SageTomo:

System for the reconstruction of Electrical Impedance Tomographies in real-time, based on Machine Learning techniques

Martín Aller Domínguez
(martin.aller.dominguez@gmail.com)

Table of contents

1.   Introduction

In some industrial and medical environments, it is necessary to obtain information from the inside of bodies using non-invasive methods. Techniques for obtaining information in two or three dimensions are known as tomographies. Computed Tomography (CT) is the most well-known tomographic technique. However, it implies the use of X-rays, so utilizing this technique is not feasible in most industrial environments. In this work, it is proposed as an alternative to the CT the use of the Electrical Impedance Tomography (EIT).

Using the EIT, the conductivity values from the inside of a body are measured. In order to do so, electrodes are placed around the body, and some of them supply an electrical current. The electrical current passes through the body and is captured by the remaining electrodes. With the measured voltages, it is necessary to estimate the body's conductivity values, which are related to some physical internal feature that we wish to study (e.g., moisture distribution). This problem, known as the **inverse problem**, is highly non-linear and ill-posed.

Traditionally, two kinds of algorithms were used to solve the inverse problem. The first one consists of algorithms that assume some kind of linearity in the response of the body to the supplied voltages (losing precision to gain efficiency). The second type includes iterative algorithms, so they are more precise than the previous ones, but they consume an extreme amount of computational and time resources. None of the prior algorithms would be suitable for specific industry problems that require getting highly accurate results in real-time, so some authors have proposed the use of Machine Learning techniques to solve the inverse problem.

2.   SageTomo

The software SageTomo has been developed to solve the inverse problem and reconstruct EIT images. Its main functionalities will be described in this section.

SageTomo is able to reconstruct EIT images using three types of Machine Learning models: Neural Networks, Random Forests and Support Vector Machines.

Bodies used by the software are bidimensional and circular. They are called meshes and are divided into 844 triangular cells, each of them with an electrical conductivity value, measured in S/m. Figure 2.1 shows a mesh image as an example.
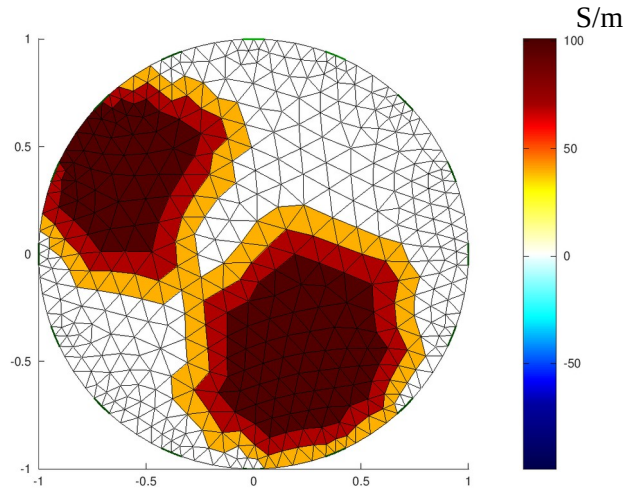
Figure 2.1. Image of a mesh with two artifacts.

The generation of the electrical current has been simulated by the software EIDORS, using a setting of 16 electrodes and an adjacent stimulation pattern. This setting produces 208 voltages measures for each mesh. Thus, the Machine Learning problem that must be solved consists of predicting the 844 conductivities (output) from the 208 voltages (input). Each predicted conductivity value is associated with one cell in the mesh. Figure 2.2 shows a visual representation of the process.
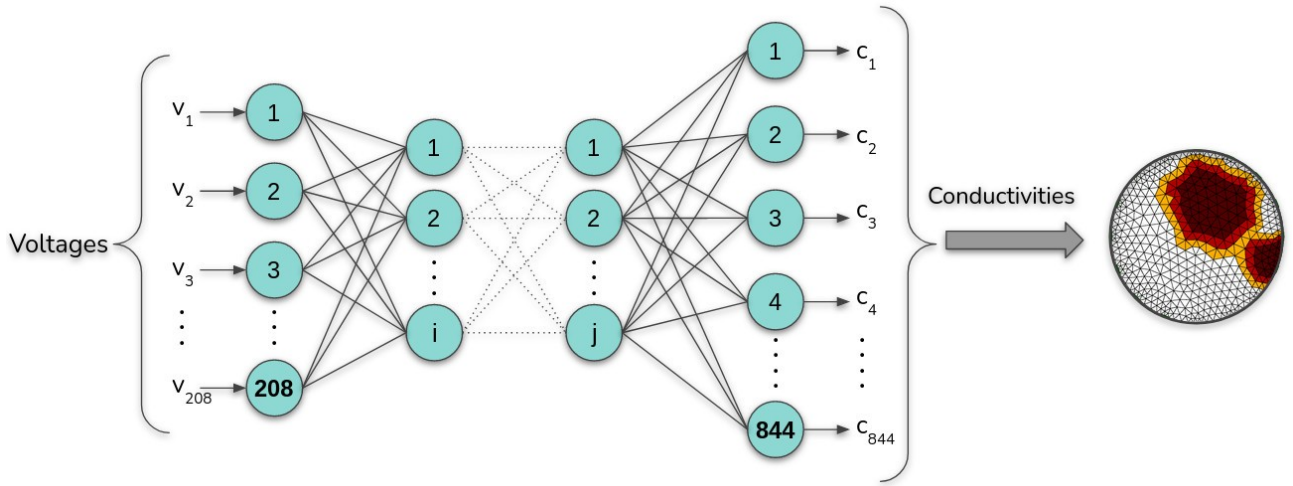

Figure 2.2. Visualization of the Machine Learning problem to solve.

The developed software has the following functionalities:

- **Reconstruction of EIT images**. This reconstruction consists of predicting the conductivity values using the voltages measures as the input of the Machine Learning models.

- **Training and storage of Machine Learning models**. Models trained by the users can be stored and compared with the existing models in the system.

- **Generation and storing of datasets.** Each dataset consists of a set of meshes. Each mesh has 208 voltage values (input) and 844 conductivity values (output). Datasets are stored in a PostgreSQL database.

- **Task management**. Training of models and generation of datasets are computationally expensive tasks. The software allows users to send these kinds of tasks to a task queue.

- **Users management**. Users have a personal account and they can train and generate models and datasets, which can be both private or public to the rest of the users.

3. Software architecture and technologies

SageTomo has a Service Oriented Architecture, so the application contains two modules: the frontend -developed using the ReactJS framework- and the backend -developed using the Django REST framework-.

The Service-Oriented Architecture and the fact that the frontend and the backend are completely decoupled are interesting features for this kind of software because it implies that the backend services can be used by other industrial or researching software. Furthermore, using a task queue for the Machine Learning models training and datasets generation ensures the scalability of the offered services.

3.1. Technologies

In order to develop SageTomo, a great variety of very different technologies were used, as can be observed in figure 3.1. Despite the heterogeneity of the frameworks and programming languages utilized, the integration of all these technologies was completed successfully.

Figure 3.1. Technologies used to develop SageTomo.

The most relevant technologies employed are the next ones:

- **Django REST**. Django REST is a Python framework to develop REST APIs. The backend was developed using this framework. In spite of the fact that Java is the language used to develop backends generally, for this project, it was considered that Python was the ideal language for one reason: Python is one of the most suitable languages for Data Science and Artificial Intelligence projects. Using Python to develop the backend allows embodying in the software other technologies like Keras or Scikit-learn effortlessly.

- **Keras and Scikit-learn**. These libraries were used to implement all the functionalities related to Machine Learning models. On the one hand, Keras was utilized to incorporate the functionality of training Neural Networks. On the other hand, Scikit-learn was used for Random Forests and SVMs.

- **EIDORS**. The EIDORS library was used to simulate the Electrical Impedance Tomographies. Although EIDORS is a MATLAB library, most of its functions can be run from Octave. In this project, it was used the Python module *oct2py*, which allows running Octave functions from Python.

- **Celery**. Celery is a task queue implemented in Python. The training of models and generation of datasets are computationally expensive tasks, so it is convenient to execute them asynchronously. Celery was used to manage these tasks efficiently.

- **C++**. In order to implement the functionality of generating new datasets, it was used the algorithm developed by the research group COGRADE (University of Santiago de Compostela). They implemented this algorithm using C++. Their code was embodied in SageTomo as an external module.

- **React**. The frontend was developed utilizing React, a JavaScript framework. React is based on the use of *components*, which are JavaScript classes.

- **Bootstrap**. The graphical interface was developed using this popular framework.

