iSoft

Statistical Analysis
Software Architecture Document (SAD)

CONTENT OWNER: Carolina Barros, Mafalda Landeiro, Manuel Correia e Nuno Bento

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
• 1.0	• 1.1	• 22/12/2016

Table of Contents

List of Figures

List of Tables

1 Documentation Roadmap

The Documentation Roadmap should be the first place a new reader of the SAD begins. But for new and returning readers, it is intended to describe how the SAD is organized so that a reader with specific interests who does not wish to read the SAD cover-to-cover can find desired information quickly and directly.

Sub-sections of Section 1 include the following.

- Section **Erro! A origem da referência não foi encontrada.** ("How the SAD Is Organized") explains the information that is found in each section of the SAD. This tells you what section(s) in this SAD are most likely to contain the information you seek.
- Section **Erro! A origem da referência não foi encontrada.** ("Viewpoint Definitions") explains the *viewpoints* (as defined by IEEE Standard 1471-2000) used in this SAD. For each viewpoint defined in Section 1.2, there is a corresponding view defined in Section 3 ("Views"). This tells you how the architectural information has been partitioned, and what views are most likely to contain the information you seek.
- Section 1.1 ("How a View is Documented") explains the standard organization used to document architectural views in this SAD. This tells you what section within a view you should read in order to find the information you seek.

1.1 How the SAD Is Organized

CONTENTS OF THIS SECTION: This section provides a narrative description of the major sections of the SAD and the overall contents of each. Readers seeking specific information can use this section to help them locate it more quickly.

This SAD is organized into the following sections:

- **Section 1 ("Documentation Roadmap") provides information about this document and its intended audience**. It provides the roadmap and document overview. Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use it, and where information may be found. Section 1 also provides information about the views that are used by this SAD to communicate the software architecture.
- **Section 2 ("Architecture Background") explains why the architecture is what it is.** It provides a system overview, establishing the context and goals for the development. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. It includes information about evaluation or validation performed on the architecture to provide assurance it meets its goals.

- **Section 3 (Views") and Section 4 ("Relations Among Views") specify the software architecture.** Views specify elements of software and the relationships between them. A view corresponds to a viewpoint (see Section 1.5), and is a representation of one or more structures present in the software (see Section 1.2).
- **Sections 5 ("Referenced Materials") and 6 ("Directory") provide reference information for the reader.** Section 5 provides look-up information for documents that are cited elsewhere in this SAD. Section 6 is a directory, which is an index of architectural elements and relations telling where each one is defined and used in this SAD. The section also includes a glossary and acronym list.

1.2 Viewpoint Definitions

CONTENTS OF THIS SECTION: This section provides a short textual definition of a viewpoint and how the concept is used in this SAD. The section describes viewpoints that may be used in the SAD. The specific viewpoints will be tailored by the organization.

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471].

As described in Section **Erro! A origem da referência não foi encontrada.**, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A *view* is the specification of one or more of these structures, and documenting a software architecture, then, is a matter of documenting the relevant views and then documenting information that applies to more than one view [Clements 2002].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system.

The remainder of Section 1.5 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	**Viewpoint(s) that apply to that class of stake-holder's concerns**
Users	C&C View, Deployment View.
Acquirers	C&C View, Deployment View.
Developers	Functional View, Context Diagram View, Module De-composition & Uses View, C&C View, Module Layer View, Deployment View, Uses Cases
Maintainers	Functional View, Context Diagram View, Module De-composition & Uses View, C&C View, Module Layer View, Deployment View, Uses Cases

1.3 How a View is Documented

CONTENTS OF THIS SECTION: This section describes how the documentation for a view is structured and organized. If you change the *organization* of information in Section 3, then you should also change its description in here. Otherwise, this section is all boilerplate.

If you choose to document all information in a view in a single presentation, then you will not need view packets. In that case, the template is as follows:

- Section 3.i: Name of view
- Section 3.i.1: View description
- Section 3.i.2: Primary presentation. This section presents the elements and the relations among them that populate this view packet, using an appropriate language, languages, notation, or tool-based representation.
- Section 3.i.3: Element catalog. Whereas the primary presentation shows the important elements and relations of the view packet, this section provides additional information needed to complete the architectural picture. It consists of subsections for (respectively) elements, relations, interfaces, behavior, and constraints.
- Section 3.i.4: Context diagram. This section provides a context diagram showing the context of the part of the system represented by this view packet. It also designates the view packet's scope with a distinguished symbol, and shows interactions with external entities in the vocabulary of the view.
- Section 3.i.5: Variability mechanisms. This section describes any variabilities that are available in the portion of the system shown in the view packet, along with how and when those mechanisms may be exercised.
- Section 3.i.6: Architecture background. This section provides rationale for any significant design decisions whose scope is limited to this view packet.

Section 3 of this SAD contains one view for each viewpoint listed in Section 1.5. Each view is documented as a set of view packets. A view packet is the smallest bundle of architectural documentation that might be given to an individual stakeholder.

Each view is documented as follows, where the letter *i* stands for the number of the view: 1, 2, etc.:

- Section 3.i: Name of view.
- Section 3.i.1: View description. This section describes the purpose and contents of the view. It should refer to (and match) the viewpoint description in Section 1.5 to which this view conforms.
- Section 3.i.2: View packet overview. This section shows the set of view packets in this view, and provides rationale that explains why the chosen set is complete and non-duplicative. The set of view packets may be listed textually, or shown graphically in terms of how they partition the entire architecture being shown in the view.
- Section 3.i.3: Architecture background. Whereas the architecture background of Section 2 pertains to those constraints and decisions whose scope is the entire architecture, this section provides any architecture background (including significant driving requirements, design approaches, patterns, analysis results, and requirements coverage) that applies to this view.
- Section 3.i.4: Variability mechanisms. This section describes any architectural variability mechanisms (e.g., adaptation data, compile-time parameters, variable replication, and so forth) described by this view, including a description of how and when those mechanisms may be exercised and any constraints on their use.
- Section 3.i.5: View packets. This section presents all of the view packets given for this view. Each view packet is described using the following outline, where the letter *j* stands for the number of the view packet being described: 1, 2, etc.
  - Section 3.i.5.j: View packet #j.
  - Section 3.i.5.j.1: Primary presentation. This section presents the elements and the relations among them that populate this view packet, using an appropriate language, languages, notation, or tool-based representation.
  - Section 3.i.5.j.2: Element catalog. Whereas the primary presentation shows the important elements and relations of the view packet, this section provides additional information needed to complete the architectural picture. It consists of the following subsections:
    - Section 3.i.5.j.2.1: Elements. This section describes each element shown in the primary presentation, details its responsibilities of each element, and specifies values of the elements' relevant *properties*, which are defined in the viewpoint to which this view conforms.
    - Section 3.i.5.j.2.2: Relations. This section describes any additional relations among elements shown in the primary presentation, or specializations or restrictions on the relations shown in the primary presentation.
    - Section 3.i.5.j.2.3: Interfaces. This section specifies the software interfaces to any elements shown in the primary presentation that must be visible to other elements.
    - Section 3.i.5.j.2.4: Behavior. This section specifies any significant behavior of elements or groups of interacting elements shown in the primary presentation.
    - Section 3.i.5.j.2.5: Constraints: This section lists any constraints on elements or relations not otherwise described.
  - Section 3.i.5.j.3: Context diagram. This section provides a context diagram showing the context of the part of the system represented by this view packet. It also designates the

view packet's scope with a distinguished symbol, and shows interactions with external entities in the vocabulary of the view.

- Section 3.i.5.j.4: Variability mechanisms. This section describes any variabilities that are available in the portion of the system shown in the view packet, along with how and when those mechanisms may be exercised.
- Section 3.i.5.j.5: Architecture background. This section provides rationale for any significant design decisions whose scope is limited to this view packet.
- Section 3.i.5.j.6: Relation to other view packets. This section provides references for related view packets, including the parent, children, and siblings of this view packet. Related view packets may be in the same view or in different views.

2 Architecture Background

2.1 Problem Background

CONTENTS OF THIS SECTION: The sub-parts of Section 2.1 explain the constraints that provided the significant influence over the architecture.

2.1.1 System Overview

CONTENTS OF THIS SECTION: This section describes the general function and purpose for the system or subsystem whose architecture is described in this SAD.

The name of our system is iStat, a system to analyse statistic data, to perform operations and to transform numeric data, through a web application. The user should be able to work with multiple sets of data at once: apply statistical calculations, perform transformations and draw graphics. Our system is composed by three subsystems: a web component named "iStat.com" and responsible for the presentation and user's interaction; a persistence component named "iStatDB" responsible for data storage; finally, an API component named "iStat.com/api" responsible for managing the entire application's logic.

2.1.2 Goals and Context

CONTENTS OF THIS SECTION: This section describes the goals and major contextual factors for the software architecture. The section includes a description of the role software architecture plays in the life cycle, the relationship to system engineering results and artifacts, and any other relevant factors.

The iStat architecture is spared in three concerns: user interface, logic and persistence. This separation was made with the purpose of having more flexibility on the application.

The user interface is a RIA, so it has a better accessibility, performance and availability, since it has a great user experience, only made requests to the server on more complex operations and the user can be working while the data is being transferred.

The logic has three layers (iServices, iBLL and iDAL) and applies the principle of responsibility, which allows to have extensibility. The fact of logic not being together with the user interface gives more security to the system and it also allows to make modifications on each part without having to do big changes on the other, like for example the implementation for mobile devices.

The team made the following assumptions for the design of iStat architecture:

• Use of external libraries for import/export datasets (which is not decided yet) and graphics design (Chart.js).

With the purpose of making the development easier, we decide to use some external libraries for developing these functionalities. The choice of Chart.js was made because is a library with the charts that are need, easy to implement and responsive.

• The web application only runs in online mode.

This decision was made for ensuring that the user is always online and he has access to all the business logic.

• Use of non-relational database.

Given the lack of business rules and the simplicity of data manipulation (inserts, updates, and readings). Also, we wanted to improve the performance of data manipulation and simplify the database management.

2.1.3 Significant Driving Requirements

CONTENTS OF THIS SECTION: This section describes behavioral and quality attribute requirements (original or derived) that shaped the software architecture. Included are any scenarios that express driving behavioral and quality attribute goals, such as those crafted during a Quality Attribute Workshop (QAW) [Barbacci 2003] or software architecture evaluation using the Architecture Tradeoff Analysis Method^{SM} (ATAM^{SM}) [Bass 2003].

During the evaluation of the use cases provided, for this phase of the work we took into account those that would have a greater impact from the point of view of system's architecture. Thus, we chose to describe three use cases: the calculation of the dataset, which includes all possible operations, from median to standard deviation calculation; the save of the dataset, which symbolizes the process of saving a dataset on database; finally, we considered relevant to describe the import/export process of the dataset since we used an external library for conversion, export and import of different types of files.

^{SM} Quality Attribute Workshop and QAW and Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

3 Views

CONTENTS OF THIS SECTION: The sub-parts of Section 3 specify the views corresponding to the viewpoints listed in Section 1.5.

This section contains the views of the software architecture. A view is a representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. Concretely, a view shows a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

Architectural views can be divided into three groups, depending on the broad nature of the elements they show. These are:

- Module views. Here, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility, and are assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as: What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?
- Component-and-connector views. Here, the elements are runtime components (which are principal units of computation) and connectors (which are the communication vehicles among components). Component and connector structures help answer questions such as: What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel? How can the system's structure change as it executes?
- Allocation views. These views show the relationship between the software elements and elements in one or more external environments in which the software is created and executed. Allocation structures answer questions such as: What processor does each software element execute on? In what files is each element stored during development, testing, and system building? What is the assignment of the software element to development teams?

These three kinds of structures correspond to the three broad kinds of decisions that architectural design involves:

- How is the system to be structured as a set of code units (modules)
- How is the system to be structured as a set of elements that have run-time behavior (components) and interactions (connectors) ?
- How is the system to relate to non-software structures in its environment (such as CPUs, file systems, networks, development teams, etc.)?

Often, a view shows information from more than one of these categories. However, unless chosen carefully, the information in such a hybrid view can be confusing and not well understood.

The views presented in this SAD are the following:

Name of view	**Viewtype that defines this view**	**Types of elements and relations shown**		**Is this a module view?**	**Is this a component-and-connector view?**	**Is this an allocation view?**

3.1 Functional View

3.1.1 View Description

The functional view is the easiest view for stakeholders to understand. This view defines the architectural elements that deliver the system's functionality. The goal is to specify what is really architectural relevant, or in other words, what has a visible impact on system’s architecture. As the rules says, our focus was to select, from the use cases given, the ones that were relevant for the system’s architecture and we agree that the use cases of save dataset, the calculation and finally, the import/export a dataset were the ones that could represent the variety of functionalities required for this application.

