# picoAds

**Michael Sioutis**

Department of Informatics and Telecommunication
National and Kapodistrian University of Athens
sioutis@di.uoa.gr

September 2011

**Abstract**

This document serves as a technical report for the `picoAds` implementation, designed and implemented by postgraduate student Michael Sioutis for the course of e-Commerce technologies.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The picoAds website was designed and implemented by postgraduate student Michael Sioutis for the course of e-Commerce technologies.

Figure 1.1 shows a preview of the eStore website at the time of writing this report.

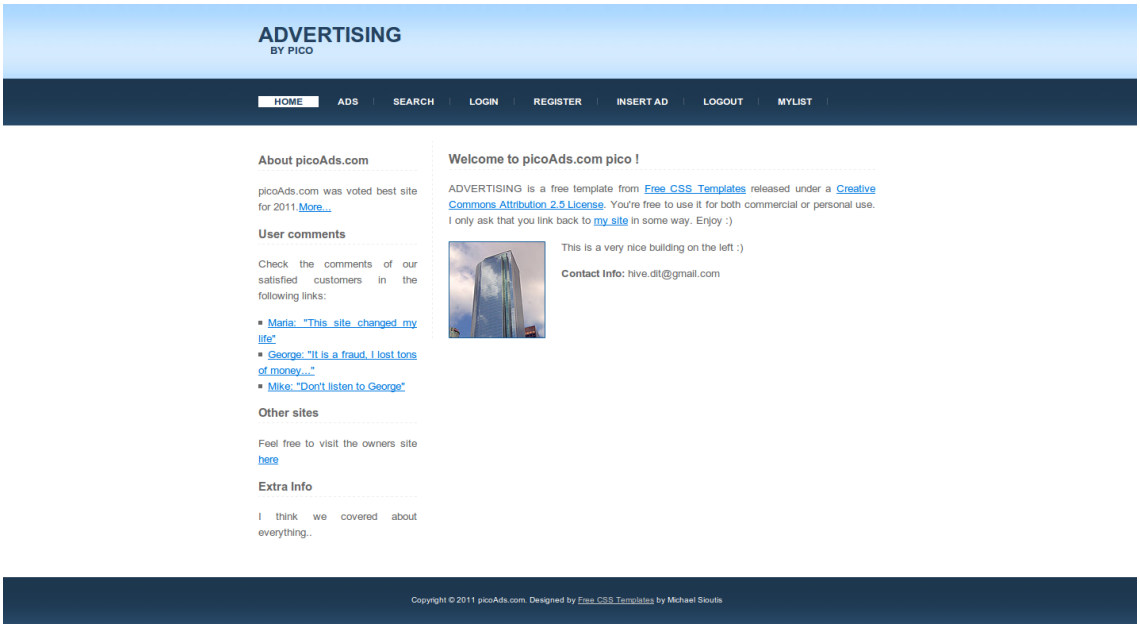


Figure 1.1: picoAds

# Chapter 2

# Technical Infrastructure

In this chapter we present the technical details of the picoAds project.

## 2.1 Software Details

For the purposes of the project the following technologies were used:

- **Netbeans IDE[1].** This is a fully-featured, free, open-source Java Integrated Development Environment for software developers, written completely in Java, with many modules available. The version of choice was 6.8 supplemented with two plugins to enable Web Services support, in terms of design and implementation.

  Specifically, we installed the following two plugins:

  - SOA
  - XML Schema and WSDL

  These plugins were available from the following update center: `http://dlc.sun.com.edgesuite.net/netbeans/updates/6.7.1/uc/final/beta/`

- **Sun GlassFish Enterprise Server[2].** GlassFish is an open source application server project led by Sun Microsystems for the Java EE platform. It uses a derivative of Apache Tomcat as the servlet container for serving Web content, with an added component called Grizzly which uses Java New I/O (NIO) for scalability and speed.

  We used both versions 3.0.1 and 2.1.1, as the latter one was needed to deploy our SOA composite application. To run both servers in parallel, we manually edited the `domain.xml` of GlassFish server 2.1.1, enabling different listening ports to the ones of GlassFish version 3.0.1.

  Additionally, we used the `updatetool` to update GlashFish server 2.1.1 with the `Composite Applicationss` package.

[1] `http://netbeans.org/`
[2] `http://glassfish.java.net/`

- **MySQL**[3]**.** This is a popular open source database best known for its high performance, high reliability and ease of use. Version 14.14 distribution 5.1.41 with default settings was used to store the data needed for the project.

- **phpMyAdmin**[4]**.** phpMyAdmin is a free software tool written in PHP intended to handle the administration of MySQL over the World Wide Web. phpMyAdmin supports a wide range of operations with MySQL, while it still provides the ability to directly execute any SQL statement. Version 3.3.2 was used so that all members of the team could access a common remote databse through a shared account. The page of access is: `http://pic0.dyndns.org/phpmyadmin/`.

- **Dropbox**[5]**.** We used Dropbox as an easy way to store, sync, and, share files online. Dropbox is a Web-based file hosting service operated by Dropbox, Inc. that uses cloud computing to enable users to store and share files and folders with others across the Internet using file synchronization.

## 2.2 Database Structure

A functional online ad store needs a database in order to store and retrieve information about its users, ads, lists, and anything that is relevant for online advertising. We approached the database structure having in mind simplicity and functionality. A database with the name `ad` was created and a brief description of its tables is provided:

- **advertisement**. Table 2.1 shows the structure of table `advertisement`. Field `adid` is the primary key, field `shortDisc` stores the short description of an advertised item, field `category` stores the type of an advertised item, whether it's about a home or a shop, field `longDisc` stores the long description of an advertised item, field `type` stores the use of an advertised item, whether it's for sale or rental, field `price` stores the price of an advertised item, field `image` stores the image url of an advertised item, field `location` stores the location of an advertised item which is later used to enable map viewing of the item, field `contact` stores the contact information of the user who created the advertisement, and field `day` stores information about the remaining days that the advertisement will be available online.

  This table embodies the notion of an advertisement placed on our site. Figure 2.1 shows an example of how this table is populated.

Table 2.1: Structure of table advertisement

| Field | Type | Null | Default | Comments | MIME |
|---|---|---|---|---|---|
| ***adid*** | int(11) | No | | | |
| shortDisc | text | No | | | |
| category | varchar(15) | No | | | |
| longDisc | longtext | No | | | |
| type | varchar(15) | No | | | |

[3] `http://www.mysql.com/`
[4] `http://www.phpmyadmin.net/home_page/index.php`
[5] `https://www.dropbox.com/`

Table 2.1: Structure of table advertisement (continued)

| Field | Type | Null | Default | Comments | MIME |
|---|---|---|---|---|---|
| price | int(11) | No | 0 | | |
| image | text | No | | | |
| contact | text | No | | | |
| location | text | No | | | |
| day | int(11) | No | 0 | | |

- **user**. Table 2.2 shows the structure of table `user`. Field `userId` is the primary key, field `name` stores the name and surname of a user, field `password` stores the password of a user, field `email` stores the e-mail of a user, field `creditcard` stores the credit card number of a user, field `chargeAmount` stores the amount of money charged in the credit card of a user, field `chargeType` stores information about the type of charging package the user chose, which can be a specified number of ads or a time period, field `chargeInfo` (which is interpreted in conjuction with field `chargeType`) stores information about the charging package the user chose, regarding the number of ads he bought for viewing their confidential information or the time period he bought for free access, and field `datetime` keeps record of the date an time the user registered to the site.

  This table embodies the notion of a user on our site. Figure 2.2 shows an example of how this table is populated.

Table 2.2: Spacetructure of table user

| Field | Type | Null | Default | Comments | MIME |
|---|---|---|---|---|---|
| ***userId*** | int(11) | No | | | |
| name | varchar(32) | No | | | |
| age | int(11) | No | | | |
| username | varchar(32) | No | | | |
| password | varchar(32) | No | | | |
| email | varchar(32) | No | | | |
| creditcard | varchar(32) | No | | | |
| chargeAmount | int(11) | No | | | |
| chargeType | varchar(32) | No | | | |
| chargeInfo | int(11) | No | | | |
| datetime | datetime | No | | | |

- **{user}List**. Table 2.3 shows the structure of table {`user`}`List`. We placed brackets around "user", because such a table can be created for any one of the users, by concatenating the user's `username` with the fixed string "List", ensuring its uniqueness. For example, table `picoList` belongs to user with username `pico`. Field `adid` is an indexed key, which also serves as a foreign key that references field `adid` (primary key) of table `advertisement` and gets cascaded on deletion of the latter key.

  This table embodies the notion of a user's list, which is a functionality that will be explained

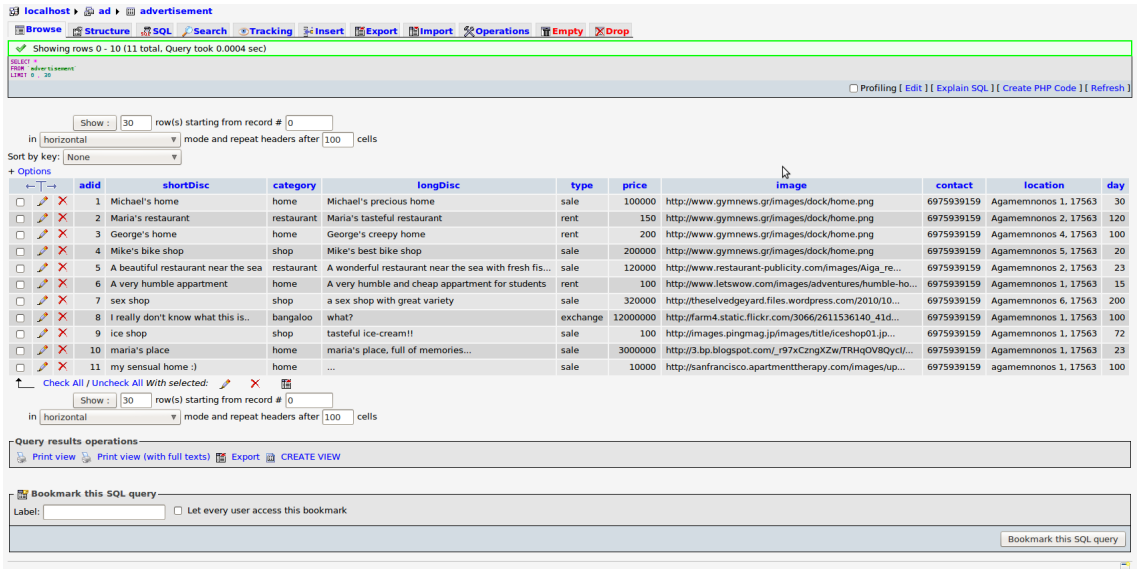


Figure 2.1: A populated example of the `advertisement` table

in a later chapter. Figure 2.3 shows an example of how this table is populated.

Table 2.3: Structure of table {user}List

| Field | Type | Null | Default | Comments | MIME |
|---|---|---|---|---|---|
| *adid* | int(11) | Yes | NULL | | |

- **profiles**. Table 2.4 shows the structure of table `profiles`. Field `profileId` is the primary key, field `username` is an indexed key, which also serves as a foreign key that references field `userId` (primary key) of table `user` and gets cascaded on deletion of the latter key, field `category` stores the type of an advertised item, whether it's about a home or a shop, field `type` stores the use of an advertised item, whether it's for sale or rental, and fields `minPrice` and `maxPrice` store the price range of an advertised item,

  This table embodies the notion of a user's profile(s), which is a functionality that will be explained in a later chapter. Figure 2.4 shows an example of how this table is populated.

