Software Architecture Document

Insurance Premium Prediction

Version 1.0

25/08/2021

Version Control

Version	Description	Responsible Party	Date
1.0	Initial version	Motheeshkumar Jay	25-08-2021

1 Introduction

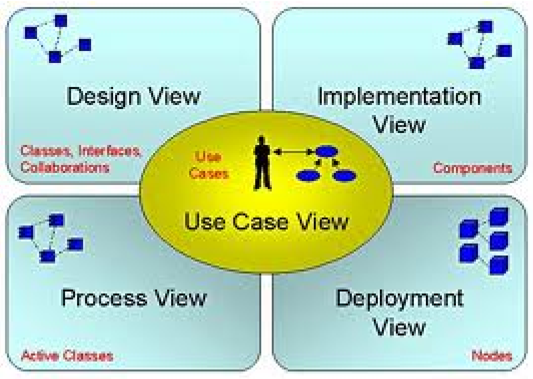
This document provides a high-level overview and explains the architecture of the insurance premium prediction.

The document defines goals of the architecture, the use cases supported by the system, architectural styles and components that have been selected. The document provides a rationale for the architecture and design decisions made from the conceptual idea to its implementation.

1.1 Purpose

The Software Architecture Document (SAD) provides a comprehensive architectural overview of the Insurance premium prediction (IPP). It presents several different architectural views to depict the different aspects of the system.

To depict the software as accurately as possible, the structure of this document is based on Philippe Kruchten’s “4+1” model view of architecture [Kruchten].



1.2 Scope

The scope of this SAD is to explain the architecture of the Insurance premium prediction (IPP).

This document describes the various aspects of the IPP system design that are architecturally significant. These elements and behaviours are fundamental for guiding the construction of the IPP and for understanding this project.

1.3 Definitions, Acronyms and Abbreviations

- WWW – World Wide WEB
- HTTP – HyperText Transfer Protocol
- SAD – Software Architecture Document
- IPP– Insurance premium prediction
- User – Any user who has access to a URL.

2 Architectural Representation

This document details the architecture using the views defined in the "4+1" model [Kruchten]. The views used to document the IPP system are:

Use Case view

Audience: all the stakeholders of the system, including the end-users.

Area: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system. Describing the actors and use cases for the system, this view presents the needs of the user and is elaborated further at the design level to describe discrete flows and constraints in more detail. This domain vocabulary is independent of any processing model or representational syntax (i.e. XML).

Related Artefacts: Use-Case Model, Use-Case documents

Logical view

Audience: Designers.

Area: Functional Requirements: describes the design's object model. Also describes the most important use-case realizations and business requirements of the system.

Related Artefacts: Design model

Process view

Audience: Data specialists, Database administrators

Area: Persistence: describes the architecturally significant persistent elements in the data model as well as how data flows through the system.

Related Artefacts: Data model.

Deployment view

Audience: Deployment managers.

Area: Topology: describes the mapping of the software onto the hardware and shows the system's distributed aspects. Describes potential deployment structures, by including known and anticipated deployment scenarios in the architecture we allow the implementers to make certain assumptions on network performance, system interaction and so forth.