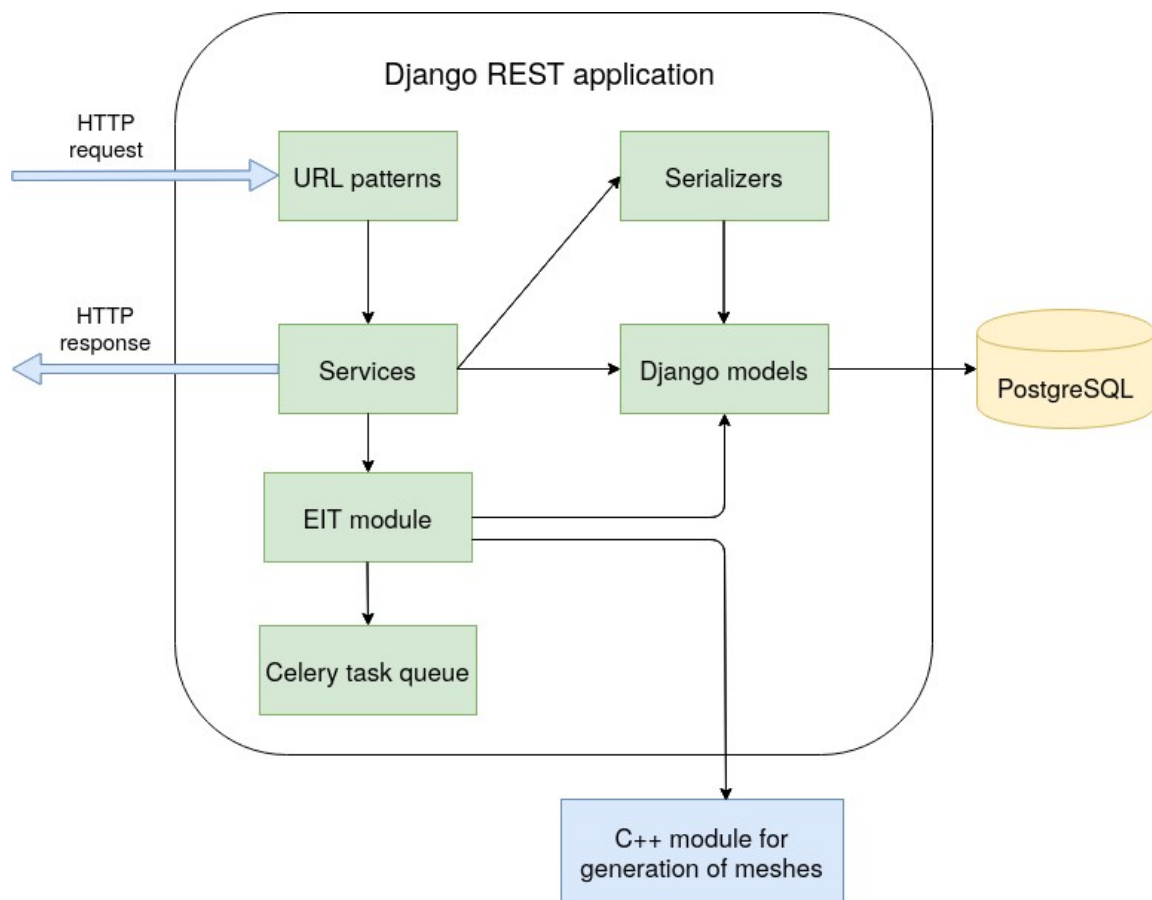## 3.2. Backend

3.2.1. Architecture diagram

Figure 3.2. Backend architecture diagram.

Figure 3.2 shows the backend architecture diagram. The main backend components are the following ones:

- **URL patterns**. When the backend receives an HTTP request, the URL pattern module analyzes the defined patterns and, when it finds the first pattern that matched the requested

6

URL, it calls the services related to that URL. Every defined pattern is an API REST endpoint.

- **Serializers**. Resources returned by the services are sent in the JSON format, so it is necessary to have classes that serialize the Django models, converting them to the JSON format.

- **Services**. In this module, the API REST services are defined. Every service returns an HTTP response with the requested resources or an error code when the request cannot be processed properly.

- **Django Models**. Classes that will be mapped to the database are defined in this module. In SageTomo, there exist classes for the datasets, the meshes, the Machine Learning models, and the users, among others.

- **EIT module**. This is a standard Python module that contains functions that implement the functionalities of the tomographic system: image reconstruction, model training, datasets generation, etc.

- **Celery tasks queue**. Some of the functions of the EIT module generate and send jobs to a Celery task queue.

- **C++ module for meshes generation**. This external module is called by the system when it is necessary to generate a new dataset.

- **PostgreSQL**. Machine Learning models are stored in a DB, and PostgreSQL has been used as DBMS.

3.2.2. Class diagram for the DB access module
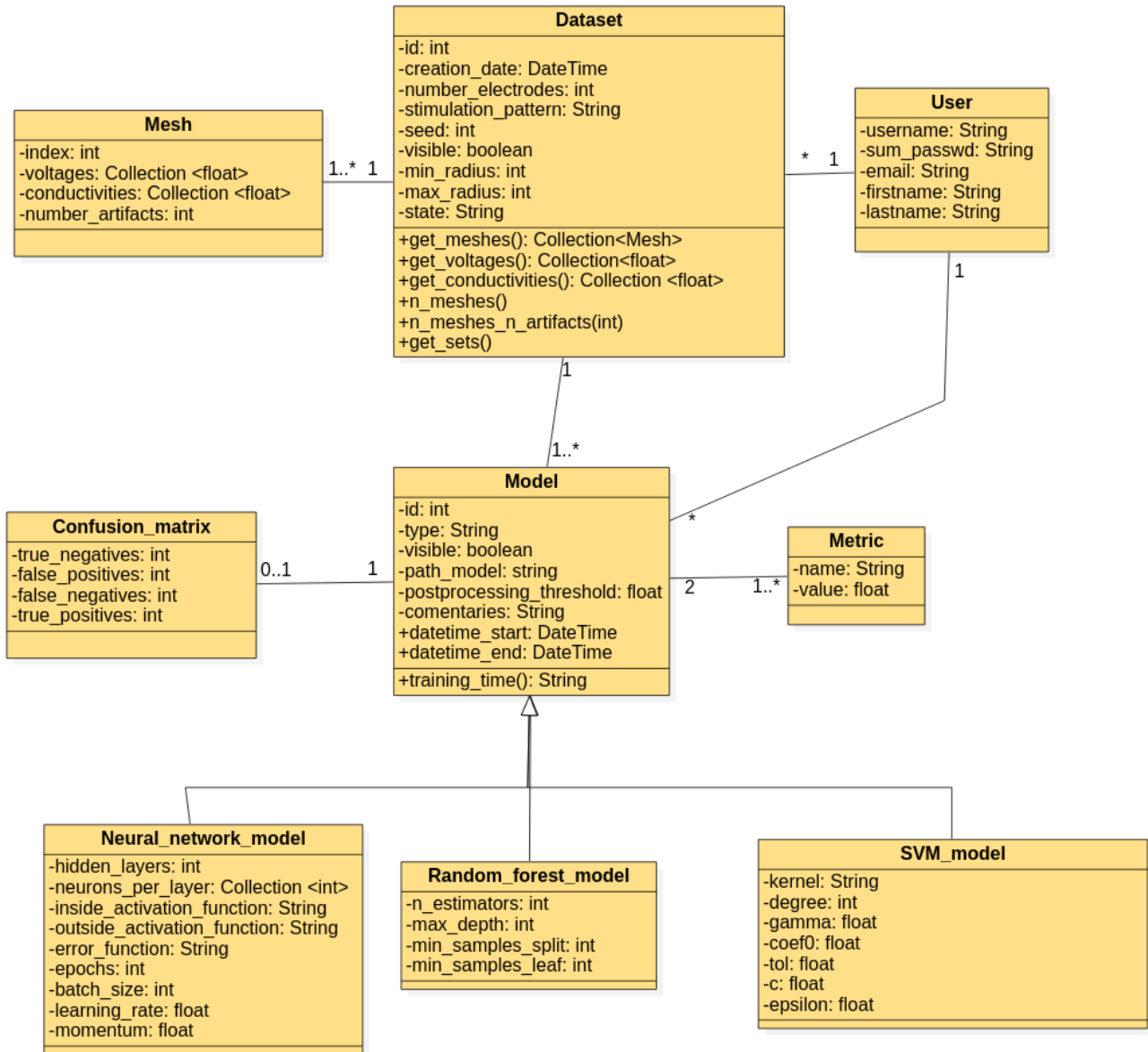
Figure 3.3. Class diagram for the DB access module

Classes presented in the diagram from figure 3.3 are mapped by Django into tables in the BD. All these classes are defined in the file *models.py,* following the Django conventions. It can be observed that the datasets are composed of a set of meshes. Each dataset is created by a user, and each user can generate an unlimited number of datasets. On the other hand, one dataset can be used to train different Machine Learning models. Each Machine Learning model has several metrics associated with it, and these are used to measure the model performance. Taken into account that Machine Learning models can be of three different types (neural networks, random forest and support vector machines), there is an inheritance relationship between the class *Model* and the classes

Neural_network_model, *Random_forest_model* and *Support_vector_machine_model*. These three classes inherit all the attributes and methods from the class *Model*.