  Structure: profiles

Table 2.4: Structure of table profiles

| Field | Type | Null | Default | Comments | MIME |
|---|---|---|---|---|---|
| ***profileId*** | int(11) | No | | | |
| *username* | varchar(32) | No | | | |
| category | varchar(15) | No | | | |
| type | varchar(15) | No | | | |
| minPrice | int(11) | No | | | |
| maxPrice | int(11) | No | | | |

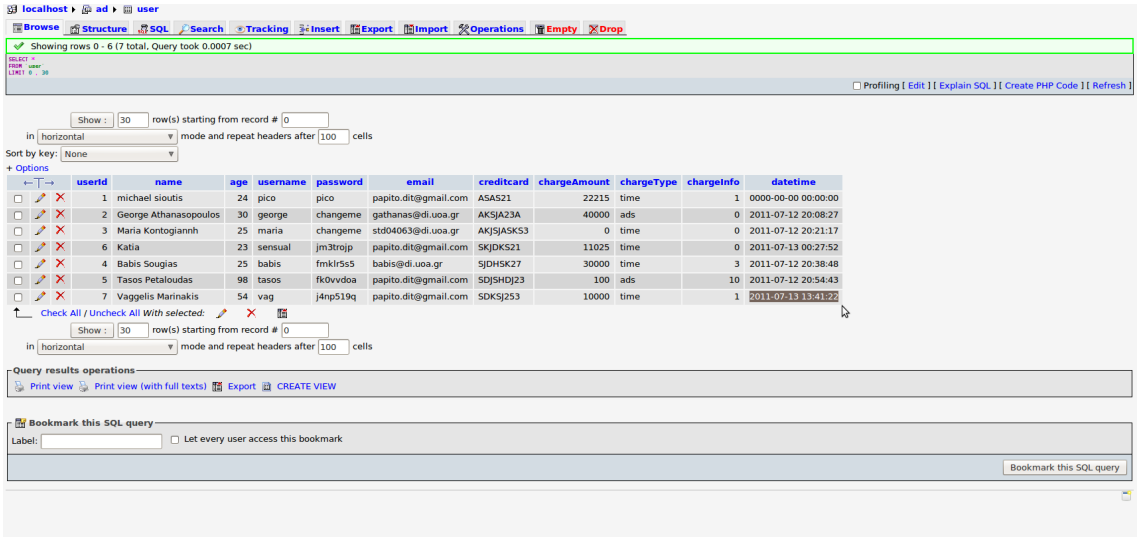


Figure 2.2: A populated example of the `user` table

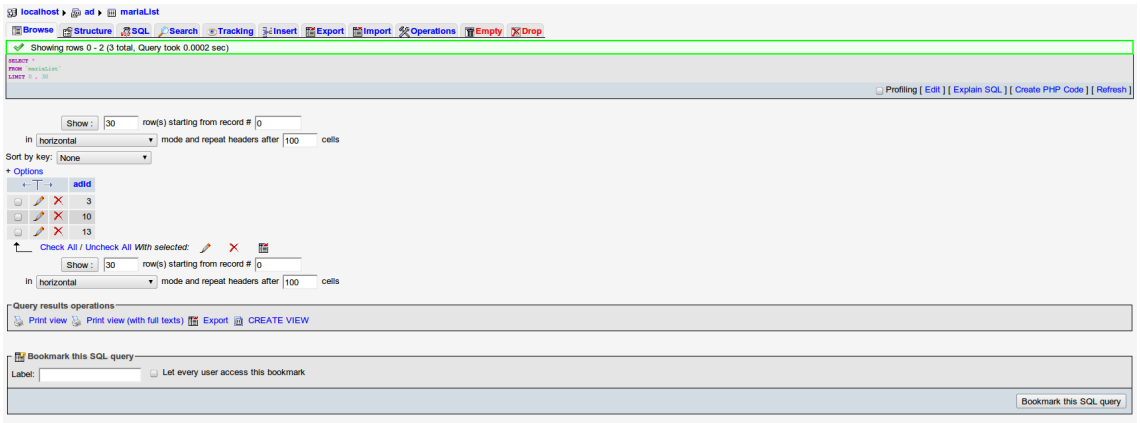


Figure 2.3: A populated example of the {`user`}`List` table

**Event Scheduling.** To lower the values of field `day` of table `advertisement` as the days progress, we scheduled an event that runs every day and executes a set of SQL queries. This event was scheduled with use of `cron`[6], by adding the following entry in a system's cron file:

```
00 00 * * *  mysql -h pic0.dyndns.org -u pico -pmytsalga ad -e \
"update advertisement set day = day - 1; delete from advertisement where day = 0;"
```

At first we considered employing the built-in event scheduler of MySQL, but the cron approach was far too simple to ignore.

[6] `http://en.wikipedia.org/wiki/Cron`

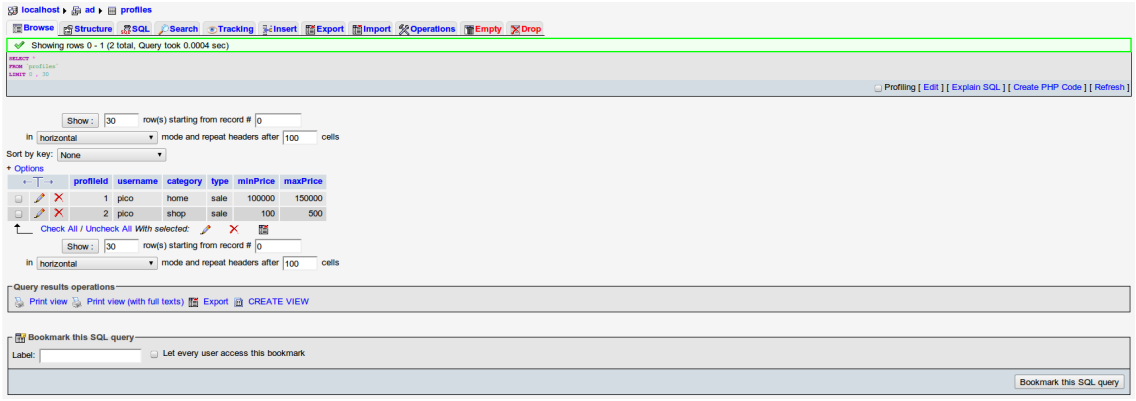


Figure 2.4: A populated example of the `profiles` table

## 2.3 CSS template

A free CSS template was employed from freeCSStemplates.org[7] modified appropriately to fit the purposes of the project. We used text editor `gedit`[8] to edit the CSS template.

## 2.4 Online Repository

The project is available online, hosted in *pic0.dyndns.org* server, through a chrooted[9] SFTP session, for the following acount credentials:

- username: `mosquito`
- password: `brun1s86`

Thus, one can download the project, as shown below:

```
$ sftp mosquito@pic0.dyndns.org
Connecting to pic0.dyndns.org...
mosquito@pic0.dyndns.org's password:
sftp> get ad.tar.bz2
```

[7] `http://www.freecsstemplates.org`
[8] `http://projects.gnome.org/gedit/`
[9] `http://en.wikipedia.org/wiki/Chroot`

# Chapter 3

# Installation Instructions

## 3.1 Required Tools

- Java Developer Kit v6.x
- Netbeans IDE 6.8
- The following glassfish servers:
  - Glassfish v3.0.1
  - Glassfish v2.1.1

## 3.2 Netbeans IDE installation

To successfully install Netbeans IDE you should do the following steps in the specified order:

1. Download and install Java Developer Kit v6.x
2. Download and install Netbeans IDE 6.8
3. Download and install the required Glassfish servers 2.1.1 and 3.0.1

### 3.2.1 Required plugins

To enable Web Services support for Netbeans IDE 6.8, we must supplement it with two required plugins, namely:

- SOA
- XML Schema and WSDL

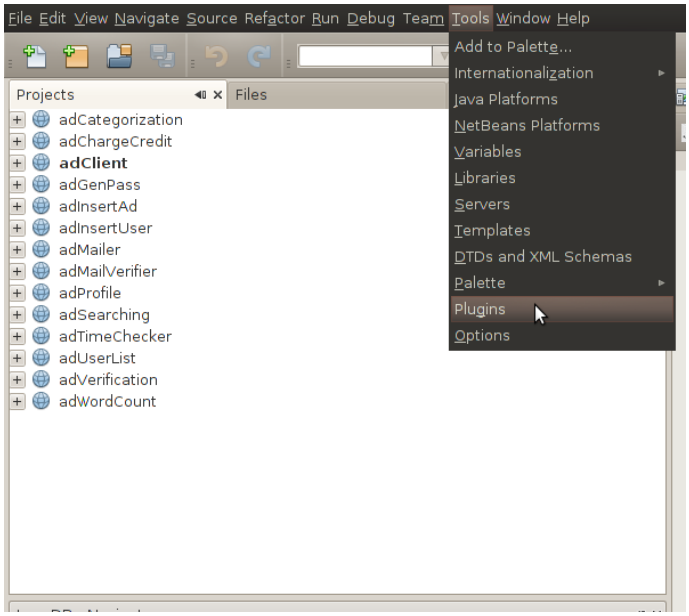


Figure 3.1: Netbeans IDE plugin wizard

To do this, start Netbeans IDE by double clicking the Netbeans IDE application icon. Then go to the `Plugins` menu under the `Tools` tab in the main menu bar, as shown in Figure 3.1. The `Plugins` wizard will open.

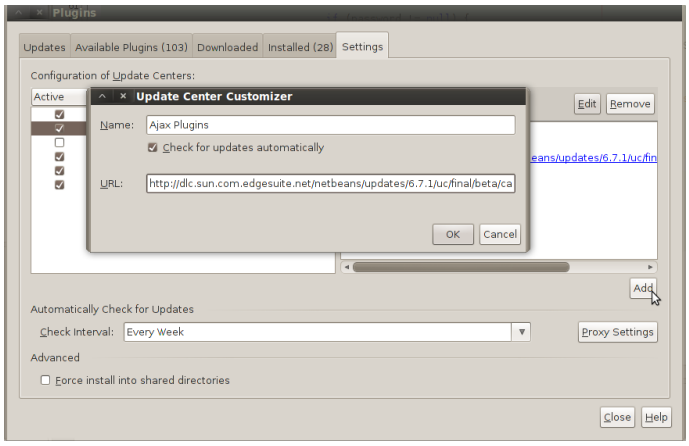


Figure 3.2: Creation of new update center

Go to the `Settings` tab and click on `Add`, as shown in Figure 3.2.Enter `Ajax Plugins` in the `Name` box and `http://dlc.sun.com.edgesuite.net/netbeans/updates/6.7.1/uc/final/beta/` in the `URL` box. Click on `Ok`. We have just created a new update center for Netbeans IDE.

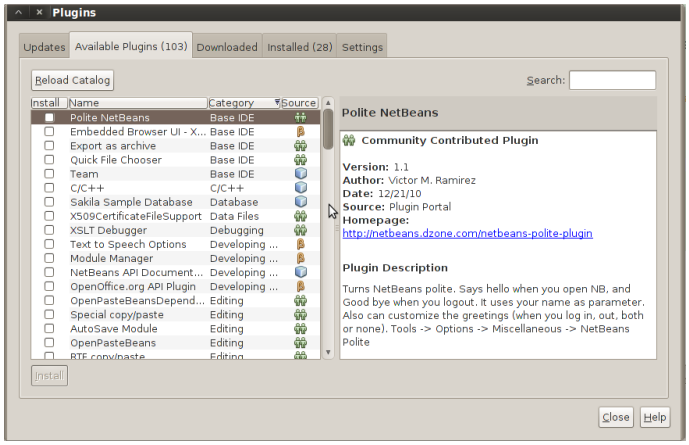


Figure 3.3: List of available plugins

Next, go to the `Available Plugins` tab and click on `Reload Catalog`. This will grab all plugins from our newly created update center, and will present you with a list of all available plugins, as shown in Figure 3.3. Search for the `SOA` and `XML Schema and WSDL` plugins, and check their boxes. Click on `Install`.

