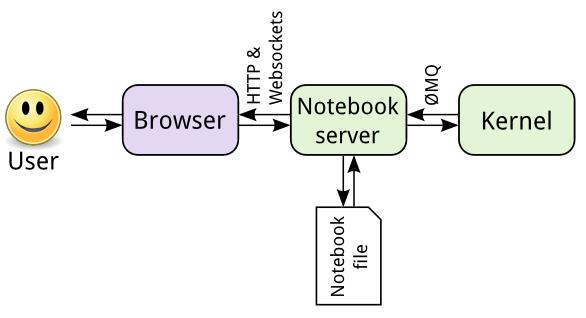
Chapter 5 DETAILED DESIGN

messages sent over ZeroMQ sockets. The core execution machinery for the kernel is shared with terminal IPython as shown in the figure 5.1. A kernel process can be connected to more than one frontend simultaneously. In this case, the different frontends will have access to the same variables.

**5.1.1.2 The Notebook**

The Notebook frontend does something extra. In addition to running your code, it stores code and output, together with markdown notes, in an editable document called a notebook. When you save it, this is sent from your browser to the notebook server, which saves it on disk as a JSON file with a .ipynb extension.



**Figure 5.2 User to Notebook Interface**

The notebook server, not the kernel, is responsible for saving and loading notebooks, so you can edit notebooks even if you don’t have the kernel for that language you just won’t be able to run code. The kernel doesn’t know anything about the notebook document: it just gets sent cells of code to execute when the user runs them.

**5.2 The Notebook design**

A design notebook is a way for a designer or engineer to keep a history of his or her desugn project from start to finish. It is a place to record research, observation, ideas, drawings, comments, and questions during the design process.

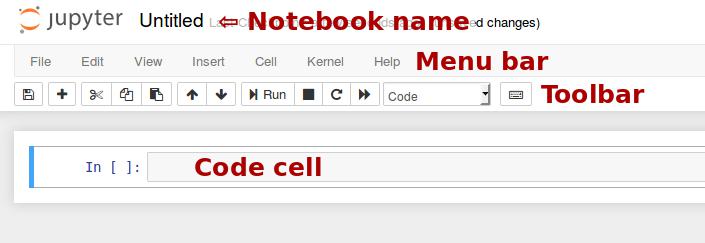
**5.2.1 Notebook dashboard**

When you create a new notebook document, you will be presented with the **notebook name**, a **menu bar**, a **toolbar** and an empty **code cell**.

|  |  |  |
| --- | --- | --- |
| Department of CSE, RNSIT | 2018-2019 | 21 |

Chapter 5 DETAILED DESIGN

* **Notebook name**: The name displayed at the top of the page, next to the Jupyter logo,reflects the name of the .ipynb file. Clicking on the notebook name brings up a dialog which allows you to rename it. Thus, renaming a notebook from “Untitled0” to “My first notebook” in the browser, renames the Untitled0.ipynb file to My first notebook.ipynb.
* **Menu bar**: The menu bar presents different options that may be used to manipulatethe way the notebook functions.
* **Toolbar**: The tool bar gives a quick way of performing the most-used operationswithin the notebook, by clicking on an icon.
* **Code cell**: the default type of cell; read on for an explanation of cells.



**Figure 5.3 Notebook Dashboard**

**5.2.2 Structure of a Notebook document**

The notebook consists of a sequence of cells. A cell is a multiline text input field, and its contents can be executed by using Shift-Enter, or by clicking either the “Play” button the toolbar, or “**Cell Run**” in the menu bar. The execution behavior of a cell is determined by the cell’s type. There are three types of cells: code cells, markdown cells, and raw cells. Every cell starts off being a **code cell**, but its type can be changed by using a **drop-down on the toolbar.**

* **Code cells**

A *code cell* allows you to edit and write new code, with full syntax highlighting and tab completion. The programming language you use depends on the *kernel*, and the default kernel (IPython) runs Python code. When a code cell is executed, code that it contains is sent to the kernel associated with the notebook. The results that are returned from this computation are then displayed in the notebook as the cell’s *output*.