3.2.3. Class diagram for the EIT module
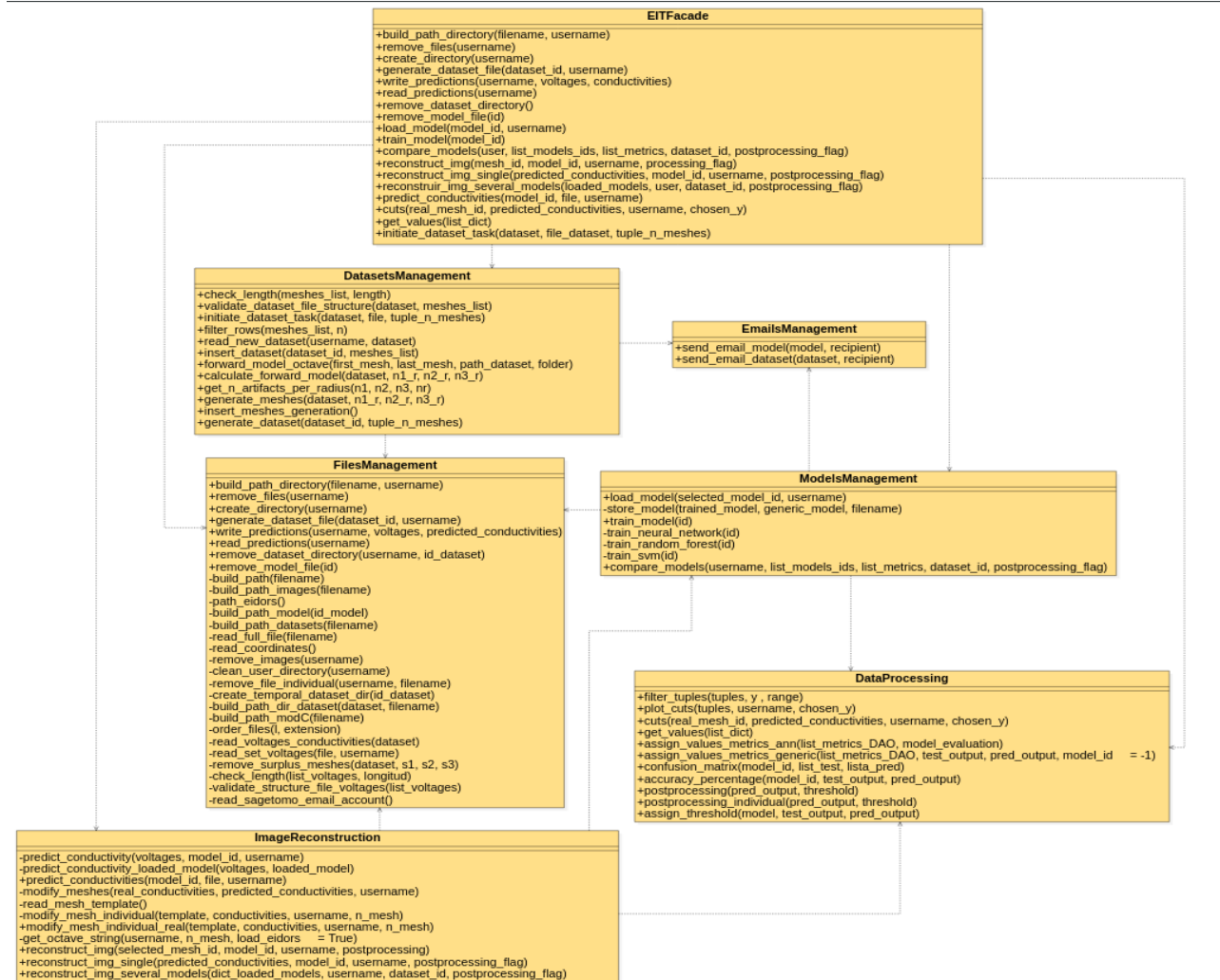
Figure 3.4. Class diagram for the EIT module.

Figure 3.4 shows the class diagram for the EIT module. All the classes from the EIT module are static (all their methods are static). These classes implement the main functionalities of the system. The classes from this module are the next ones:

- *ModelsManagement*. It includes the methods required to train, load and compare different kinds of models from the system.

- **DatasetsManagement**. It includes the methods required to generate or upload datasets to the system.

- **ImageReconstruction**. It includes the methods required to make predictions about the conductivity values of meshes as well as the reconstructions of the images associated with such meshes.

- **DataProcessing**. It includes the methods required to calculate performance metrics and apply postprocessing techniques to the predictions from the Machine Learning models.

- **FilesManagement**. Some of the classes in the EIT module need to generate or use files. This class includes the methods required to manage the different kinds of files used by the rest of the classes.

- **EmailsManagement**. It includes the methods required to send emails to those users who have sent tasks to the task queue. These users receive an email every time one of their tasks has finished.

- **EITFacade**. This class is used to implement the *Facade pattern*, avoiding unnecessary dependencies between the EIT module and the rest of the modules.

3.2.4. Sequence diagram for the reconstruction of an EIT image
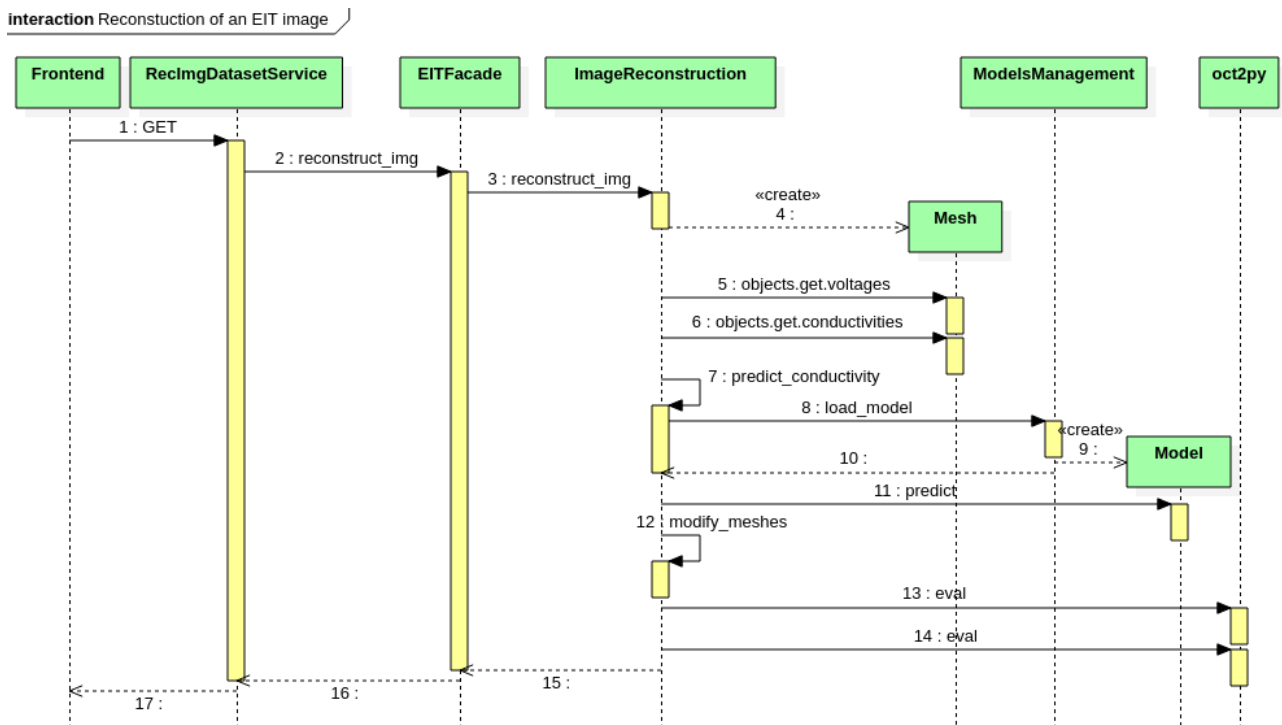
Figure 3.5. Sequence diagram for the reconstruction of an EIT image

Figure 3.5 shows the sequence diagram for the reconstruction of the image of a mesh from a dataset. The steps included in the diagram are the next ones:

1. An HTTP request is sent from the frontend to the image reconstruction service, using GET as the request verb. In the request parameters, the dataset ID and the mesh index are included in order to identify the specific mesh whose image will be reconstructed.

2. The method *reconstruct_img* (a method from the EIT module facade) is called from the service.

3. The facade method calls the method *reconstruct_img* from the class *ImagesReconstruction*.

4. The database is queried in order to obtain the information from the mesh whose image will be reconstructed.

5, 6. The voltages and conductivity values of the mesh are obtained.

7, 8, 9, 10. The method *predict_conductivity* is called and this one, in turn, calls the method *load_model*, from the class *ModelsManagement*. An object from the class *Model* is instantiated.

11. The voltages of the mesh are supplied as the input of the loaded Machine Learning model, and this one predicts the conductivity values.

12. From a base file that contains the information to represent an empty mesh, two new files are generated in an adequate format for EIDORS. One of the files contains the actual conductivities of the mesh, while the other one contains the predicted ones.

13, 14. The images are reconstructed by running Octave code. For that purpose, the Python module *oct2py* is used. This module reconstructs the images from the files generated in the step 12.

16, 17. The frontend receives the URLs of the generated images, as well as the list of predicted impedances.