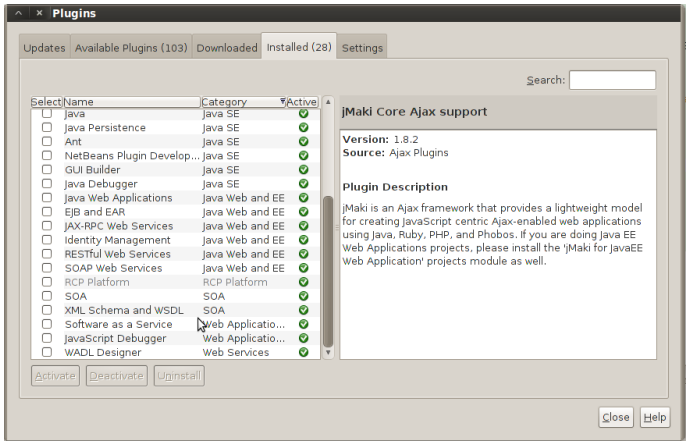


Figure 3.4: List of installed plugins

As a final step, you can verify that the plugins have been installed by going to the `Installed` tab and viewing the `SOA` and `XML Schema and WSDL` plugins, as shown in Figure 3.4.

## 3.3 Installation and adjustment of Glassfish server 2.1.1

To successfully install Glassfish server 2.1.1 you should do the following steps in the specified order:

1. Download and install Glassfish server 2.1.1, following the instructions from here: `http://glassfish.java.net/downloads/v2.1.1-final.html`
2. Register the new server instance in Netbeans IDE 6.8, following the instructions provided in the following link: `http://download.oracle.com/docs/cds/E19879-01.zip`

3. Further instructions regarding adjustments and functionality of the server are provided in the following link: `http://www.oracle.com/technetwork/indexes/documentation/index.html`

The initial settings for the administrator of the Glassfish server are:

- username: `admin`
- password: `adminadmin`

### 3.3.1 Required plugins

To enable Composite Applications support for Glassfish server 2.1.1, we must update it with some packages using a graphical tool that can be started from the command line.

To start the tool, open a terminal and type:

```
$ sh <glassfish installation dir>/updatecenter/bin/updatetool
```

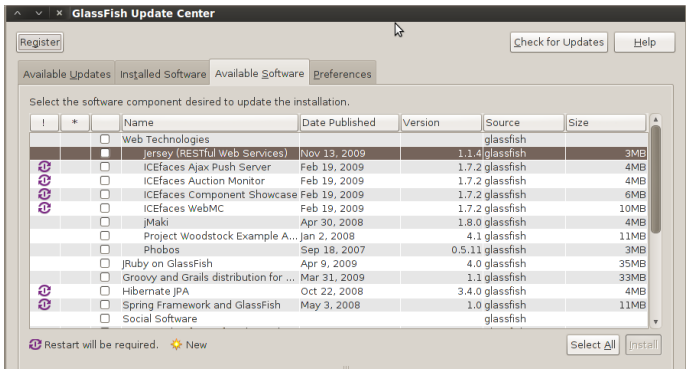


Figure 3.5: List of available software packages

You will be presented with a list of available software packages, as shown in Figure 3.5. Choose the `Composite Applications` package, and click on `Install`.

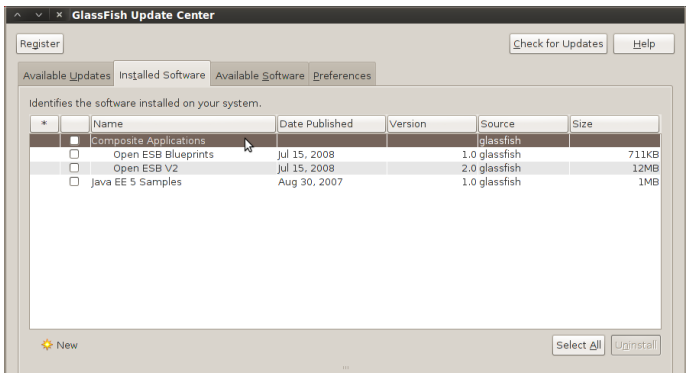


Figure 3.6: List of installed software packages

As a final step, you can verify that the package has been installed by going to the `Installed Software` tab and viewing the `Composite Applications` package, as shown in Figure 3.6.

## 3.4 Deployment of picoAds project

To deploy the project, copy and paste the contents of the `ad` folder to your prefered Netbeans workspace. The `ad` folder essentially contains all Java Web and SOA Composite applications required to run our picoAds implementation.

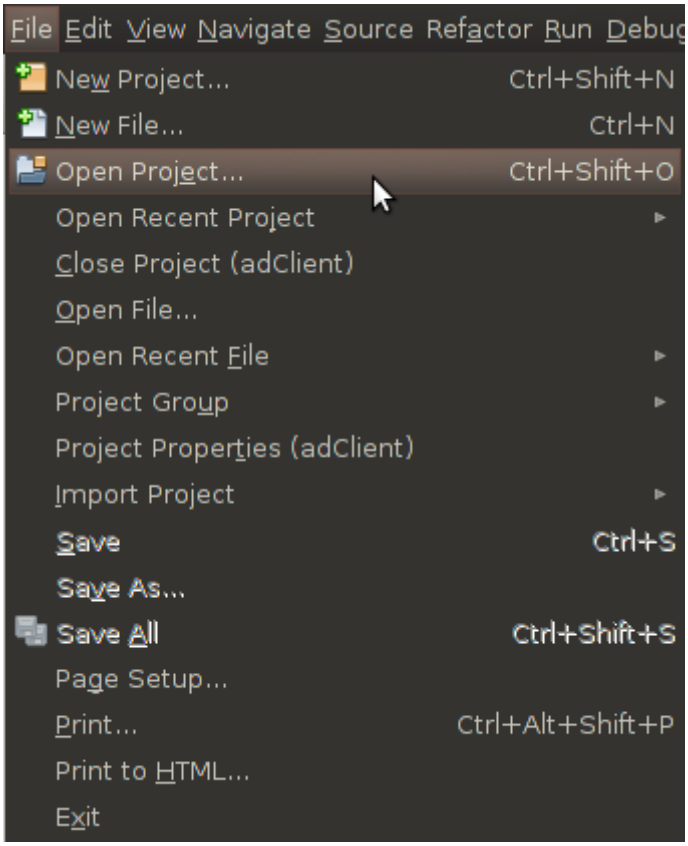


Figure 3.7: Open project wizard

Then, start Netbeans IDE by double clicking the Netbeans IDE application icon. Go to the `Open Project...` menu under the `File` tab in the main menu bar, as shown in Figure 3.7. The `Open Project...` wizard will open.

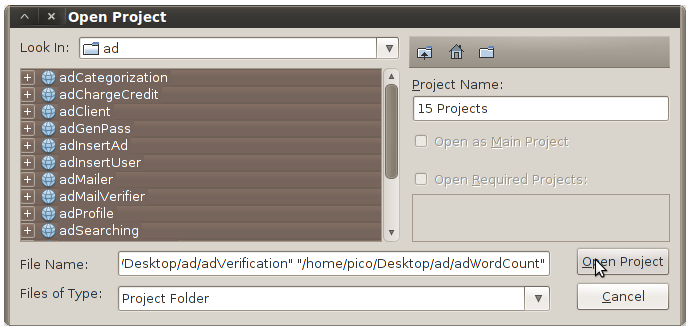


Figure 3.8: Open all applications belonging to the picoAds implementation

Then, select and highlight all applications, and click on `Open Project`, as shown in Figure 3.8. All required projects are now open and shown under the `Projects` tab.

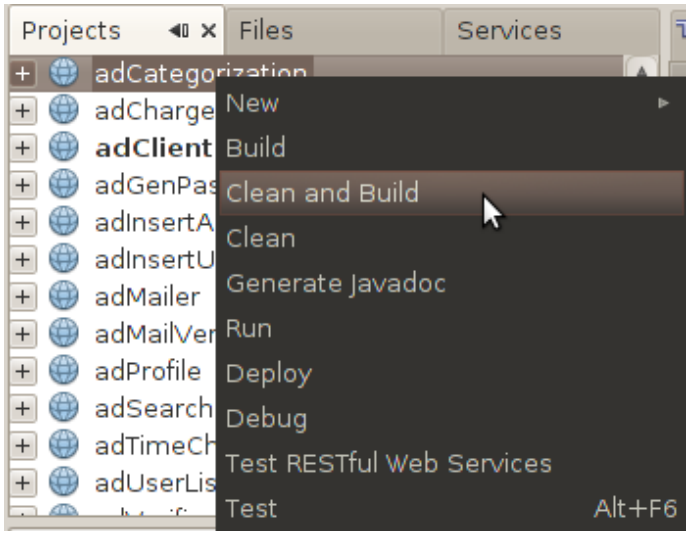


Figure 3.9: Clean and build all applications

Right click upon each one of them and go to the `Clean and build` menu, as shown in Figure 3.9. Select it to clean and build the current project.

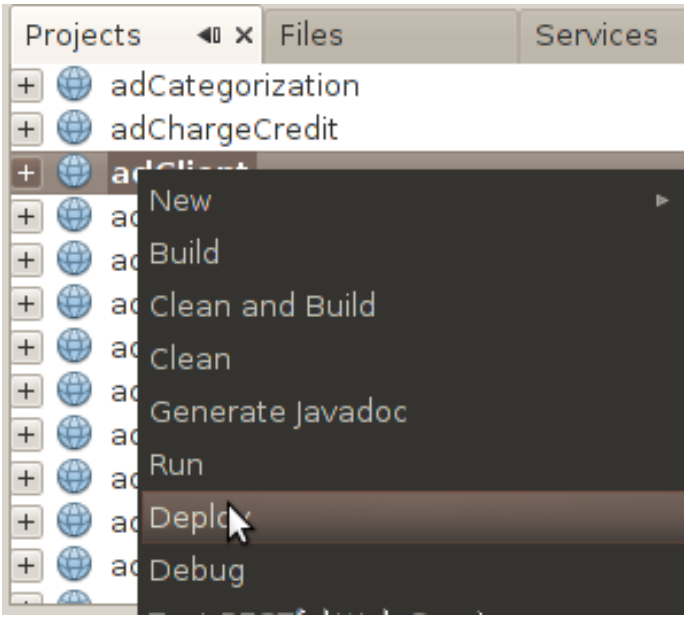


Figure 3.10: Deploy all aplications

Right click upon each one of them and go to the `Deploy` menu, as shown in Figure 3.10. Select it to clean and build the current project. **Important notice:** Leave the `adClient` project last, since it is the single master client project that references deployed services from all others.

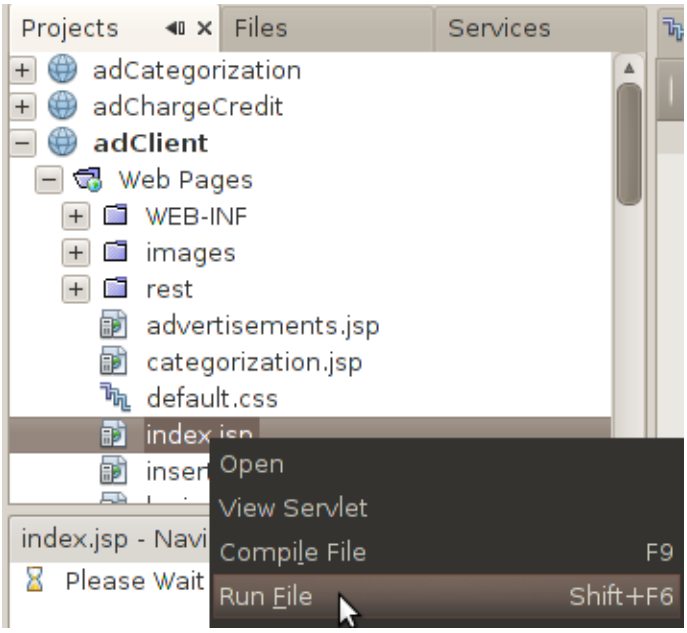


Figure 3.11: Run picoAds

At last, expand the `adClient` project, right click on `index.jsp`, and select the `Run` menu, as shown in Figure 3.11. The default browser for your system will start, and go to the index page of

the `picoAds` website, as shown in Figure 1.1.

If the browser doesn't start by its own, start it manually and go to `http://localhost:8080/adClient/index.jsp`.