3.1.2 Primary Representation

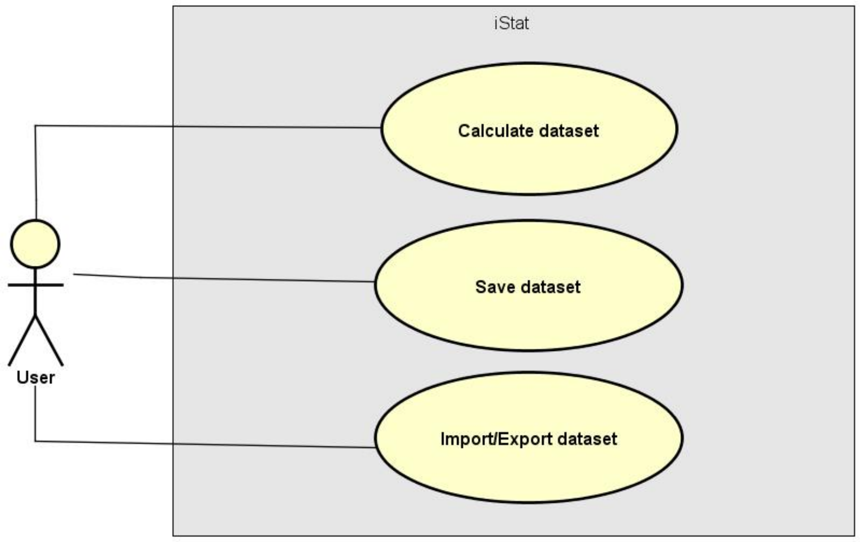

Illustration 1 - Functional Diagram

3.1.3 Element Catalog

3.1.3.1 Elements

- User : Our application user.

3.1.3.2 Relations

Not applicable.

3.1.3.3 Interfaces

Not applicable.

3.1.3.4 Behavior

Not applicable.

3.1.3.5 Constraints

Not applicable.

3.1.4 Context Diagram

3.1.5 Variability mechanisms

3.1.6 Architecture Background

3.2 Context Diagram View

3.2.1 View Description

A context diagram view is a high level view of a system and defines the boundary between the system, or part of it, and its environment, showing the entities that interact with it.

In our diagram, the user of the application is represented and his interactions with the "iStat", which means the application. In turn, the application communicates with external libraries, which have the responsibility to help the system deal with graphic design (‘UtilsGraph’) and with the files manipulation (‘UtilsInOut’).

3.2.2 Primary Representation

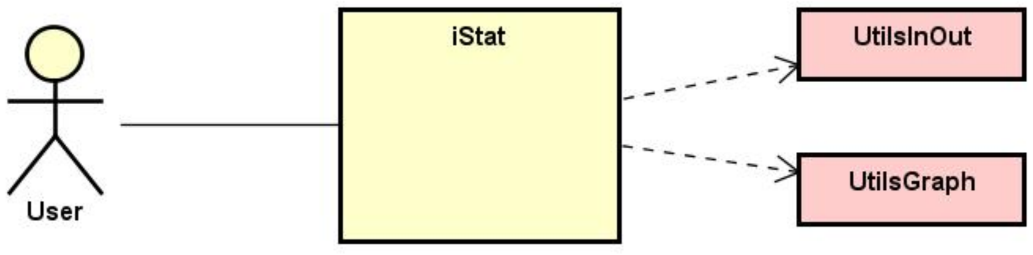

Illustration 2 - Context Diagram

3.2.3 Element Catalog

3.2.3.1 Elements

- User: Our application user;
- iStat: Our application;
- UtilsGraph: Our Graph API external library;
- UtilsInOut: Our Import/Export API external library.

3.2.3.2 Relations

Not applicable.

3.2.3.3 Interfaces

Not applicable.

3.2.3.4 Behavior

Not applicable.

3.2.3.5 Constraints

Not applicable.

3.2.4 Context Diagram

3.2.5 Variability mechanisms

3.2.6 Architecture Background

3.3 Module Decomposition & Uses View

3.3.1 View Description

A high-level module decomposition & uses view is about partitioning the system into parts that are easier to conceive, understand, program and maintain. Giving independence to the components, they can be simply replaced by others, keeping the system working.

Our web application communicates with the iStat.com/api, which is written in Java, through http requests, and this one interacts with the database by **jdbc** requests. Finally, both application and server uses external API's to create graphs and import/export data files, respectively.

3.3.2 Primary Representation

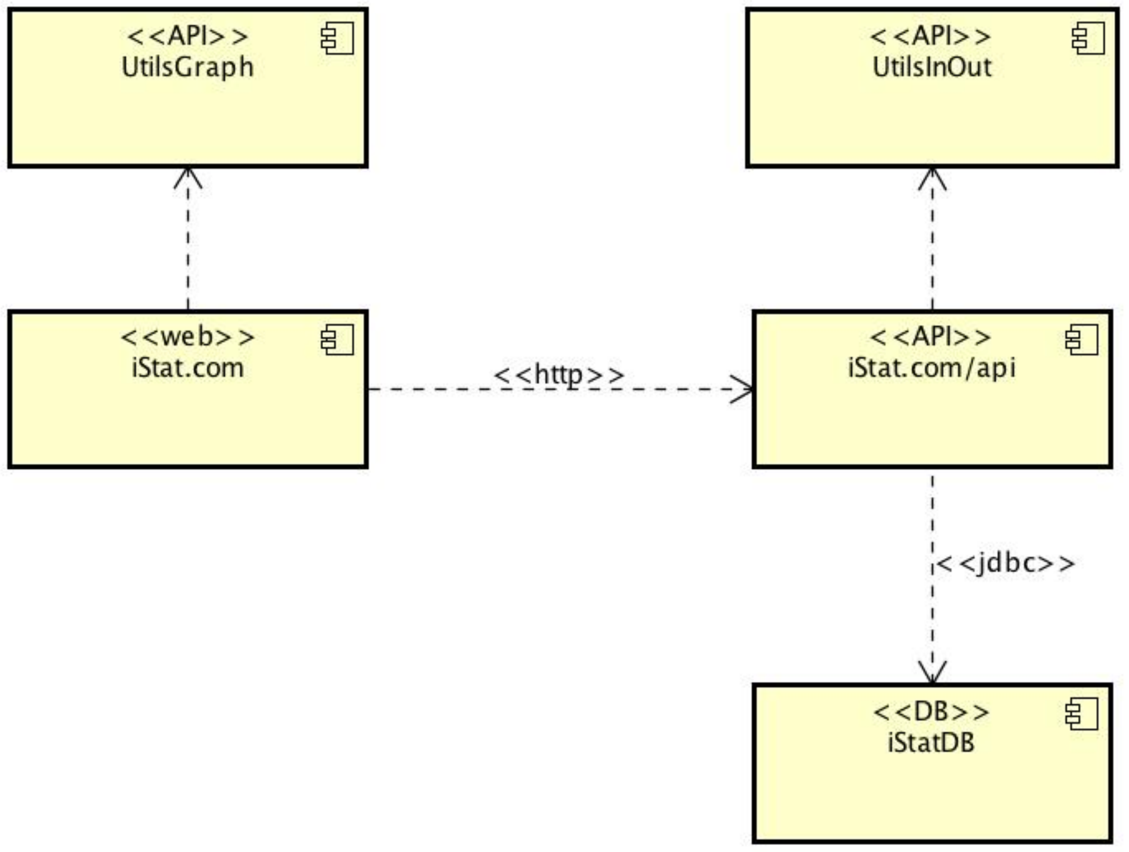

Illustration 3 - Module Decomposition & Uses Diagram

3.3.3 Element Catalog

3.3.3.1 Elements

- iStat.com: Our web application, which uses Bootstrap and JQuery;
- iStat.com/api: Our API, which is written in Java and uses the framework Spring;
- iStatDB: Our database, which is MongoDB, a non-relational database;
- UtilsGraph: External library to draw graphs;
- UtilsInOut: External library to import and export different types of data files.

3.3.3.2 Relations

- iStat.com > UtilsGraph: Our application interacts with an external library to create graphs, on client side, using JavaScript;

- iStat.com > iStat.com/api: Our application interacts with the API, using http requests. The API has a group of services that allows to enrich the user interface;
- iStat.com/api > UtilsInOut: Our API interacts with an external library to import and export data files;
- iStat.com/api > iStatDB: Our API interacts with the database for data persistence, using jdbc.

3.3.3.3 Interfaces

Not applicable.

3.3.3.4 Behavior

Not applicable.

3.3.3.5 Constraints

Not applicable.

3.3.4 Context Diagram

3.3.5 Variability mechanisms

3.3.6 Architecture Background

3.4 C&C View

3.4.1 View Description

A high-level C&C view enable us to see the system as a collection of runtime entities called components. During execution, the components need to interact with others to support the system services. This interaction is provided by connectors.

Our view is mainly composed by three components: the web application "**iStat.com**", the API "**iStat.com/api**" and the database "**iStatDB**". The application interacts with the API by five different connectors, each one of them using a unique interface, but they all communicate using http requests. That decision was made with the purpose of maintain the single responsibility principle and to guarantee the possibility of future implementations, like for example, for mobile devices. Finally, the API communicates with the database, using **jdbc** requests.

3.4.2 Primary Representation

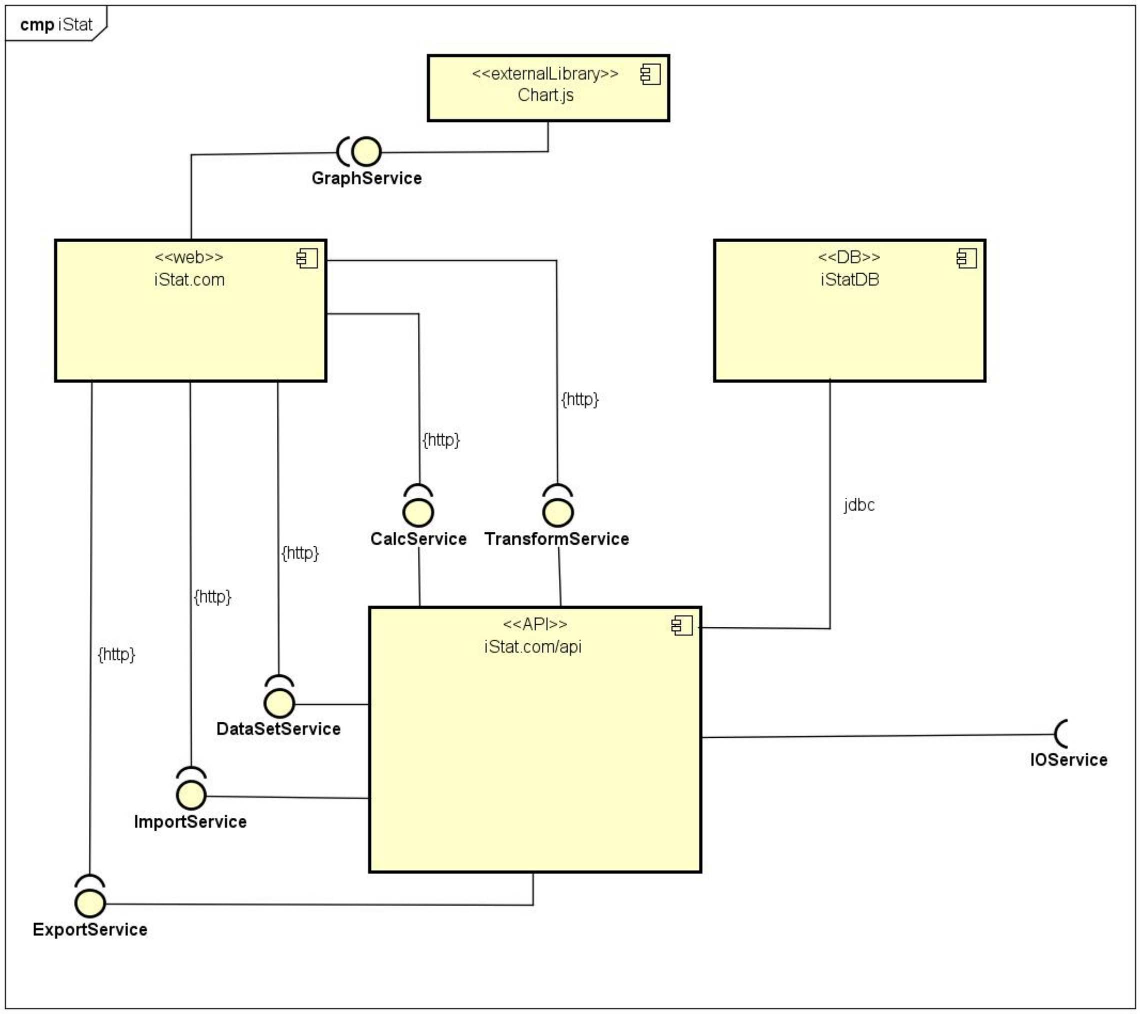

Illustration 4 - C&C Diagram

3.4.3 Element Catalog

3.4.3.1 Elements

- iStat.com: Our web application;
- iStat.com/api: Our API;
- iStatDB: Our database.