|  |  |  |
| --- | --- | --- |
| Department of CSE, RNSIT | 2018-2019 | 22 |

Chapter 5 DETAILED DESIGN

* **Markdown Cells**

You can document the computational process in a literate way, alternating descriptive text with code, using *rich text*. In IPython this is accomplished by marking up text with the Markdown language. The corresponding cells are called *Markdown cells*. The Markdown language provides a simple way to perform this text markup, that is, to specify which parts of the text should be emphasized (italics), bold, form lists, etc. When a Markdown cell is executed, the Markdown code is converted into the corresponding formatted rich text. Markdown allows arbitrary HTML code for formatting.

* **Raw Cells**

*Raw* cells provide a place in which you can write *output* directly. Raw cells are notevaluated by the notebook. Raw cells arrive in the destination format unmodified. For example, you can type full LaTeX into a raw cell, which will only be rendered by LaTeX.

**5.2.3 The Output format**

A code cell can have a variety of outputs (stream data or rich mime-type output). These correspond to messages produced as a result of executing the cell. All outputs have an output\_type field, which is a string defining what type of output it is.

**5.2.3.1 Stream output and Display data**

{

"output\_type" : "stream",

"name" : "stdout", # or stderr

"text" : ["multiline stream text"],

}

Rich display outputs, as created by display\_data messages, contain data keyed by mime-type. This is often called a mime-bundle, and shows up in various locations in the notebook format and message spec. The metadata of these messages may be keyed by mime-type as well. This program explanation can be seen below.

{

"output\_type" : "display\_data",

"data" : {

|  |  |  |
| --- | --- | --- |
| Department of CSE, RNSIT | 2018-2019 | 23 |

Chapter 5 DETAILED DESIGN

"text/plain" : ["multiline text data"],

"image/png": ["base64-encoded-png-data"],

"application/json": {

* JSON data is included as-is "json": "data",

},

},

"metadata" : {

"image/png": {

"width": 640,

"height": 480,

},

},

}

**5.2.3.2 Execute result**

Results of executing a cell are stored in execute\_result outputs. execute\_result outputs are identical to display\_data, adding only a execution\_count field, which must be an integer. This program explanation can be seen below.

{

"output\_type" : "execute\_result",

"execution\_count": 42,

"data" : {

"text/plain" : ["multiline text data"],

"image/png": ["base64-encoded-png-data"],

"application/json": {

* JSON data is included as-is "json": "data",

},

},

"metadata" : {

"image/png": {

"width": 640,

"height": 480,

},

},

}

|  |  |  |
| --- | --- | --- |
| Department of CSE, RNSIT | 2018-2019 | 24 |

Chapter 5 DETAILED DESIGN

**5.2.3.3 Error**

Failed execution may show a traceback.

{

'ename' : str, # Exception name, as a string

'evalue' : str, # Exception value, as a string

* The traceback will contain a list of frames,
* represented each as a string.

'traceback' : list,

}

**5.2.3.4 Raw NBConvert cells**

A raw cell is defined as content that should be included *unmodified* in nbconvert output. For example, this cell could include raw LaTeX for nbconvert to pdf via latex, or restructured text for use in Sphinx documentation. The notebook authoring environment does not render raw cells.

The only logic in a raw cell is the format metadata field. If defined, it specifies which nbconvert output format is the intended target for the raw cell. When outputting to any other format, the raw cell’s contents will be excluded.

{

"cell\_type" : "raw",

"metadata" : {

* the mime-type of the target nbconvert format.
* nbconvert to formats other than this will exclude this cell. "format" : "mime/type"

},

"source" : ["some nbformat mime-type data"]

}

|  |  |  |
| --- | --- | --- |
| Department of CSE, RNSIT | 2018-2019 | 25 |