TECHNISCHE UNIVERSITÄT ILMENAU
Department of Databases and Information Systems

Research Project

Summer Semester 2014

AutoStudio: A generic Web App to transform flow-designs into action

Nikhil Rane
Matrikel 51384

Prof. Kai-Uwe Sattler

Ilmenau, September 17, 2014

Contents

List of Figures

List of Tables

1 Motivation

"Data is a precious thing and will last longer than the systems themselves."

- Tim Berners-Lee.

In today's world, the amount of data being generated is enormous, and so is the demand for its processing. With the growth of applications focusing on Data Mining, there is always a requirement of devising innovative ways of processing more data in less time. The nature of data varies highly based upon the systems it is generated from to the point when it becomes available. One of these natures is "streamed data". There are many frameworks to process streamed data; however, the one we use here is PipeFabric. It is a framework for processing streams of tuples, where the processing steps are described by queries formulated as dataflow graphs of operators. PipeFabric represents the execution engine by providing only a set of operators and utility classes. It is mainly intended to be used in conjunction with the PipeFlow language. [DISGa] PipeFlow is inspired by the Pig language for Hadoop and is designed for specifying dataflow programs. However, these programs are not compiled for Hadoop, but for the data stream engine PipeFabric. As PipeFabric is basically a framework of operators implemented in C++, the target language of PipeFlow is also C++. [DISGb]

PipeFlow specifies the dataflow by a sequence of statements. Each statement declares an operator manipulating tuples. An operator is fed tuples via one or more input pipes and can publish tuples over its output pipe. Each operator defines a specific task and has its own clauses and parameters.[DISGb] Currently, the user has to manually perform all tasks of creating a PipeFlow script, compiling, executing, monitoring, etc. This forces the user to be aware of the operators and its constructs. The user also needs to manually check for status of programs in execution, which becomes cumbersome as the number of programs increases. Moreover, this manual approach could lead to time & effort wastage if there occur any errors or misconfigurations. This motivated us to design an application to automate the whole process; and as a result, AutoStudio was implemented.

AutoStudio is a simple and responsive web application to run cross-platform using HTML5, Draw2D Touch and Node.js. In addition to above mentioned tasks, it also provides additional functionalities which makes it a "One-Stop-App" to handle the complete flow. The implementation of AutoStudio started only with PipeFlow in consideration. However, it is designed to be generic, and hence can be extended to handle any application similar to this workflow. For instance, designing a Pig flow, transforming into Pig language, then compiling into Hadoop programs and finally executing & monitoring the same. Moreover, AutoStudio will also return any intermediate messages in real-time and email you when the execution is complete! To model diverse applications, AutoStudio uses the terminology of "apps". Each of these apps represents a specific type of workflow and has a unique name. For instance, the name for PipeFlow and PipeFabric workflow is "PipeStudio".

In the course of subsequent chapters, we will explain AutoStudio with reference of the PipeStudio app. The next chapter explains the terminologies and workflow. Chapter 3 briefly explains all the technologies and frameworks used, followed by chapter 4 which explains the design. Chapter 5 provides an implementation overview and future scope.. Finally Chapter 6 concludes the report with Conclusions followed by Bibliography.

2 Terminologies and Workflow

2.1 AutoStudio

AutoStudio is a web application running in a web browser. It relies heavily on `JSON` for data exchange and pre-compiled templates (Hogan.js) for UI. To perform any task, a user has to login to AutoStudio with a `User ID` and `Password`. New users can easily create their own `User ID` using the `Signup` option. Once the user is successfully logged in, the user's `Home` page is displayed with a list of **Apps** and **Flow-Designs** sorted in the order they were last accessed. This is the point where the workflow begins. Below is a brief description of the terms which make up AutoStudio:

2.1.1 App

App is AutoStudio's terminology of referring to the design part of a Workflow (2.1.3). An app defines a set of Operators, Connection types, Properties, Parameters, etc. which have a pre-defined semantics. The semantics are later applied to a Flow-Design (2.1.2) to transform it into a target language script; for instance, PipeFlow script. The app we are focusing on here is PipeStudio.

2.1.2 Flow-Design

The diagram a user creates in AutoStudio which is later transformed into a script is known as a Flow-Design. A flow-design corresponds to a specific app having pre-defined semantics. AutoStudio allows CRUD operations on a flow-design and stores it primarily as JSON on server. However, it can also be exported in JSON, SVG or PNG formats right from AutoStudio at click of a button.