### 3.4.1 Browser troubleshooting

You may find that the `picoAds` website doesn't look or function as described in Chapter 4. This can be due to the following reasons, which you should fix:

- Disabled Javascript
- Enabled Adblock protection

## 3.5 Database connection

For the sake of deployment simplicity, the user doesn't need to create any database connections. All services that need access to a database, do so by utilizing JNDI connections. An example of a JNDI connection for a database dependent service, described by a xml file, is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
    <jdbc-connection-pool>
        <property name="serverName" value="pic0.dyndns.org"/>
        <property name="portNumber" value="3306"/>
        <property name="databaseName" value="ad"/>
        <property name="User" value="pico"/>
        <property name="Password" value="mytsalga"/>
        <property name="URL" value="jdbc:mysql://pic0.dyndns.org:3306/ad"/>
        <property name="driverClass" value="com.mysql.jdbc.Driver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" pool-name="mysql_ad_picoPool" jndi-name="myAdDB"
object-type="user"/>
</resources>
```

One can clearly see that the above decription of a JNDI connection guides the service to a database connection with the following options:

- host: `pic0.dyndns.org`
- port: `3306`
- user: `pico`
- password: `mytsalga`
- database: `ad`

Driver `com.mysql.jdbc.Driver` is used, since we are connecting to a remote MySQL database, and `jdbc:mysql://pic0.dyndns.org:3306/ad` is the connection's URL.

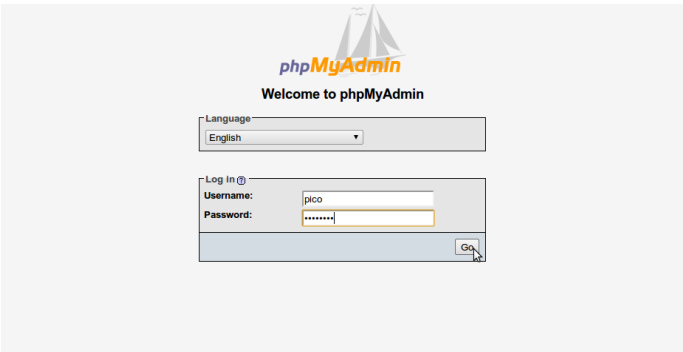


Figure 3.12: phpMyAdmin login page

For performing administrative tasks upon database `ad`, a graphical software tool is provided, phpMyAdmin[1], which can be accessed from the following page: `http://pic0.dyndns.org/phpmyadmin/`. There, you will be presented with the login page, as shown in Figure 3.12. Enter `pico` for the `username` field and `mytsalga` for the `password` field and click on `Go`.

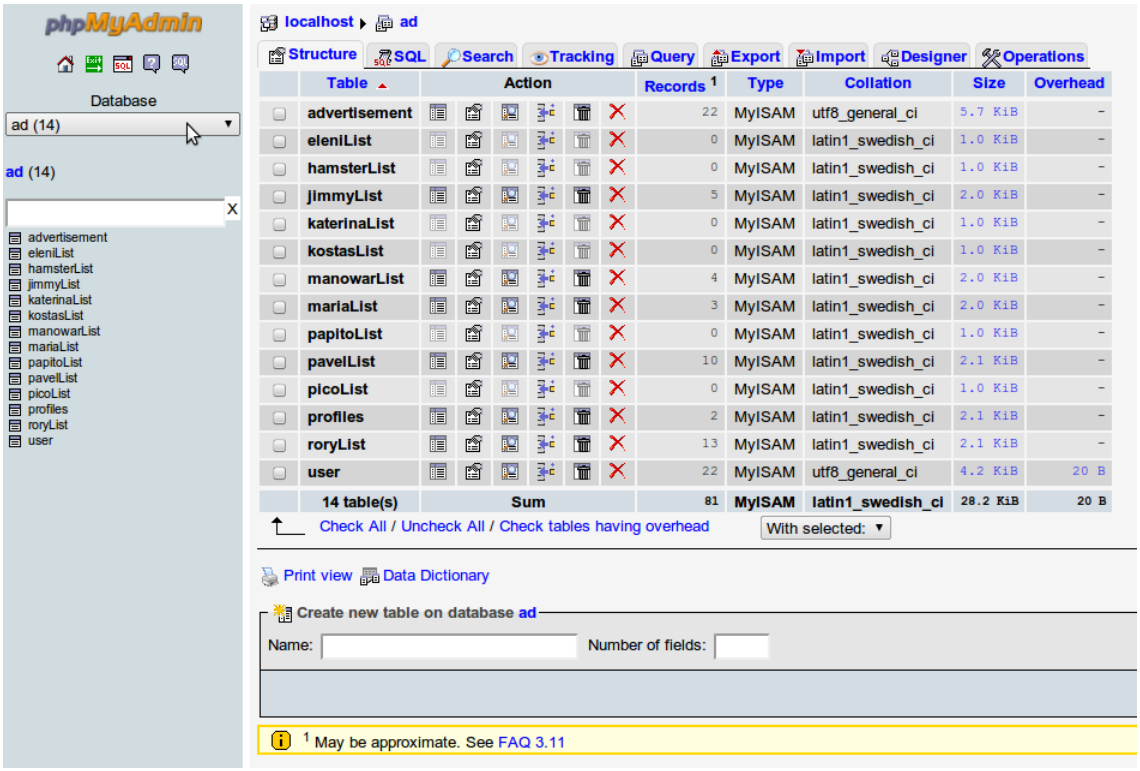


Figure 3.13: phpMyAdmin user interface

Then you will be presented with the phpMyAdmin user interface for database `ad`, as shown in Figure 3.13. You can perform all administrative tasks from that interface. Documentation is provided in the following wiki: `http://wiki.phpmyadmin.net/`

[1] `http://www.phpmyadmin.net/home_page/index.php`

### 3.5.1 Local database installation

Section 3.5 assumes remote database usage for the `picoAds` project. This makes installation of the project to various hosts very easy, since only one central MySQL database server is needed, thus, only one host should serve as a database server machine. Regural dump files of the database are created, in case of a hardware failure.

However, if one wants to run the `picoAds` project entirely and strictly locally, the following additional software is required:

- MySQL Community Server

After the MySQL database server is installed and started, connect to the server as shown below:

```
$ mysql -u root -p
```

Then, create the database `ad`, as shown below:

```
mysql> create database ad;
```

Then, create local user `pico`, with password `mytsalga`, and grant him all privilleges on database `ad`, as shown below:

```
mysql> CREATE USER 'pico'@'localhost' IDENTIFIED BY 'mytsalga';
mysql> GRANT ALL PRIVILEGES ON `ad`.* TO 'pico'@'localhost';
```

Then, quit the server, as shown below:

```
mysql> \q
```

Then, grab the latest dump file from database `ad` (a dump file named `ad.sql` is provided in the project's root folder), and restore the `ad` database, as shown below:

```
$ mysqldump -u pico -pmytsalga ad < ad.sql
```

Verify that you can run the following set of commands:

```
$ mysql -u pico -pmytsalga ad
mysql> show tables;
+---------------+
| Tables_in_ad  |
+---------------+
| advertisement |
| user1List     |
| user2List     |
| user3List     |
| ...           |
| profiles      |
| user          |
+---------------+
X rows in set (0.00 sec)
```

Then, edit the `/etc/hosts` file by changing the $1^{st}$ entry from:

```
127.0.0.1 localhost
```

to:

```
127.0.0.1 localhost  pic0.dyndns.org
```

As a final touch, a cron job needs to be created, that lowers the values of field `day` of table `advertisement` as the days progress. This can be done by invoking the `crontab` program, as shown below:

```
$ crontab -e
00 00 * * *  mysql -h pic0.dyndns.org -u pico -pmytsalga ad -e \
"update advertisement set day = day - 1; delete from advertisement where day = 0;"
^X
crontab -l
```

By now, you should not have any problem deploying and running the `picoAds` project entirely and strictly locally ☺

# Chapter 4

# Functionality

In this chapter we show the functionality of the picoAds project, by describing the web services implemented, and by presenting a very illustrative scenario that makes use of the technologies developed.

## 4.1 Web Services

The following services were implemented:

- **adCategorization.** This RESTful service implements functionality concerning the detailed categorization and presentation of advertisements on picoAds.
- **adChargeCredit.** This Web service implements functionality concerning the user's credit card charging in picoAds.
- **adGenPass.** This Web service implements functionality concerning the generation of a strong 8-character long alpharithmetic password. It imports the `java.security.SecureRandom` package to achieve this.
- **adInsertAd.** This Web service implements functionality concerning the addition of a new advertisement on picoAds.
- **adInsertUser.** This Web service implements functionality concerning the registration of a new user on picoAds. The web service also handles the case when a user is trying to register with an allready used username.
- **adMailer.** This Web service implements functionality concerning the delivery of mails to users in various ocsasions (e.g.,registration of the user, placement of ad by the user). It imports the `javax.mail.*` package, and uses a smarthost to deliver the e-mail. For the sake of this project we used `mailhost.di.uoa.gr` as a smarthost. This means that one can send an e-mail only when connected to the UoA Department of Informatics network. The smarthost should be changed appropriately in any other case.
- **adMailVerifier.** This Web service implements functionality concerning the verification of a users e-mail. This is done by importing the `java.util.regex.Pattern` package, and

validating the users e-mail against the *RFC 2822* internet message format, which is described by a regural expression.

- **adSearching.** This Web service implements functionality concerning ad search by users. It is responsible for offering the user a variety of search forms, to select and combine different search criterias.
- **adTimeChecker.** This Web service implements functionality concerning the validation of a user's registration date and time against the current system's date and time, in case he had opted for the time period registration package. It imports the `java.sql.Timestamp` package, and uses an *if-else* structure to decide upon the validity of a user's timestamp.
- **adUserList.** This Web service implements functionality concerning the user list feature in picoAds. It creates a list for a user upon his request, and keeps record of the advertisements he visits and has full access to.
- **adVerification.** This Web service implements functionality concerning the verification of the user's input data when placing a new ad on picoAds. It simply checks whether all required information is provided.
- **adVerification.** This Web service implements functionality concerning the counting of words for some input text.