3.4.3.2 Relations

- iStat.com > iStat.com/api: Our web application interacts with the API, using http requests;
- iStat.com > Chart.js: Our web application interacts with an extern library to create graphs;
- iStat.com/api > iStatDB: Our API interacts with de database for data persistence, using **jdbc** requests;
- iStat.com/api > UtilsInOut: Our API interacts with an extern library to import and export data files.

3.4.3.3 Interfaces

- CalcService: Gives an interface for the web application communicate with the API, using http requests. This interface allows to get all the services related with the calculation operation (For example: calculate the median);
- TransformService: Gives an interface for the web application communicate with the API, using http requests. This interface allows to get all the services related with the transform operation (For example: add a scalar);
- DataSetService: Gives an interface for the web application communicate with the API, using http requests. This interface allows to get all the services related with the dataset manipulation (For example: save dataset);
- ImportService: Gives an interface for the web application communicate with the API, using http requests. This interface allows to get all the services related with the dataset import (For example: import XML);
- ExportService: Gives an interface for the web application communicate with the API, using http requests. This interface allows to get all the services related with the dataset export (For example: export as XML);
- GraphService: Gives an interface for the web application communicate with the external library. This interface allows to get all the services related with the graph drawing (For example: pie chart);
- IOService: Gives an interface for the API communicate with the external library. This interface allows to get all the services related with the files manipulation (For example: convert from json to xml file).

3.4.3.4 Behavior

Not applicable.

3.4.3.5 Constraints

Not applicable.

3.4.4 Context Diagram

3.4.5 Variability mechanisms

3.4.6 Architecture Background

3.5 Module Layer View

3.5.1 View Description

A module layered diagram describes the architecture of the code, giving to us an overview of the components and also the relations and dependencies between them.

Our system is mainly composed by three layers. The "**iStat.com**" layer knows the "**iStat.com/api**" layer, which aggregates three segments. The first, nominated "**iServices**" contains the interfaces consumed by the services and he knows the "**iBLL**" segment. This one is responsible for the business logic and it knows the "**iDAL**" segment, where the data is going to be manipulated. Finally, this last one is connected to the "**iStatDB**" layer, responsible for the data persistence.

3.5.2 Primary Representation

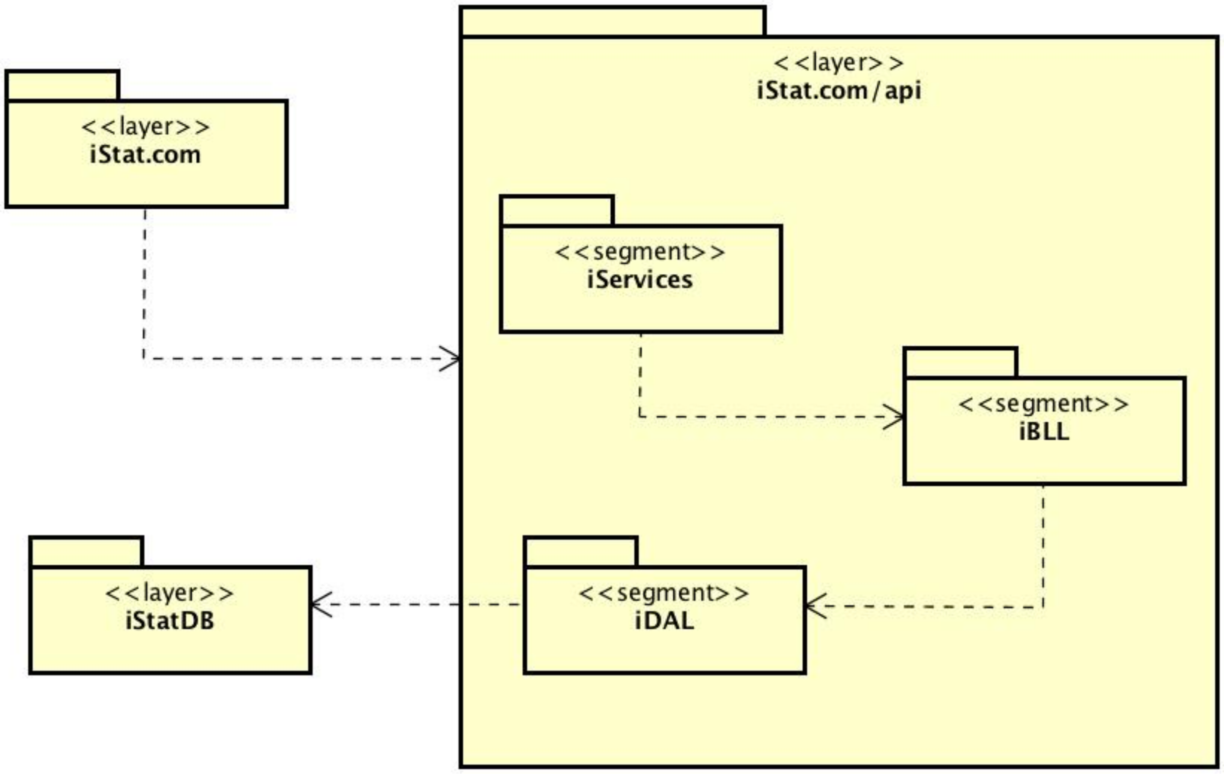

Illustration 5 - Module Layer Diagram

### 3.5.3 Element Catalog

#### 3.5.3.1 Elements

- iStat.com: Contains all the elements that are used on client side.
- iStatDB: Contains the application database.
- iStat.com/api:
    - iServices: Contains the interfaces consumed by the services;
    - iBLL: Responsible for the business logic;
    - iDAL: Responsible for the data manipulation.

#### 3.5.3.2 Relations

- iStat.com > iStat.com/api: Our client side communicates with the API, for dealing with business logic, using the services interfaces.

- iServices > iBLL: The segment of services are responsible for delegating the requests for the corresponding business logic.
- iBLL > iDAL: The segment of business logic is responsible for transmitting the necessity of dealing with data, using DAL.
- iDAL > iStatDB: The segment of data manipulation is responsible for knowing were to get and change the data, which are on database.

3.5.3.3 Interfaces

Not applicable.

3.5.3.4 Behavior

Not applicable.

3.5.3.5 Constraints

Not applicable.

3.5.4 Context Diagram

3.5.5 Variability mechanisms

3.5.6 Architecture Background

3.6 Deployment View

3.6.1 View Description

A deployment diagram describes the system architecture in terms of hardware and its relationship, with the different components (software).

We have two main nodes, one of them is the browser and the other is our Linux server. The web application runs in a browser, which can be any browser, and it communicates with our Linux server by http requests.

The iStat.com is a RIA, using Bootstrap, because it allows the system to have a richer user experience, it has better performance, since the connections to the server are only made if necessary and let the user work while is the data is being transferred in background. For this we choose the language JavaScript, since there is a lot of documentation and all the team has already experience with that.

Finally, our Linux server contains two components, an API written in Java and a MongoDB database. Java was chosen since it was the common knowledge between all team members and it is a language with a variety of libraries and frameworks (for example: Spring) that could make the implementation easier. As we explain before, we chose a non-relational database, so MongoDB was chosen because it has a great performance and a lot of documentation, it is easy to replicate and to scale and it has high availability. Even though, having the API and the database on the same server can generate some performance issues, we decide that it is better this way because like this we can reduce latency of exchanging data.

3.6.2 Primary Representation

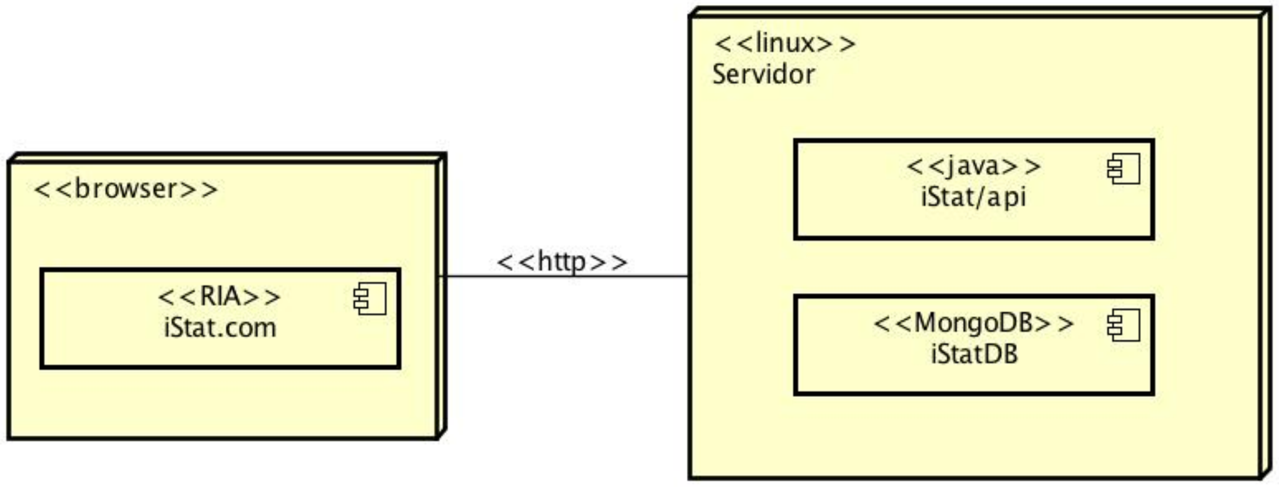

Illustration 6 - Deployment Diagram

3.6.3 Element Catalog

3.6.3.1 Elements

- iStat.com: Our web application, which is a RIA;
- iStat/api: Our API, written in Java.
- iStatDB: Our database, which is in MongoDB.

3.6.3.2 Relations

- Browser > Linux: All the connections between browser and server are made by http requests.

3.6.3.3 Interfaces

Not applicable.

3.6.3.4 Behavior

Not applicable.

3.6.3.5 Constraints

Not applicable.

3.6.4 Context Diagram

3.6.5 Variability mechanisms

3.6.6 Architecture Background

4 Uses Cases

4.1 Use Case 1: Calculate Median

4.1.1 View Description

Over this subsection, we will describe the use case "Calculate dataset", specifically for the median calculation, which can be applied to either a column, a row or even a complete dataset. We chose to do three sequence diagrams: the first shows the interaction of the client application with the backend's services; the second only describes the interaction in the client application part; and the last one the interaction in the backend's services part. For each sequence diagram represented, a class diagram was created. In this case, the class diagrams have already contemplated the operations relative to other types of calculations possible to realize. For this use case, it was used the singleton on CalcService, because the increase of requests to the web service could affect the application performance, since it has to be always creating this class.

4.1.2 Sequence Diagram: Client - Backend

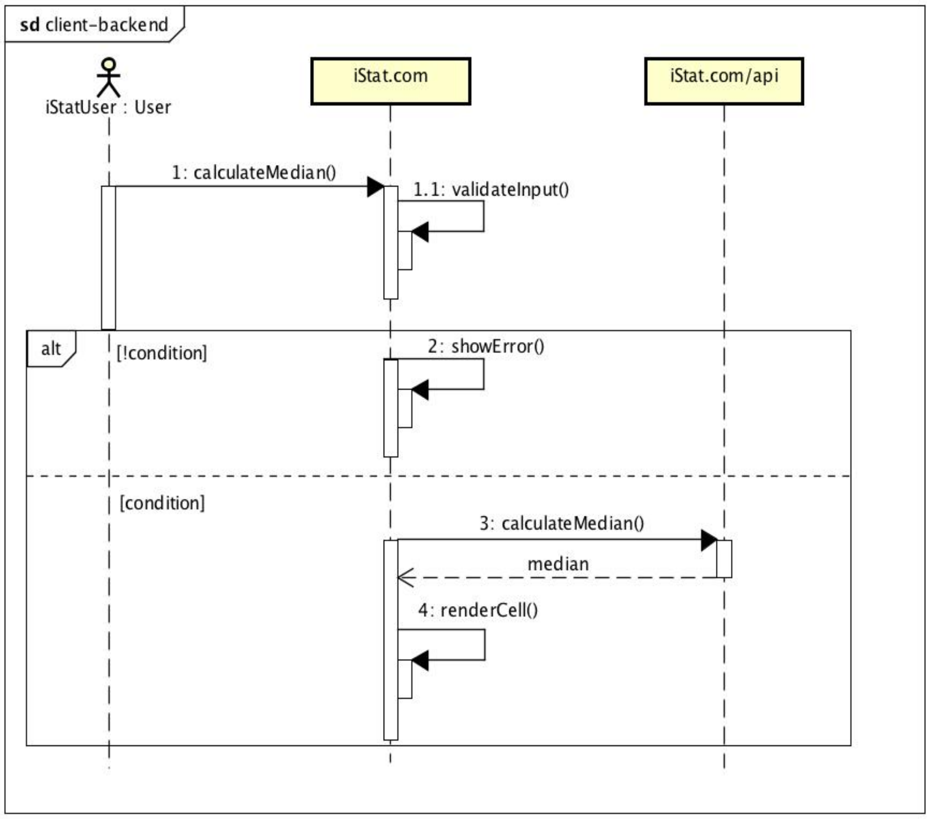

Illustration 7 - Sequence Diagram: Calculate Median (Client - Backend)