2.1.3 Workflow

The point at which a user launches an app and starts creating a flow-design to the point when it actually executes on server is called as Workflow. Workflows are typically tied to an application stack, for instance PipeStudio, PipeFlow and PipeFabric make up our stream processing workflow.

2.1.4 Flow-Design status

A flow-design undergoes different statuses while it is in a workflow. To quickly have an idea of the stage a flow-design resides in, each flow-design is assigned a status. These are as follows:

Unparsed

Assigned to a flow-design which is created but the script for it is not yet generated.

Scripted

Assigned to a flow-design whose script has been successfully generated.

Executing

A flow-design which is currently in execution holds this status. The user can monitor real-time stats for such flow-designs right from the `Home` page.

Execution Complete

A flow-design which is executed in past (successfully or with error) has this status. The user can easily download all files related to this execution from the app. For notification purposes, a user can only select if a notification is to be sent to a registered email ID when the execution is complete.

3 Requirements and Installation

3.1 Technologies and Frameworks

AutoStudio prominently uses Open Source softwares and frameworks. It should work out-of-the-box in any modern browser on any platform supporting HTML5. However, there could be some UI differences due inconsistencies within browsers and screen sizes.

3.1.1 Client side

HTML5, JavaScript, jQuery (and helping libraries), Twitter Bootstrap, Hogan.js

Mostly used for GUI, AJAX requests, file upload & download, etc. AutoStudio extensively uses pre-compiled Hogan templates. The source code for these templates is also included in `autostudio/public/src/templates/` directory. The data returned from server is simply passed to these templates for quick rendering.

Draw2D Touch

Enables creation of diagram applications in a browser. AutoStudio uses Draw2D Touch for creating and manipulating operators and connections. [Hera]

3.1.2 Server side

Node.js Web Server

The logic on server side is completely implemented in Node.js and its supporting modules. The event-driven, non-blocking I/O model of Node.js makes it fast, light-weight and efficient. It is greatly suitable for data-intensive real-time applications. [noda] [Tei12] Table 3.1 lists Node.js supporting modules and a brief information about them.

Database

On the database side, the most popular No-SQL database MongoDB is used. It is open source, JSON-style document oriented and supports conventional as well as additional properties for data-intensive applications. [mon]

Module	Description
Bunyan	It is a simple and fast JSON logging library for Node.js services. [bun]
Express	Express is a web application framework, providing a set of features for building single and multi-page hybrid web applications. [expb]
Express-session	A simple session middleware for Express. [expa] [bun]
Formidable	It is a module for parsing form data, especially file uploads. AutoStudio uses it with features of HTML5 for uploading files when starting execution of a flow-design. [for]
MongoDB Driver	Node.js driver for MongoDB. [mon]
Nconf	Used as an object-store for easy storage and retrieval of configuration properties for AutoStudio, its apps, parsers, etc. [nco]
Nodemailer	Used to send emails from Node.js. AutoStudio uses this module to send notifications to registered email IDs when execution of a flow-design is complete. [nodb]
Socket.io	A realtime framework server for Node.js. AutoStudio uses this module extensively to send real-time stats to client when a flow-design is in execution. [soc]

Table 3.1: Node.js Supporting Modules

3.2 Installation

Installing AutoStudio and running is fairly simple and quick. The complete source code is included with a `package.json` file which lists all Node.js modules that are needed for AutoStudio to run. Here are the steps:

1. Install NPM Package Manager and MongoDB
2. Start MongoDB daemon using the command
   `$ mongod <options>`
3. Download the source code from Git repository
4. In Terminal, navigate to the directory where source code is downloaded
5. Fire the command
   `$ npm install`
6. After successful installation, fire the command
   `$ node server.js`
   And that's it! AutoStudio should be up and running on port 80.

4 Design

4.1 Design

The core design of AutoStudio's apps is driven by Operators, Connections and their properties. As AutoStudio uses Draw2D Touch for diagramatic representations, it follows the same policy of creating Classes for its implementation. Draw2D Touch has numerous classes for modeling each of its components. The complete documentation with examples could be found on its website. [Herb]

4.1.1 App model

The client side for each app in AutoStudio is modeled in a single JSON file. Based upon the user's selection from `Home` page, the right JSON file is fetched from server. This JSON file contains complete information about Operators, Connection Types, their properties, paths, etc. Table 4.1 provides a brief information.