- **adProfile.** This RESTful service implements functionality concerning the profiles of users in picoAds.
- **AdInsertion.** This BPEL process orchestrates services responsible for the insertion of a new ad.
- **AdInsertionSOA.** This SOA composite application deploys the *AdInsertion* service.
- **UserRegistration.** This BPEL process orchestrates services responsible for the registration of a new user.
- **UserRegistrationSOA.** This SOA composite application deploys the *UserRegistration* service.

## 4.2 Scenario

We will proceed by presenting a scenario that goes through all the functionality of picoAds, accompanied by explanatory figures.

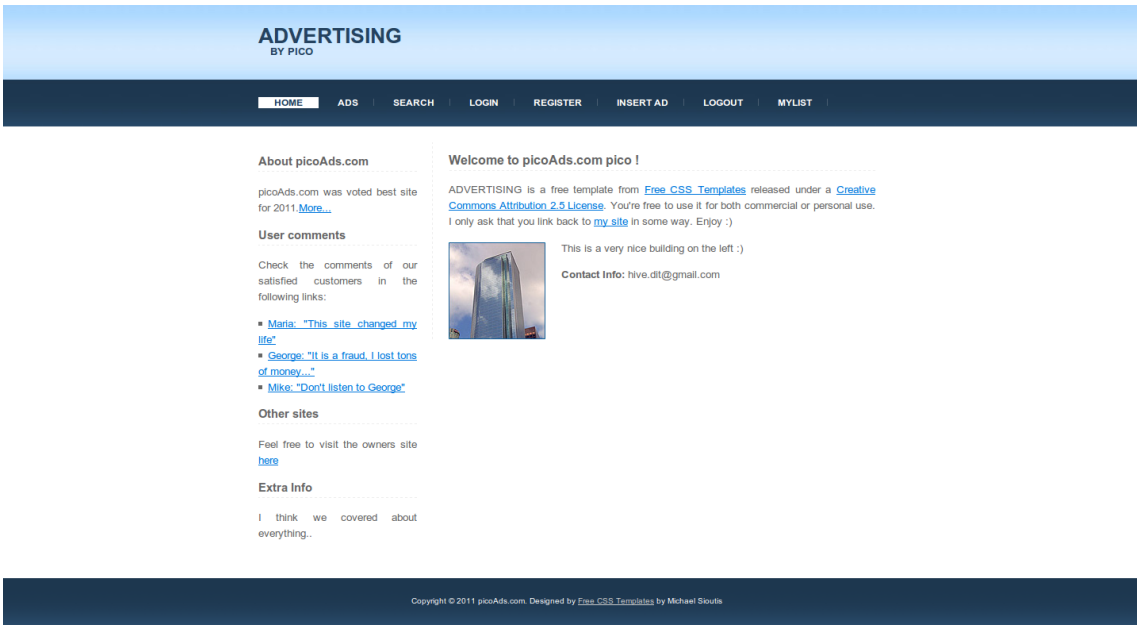


Figure 4.1: Home page

After having deployed picoAds following the installation instructions in Chapter 3, we open an internet browser and go to `http://localhost:8080/adClient`. We are presented with the website's home page as shown in Figure 4.1. In the home page we can find information about the site, as well as contact information.

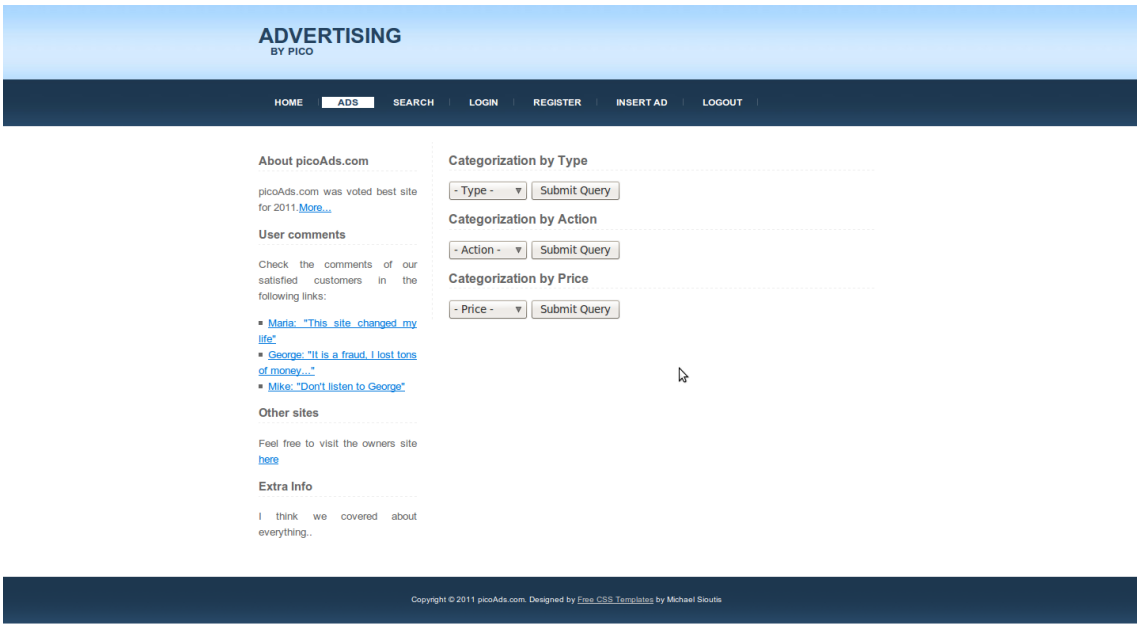


Figure 4.2: Categorization functionality

We hit the *ADS* button in the menu bar and we are presented with a select form and a submit button. The first level of categorization categorizes the ads by their type, meaning you can select *home*, *restaurant*, *shop*, or *all*, and then hit the submit button for your request to be granted. As soon as you do that, a second level of categorization appears that further categorizes the ads by the action they are offered for. Now you can choose whether you are interested in *sale* or *rental* ads. If you can't make up your mind, just select *all*. A last level of categorization appears, requesting the user's selection of cost. Again there are multiple options to choose from, specifically, *cheap*, *expensive*, and *all*. At this point the categorization should look like Figure 4.2.

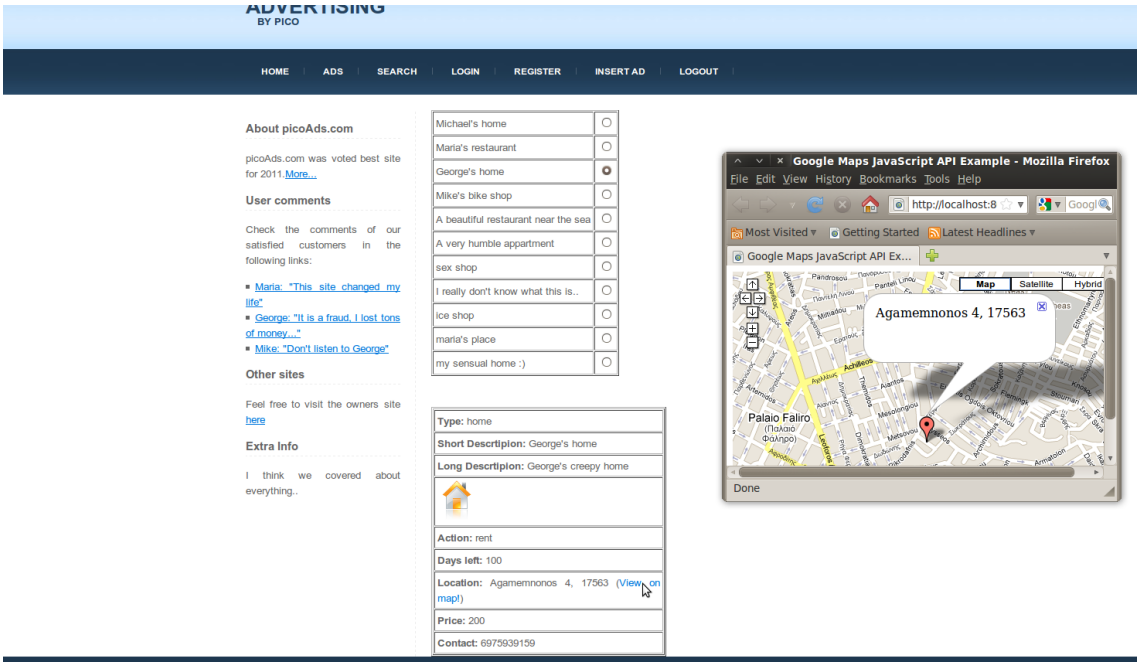


Figure 4.3: Results of categorization

As you might have guessed, the categorization we offer is capable of $4 \times 3 \times 3 = 36$ combinations, allowing the user to browse a variety of different ad categorizations. When you select an option and hit the last submit button, you are forwarded to the results page, as shown in Figure 4.3. There you are presented with a list of ads and a short description for each one of them. You can choose to navigate through the ads by clicking upon them and viewing all their information. Furthermore, if you click on the "View on map!" url, a window opens that shows the advertised item's location on a google map.

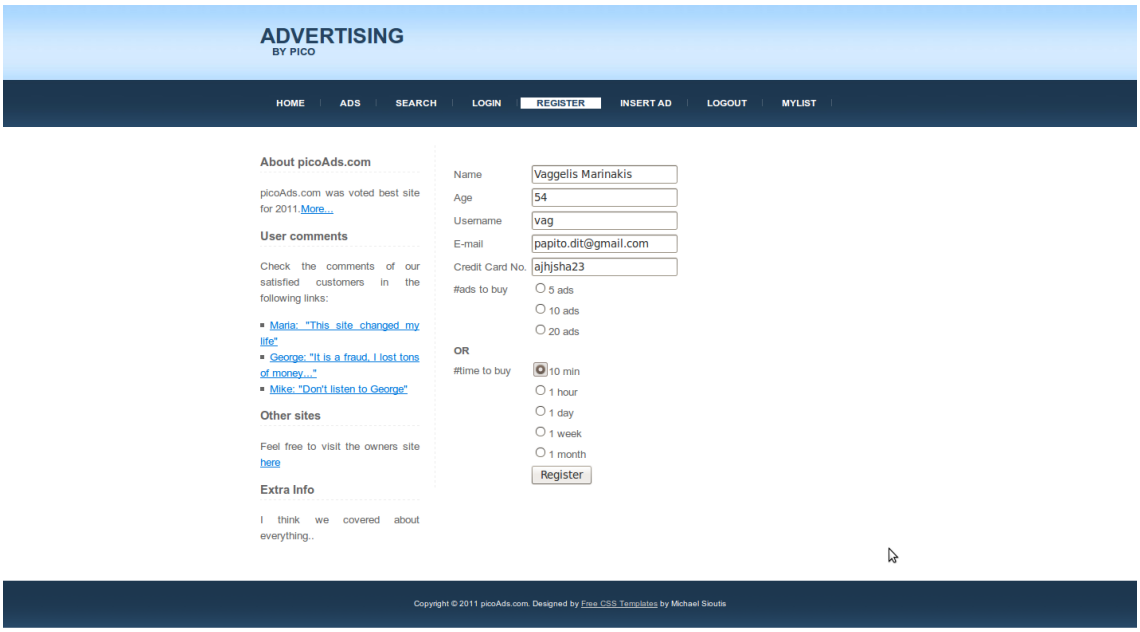


Figure 4.4: Registration functionality

Now it's time to create a new account. We hit the *REGISTER* button and we are presented with a registration form. After filling in the required information the form should look like Figure 4.4. At this point you can also select the charging package you want to have when logging into the site. You can either buy free access to confidential information of 5,10 or 20 ads, or a free access time period of 10 minutes, 1 hour, 1 day, 1 week, or 1 month.

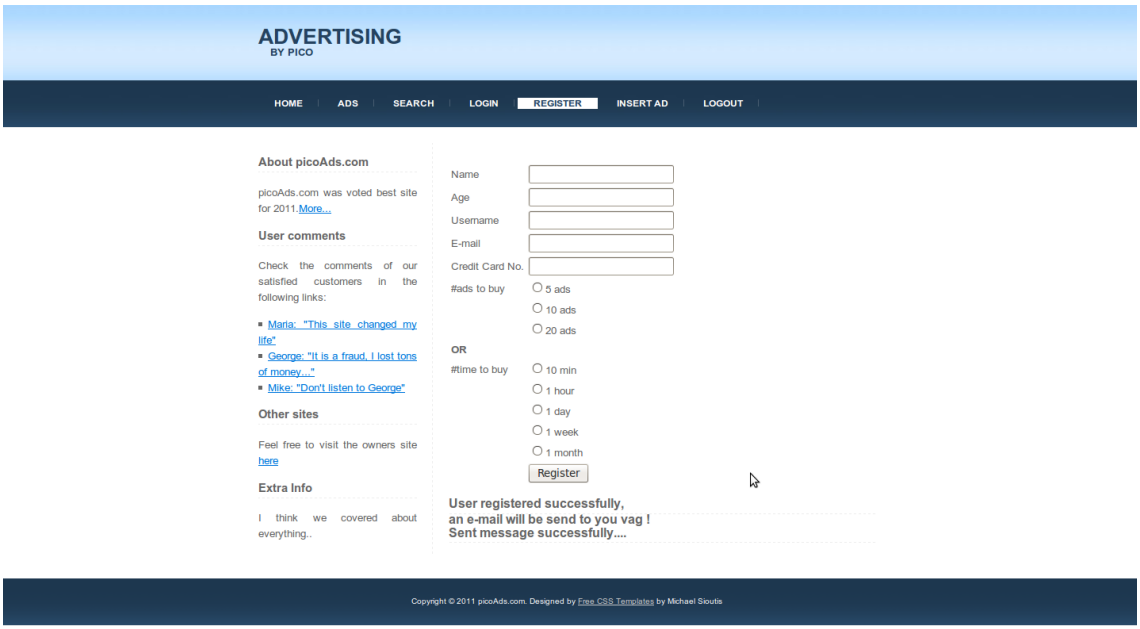


Figure 4.5: Results of registration

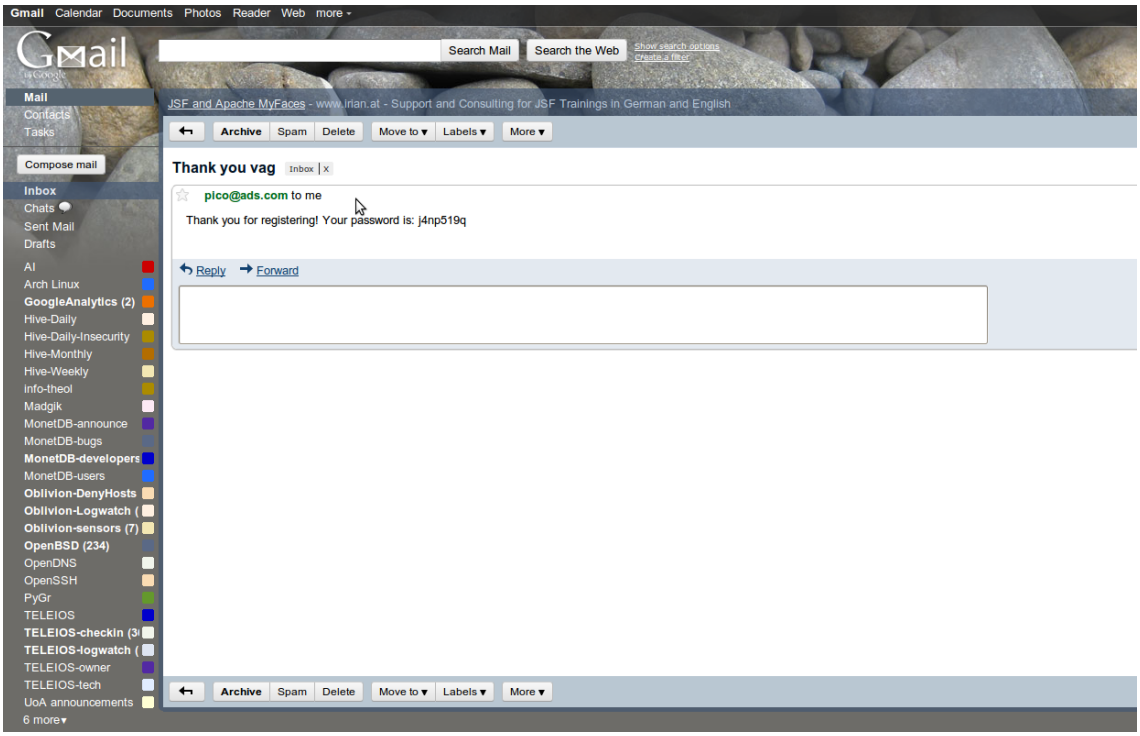


Figure 4.6: Mail sent upon successful registration