4.1.3 Sequence Diagram: Client

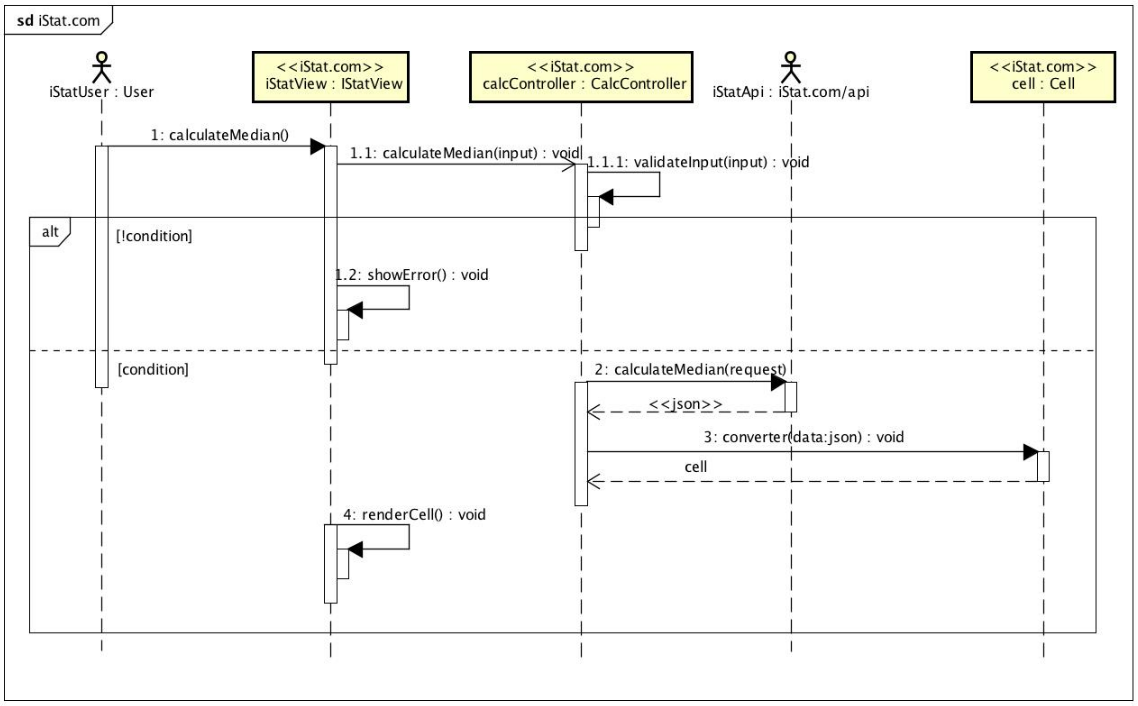
sd iStat.com
iStatUser : User
<<iStat.com>> iStatView : IStatView
<<iStat.com>> calcController : CalcController
iStatApi : iStat.com/api
<<iStat.com>> cell : Cell
1: calculateMedian()
1.1: calculateMedian(input) : void
1.1.1: validateInput(input) : void
alt
[!condition]
1.2: showError() : void
[condition]
2: calculateMedian(request)
<<json>>
3: converter(data:json) : void
cell
4: renderCell() : void

Illustration 8 - Sequence Diagram: Calculate Median (Client)

4.1.4 Sequence Diagram: Backend

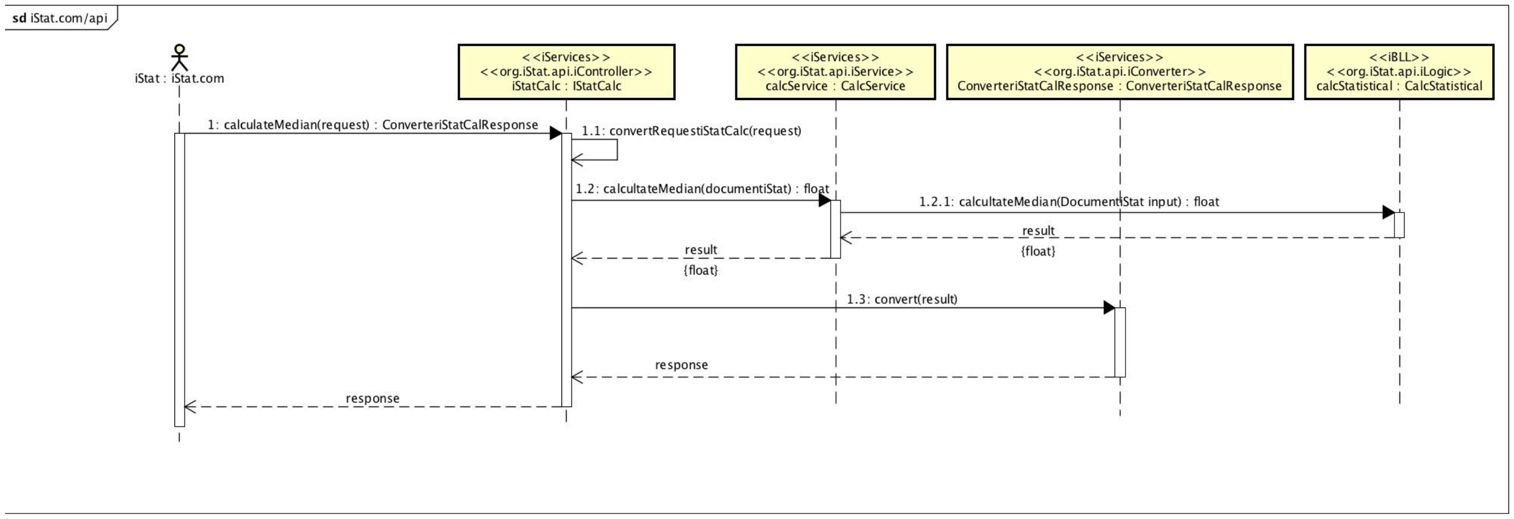
sd iStat.com/api
iStat : iStat.com
<<iServices>> <<org.iStat.api.iController>> iStatCalc : IStatCalc
<<iServices>> <<org.iStat.api.iService>> calcService : CalcService
<<iServices>> <<org.iStat.api.iConverter>> ConverteriStatCalResponse : ConverteriStatCalResponse
<<iBLL>> <<org.iStat.api.iLogic>> calcStatistical : CalcStatistical
1: calculateMedian(request) : ConverteriStatCalResponse
1.1: convertRequestiStatCalc(request)
1.2: calcultateMedian(documentiStat) : float
1.2.1: calcultateMedian(DocumentiStat input) : float
result
{float}
result
{float}
1.3: convert(result)
response
response

Illustration 9 - Sequence Diagram: Calculate Median (Backend)

4.1.5 Class Diagram: Client

4.1.6 Class Diagram: Backend

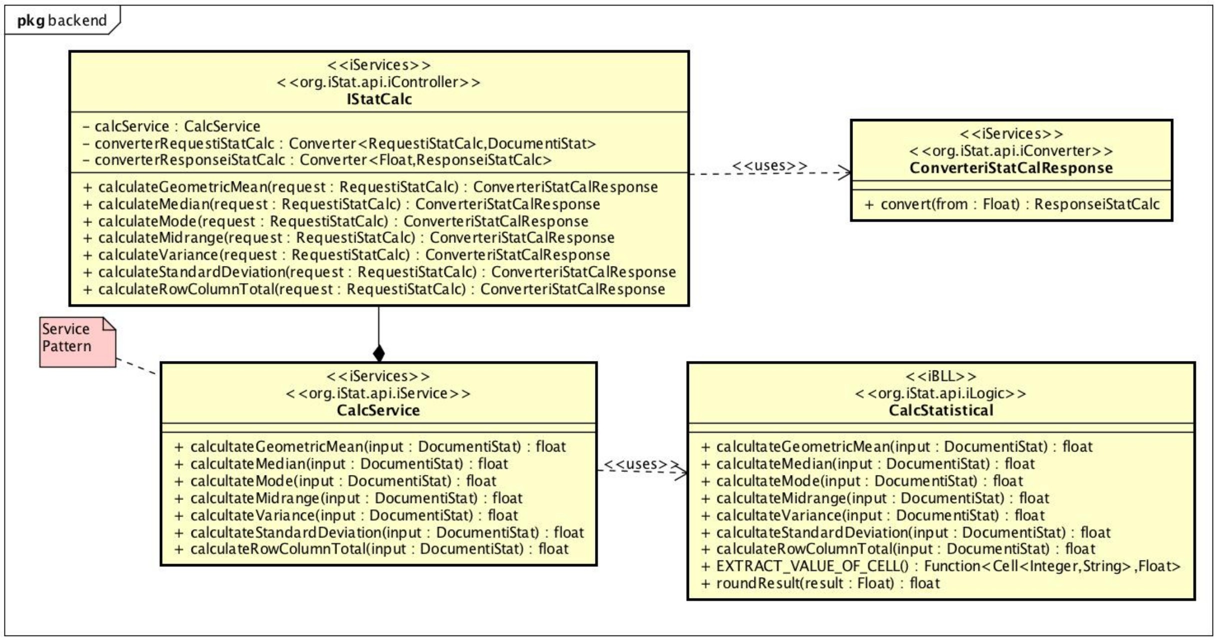
pkg backend
<<iServices>>
<<org.iStat.api.iController>>
IStatCalc
- calcService : CalcService
- converterRequestiStatCalc : Converter<RequestiStatCalc,DocumentiStat>
- converterResponseiStatCalc : Converter<Float,ResponseiStatCalc>
+ calculateGeometricMean(request : RequestiStatCalc) : ConverteriStatCalResponse
+ calculateMedian(request : RequestiStatCalc) : ConverteriStatCalResponse
+ calculateMode(request : RequestiStatCalc) : ConverteriStatCalResponse
+ calculateMidrange(request : RequestiStatCalc) : ConverteriStatCalResponse
+ calculateVariance(request : RequestiStatCalc) : ConverteriStatCalResponse
+ calculateStandardDeviation(request : RequestiStatCalc) : ConverteriStatCalResponse
+ calculateRowColumnTotal(request : RequestiStatCalc) : ConverteriStatCalResponse
<<uses>>
<<iServices>>
<<org.iStat.api.iConverter>>
ConverteriStatCalResponse
+ convert(from : Float) : ResponseiStatCalc
Service Pattern
<<iServices>>
<<org.iStat.api.iService>>
CalcService
+ calculateGeometricMean(input : DocumentiStat) : float
+ calculateMedian(input : DocumentiStat) : float
+ calculateMode(input : DocumentiStat) : float
+ calculateMidrange(input : DocumentiStat) : float
+ calculateVariance(input : DocumentiStat) : float
+ calculateStandardDeviation(input : DocumentiStat) : float
+ calculateRowColumnTotal(input : DocumentiStat) : float
<<uses>>
<<iBLL>>
<<org.iStat.api.iLogic>>
CalcStatistical
+ calcultateGeometricMean(input : DocumentiStat) : float
+ calculateMedian(input : DocumentiStat) : float
+ calculateMode(input : DocumentiStat) : float
+ calculateMidrange(input : DocumentiStat) : float
+ calculateVariance(input : DocumentiStat) : float
+ calculateStandardDeviation(input : DocumentiStat) : float
+ calculateRowColumnTotal(input : DocumentiStat) : float
+ EXTRACT_VALUE_OF_CELL() : Function<Cell<Integer,String>,Float>
+ roundResult(result : Float) : float

Illustration 10 - Class Diagram: Calculate (Client)