3.3. Frontend

In order to develop the frontend, it was used React, a JavaScript framework. React is based on the use of *components*, which are JavaScript classes. Every one of the views of the application is composed of one or more React components. The utilization of this framework facilitated the implementation of dynamic views and behaviors. Figure 3.6 shows a package diagram with the different React components defined for the frontend.
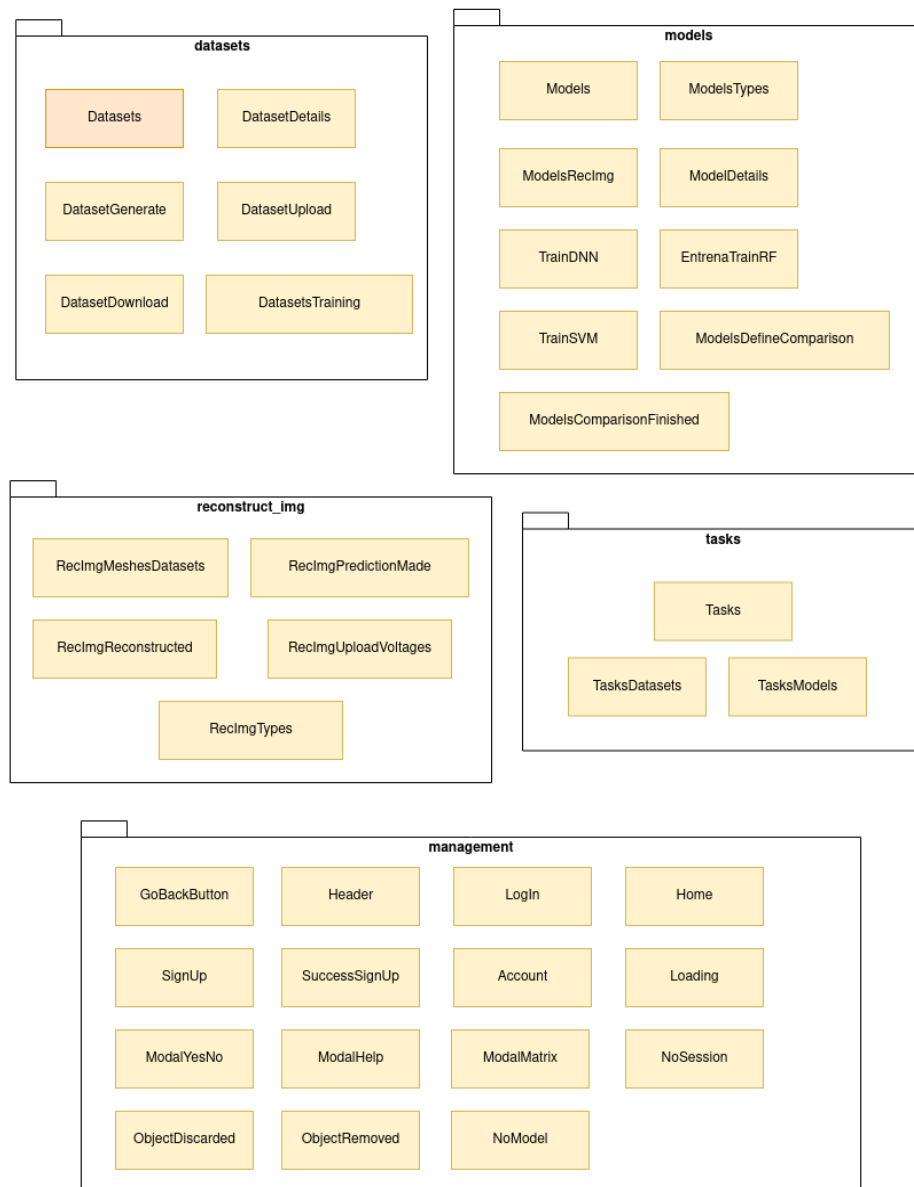
Figure 3.6. High-level package diagram of the frontend

3.3.1. Views

In this section, the main views of SageTomo will be presented.

3.3.1.1.   Main view
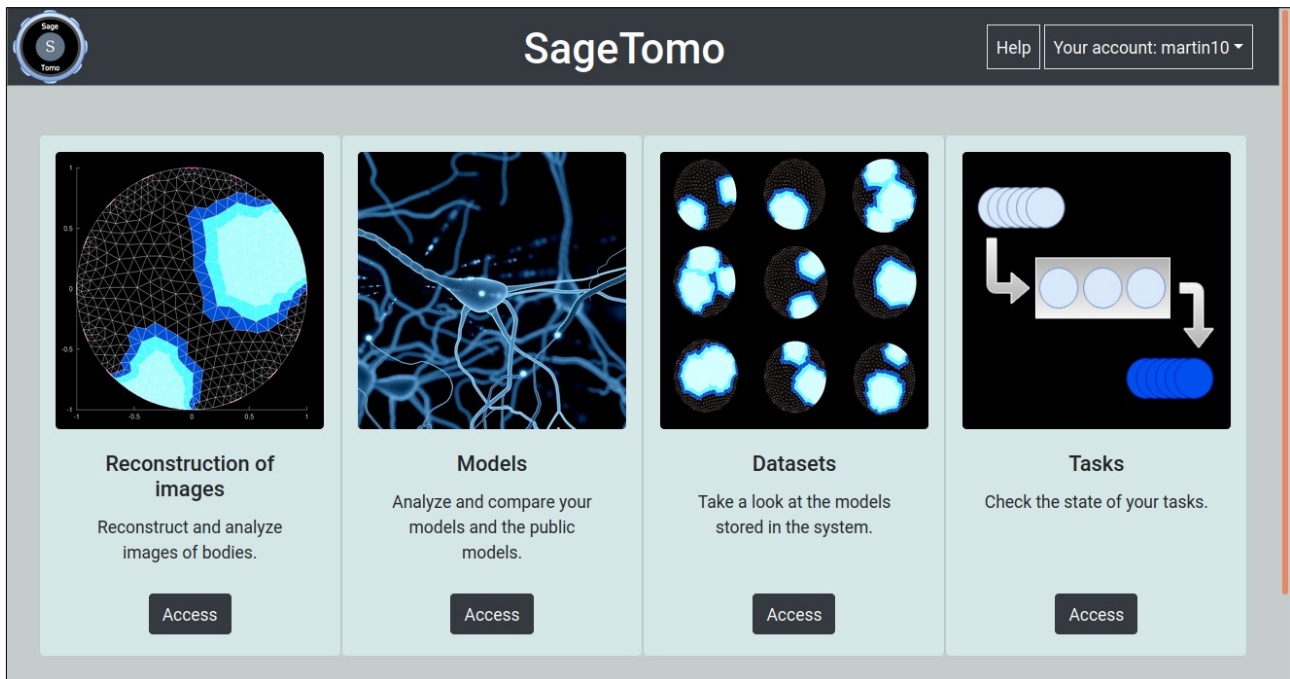
Figure 3.7. Main view.

Once users log in, they will access the main view of SageTomo. In the main view, they will be able to navigate through the different sections of the application:
- Reconstruction of images.
- Models.
- Datasets.
- Tasks.

3.3.1.2.   Reconstruction of images

Dataset and mesh selection

Figure 3.8. Selection of mesh for reconstrucion (I).

Figure 3.9. Selection of mesh for reconstruction (II).

Once the users select a Machine Learning model to make the predictions, they will have to choose the dataset (figure 3.8) and the specific mesh (3.9) whose image they wish to reconstruct.

Reconstructed image

Once the EIT reconstruction is done, the application will show two images: the actual image (left) and the image reconstructed by the Machine Learning model (right). Moreover, users can select a point in the Y-axis to make a transversal cut. If they click on the option *Analyze section*, a plot will be generated with the conductivity values (the real ones and the predicted ones) along the X-axis, in the point specified by the user.
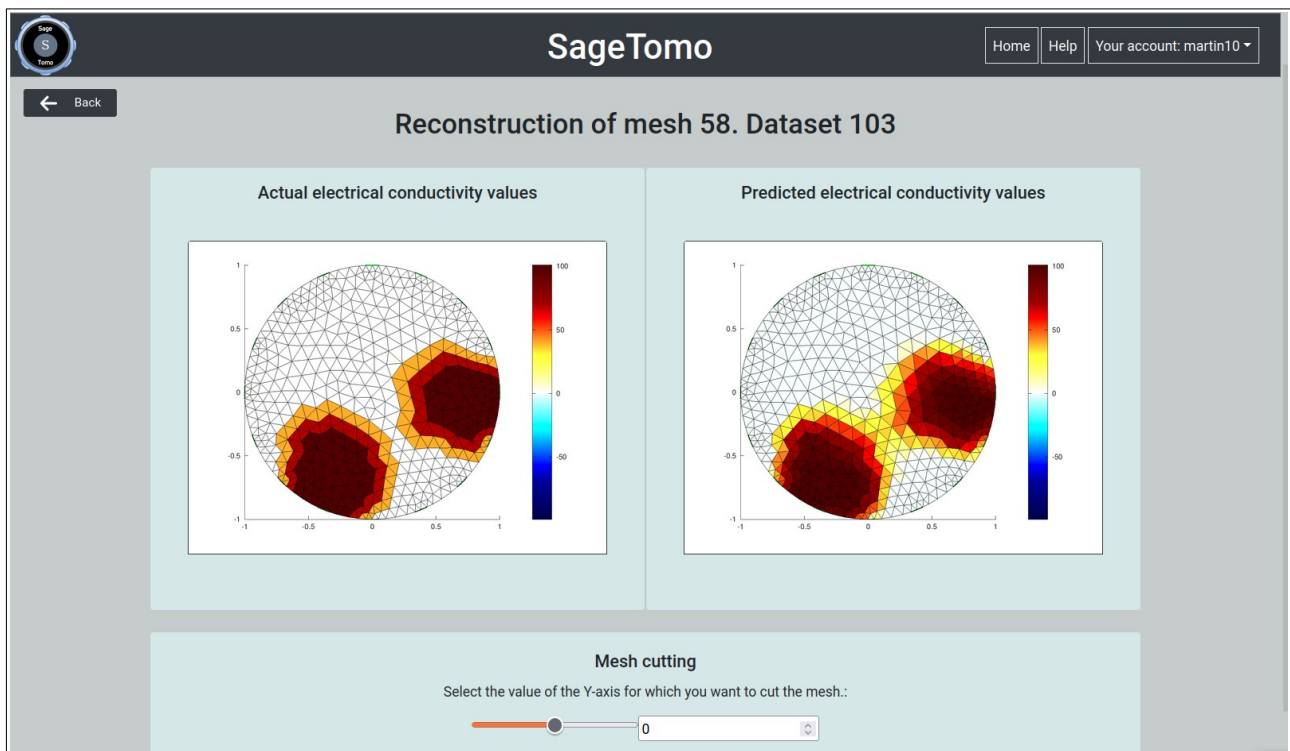
Figure 3.10. Actual and reconstructed images.
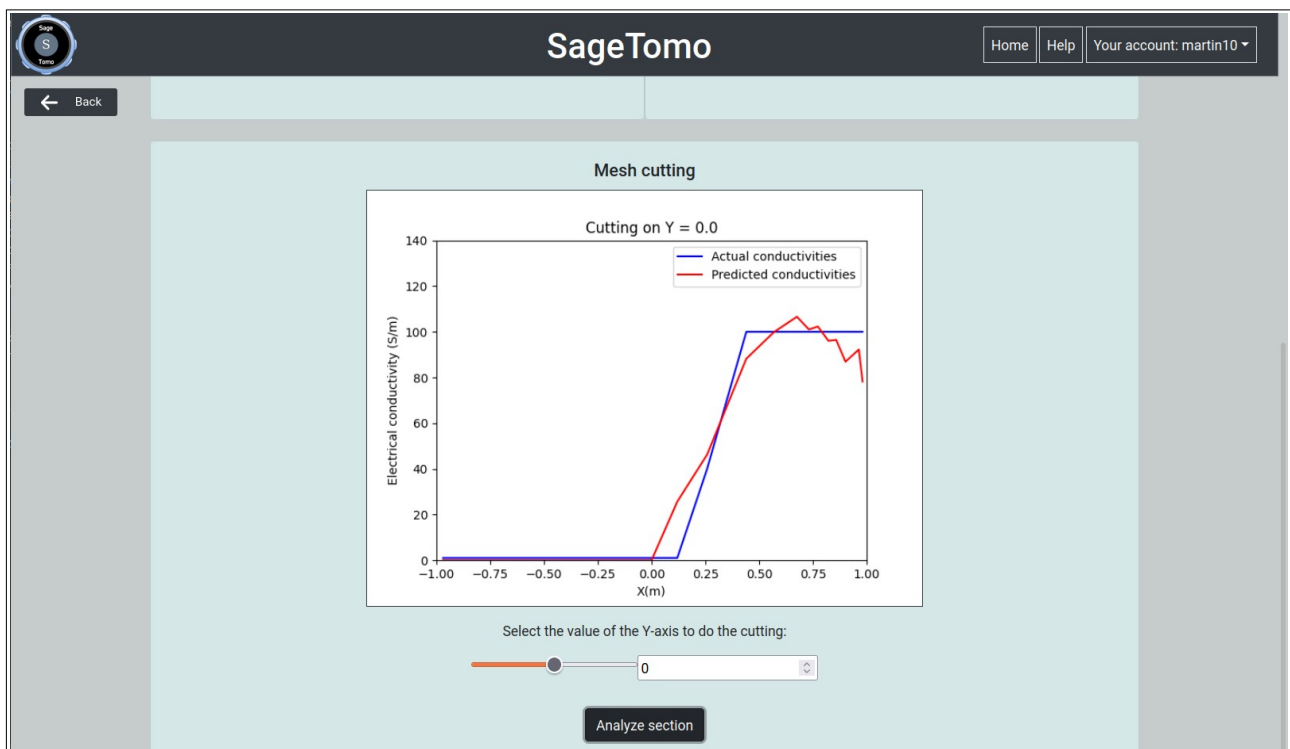
Figure 3.11. Mesh cutting.