After you hit the submit button, information about your registration success (or failure) will be presented to you under the form as shown in Figure 4.5. Registration failure can happen when the user enters an invalid e-mail account, or when he is trying to register with an allready used username. Provided the registration was successful an e-mail will be sent to you at the e-mail adress you provided as shown in Figure 4.6. The message will have a welcoming title, and will contain the password in the message body.

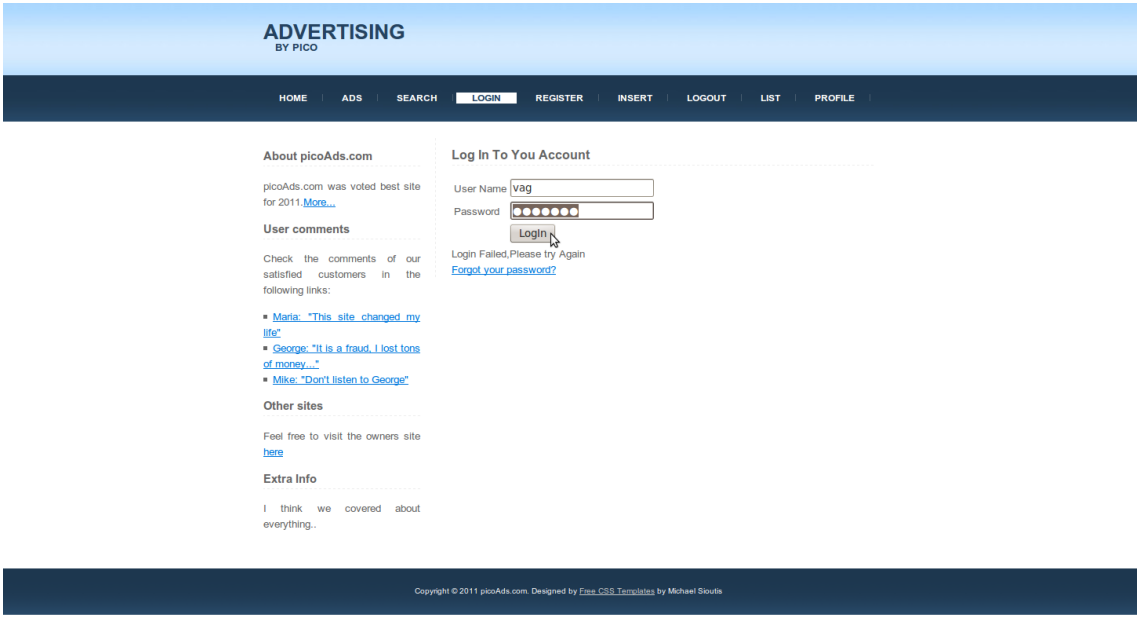


Figure 4.7: Login page

Since we registered, it's time to login to the site. We hit the *LOGIN* button in the menu bar and we are presented with the login page that asks us for a username and a password as shown in Figure 4.7. If you give a wrong username or password the message "Login failed, please try again" will appear under the form, and the link "Forgot your password?" will appear right below the previous message, which offers password recovery functionality. Also, the password appears on the screen in the form of bullets, for security reasons. If the username and password you entered are correct, you will be redirected to the home page as soon as you hit the submit button.

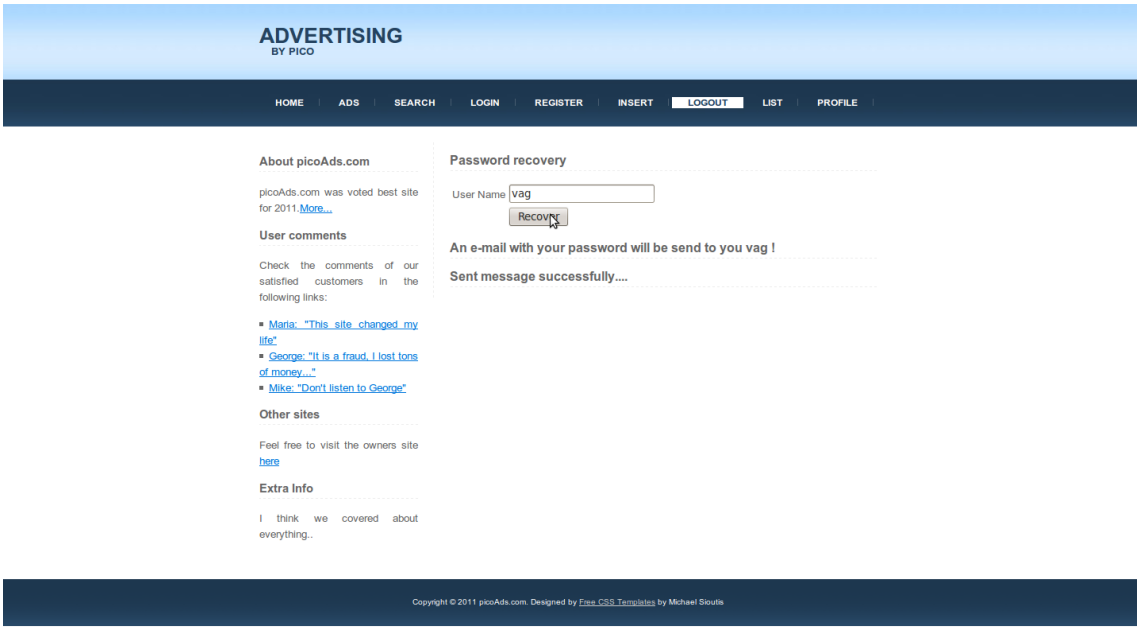


Figure 4.8: Password recovery page

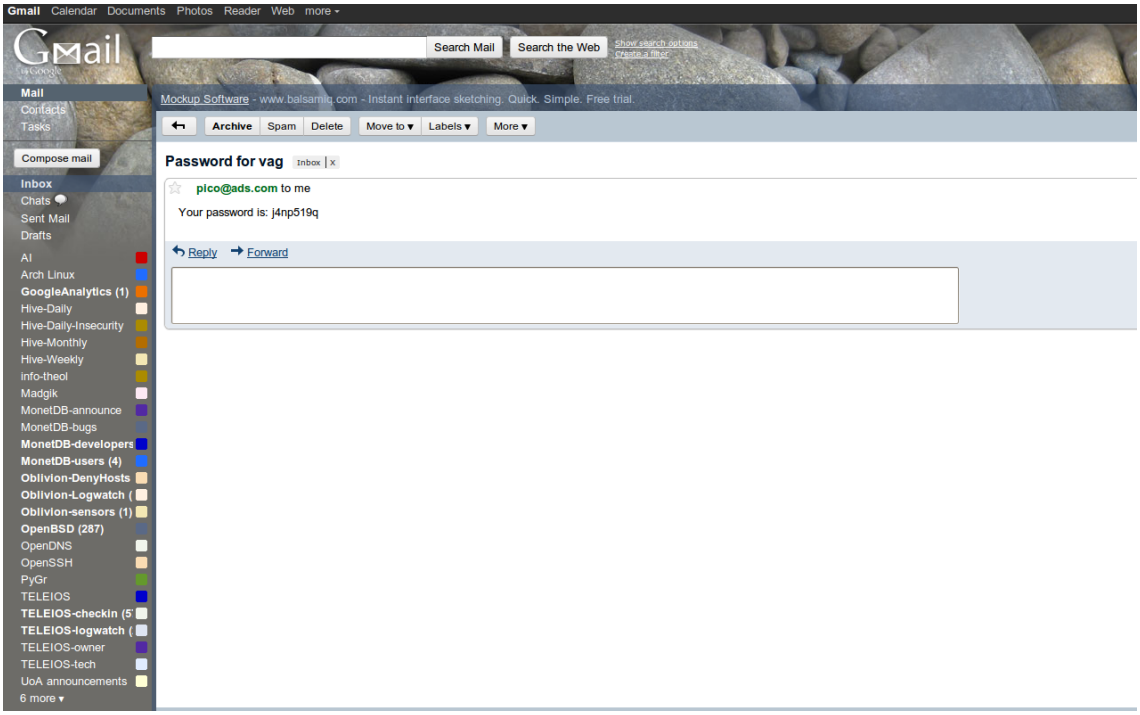


Figure 4.9: Mail sent upon successful password recovery

Otherwise, if the password you entered is incorrect, because you cannot recall it, you may choose to recover it by pressing the "Forgot your password?" link. After you press the link, you are redirected to a password recovery page as shown in Figure 4.8. There you are only required to enter your username. If the username you provide is incorrect, the message "No user `username` exists!" will appear, otherwise a mail will be sent to the user's e-mail with the recovered password as shown in Figure 4.9, and appropriate messages will appear under the form.

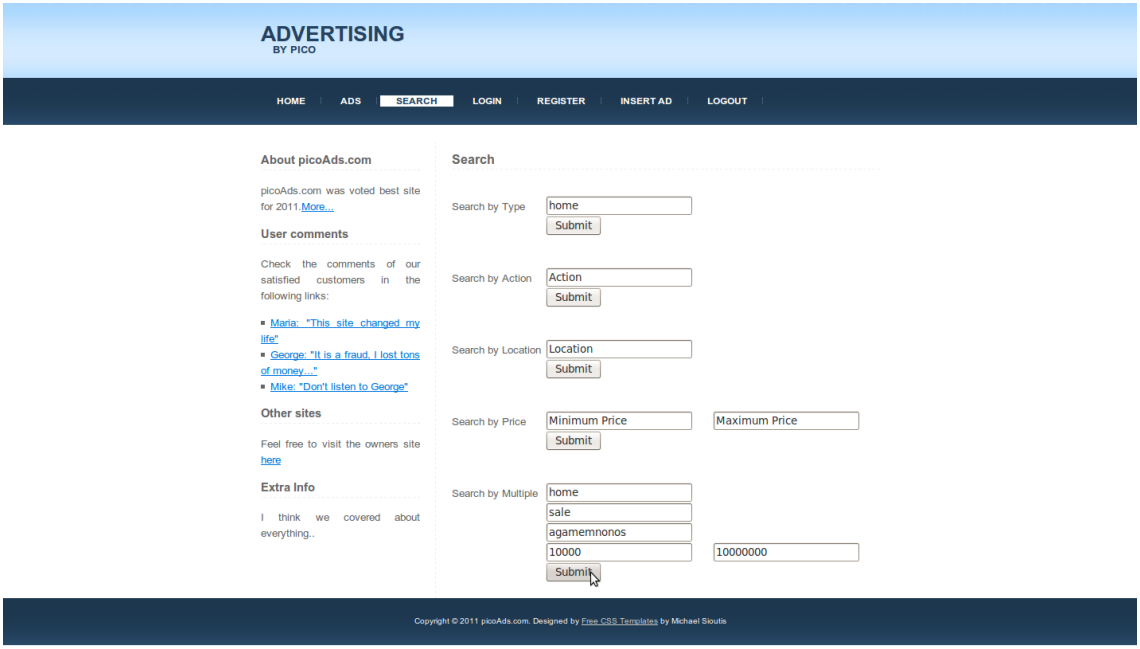


Figure 4.10: Search functionality

We go on with the search functionality of picoAds. By hitting the *SEARCH* button in the menu bar you are presented with multiple search forms as shown in Figure 4.10. There you can search for ads either by their type, the action they are ment for, their price range, thei location, or a combination of the previous criterias. For the sake of the example we will search for a home, that is for sale,has a price range from 10 thousand euros to 10 million euros, and is located in an area that is called something like "agamemnonos". We have filled all available option, and now we hit the submit button.

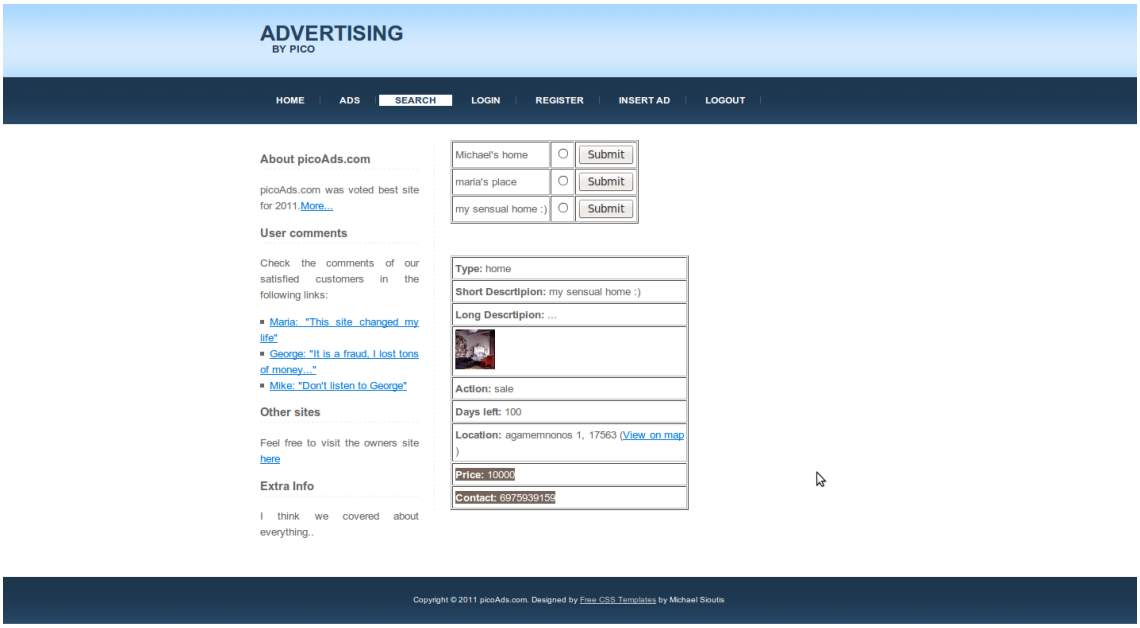


Figure 4.11: Search results

We are then forwarded to a results page. The results of our search are shown in Figure 4.11. There you are presented with a list of ads and a short description for each one of them. You can choose to navigate through the ads by clicking upon them, hitting the submit button and viewing all their information. Furthermore, if you click on the "View on map!" url, a window opens that shows the advertised item's location on a google map. Since you are now logged in and have acquired a valid registration package, you can view even confidential information of ads as shown in Figure 4.11 where the highlighted text is.

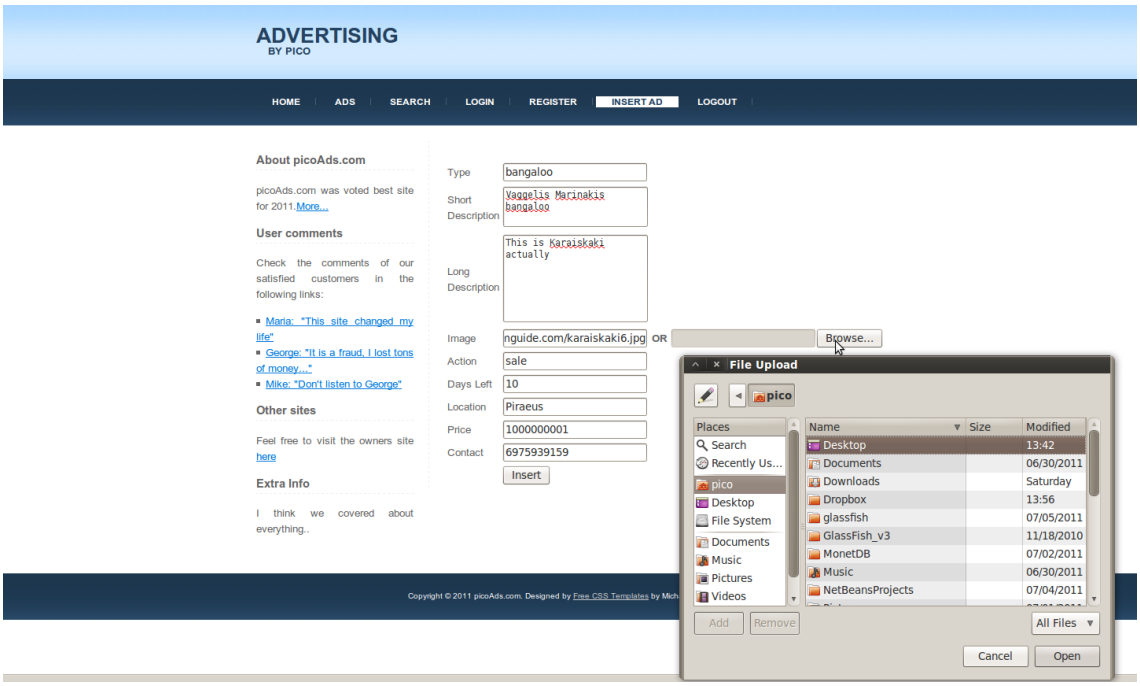