4.2 Use Case 2: Save Data Set

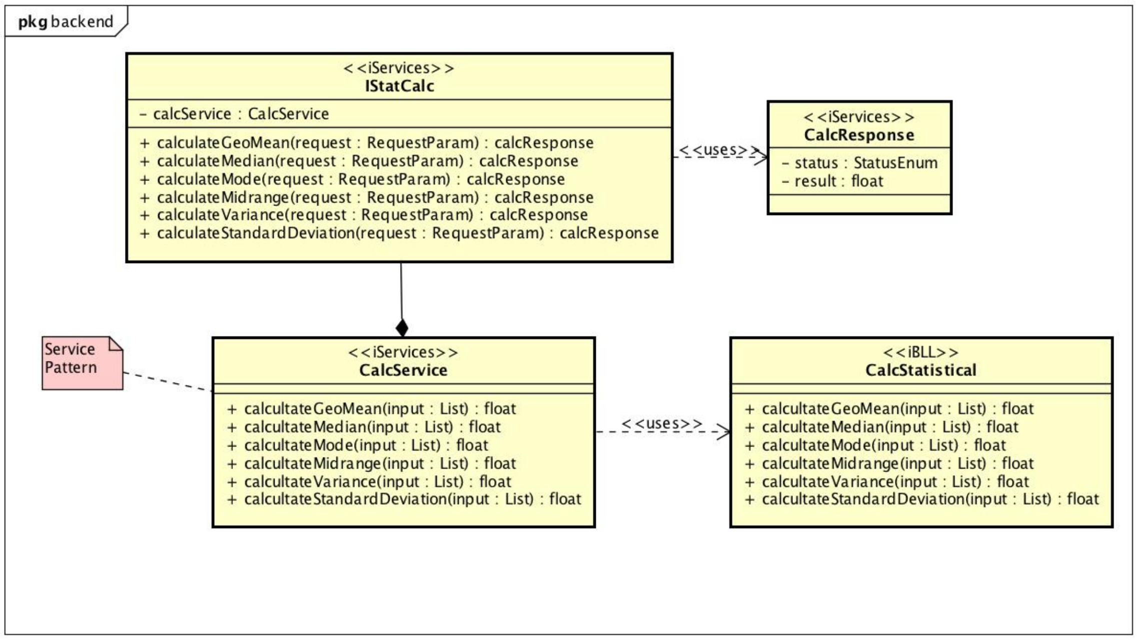
pkg backend
<<iServices>>
IStatCalc
- calcService : CalcService
+ calculateGeoMean(request : RequestParam) : calcResponse
+ calculateMedian(request : RequestParam) : calcResponse
+ calculateMode(request : RequestParam) : calcResponse
+ calculateMidrange(request : RequestParam) : calcResponse
+ calculateVariance(request : RequestParam) : calcResponse
+ calculateStandardDeviation(request : RequestParam) : calcResponse
<<uses>>
<<iServices>>
CalcResponse
- status : StatusEnum
- result : float
Service Pattern
<<iServices>>
CalcService
+ calculateGeoMean(input : List) : float
+ calculateMedian(input : List) : float
+ calculateMode(input : List) : float
+ calculateMidrange(input : List) : float
+ calculateVariance(input : List) : float
+ calculateStandardDeviation(input : List) : float
<<uses>>
<<iBLL>>
CalcStatistical
+ calculateGeoMean(input : List) : float
+ calculateMedian(input : List) : float
+ calculateMode(input : List) : float
+ calculateMidrange(input : List) : float
+ calculateVariance(input : List) : float
+ calculateStandardDeviation(input : List) : float

Illustration 11 - Class Diagram: Calculate (Backend)

4.2.1 View Description

During this subsection we will describe the use case "Save dataset", which symbolizes the process of saving a dataset on database. We chose to do three sequence diagrams: the first shows the interaction of the client application with the backend's services part; the second the interaction from the point of view of the client application; and the last one, the interaction on the backend's services side. Two class diagrams were created, the first representative of the sequence diagram for the client application and the second for the backend's services part. For this use case, it was used the singleton on DataSetService, because the increase of requests to the web service could affect the application performance, since it has to be always creating this class. It was also used a Repository that has the operations crud for a entity.

4.2.2 Sequence Diagram: Client – Backend

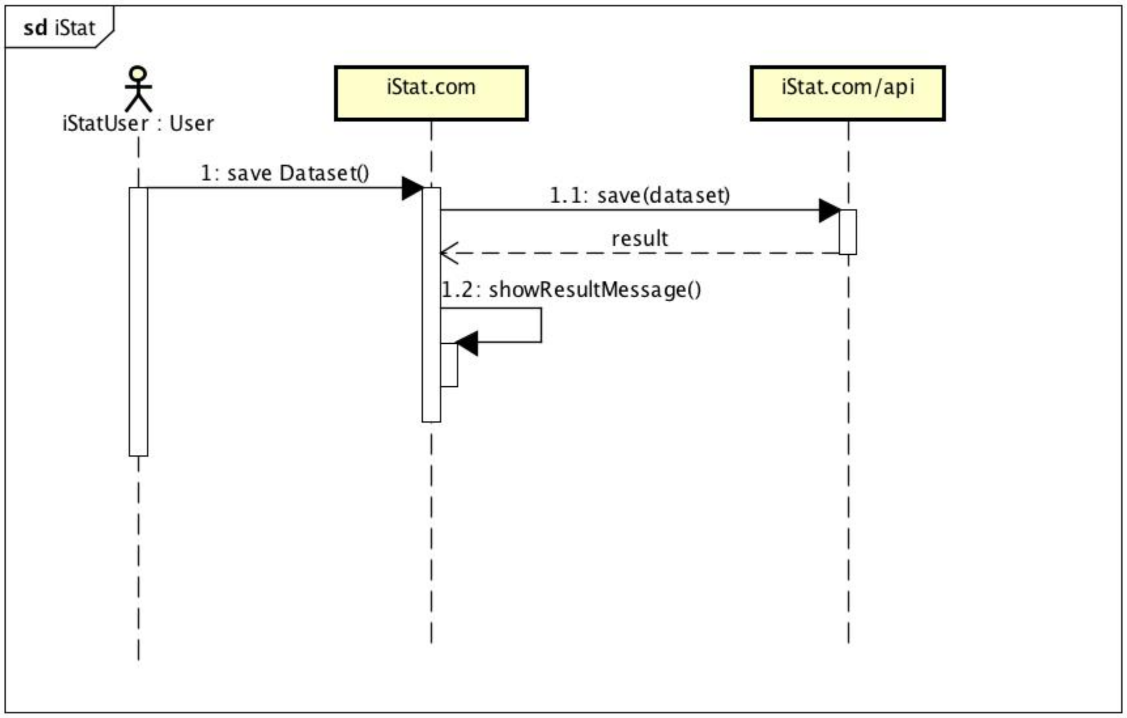

Illustration 12 - Sequence Diagram: Save Dataset (Client - Backend)

4.2.3 Sequence Diagram: Client

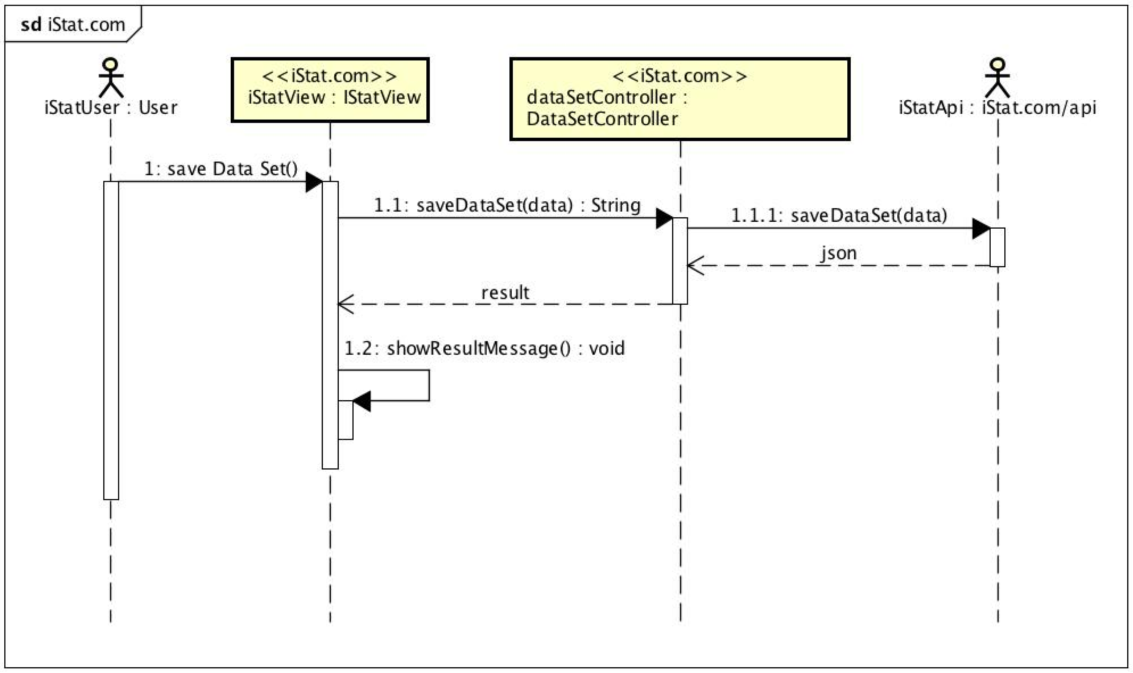
sd iStat.com
iStatUser : User
<<iStat.com>> iStatView : IStatView
<<iStat.com>> dataSetController : DataSetController
iStatApi : iStat.com/api
1: save Data Set()
1.1: saveDataSet(data) : String
1.1.1: saveDataSet(data)
json
result
1.2: showResultMessage() : void

Illustration 13 - Sequence Diagram: Save Dataset (Client)

4.2.4 Sequence Diagram: Backend

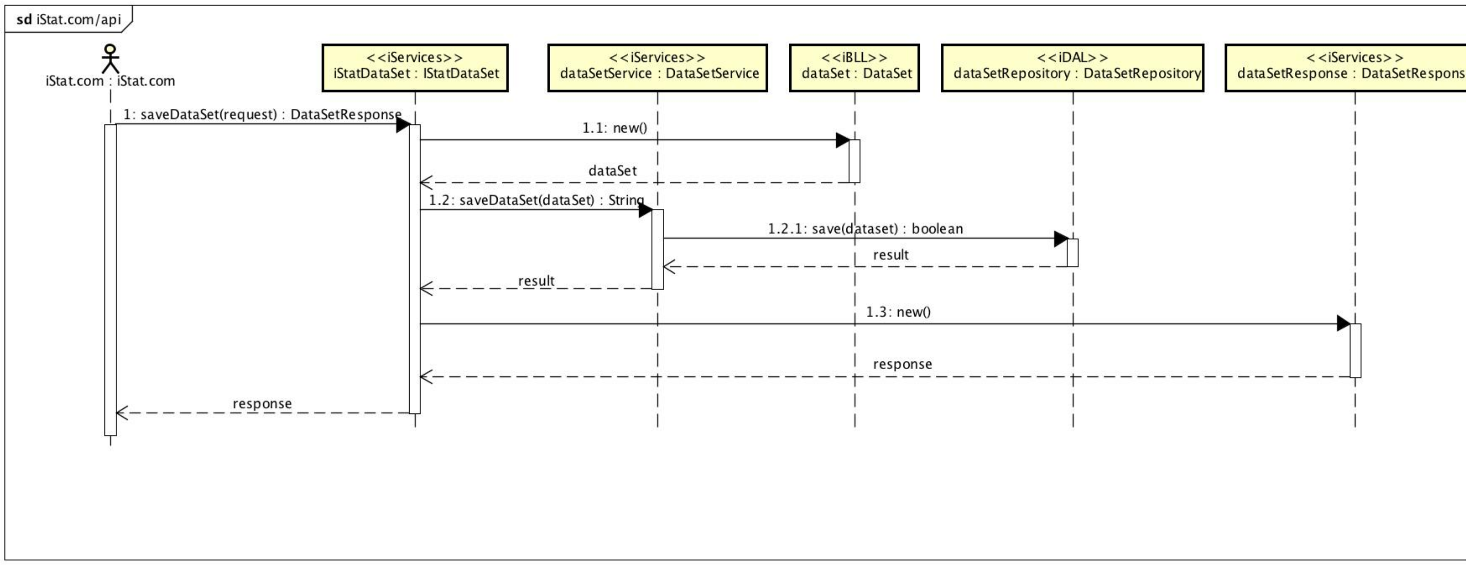
sd iStat.com/api
iStat.com : iStat.com
<<iServices>> iStatDataSet : IStatDataSet
<<iServices>> dataSetService : DataSetService
<<iBLL>> dataSet : DataSet
<<iDAL>> dataSetRepository : DataSetRepository
<<iServices>> dataSetResponse : DataSetRespons
1: saveDataSet(request) : DataSetResponse
1.1: new()
dataSet
1.2: saveDataSet(dataSet) : String
1.2.1: save(dataset) : boolean
result
result
1.3: new()
response
response

Illustration 14 - Sequence Diagram: Save Dataset (Backend)