3.3.1.3.　　Models

List of models

Figure 3.12. List of model stored in the system.

If the users select the option *Models* in the main view, they will access the list of available models in the system. Users can filter models by type. If users select the option *View details* for a particular model, they will see the specific information of such model. Figure 3.13 shows the details of a neural network. Users can also examine the confusion matrix of a model (figure 3.14).
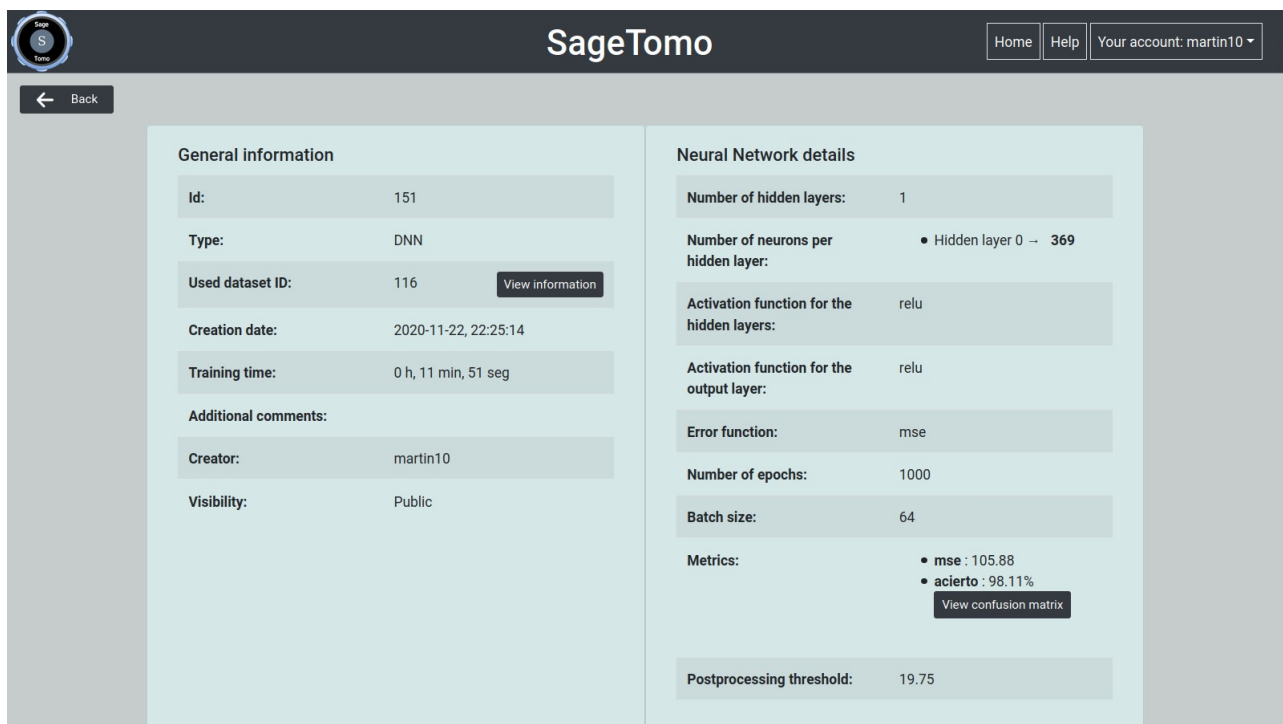
Figure 3.13. Details of Neural Network.
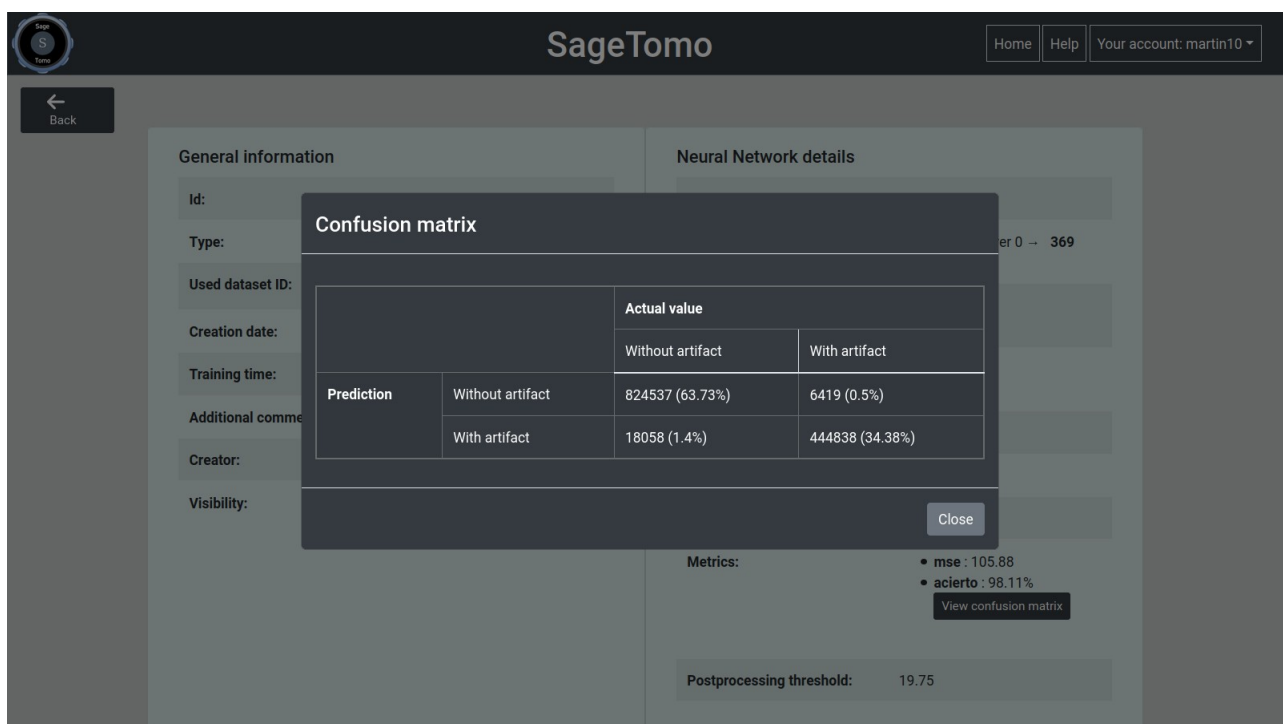
Figure 3.14. Confusion matrix.

Compare models

Users can select several models and compare them. They must choose a dataset for the comparison (figure 3.15) and the metrics they wish to use to evaluate the models.

Figure 3.15. Selection of dataset and metrics for comparison of models.