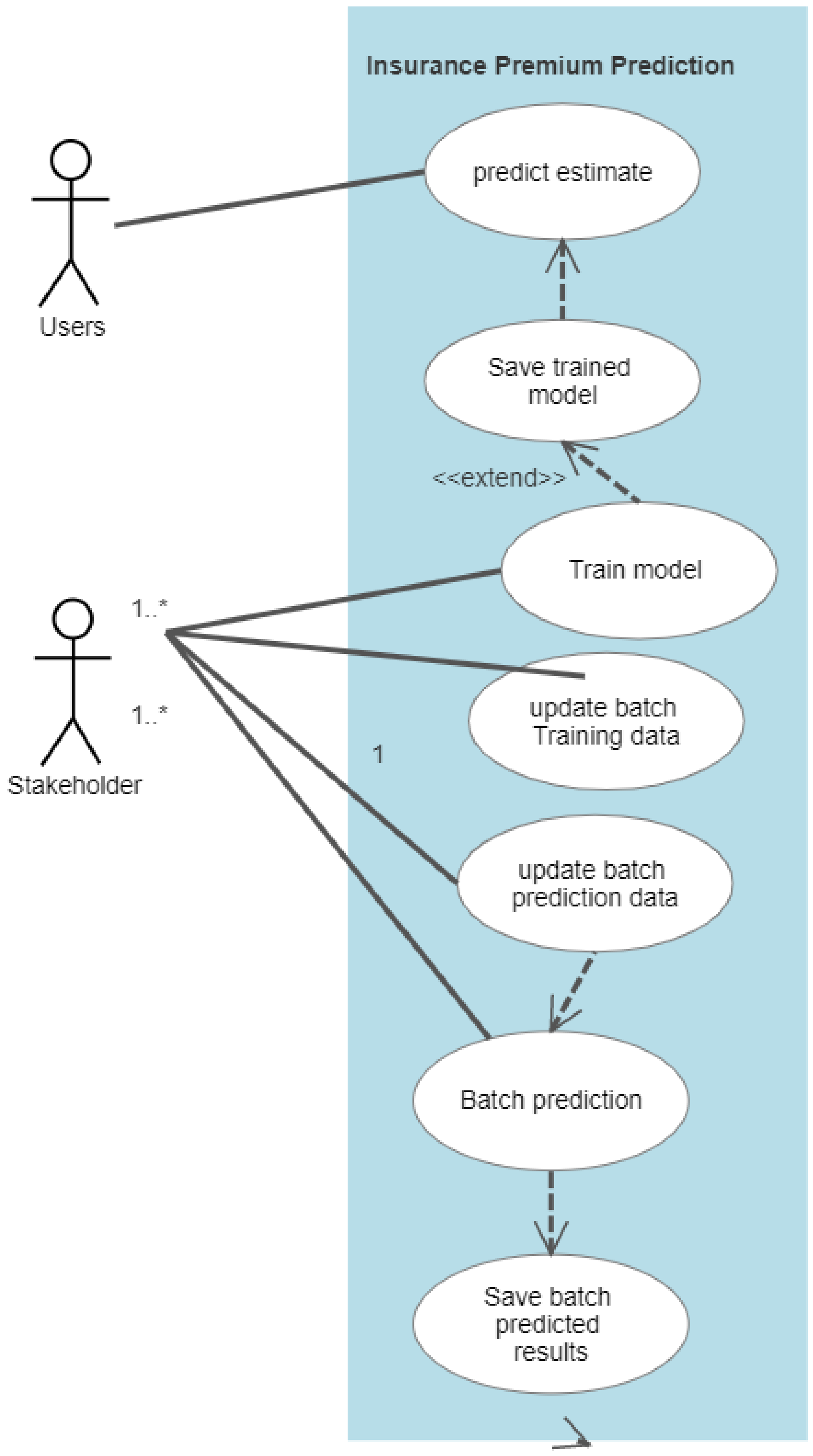
Related Artefacts: Deployment model.

3 Use-Case View

A description of the use-case view of the software architecture. The Use Case View is an important input to the selection of the set of scenarios and/or use cases that are the focus of an iteration. It describes the set of scenarios and/or use cases that represent some significant, central functionality. It also describes the set of scenarios and/or use cases that have a substantial architectural coverage (that exercise many architectural elements) or that stress or illustrate a specific, delicate point of the architecture.

The IPP use cases are –

- Home page
- Insurance Premium prediction page
- Trained model details
- API to train model after updating training dataset
- API to perform batch prediction after updating batch prediction dataset
- View batch prediction result page.


Insurance Premium Prediction
Users
predict estimate
Save trained model
<<extend>>
Train model
1..*
1..*
Stakeholder
update batch Training data
1
update batch prediction data
Batch prediction
Save batch predicted results

3.1 Home

First thing anyone will see is the insurance premium prediction form, which will ask the user to enter details related to health and family details to predict the insurance premium. By clicking on the predict button, the user will be able to get the estimation..

3.2 Batch Prediction result screen

By clicking on the batch prediction result, the client will be able to get the batch prediction result.

3.3 Trained model details screen

By clicking on the model details page, the client will be able to see the trained model details like accuracy, number of groups from training dataset, parameters, and model details.

4 Logical View

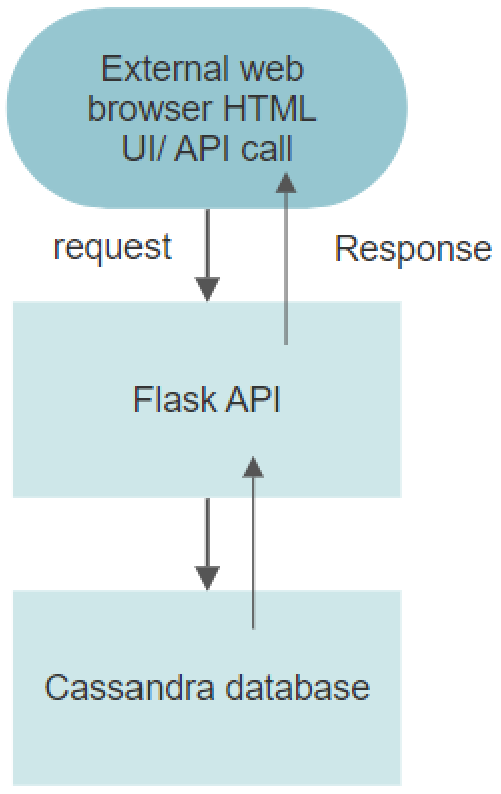
A description of the logical view of the architecture. Describes the most important classes, their organization in service packages and subsystems, and the organization of these subsystems into layers. Also describes the most important use-case realizations, for example, the dynamic aspects of the architecture. Class diagrams may be included to illustrate the relationships between architecturally significant classes, subsystems, packages and layers.

