions selected, optional, default=all
- `countries=[string\, ...]`, the countries selected, optional, default=all

Response status: 200, 400, 500
Example request:

```
GET /areadownloads?start=2017-03-01T00:00:00Z&end=2017-04-01T00
    :00:00Z&countries=Area 50\,Area 51
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{   "start": "2017-03-01T00:00:00Z",
    "end": "2017-04-01T00:00:00Z",
    "peaklensVersion": ["1.0.7", "1.0.8", "1.0.9"],
    "peaklensFlavor": ["store", "beta"],
    "osVersion": ["Android 4", "Android 5", "Android 6", "Android
        7", "Android 8"],
    "countries": ["Area 50", "Area 51"],
    "data":
    {   "type": "FeatureCollection",
        "features": [
        {   "type": "Feature",
            "geometry":
            {   "type": "Polygon",
                "coordinates": [
                    [ [0.0, 0.0], [0.0, 1.0], [1.0, 1.0], [1.0,
                        0.0], [0.0, 0.0] ]
                ]
            },
            "properties": {
                "name": "Area 50",
                "downloads": 10
            }
        },
        {   "type": "Feature",
            "geometry":
```

```
            {   "type": "Polygon",
                "coordinates": [
                    [ [2.0, 0.0], [2.0, 1.0], [3.0, 1.0], [3.0,
                        0.0], [2.0, 0.0] ]
                ]
            },
            "properties": {
                "name": "Area 51",
                "downloads": 20
                "description": "Homey Airport",
            }
        }]
    }
}
```

A.4 Request to 'event' resources

'Events' are something that happened regarding the development of PeakLens. An event is an object with the following members:

- `id: number`
- `name: string`
- **`type`**`: string`
- `description: string`
- `start: date`
- `end: date`, optional

A.4.1 Get event

Returns all the events.

Method: GET
URI: `/events`
Response status: 200, 500
Example request:

```
GET /events
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{   "data": [
    {   "id": 12,
        "name": "Release 2.45.1",
        "type": "development",
        "description": "Release of PeakLens version 3.4.5",
        "start": "2017-05-05T17:31:55Z"
    },
    {   "id": 67,
        "name": "Advertisment XYZ",
        "type": "marketing",
        "description": "Advertisment on Google Adwords",
        "start": "2017-05-01T00:00:00Z",
        "end": "2017-06-01T00:00:00Z"
    }]
}
```

A.4.2 Create event

Creates a new event with the given content.

Method: POST
URI: `/events`
Response status: 201, 400, 500
Example request:

```
POST /events
Content-Type: application/json

{   "name": "Release 1.2.3",
    "type": "development",
    "description": "Release of PeakLens 1.2.3",
    "start": "2017-06-12T16:44:13Z"
}
```

Example response:

```
HTTP/1.1 201 Create
```

A.4.3 Update event

Overwrites a new event with the given content, omitted fields are considered null.

Method: PUT

URI: `/events/:id`
Response status: 200, 400, 404, 500
Example request:

```
PUT /events/10
Content-Type: application/json

{   "name": "Release 2.3.4",
    "type": "development",
    "description": "Release of PeakLens 2.3.4",
    "start": "2017-06-12T16:44:13Z"
}
```

Example response:

```
HTTP/1.1 200 OK
```

A.4.4 Delete event

Deletes an event.

Method: DELETE
URI: `/events/:id`
Response status: 204, 400, 404, 500
Example request:

```
DELETE /events/10
```

Example response:

```
HTTP/1.1 204 No Content
```

A.5 Get installs and uninstalls

Returns the installs and uninstalls satisfying the parameters specified. The data granularity of the data returned is fixed to one day. The response object contains one or all (if no parameter has been specified in the request) the following members:

- `appVersions: string[]`, the PeakLens versions selected
- `carriers: string[]`, the carriers selected
- `countries: string[]`, the country codes selected
- `devices: string[]`, the device models selected

- `languages: string[]`, the language codes selected
- `osVersions: string[]`, the os versions selected
- `tablets: string[]`, the tablet types selected The response object always contains the following members:
- `installs: number[][]`, the series representing the installs [unix time, num. of installs]
- `uninstalls: number[][]`, the series representing the uninstalls [unix time, num. of uninstalls]

Method: GET
URI: `/installs`
Response status: 204, 400, 500
Parameters: Only one parameter at a time can be specified because of the limitation of the data available from the Google Play Store.

- `appVersions=[string\,...]`, the PeakLens versions selected, optional, default=all
- `carriers=[string\,...]`, the carriers selected, optional, default=all
- `countries=[string\,...]`, the country codes selected, optional, default=all
- `devices=[string\,...]`, the device models selected, optional, default=all
- `languages=[string\,...]`, the language codes selected, optional, default=all
- `osVersion=[string\,...]`, the os versions selected, optional, default=all
- `tablets=[string\,...]`, the tablet types selected, optional, default=all

Example request:

```
GET /installs?languages=it_IT\,es_ES
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

HTTP/1.1 200 OK
{   "languages": ["it_IT\,es_ES"],
    "installs": [
        [1511740800000,175],
```

```
        [1511827200000,114],
        [1511913600000,79]
    ],
    "uninstalls":[
        [1511740800000,95],
        [1511827200000,89],
        [1511913600000,89]
    ]
}
```