Once the comparison process has finished, users will be shown the values obtained by every model for each one of the metrics. If one of the selected metrics is the accuracy, confusion matrixes obtained by the models will be displayed (figure 3.16). Finally, users will be shown the real image of a mesh selected randomly and the reconstructions made by the Machine Learning models (figure 3.17).
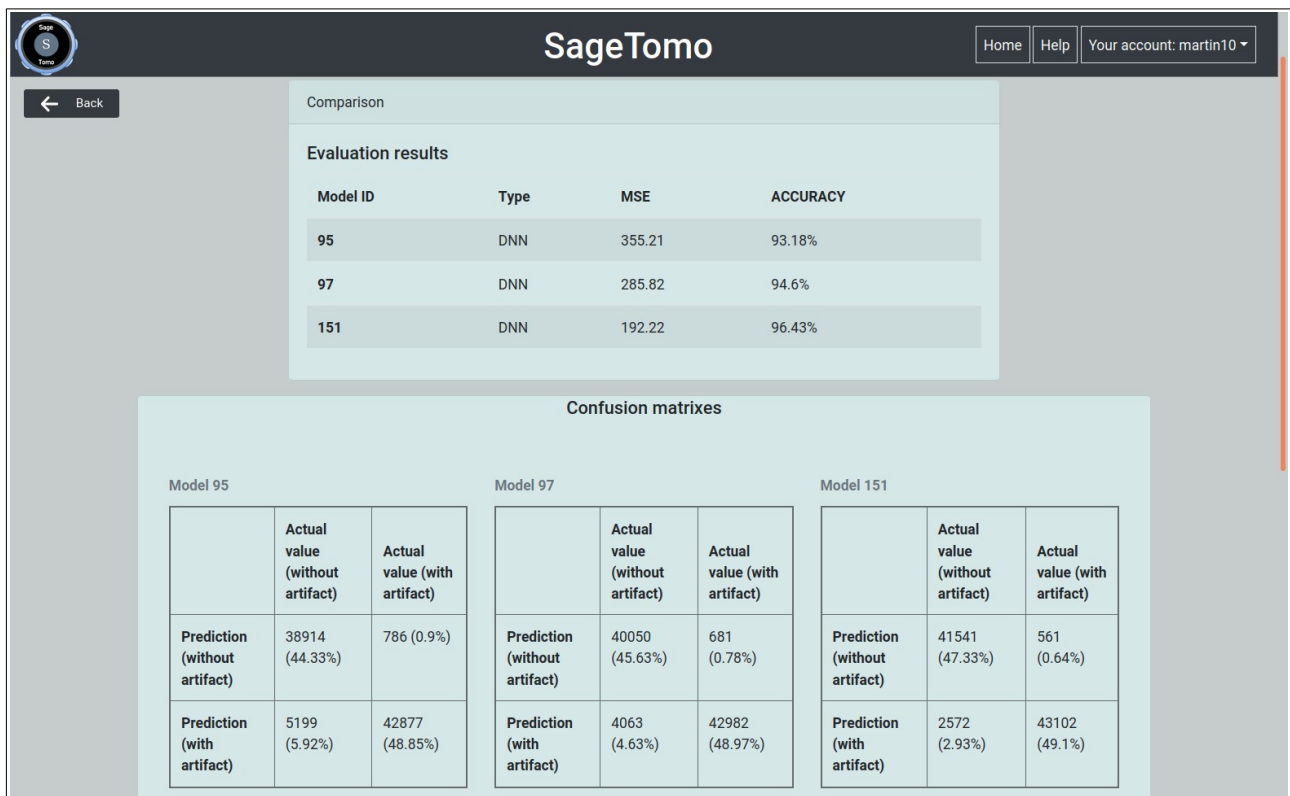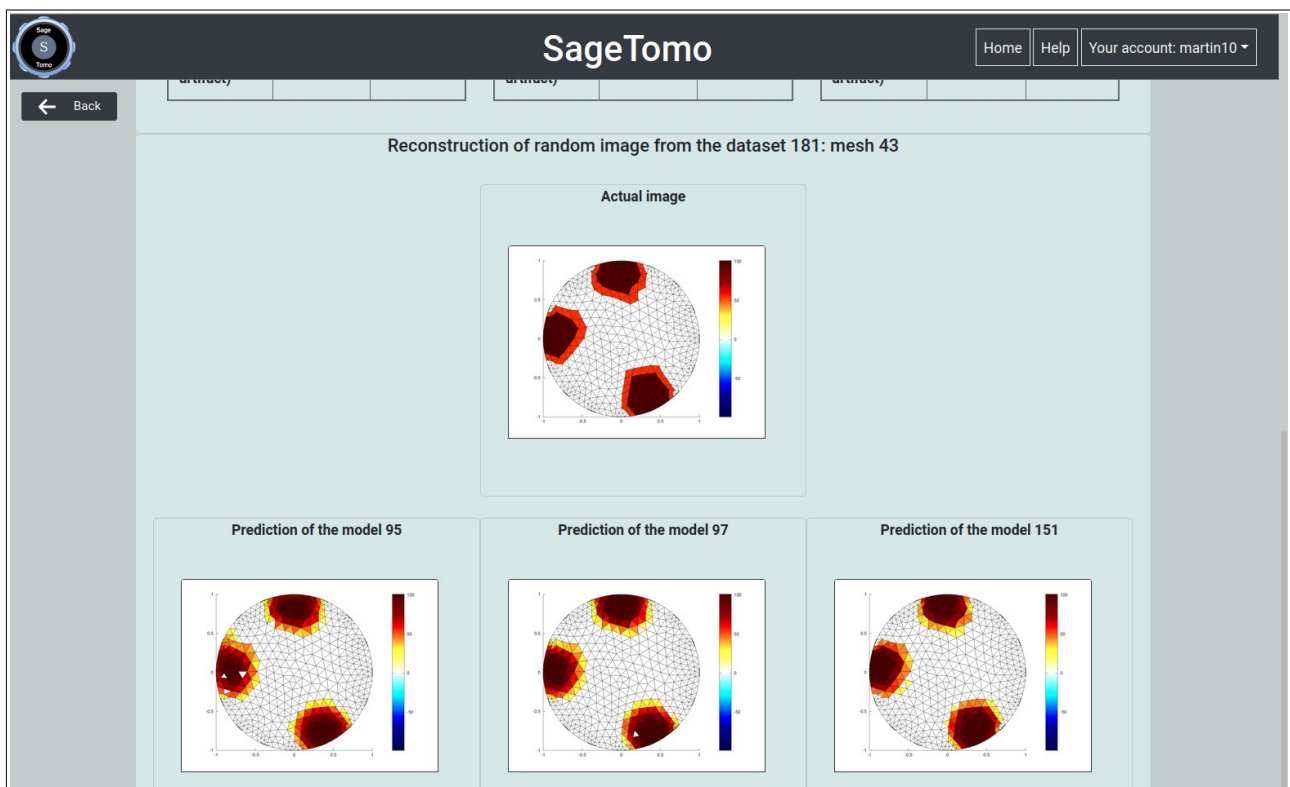
SageTomo

Home | Help | Your account: martin10 ▼

← Back

Comparison

Evaluation results

Model ID	Type	MSE	ACCURACY
95	DNN	355.21	93.18%
97	DNN	285.82	94.6%
151	DNN	192.22	96.43%

Confusion matrixes

Model 95

	Actual value (without artifact)	Actual value (with artifact)
Prediction (without artifact)	38914 (44.33%)	786 (0.9%)
Prediction (with artifact)	5199 (5.92%)	42877 (48.85%)

Model 97

	Actual value (without artifact)	Actual value (with artifact)
Prediction (without artifact)	40050 (45.63%)	681 (0.78%)
Prediction (with artifact)	4063 (4.63%)	42982 (48.97%)

Model 151

	Actual value (without artifact)	Actual value (with artifact)
Prediction (without artifact)	41541 (47.33%)	561 (0.64%)
Prediction (with artifact)	2572 (2.93%)	43102 (49.1%)

Figure 3.16. Comparison results (I).

SageTomo

Home | Help | Your account: martin10 ▼

← Back

Reconstruction of random image from the dataset 181: mesh 43

Actual image

Prediction of the model 95 | Prediction of the model 97 | Prediction of the model 151

Figure 3.17. Comparison results (II).