The logical view of the IPP consists of the 3 main packages: User Interface, Flask API and Database Services

The user interface allows users to enter details related to their health and family.

Flask API is responsible for handling the request, processing it and responding back to users.

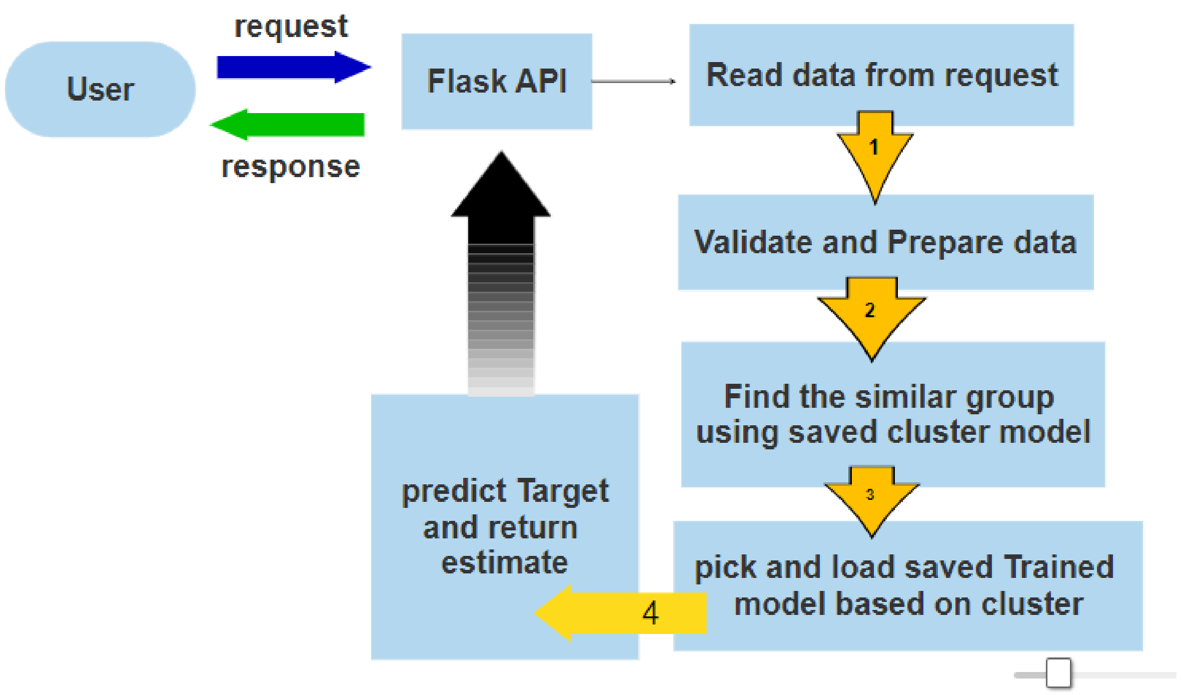
The database package maintains the training data for IPP and is responsible for retrieval, inserting, updating and deletion of data.