AutoStudio has three main classes: `GenericShape.js` to model Operators, `GenericConnection.js` to model Connections and `GenericContainer.js` to model a group of Operators and Connections. The inheritance class diagrams for these classes are shown in Fig. 4.1, Fig. 4.2 and Fig. 4.3 respectively.

The next chapter provides an overview by explaining the file structure on server and client side of AutoStudio.

Key	Description
appName	The name of the app to be displayed on UI.
urlPrefix	The prefix to URL so that the correct routines corresponding to the app are called on the server side.
fileExtension	The file extension to use in case the user does not enter one in the file name.
imgList	An array of a component's name and image path grouped as per their nature.
connectionTypes	An array of types of connections supported by this app.
<conn_name>:<conn_properties>	A JSON {key:value} pair describing a connection and its properties.
<op_name>:<op_properties>	A JSON {key:value} pair describing an operator and its properties. Properties may include color, parameters, template name, help text, etc.

Table 4.1: Description of keys in app's JSON file

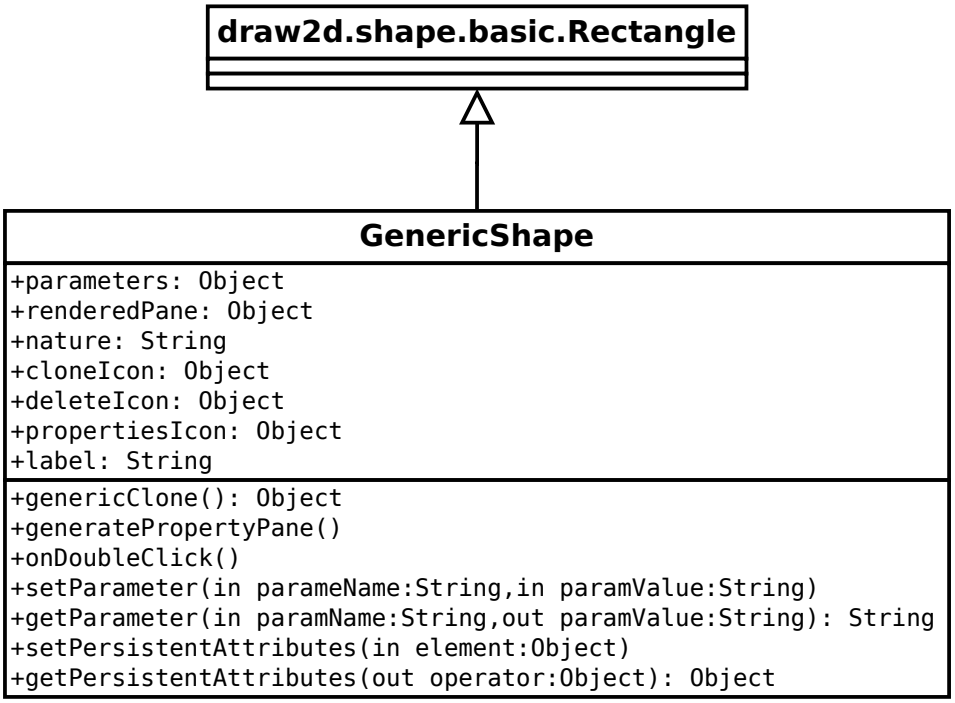

Figure 4.1: GenericShape Class Inheritance

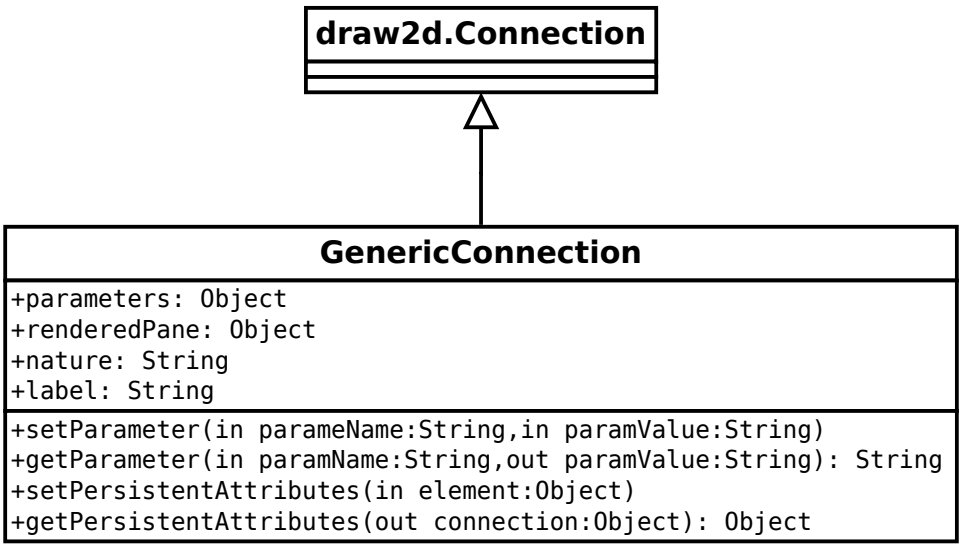

Figure 4.2: GenericConnection Class Inheritance

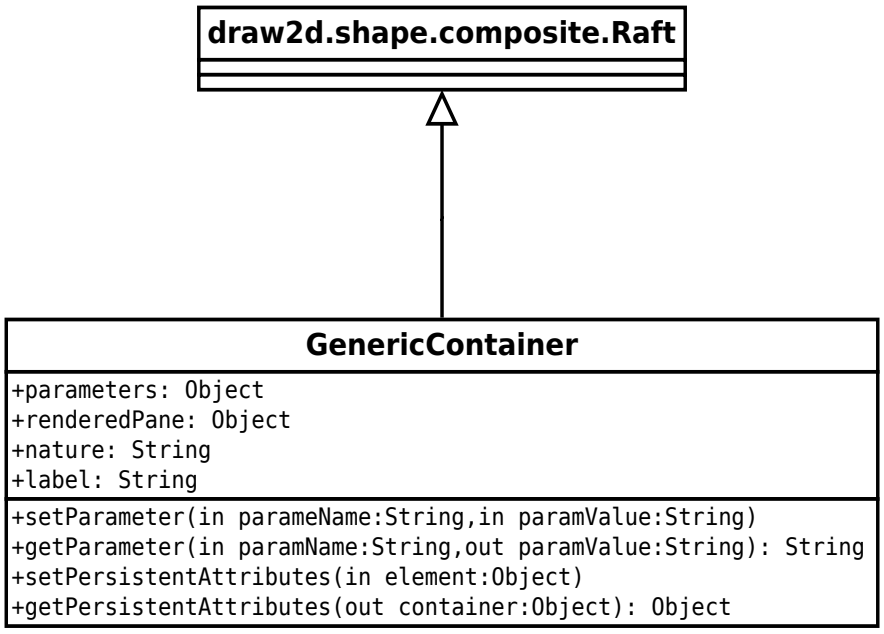

Figure 4.3: GenericContainer Class Inheritance

5 Implementation Overview