Train models

If users select the option *Train new model* in the *Models* view, they will have to choose the type of model they wish to train (figure 3.18). After that, they can set the values for the model hyperparameters. Figure 3.19 shows the hyperparameters that can be adjusted in the case of a neural network.
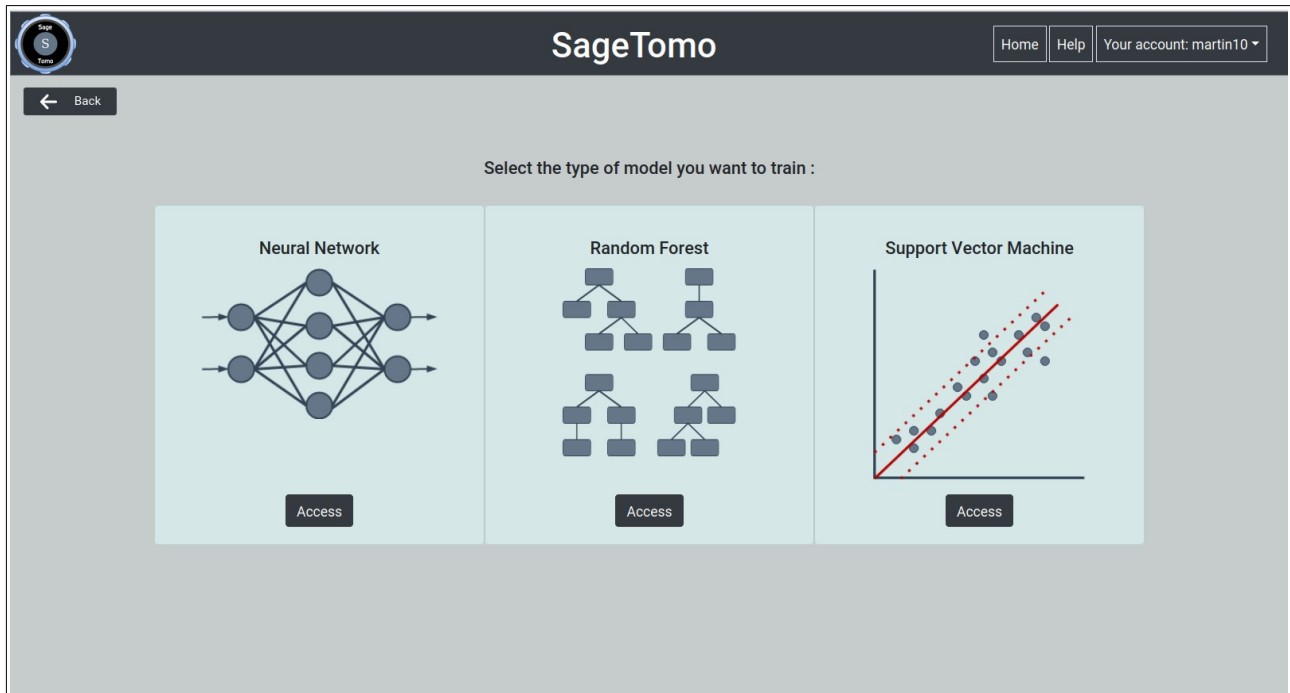

Figure 3.18. Types of models.

Figure 3.19. Training of a Neural Network.

3.3.1.4. Datasets

If the users select the option *Datasets* in the main view, they will access the list of available datasets in the system (figure 3.20). *They can obtain the specific details for every dataset by selecting the option View details* (figure 3.21).
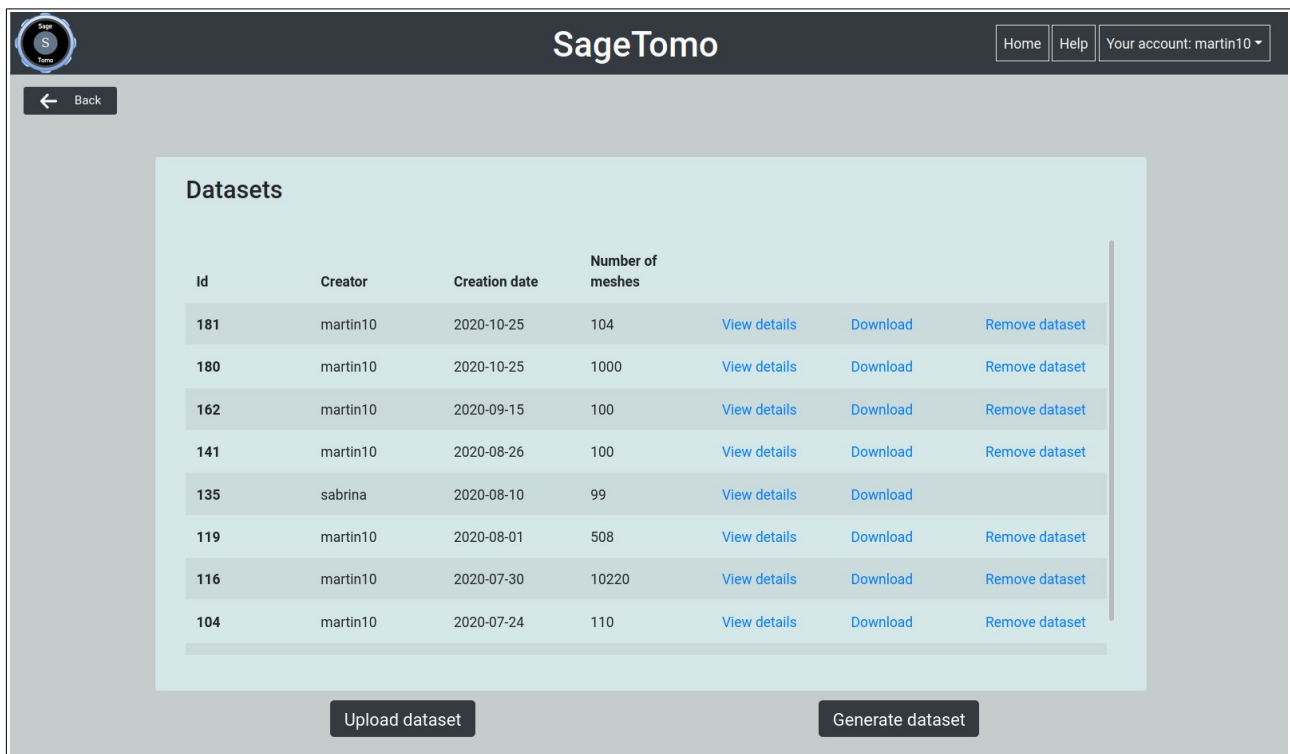
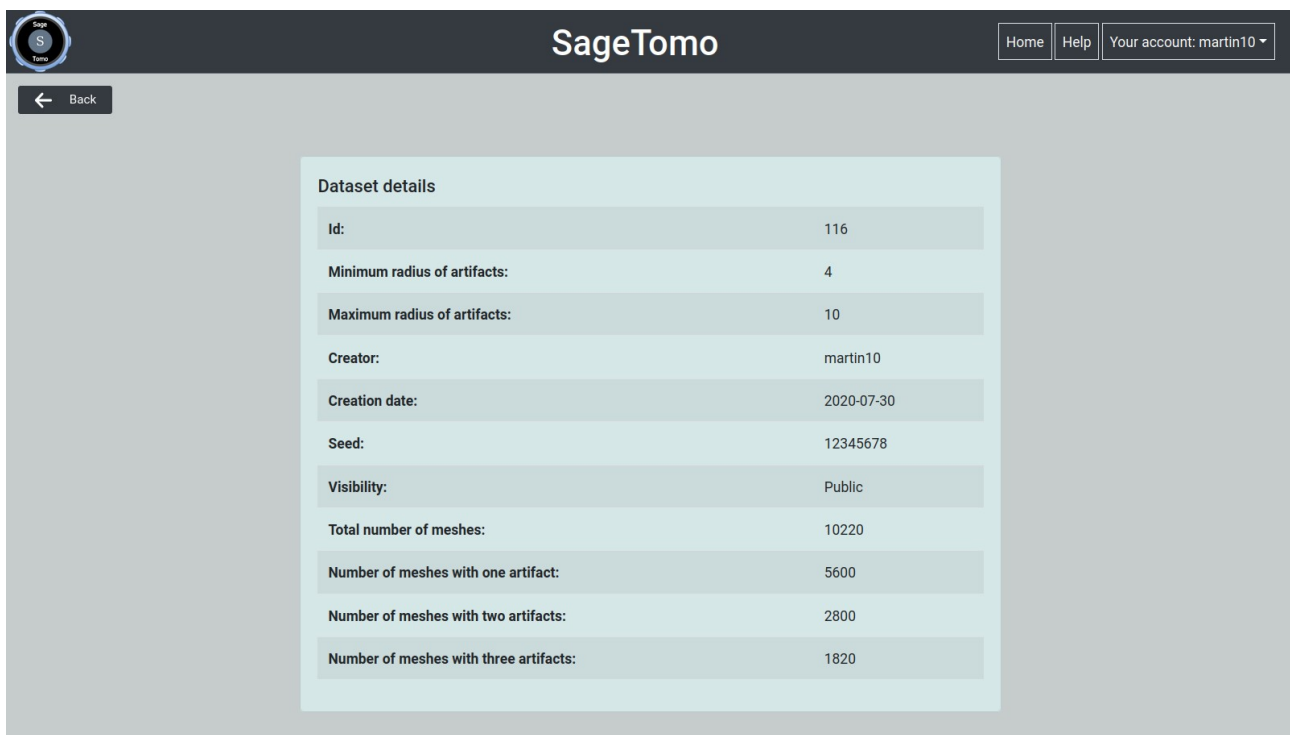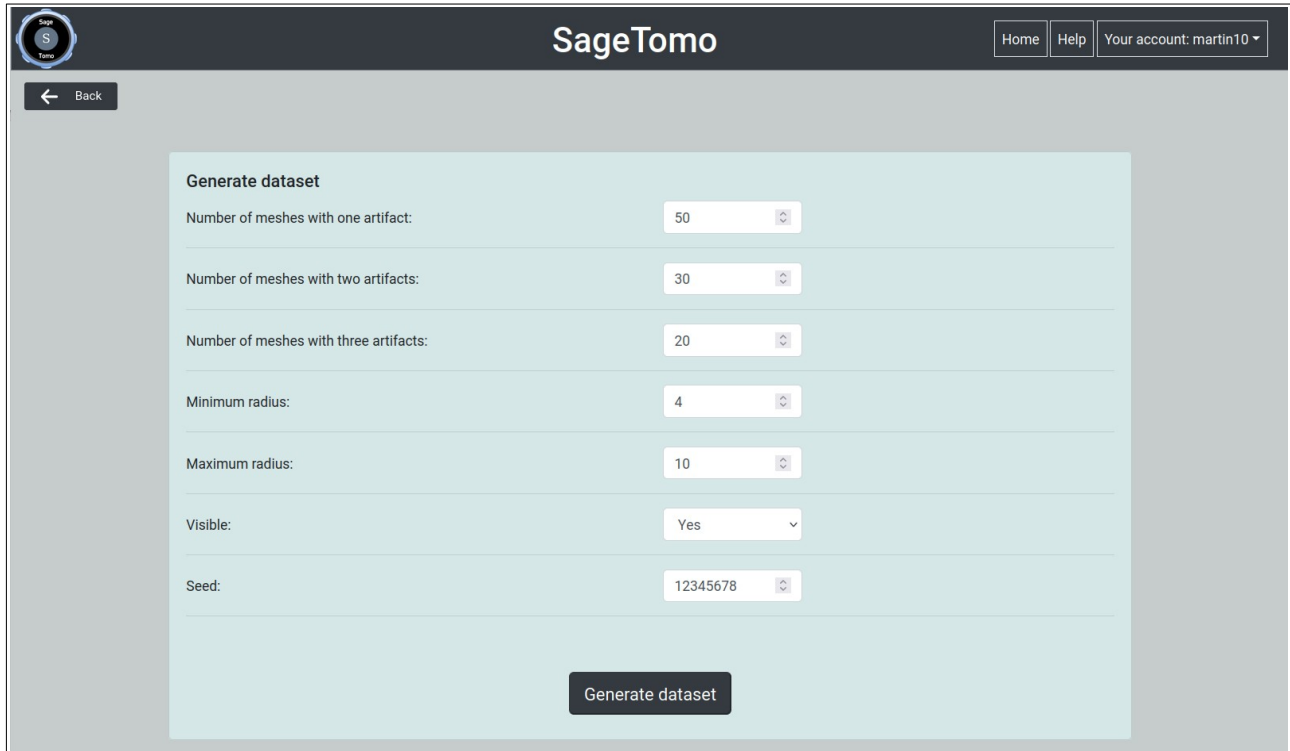*Figure 3.20. List of datasets stored in the system.*