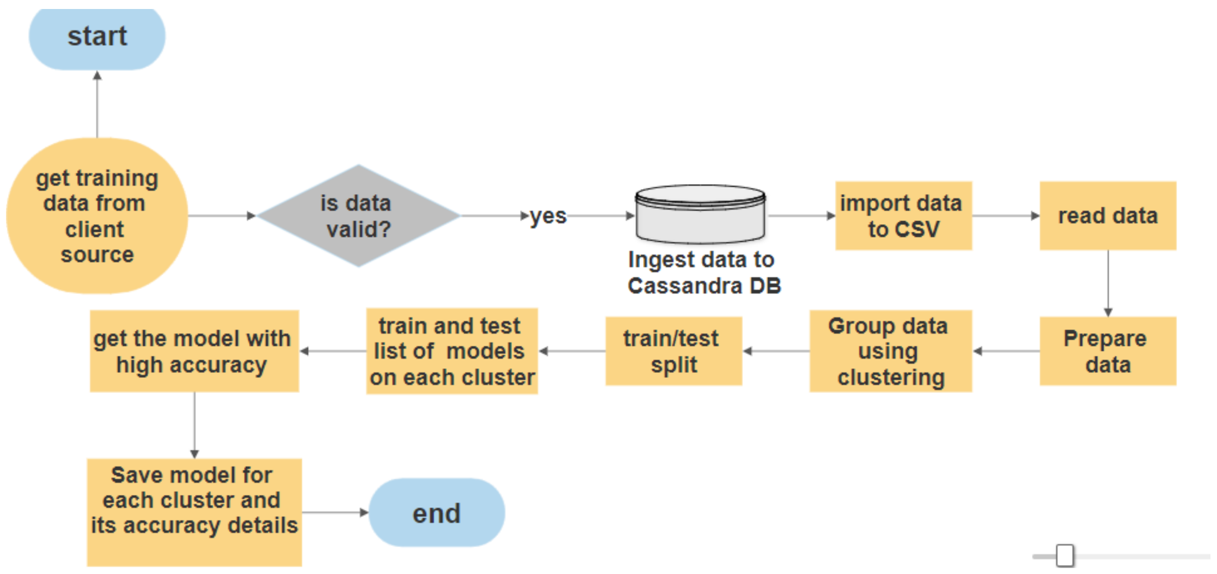
5 Process View

A description of the process-view of the architecture. Describes the tasks (processes and threads) involved in the system's execution, their interactions and configurations. Also describes the allocation of objects and classes to tasks.

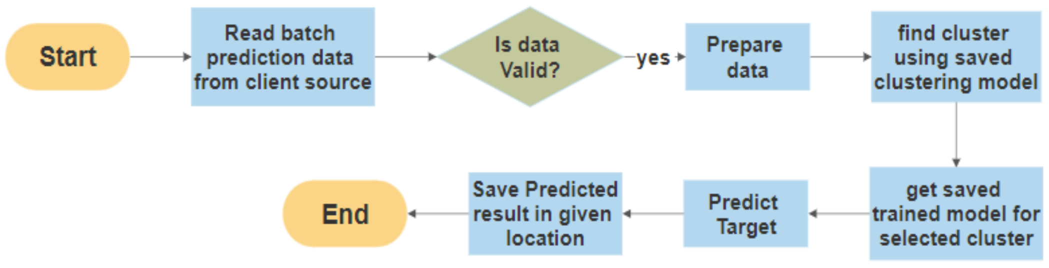
IPP Web application workflow:


User
request
response
Flask API
Read data from request
1
Validate and Prepare data
2
Find the similar group using saved cluster model
3
pick and load saved Trained model based on cluster
4
predict Target and return estimate

Model Training workflow:

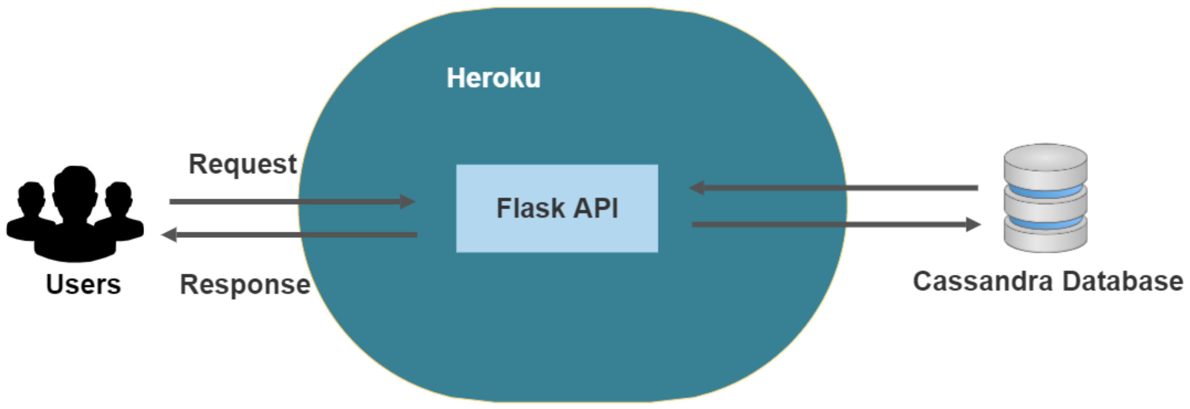

start
get training data from client source
is data valid?
yes
Ingest data to Cassandra DB
import data to CSV
read data
Prepare data
Group data using clustering
train/test split
train and test list of models on each cluster
get the model with high accuracy
Save model for each cluster and its accuracy details
end

Batch Prediction workflow:


Start
Read batch prediction data from client source
Is data Valid?
yes
Prepare data
find cluster using saved clustering model
get saved trained model for selected cluster
Predict Target
Save Predicted result in given location
End

6 Deployment View

The web application is hosted on Heroku instance.


Heroku
Request
Flask API
Users
Response
Cassandra Database