This chapter provides an overview of how components of AutoStudio are laid out on server and client sides. Below is file structure for the server followed by the file structure for the client.

5.1 Server side File Structure

Fig. 5.1 shows the file structure. Some of the directories and files are standard. However, some more have been added for AutoStudio. These are explained below.

5.1.1 config

Contains all configuration properties files which are loaded using Nconf. Multiple files exist corresponding to each app.

5.1.2 core_modules

All JavaScript files which handle a module or app are placed here. For instance, the PipeStudio app is implemented in `pipestudio.js` file. It implements actions for all URLs, parser and executor. Comments are added in `pipestudio.js` to better explain the flow.

5.1.3 executions

It contains two directories. One is `scripts` which contains shell scripts to trigger execution of programs and the other is `output` which consists of all programs, files, etc. which are generated during execution. The username, flow-design name and timestamps are used to ensure unique hierarchy for each execution.

5.1.4 logs

Consists of all logs generated on server side. The type and amount of logs can be configured in `server.js`.

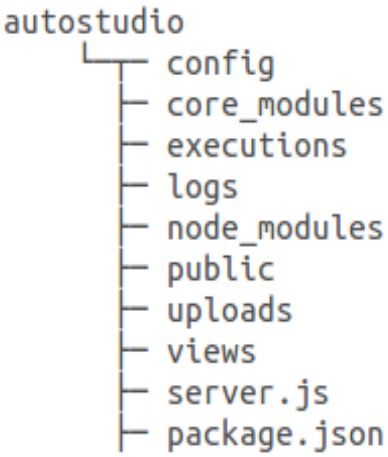

Figure 5.1: Node.js Web Server File Structure

5.1.5 node_modules

A standard Node.js directory containing files for the modules listed in `package.json` and which are installed.