Figure 3.21. Details of a dataset.

Generate dataset

Users can generate new datasets randomly. For that purpose, they must choose the wished features, as the number of meshes of the dataset.

Figure 3.22. Generation of a new dataset.

Tasks

In the *Tasks* section, users can query the lists of models being trained, as well as the lists of datasets being generated. Figure 3.24 shows the case of the lists of training models.
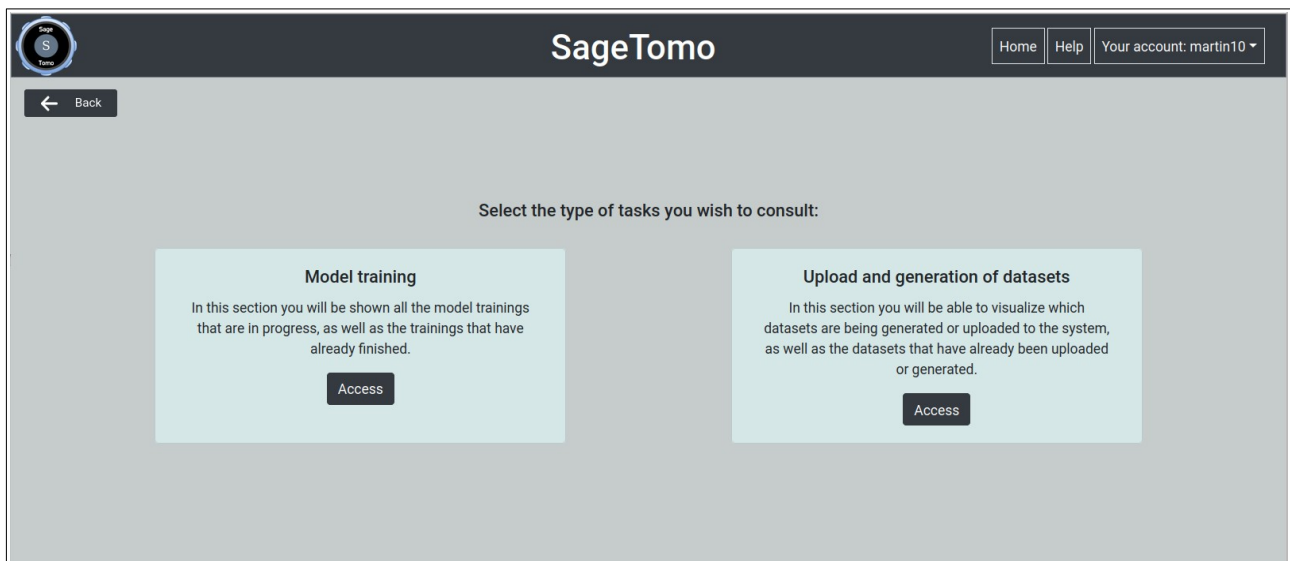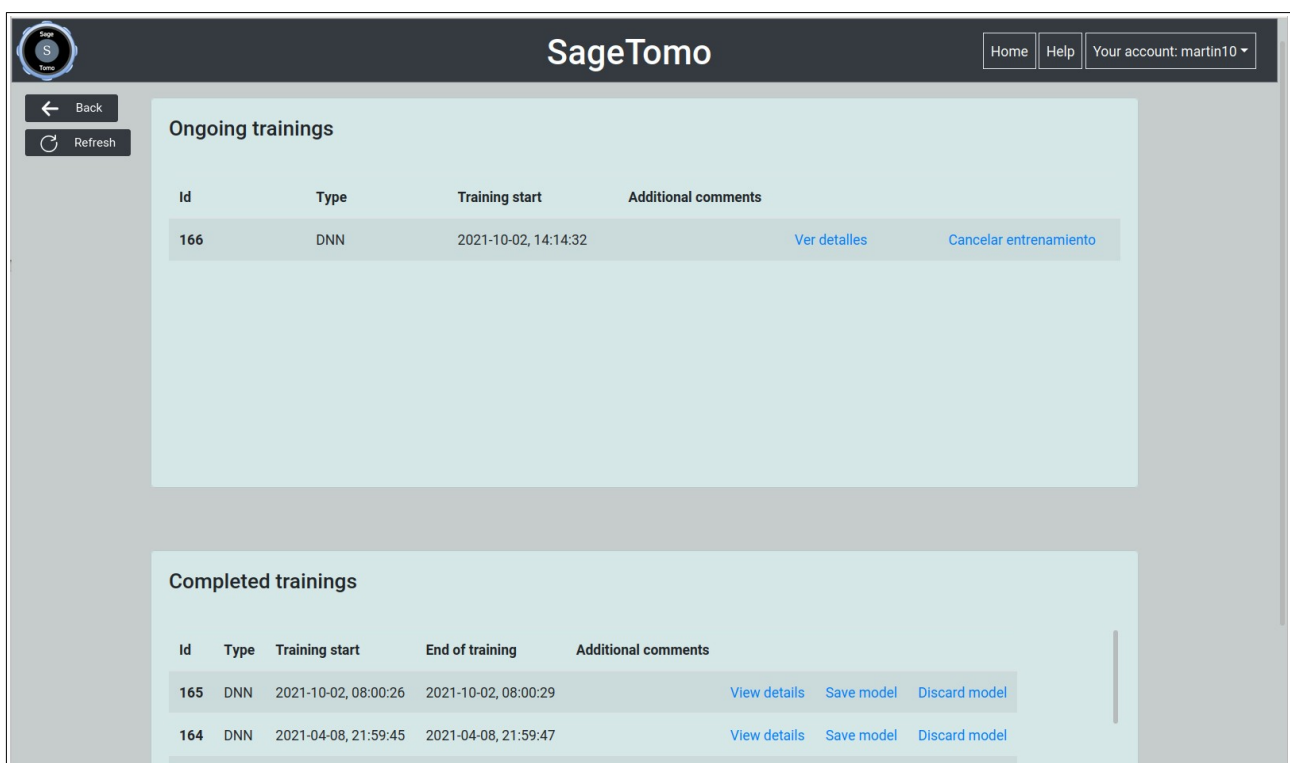
Figure 3.23. Types of tasks.

Figure 3.24. List of ongoing and completed trainings.

4.   Machine Learning

In this section, some of the more relevant Machine Learning aspects of SageTomo will be explained.

4.1.   Datasets

Datasets are composed of 1052 variables: 208 values of voltage (input variables) and 844 values of conductivity (output variables). As stated earlier, SageTomo uses Machine Learning models to predict the conductivity values from the voltage values.

4.2.   Methodology

When the users select a dataset to train a new model, the dataset meshes are shuffled randomly. Then, the dataset is divided into two sets: the training set (70% of the meshes) and the evaluation set (30% of the meshes). The training set is used to train the model. Once the model has been trained, the evaluation set is employed to evaluate it with the wished metrics.

The reason why the dataset meshes are shuffled before the training of the model is to ensure that there are different kinds of meshes both in the training set and in the evaluation set (meshes with different numbers and types of artifacts).

4.3.   Postprocessing

Predictions made by the models can have a certain noise. SageTomo allows the user to apply a postprocessing algorithm to decrease the noise in the predictions. The algorithm consists of assigning a background conductivity value (1 S/m) to all the cells in a mesh whose conductivity is below a certain threshold. In order to select the optimal thresholds, ROC curves are used.

5.   Conclusions

The developed software could be useful in a wide variety of industrial areas that require extracting information from the inside of bodies. The selected architecture and the use of a queue management system allow deploying SageTomo in a real production environment, parallelizing both the Machine Learning models training and dataset generation. SageTomo is an example of how Machine Learning techniques can be embedded in the software to improve industrial processes.