4.2.5 Class Diagram: Client

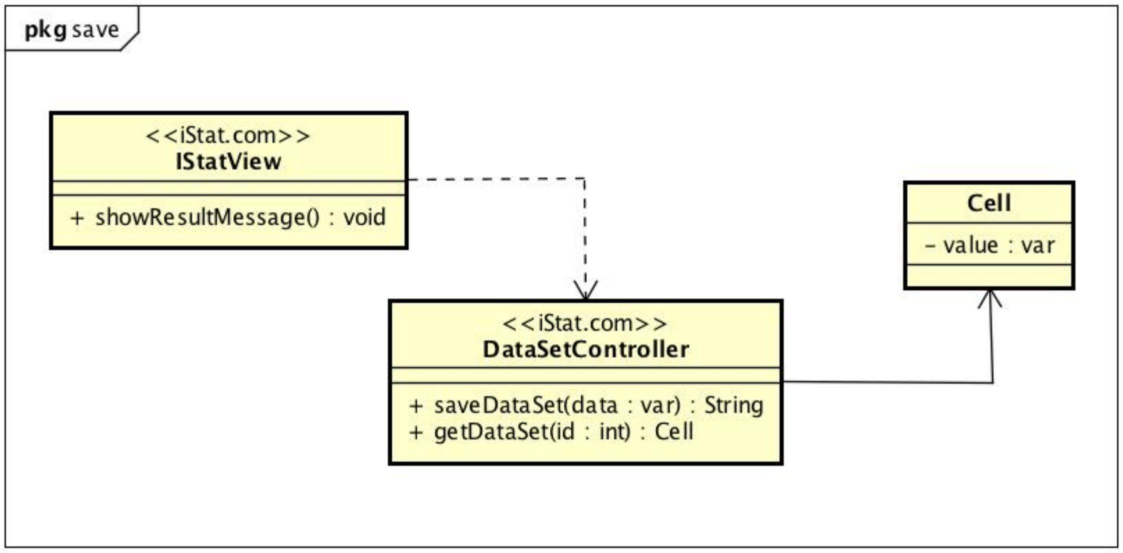
pkg save
<<iStat.com>>
IStatView
+ showResultMessage() : void
Cell
- value : var
<<iStat.com>>
DataSetController
+ saveDataSet(data : var) : String
+ getDataSet(id : int) : Cell

Illustration 15 - Class Diagram: Save Dataset (Client)

4.2.6 Class Diagram: Backend

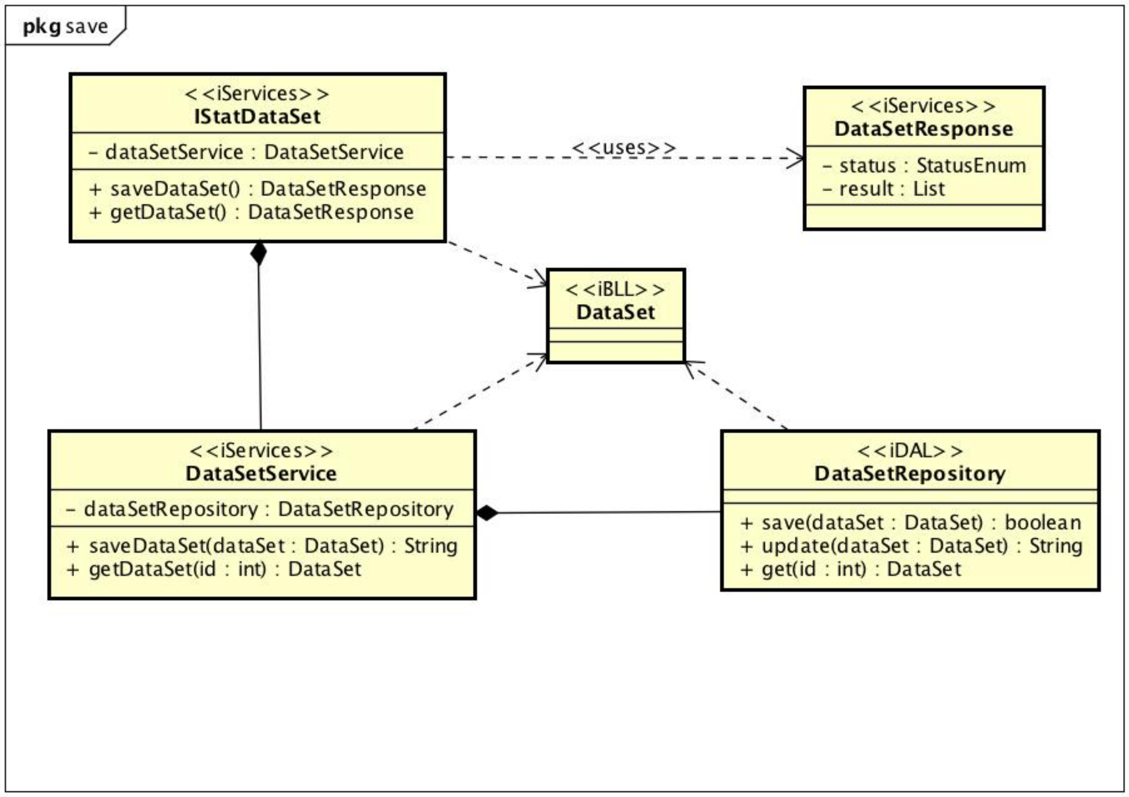
pkg save
<<iServices>>
IStatDataSet
- dataSetService : DataSetService
+ saveDataSet() : DataSetResponse
+ getDataSet() : DataSetResponse
<<uses>>
<<iServices>>
DataSetResponse
- status : StatusEnum
- result : List
<<iBLL>>
DataSet
<<iServices>>
DataSetService
- dataSetRepository : DataSetRepository
+ saveDataSet(dataSet : DataSet) : String
+ getDataSet(id : int) : DataSet
<<iDAL>>
DataSetRepository
+ save(dataSet : DataSet) : boolean
+ update(dataSet : DataSet) : String
+ get(id : int) : DataSet

Illustration 16 - Class Diagram: Save Dataset (Backend)

4.3 Use Case 3: Export Data

4.3.1 View Description

Over this subsection, we will describe the "Export dataset" use case, which symbolizes the process of exporting a dataset into different file types. We considered relevant to describe this use case, since it was used an external library to convert different file types and export them. Then, we chose to do three sequence diagrams: the first shows the interaction of the client application with the backend's services; the second shows the interaction on the client application while the third one represents the interaction in the backend's services part. For each sequence diagram portrayed, a class diagram was created. It should be noted that the process of exporting a data file is very similar to the import process described in 4.4 Use Case 4: Import Data of this document. For this use case, it was used the singleton on IOService, because the increase of requests to the web service could affect the application performance, since it has to be always creating this class.

4.3.2 Sequence Diagram: Client – Backend

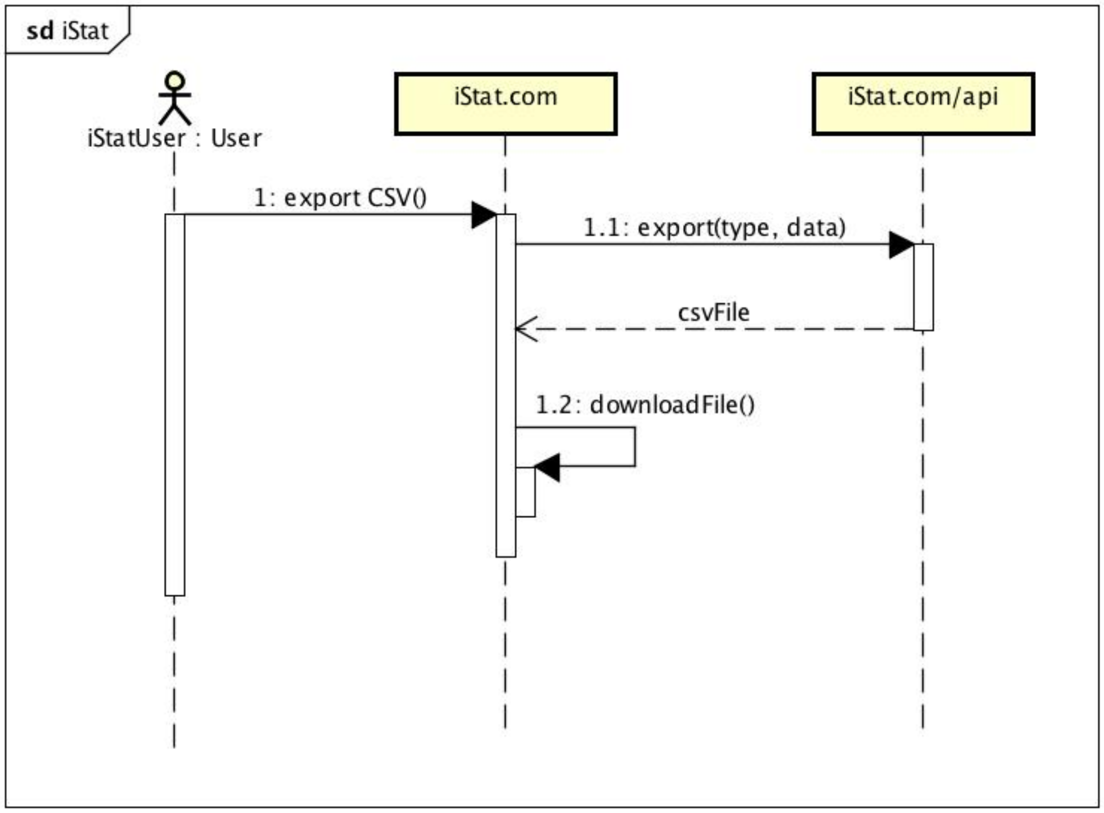

Illustration 17 - Sequence Diagram: Export Data CSV (Client – Backend)

4.3.3 Sequence Diagram: Client

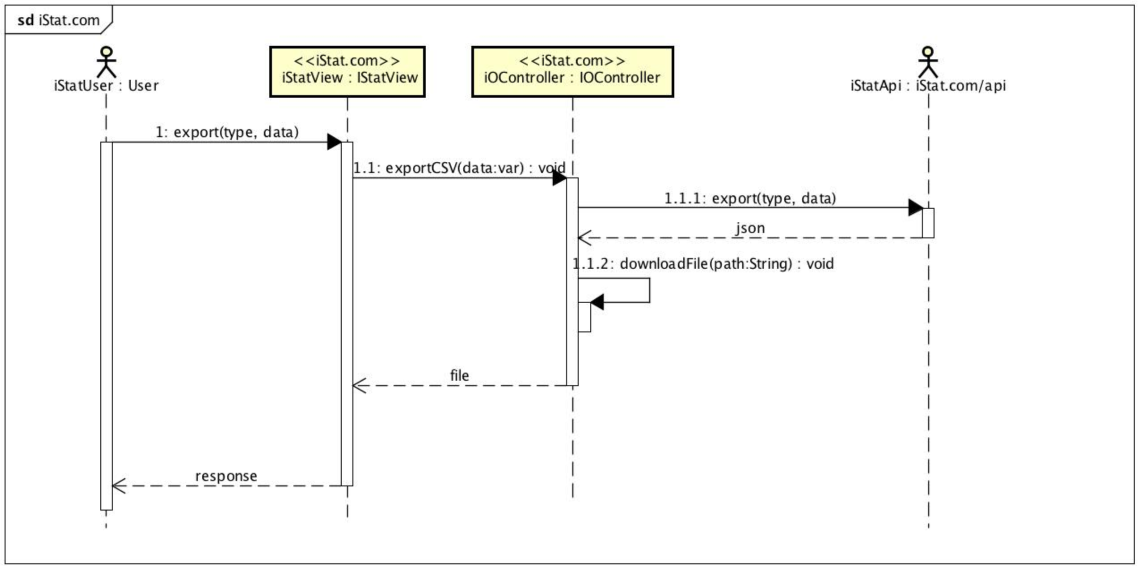

Illustration 18 - Sequence Diagram: Export Data CSV (Client)

4.3.4 Sequence Diagram: Backend

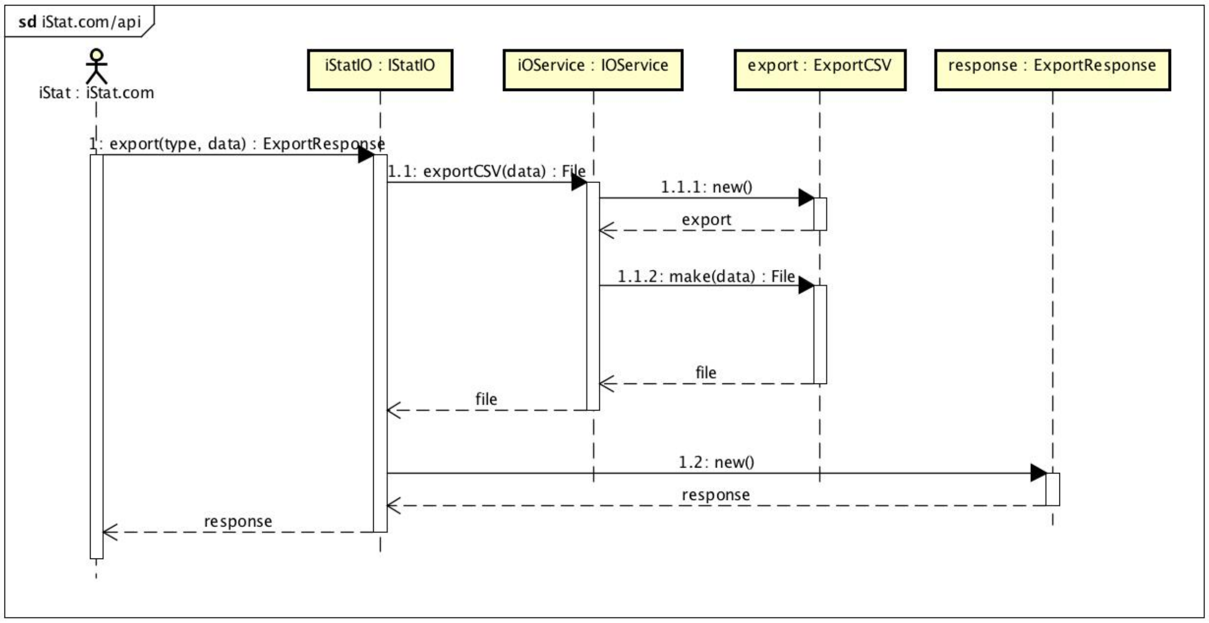

Illustration 19 - Sequence Diagram: Export Data CSV (Backend)