5.1.6 public

Contains all files which are public to the client. CSS Style Sheets, JavaScript libraries, Draw2D Touch, AutoStudio source code, etc. all reside in this directory.

5.1.7 uploads

AutoStudio provides the user with an ability to upload files when starting an execution. These files are temporarily placed in this directory and moved to the actual execution folder when upload is successfully complete.

5.1.8 views

Contains source code for any templating engines the server is configured to use.

5.1.9 package.json

Lists all modules with NPM conventions for installation and tracking.

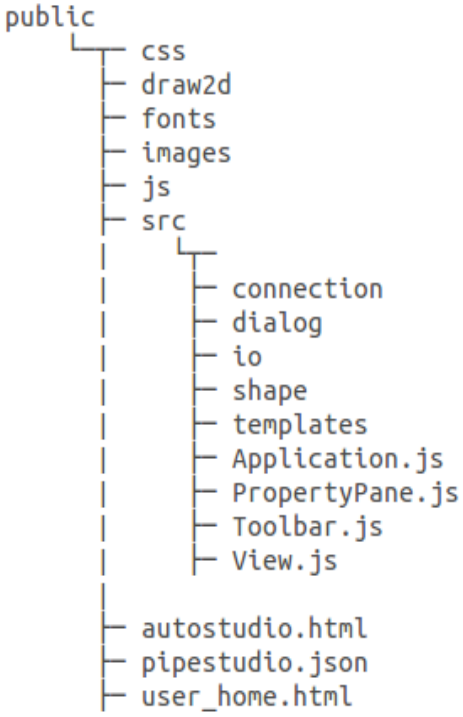

Figure 5.2: Client File Structure

5.1.10 server.js

The main web server file to run when starting AutoStudio. By default, AutoStudio is started on port 80.

5.2 Client File Structure

As explained in the previous section, the client side source code is exposed in the `public` directory. The important directories are explained below with Fig. 5.2 showing the file structure.

5.2.1 css

Contains all Cascading Style Sheets used by AutoStudio. It also includes standard CSS styles by Twitter Bootstrap and jQuery's helping libraries. The current Twitter Bootstrap version is 3.2.0.

5.2.2 draw2d

Contains source code for complete Draw2D Touch Framework. It also includes its own CSS styles and jQuery libraries. The current version included with AutoStudio is 4.9.0.

5.2.3 fonts

All external fonts and icons are stored in this folder.

5.2.4 images

Contains numerous images used by AutoStudio as a whole. It also contains images for Operators, Connections, etc.

5.2.5 js

Contains all standard and custom JavaScript libraries used by AutoStudio. The standard libraries include Twitter Bootstrap, jQuery Handsontable, jQuery Forms, etc.

5.2.6 src

Consists core logic of GUI for AutoStudio apps. There are a number of subdirectories inside `src` as explained below.

connection

Contains `GenericConnection.js` class. More classes related to connections could be added here if required.

dialog

Contains JavaScript files for displaying dialog boxes on the UI.

io

The approach Draw2D Touch uses to read JSON data and instantiate shapes (operators, connections or containers in context of AutoStudio) does not support genericness. Hence, a customized Reader class is implemented and stored under this directory.

shape

Contains classes for implementation of generic operator and generic container.

templates

Contains numerous pre-compiled templates for operator's properties, File Open Dialogs, File Download Dialogs, etc. AutoStudio heavily uses these templates for rendering parts of UI

which are repetitive and dynamic thus making it faster and responsive. The source code for these templates is also placed inside this directory with `.hogan` extension.

5.2.7 Application.js

This is the main application JavaScript file. It instantiates the View, Toolbar and PropertyPane objects to be shown on UI.

5.2.8 PropertyPane.js

The bottom right section to display generated scripts and execution stats of an open flow-design is modeled by this class.

5.2.9 Toolbar.js

The top menu strip with numerous options of creating and saving flow-designs, view options like zoom in & zoom out, grid, etc. are all modeled by the `Toolbar` class.

5.2.10 View.js

This class models the canvas for drawing area. It also registers listeners to listen for various events and execute actions accordingly.

5.2.11 autostudio.html

It is the main HTML file which paints the complete UI. It needs to include all the scripts and cascading style sheets which are used by AutoStudio.

5.2.12 pipestudio.json

The PipeStudio app is defined by this `pipestudio.json`. It contains all information about operators, their properties, images, connection types, etc. For adding more apps, there need to exist a separate JSON file in this (`public`) directory.

5.2.13 user_home.html

It renders the user's home page when the user successfully logs in.

5.3 Adding a new operator, connection type or container

As AutoStudio relies on JSON data for construction of its apps, adding an operator, connection type or a container involves only adding the required {key:value} pairs in the app's JSON file. In our example of PipeStudio, this file is `pipestudio.json`. However, this assumes that the required resources like operator's image, properties dialog, etc. are in place. If a properties dialog box is not explicitly mentioned, AutoStudio supplies a default one so that the flow does not break.

5.4 Adding a new app

An app (as a workflow) is made up of various components. These are as follows:

- **JSON file**: it contains all configurations of operators, connection types and containers.
- **Resources**: all the required resources like images, properties dialogs, etc.
- **Database**: a database entry is required to make AutoStudio aware of this new app. The flow-designs created using AutoStudio are stored in separated MongoDB collections. Hence, each app has its own collection which provides flexibility to apps, to handle their flow-designs as per requirement.
- **Parser**: to parse Draw2D Touch output into target language. For instance, in PipeStudio, the parser parses the output into PipeFlow language.
- **Executor**: a shell script or command to run the parsed script. An example of a parser and executor can be found in `pipestudio.js`.

This explains the components of AutoStudio. There are more details specific to implementation which are better documented in AutoStudio's source code using comments.

5.5 Future Scope

Data crunching applications have innumerable number of possibilities to improve the process and derive faster insights of data and its correlations. Similarly, AutoStudio also has many possibilities to make it more powerful and faster. Some of them are listed below.

- Adding new apps to support various workflows.
- Offline data storage support to handle intermittent network disconnections.
- Ability to generate short test-data based upon domain knowledge.
- Ability to schedule executions for streams of data which are available only at specific times.
- Ability to share files between users to facilitate teams to work in collaboration.
- and many more...

6 Conclusions

- Data mining applications are in constant demand of innovative ways to create data processing programs and crunch data faster.
- AutoStudio understands this need and provides a single point to manage the complete workflow.
- It is generic, extensible and works across diverse platforms offering responsive UI and real-time output.
- AutoStudio reduces the time by a significant margin to develop and test data-intensive programs with minimal supervision.

Bibliography

[bun] Bunyan. https://www.npmjs.org/package/bunyan. Last visit: 15.09.2014.

[DISGa] Databases and TU Ilmenau Information Systems Group. Pipefabric. http://dbgit.prakinf.tu-ilmenau.de/code/pipefabric/tree/master. Last visit: 14.09.2014.

[DISGb] Databases and TU Ilmenau Information Systems Group. Pipeflow. http://dbgit.prakinf.tu-ilmenau.de/code/pipeflow/wikis/home. Last visit: 14.09.2014.

[expa] Express-session. https://www.npmjs.org/package/express-session. Last visit: 15.09.2014.

[expb] Expressjs. http://expressjs.com/. Last visit: 15.09.2014.

[for] Formidable. https://www.npmjs.org/package/formidable. Last visit: 15.09.2014.

[Hera] Andreas Herz. Draw2d touch. http://www.draw2d.org/draw2d/. Last visit: 15.09.2014.

[Herb] Andreas Herz. Draw2d touch documentation. http://draw2d.org/draw2d_touch/jsdoc_5/#!/example. Last visit: 15.09.2014.

[mon] Mongodb. http://www.mongodb.org/. Last visit: 15.09.2014.

[nco] Nconf. https://www.npmjs.org/package/nconf. Last visit: 15.09.2014.

[noda] Node.js. http://nodejs.org/. Last visit: 15.09.2014.

[nodb] Nodemailer. https://www.npmjs.org/package/nodemailer. Last visit: 15.09.2014.

[soc] Socket.io. https://www.npmjs.org/package/socket.io. Last visit: 15.09.2014.

[Tei12] Pedro Teixeira. *Professional Node.js: Building Javascript Based Scalable Software*. Wrox Professional guides, first edition, October 2012.