4.3.5 Class Diagram: Client

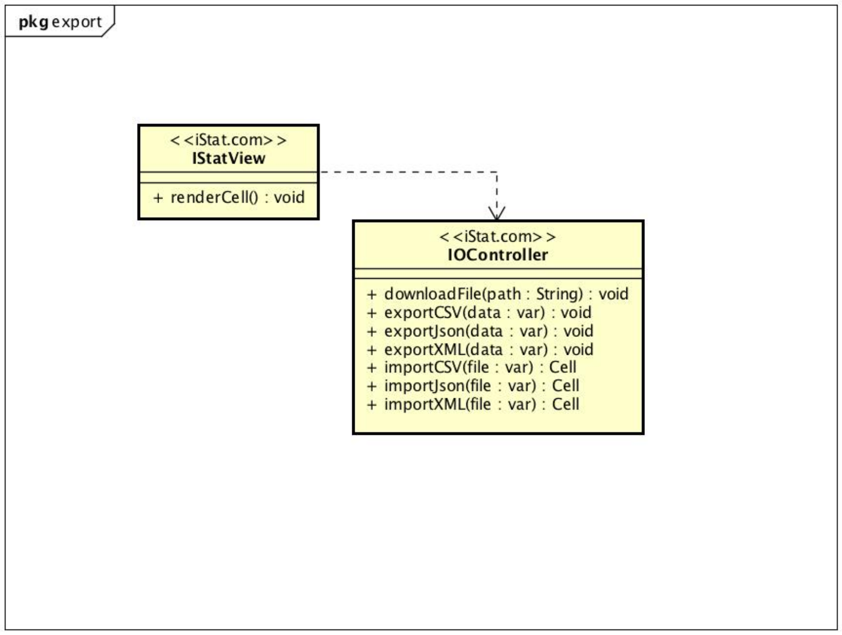
pkg export
<<iStat.com>>
IStatView
+ renderCell() : void
<<iStat.com>>
IOController
+ downloadFile(path : String) : void
+ exportCSV(data : var) : void
+ exportJson(data : var) : void
+ exportXML(data : var) : void
+ importCSV(file : var) : Cell
+ importJson(file : var) : Cell
+ importXML(file : var) : Cell

Illustration 20 - Class Diagram: Export Data (Client)

4.3.6 Class Diagram: Backend

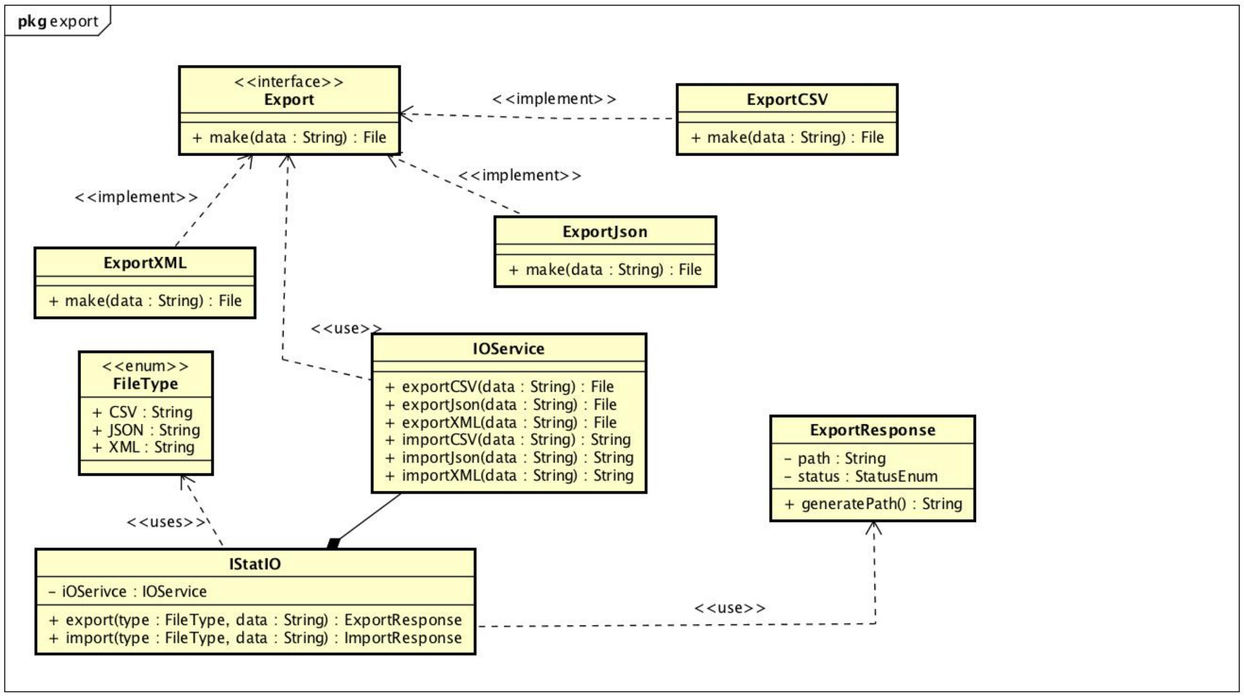
pkg export
<<interface>>
Export
+ make(data : String) : File
<<implement>>
ExportCSV
+ make(data : String) : File
<<implement>>
<<implement>>
ExportXML
+ make(data : String) : File
ExportJson
+ make(data : String) : File
<<use>>
IOService
+ exportCSV(data : String) : File
+ exportJson(data : String) : File
+ exportXML(data : String) : File
+ importCSV(data : String) : String
+ importJson(data : String) : String
+ importXML(data : String) : String
<<enum>>
FileType
+ CSV : String
+ JSON : String
+ XML : String
ExportResponse
- path : String
- status : StatusEnum
+ generatePath() : String
<<uses>>
IStatIO
- iOSerivce : IOService
+ export(type : FileType, data : String) : ExportResponse
+ import(type : FileType, data : String) : ImportResponse
<<use>>

Illustration 21 - Class Diagram: Export Data (Backend)

4.4 Use Case 4: Import Data

4.4.1 View Description

Over this subsection, we will describe the use case "Import dataset", which describes the process of importing a dataset of different types of files. We considered relevant to describe this use case, since it was used an external library to convert different types of files and import them. Then, we chose to make two sequence diagrams the first shows the interaction of the client application with the backend's services; the second shows the interaction on the client application while the third one represents the interaction in the backend's services part. For each sequence diagram portrayed, a class diagram was created. As already mentioned in the previous subsection, the process of importing a data file is very similar to the export process, described in section 4.3 Use Case 3: Export Data of this document. For this use case, it was used the singleton on IOService, because the increase of requests to the web service could affect the application performance, since it has to be always creating this class.

4.4.2 Sequence Diagram: Client – Backend

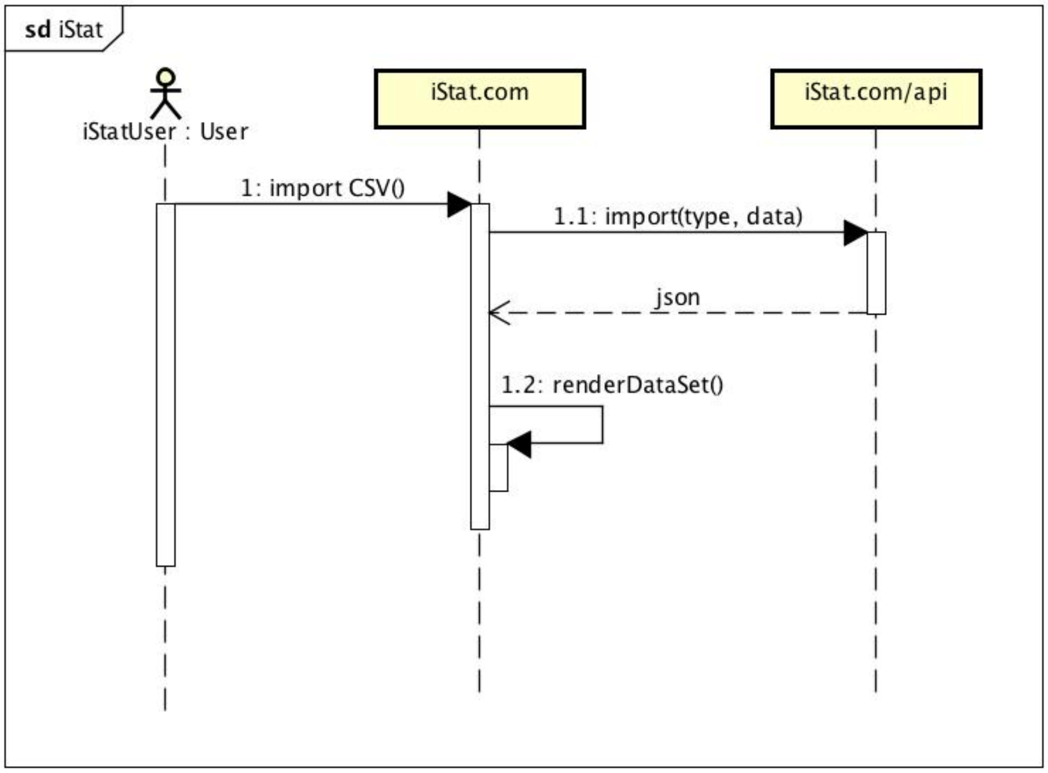

Illustration 22 - Sequence Diagram: Import Data CSV (Client – Backend)

4.4.3 Sequence Diagram: Client

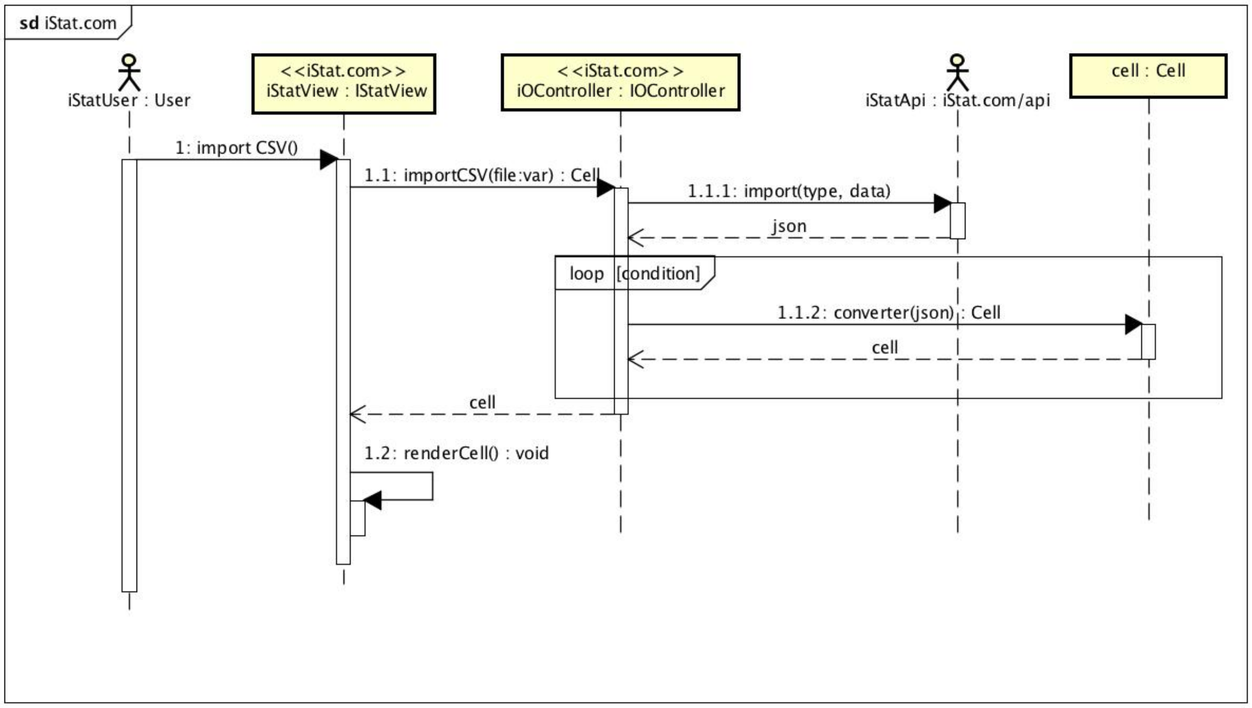

Illustration 23 - Sequence Diagram: Import Data CSV (Client)

4.4.4 Sequence Diagram: Backend

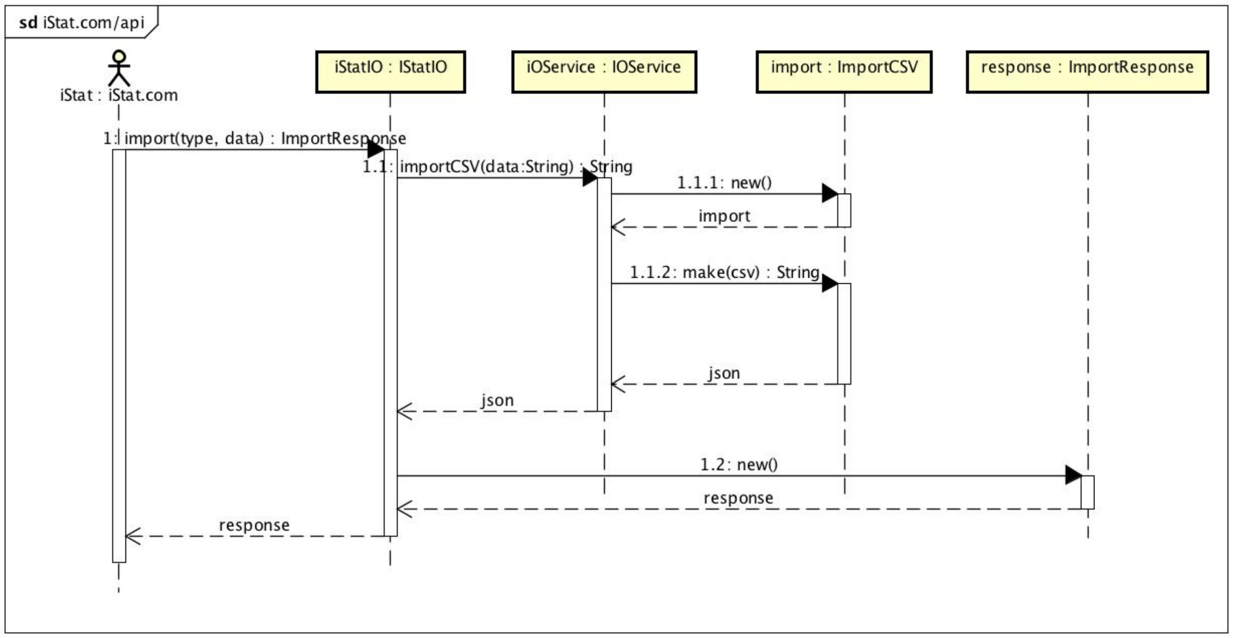

Illustration 24 - Sequence Diagram: Import Data CSV (Backend)

4.4.5 Class Diagram: Client

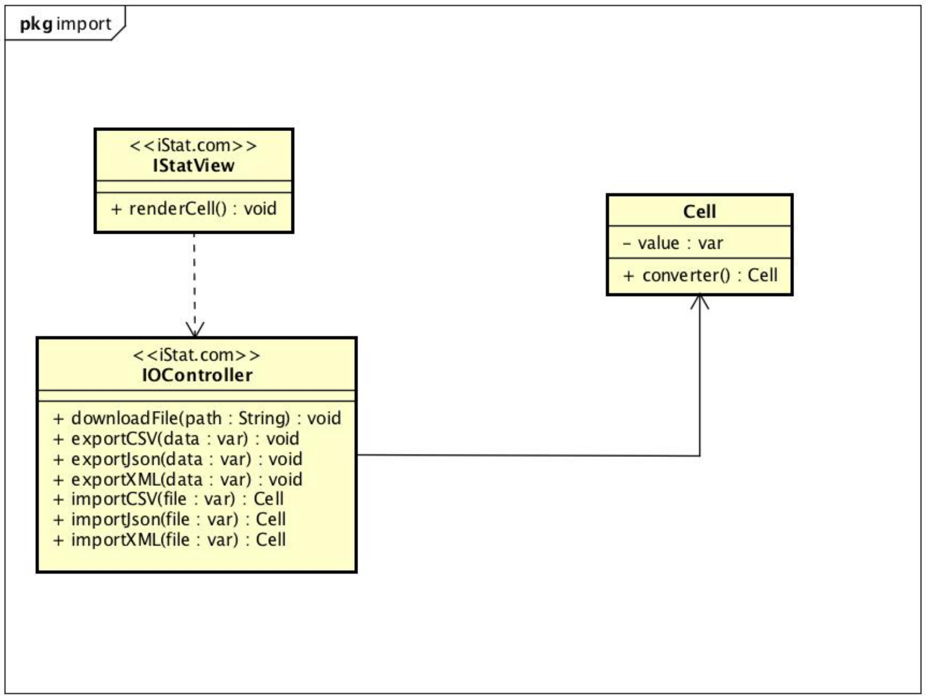
pkg import
<<iStat.com>>
IStatView
+ renderCell() : void
Cell
- value : var
+ converter() : Cell
<<iStat.com>>
IOController
+ downloadFile(path : String) : void
+ exportCSV(data : var) : void
+ exportJson(data : var) : void
+ exportXML(data : var) : void
+ importCSV(file : var) : Cell
+ importJson(file : var) : Cell
+ importXML(file : var) : Cell

Illustration 25 - Class Diagram Import Data (Client)

4.4.6 Class Diagram: Backend

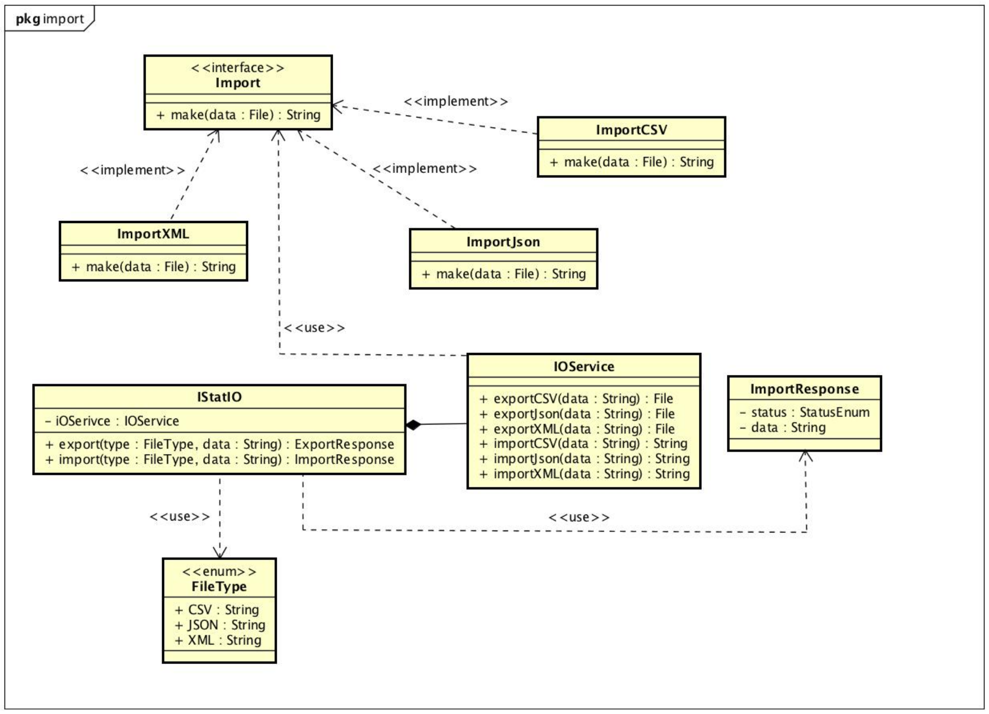
pkg import
<<interface>>
Import
+ make(data : File) : String
<<implement>>
ImportCSV
+ make(data : File) : String
<<implement>>
<<implement>>
ImportXML
+ make(data : File) : String
ImportJson
+ make(data : File) : String
<<use>>
IOService
+ exportCSV(data : String) : File
+ exportJson(data : String) : File
+ exportXML(data : String) : File
+ importCSV(data : String) : String
+ importJson(data : String) : String
+ importXML(data : String) : String
IStatIO
- iOSerivce : IOService
+ export(type : FileType, data : String) : ExportResponse
+ import(type : FileType, data : String) : ImportResponse
ImportResponse
- status : StatusEnum
- data : String
<<use>>
<<use>>
<<enum>>
FileType
+ CSV : String
+ JSON : String
+ XML : String

Illustration 26 - Class Diagram Import Data (Backend)

5 Referenced Materials

CONTENTS OF THIS SECTION: This section provides citations for each reference document. Provide enough information so that a reader of the SAD can be reasonably expected to locate the document.

Barbacci 2003	Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs)*, Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html>.
Bass 2003	Bass, Clements, Kazman, *Software Architecture in Practice,* second edition, Addison Wesley Longman, 2003.
Clements 2001	Clements, Kazman, Klein, *Evaluating Software Architectures: Methods and Case Studies,* Addison Wesley Longman, 2001.
Clements 2002	Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2002.
IEEE 1471	ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000.

6 Directory

6.1 Index

CONTENTS OF THIS SECTION: This section provides an index of all element names, relation names, and property names. For each entry, the following are identified:

- the location in the SAD where it was defined
- each place it was used

Ideally, each entry will be a hyperlink so a reader can instantly navigate to the indicated location.

6.2 Glossary

CONTENTS OF THIS SECTION: This section provides a list of definitions of special terms and acronyms used in the SAD. If terms are used in the SAD that are also used in a parent SAD and the definition is different, this section explains why.

Term	Definition
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
view packet	The smallest package of architectural documentation that could usefully be given to a stakeholder. The documentation of a view is composed of one or more view packets.

viewpoint	A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.

6.3 Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
ATAM	Architecture Tradeoff Analysis Method
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common object request broker architecture
COTS	Commercial-Off-The-Shelf
EPIC	Evolutionary Process for Integrating COTS-Based Systems
IEEE	Institute of Electrical and Electronics Engineers
KPA	Key Process Area
OO	Object Oriented
ORB	Object Request Broker
OS	Operating System
QAW	Quality Attribute Workshop
RUP	Rational Unified Process
SAD	Software Architecture Document
SDE	Software Development Environment
SEE	Software Engineering Environment
SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code
SW-CMM	Capability Maturity Model for Software

CMMI-SW	Capability Maturity Model Integrated - includes Software Engineering
UML	Unified Modeling Language

7 Sample Figures & Tables

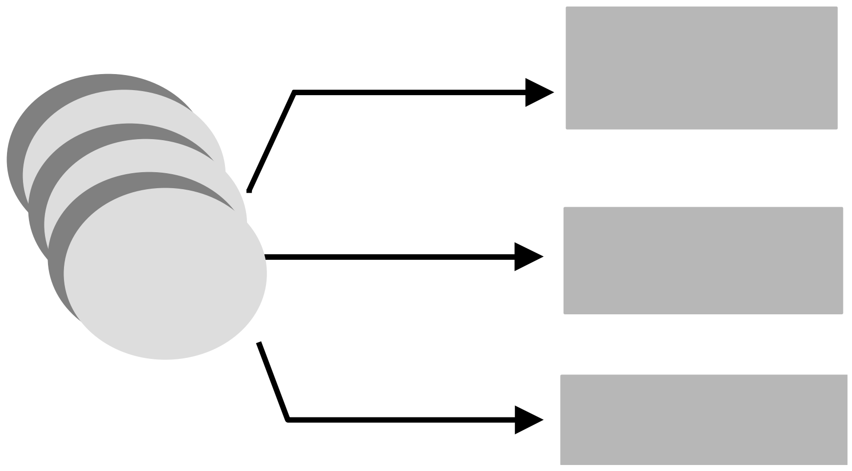

Figure 1: Sample Figure

Table 2: Sample Table

Table Heading	**Table Heading**	**Table Heading**	**Table Heading**
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body

Appendix A Appendices

CONTENTS OF THIS SECTION: Appendices may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data, API specification). As applicable, each appendix is referenced in the main body of the document where the data would normally have been provided. Appendices may be bound as separate documents for ease in handling. If your SAD has no appendices, delete this page.

A.1 Heading 2 - Appendix

A.2 Heading 2 - Appendix