Figure 4.12: Ad insertion form

It's time we inserted an ad on picoAds. You can do this by hitting the *INSERT AD* button. You are then presented with a form where you can fill in all details. After you fill in the details of the new ad you want to place on picoAds, the form will look like the one shown in Figure 4.12. For the image box of the form you can opt for either a url path to the image or an image from tour filesytem. It must be mentioned though, that for the latter case there is currently no implemented functionality that would store the image on the server that hosts picoAds.

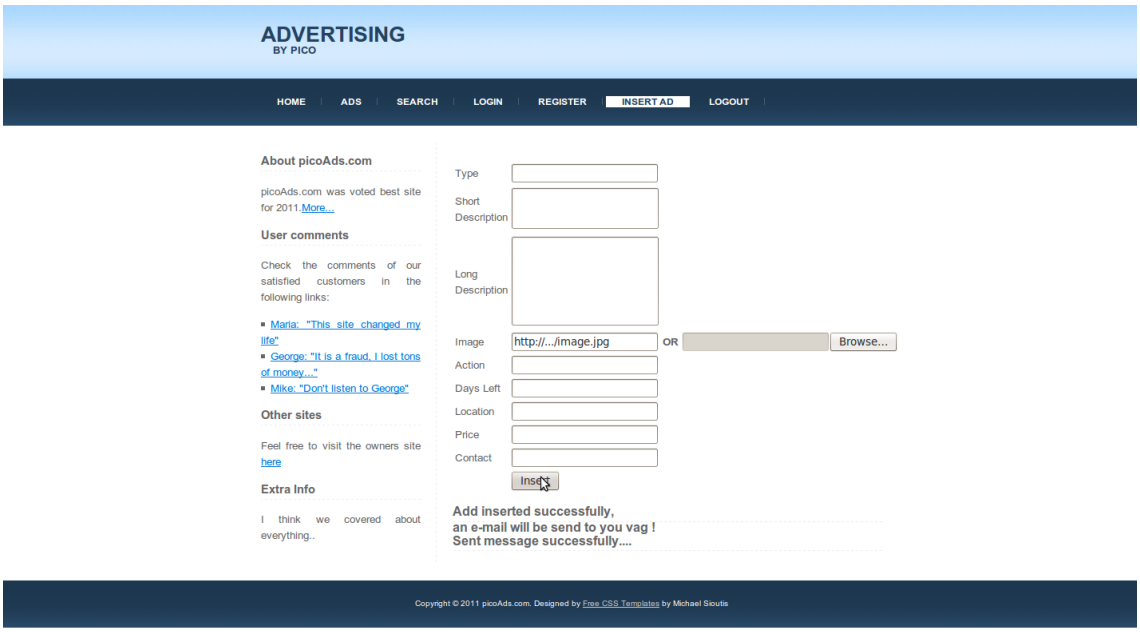


Figure 4.13: Ad insertion results

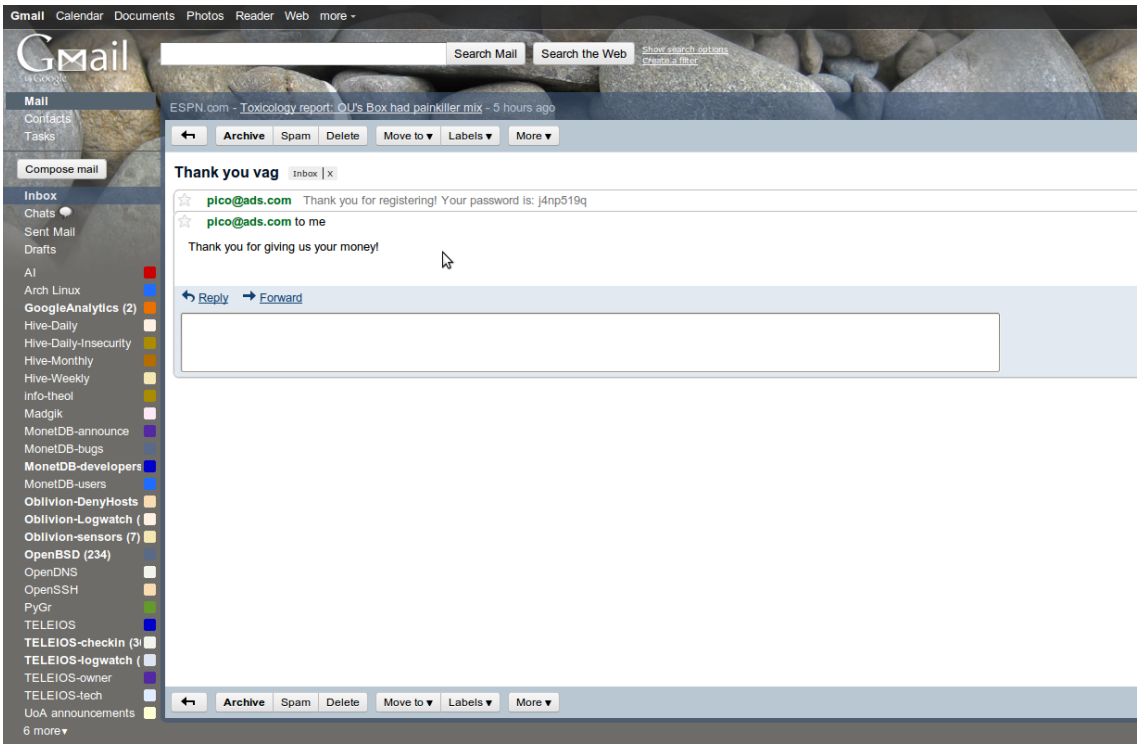


Figure 4.14: Mail sent upon successful insertion of ad

After you hit the submit button, an operation takes place that checks if all information needed for the ad was provided. Assuming all the needed information was provided you will get a small message about your insertion success under the form as shown in Figure 4.13. Upon failure the message would prompt you to try again. An e-mail will alse be sent to you at the e-mail adress you provided when you registered as shown in Figure 4.14. The message will have a thankful title, and will verify the user's ad insertion in the message body.

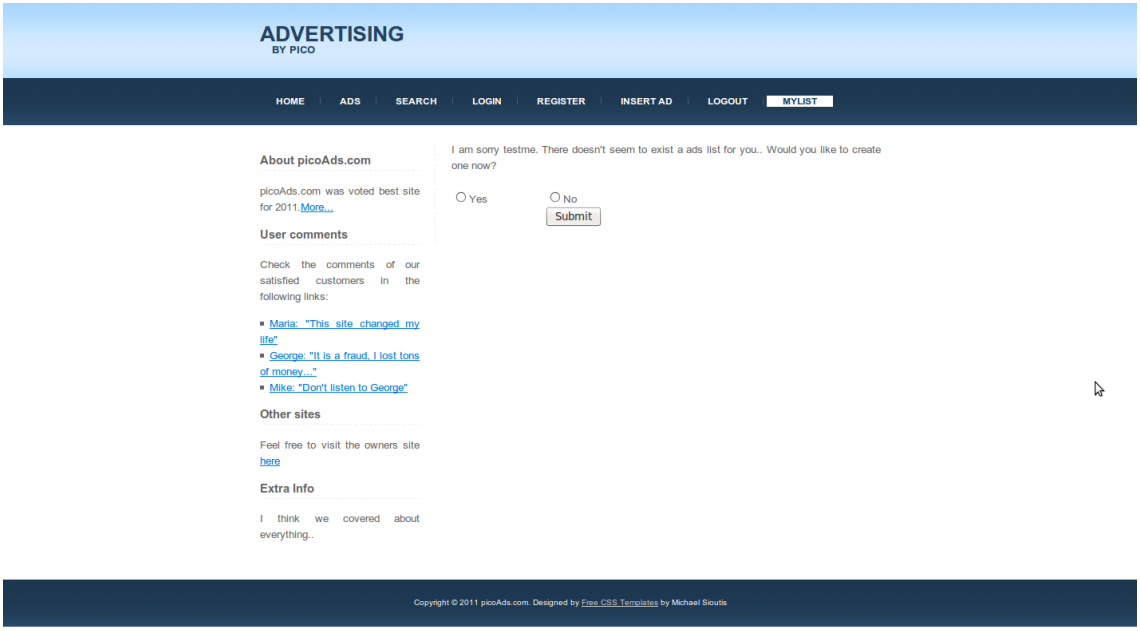


Figure 4.15: User list functionality

We continue with a powerful feature of picoAds. The user list feature. When hitting the *MYLIST* button in the menu bar an operation takes place that checks whether you allready have a list. Let's assume that you don't have a list, which is natural since you registered only minutes ago. You will be presented with a question whether you want or not to create a list for your ads as shown in Figure 4.15. If you select "yes" and hit the submit button a unique list willl be created for you where all your viewed ads, you had access to confidential information, will be stored. If you select "no" and hit the submit button, a message appears that informs you about your action and no list is created.

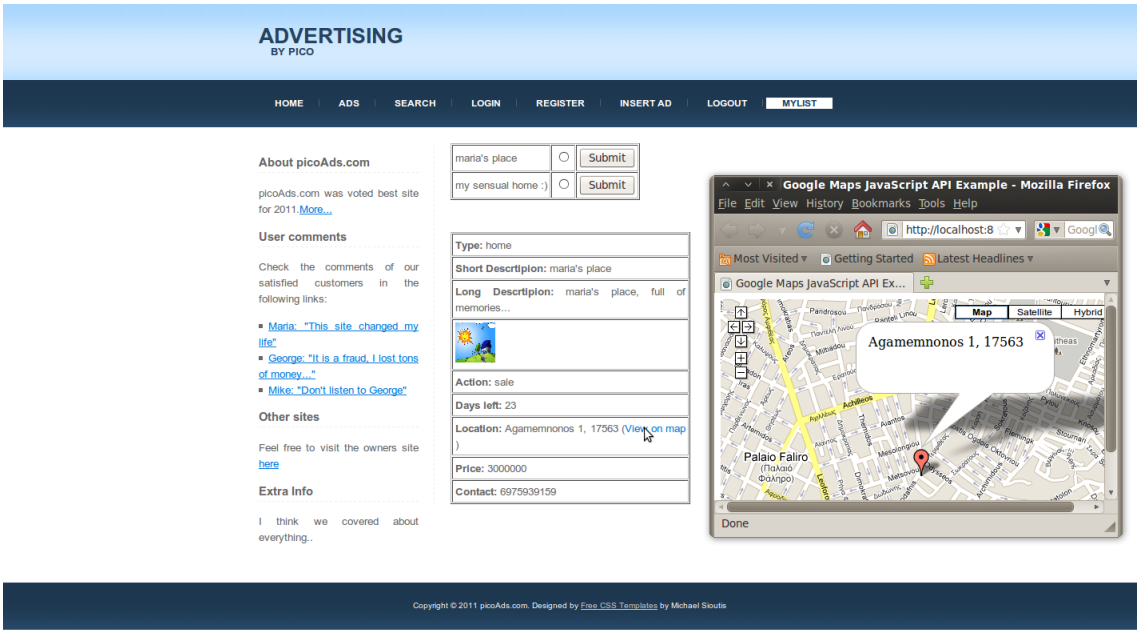


Figure 4.16: Creation of a user list

The next time you visit the user list page and since you created an ad list, the question will be replaced by a list of ads and a short description for each one of them as shown in Figure 4.17. You can choose to navigate through the ads by clicking upon them, hitting the submit button and viewing all their information. Furthermore, if you click on the "View on map!" url, a window opens that shows the advertised item's location on a google map.

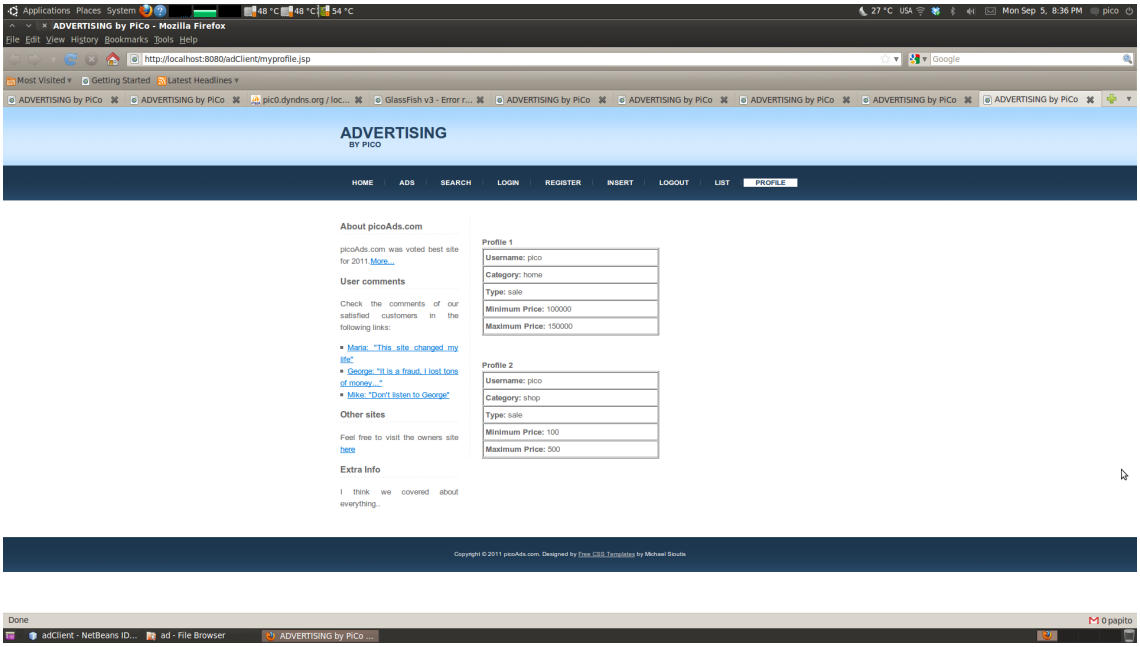


Figure 4.17: Profile of a user list

Profile of a user list can be seen when hitting the *PROFILE* button.

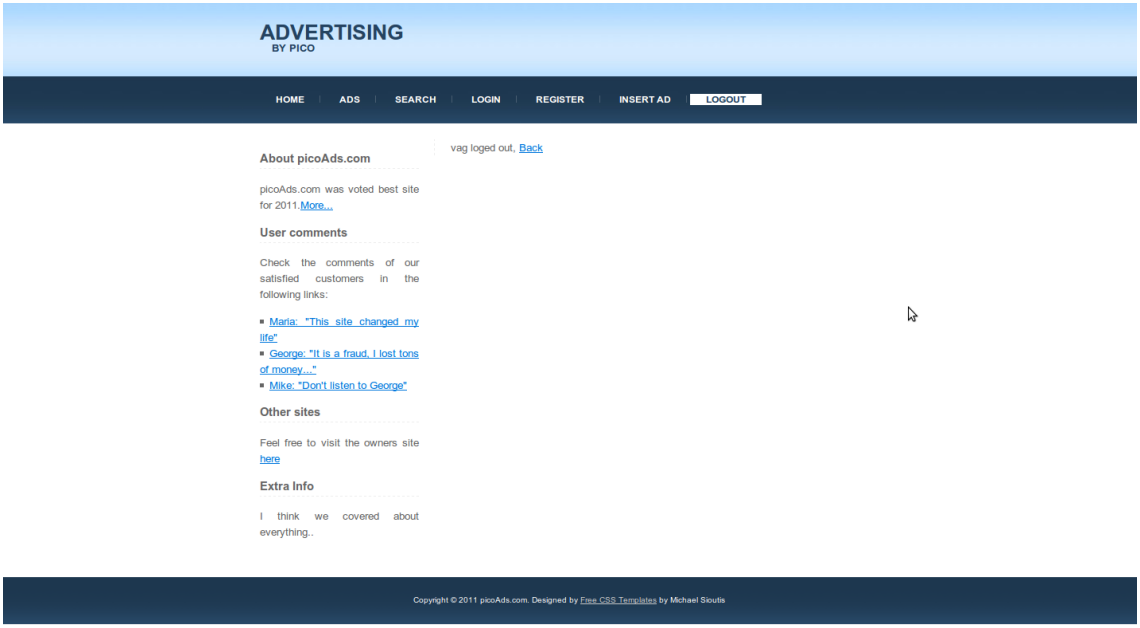


Figure 4.18: Logout page

Finally, after having successfully completed the scenario we hit the *LOGOUT* button in the menu bar and we are presented with the logout page which contains a message that informs the

user he has been looged out and a link to the home page of picoAds as shown in Figure 4.18. That's it! ☺

# Chapter 5

# Postrequisites

- When a user buys a number of advertisements to view upon registration, he essentially buys a number of total viewings of advertisements that may not differ from each other. For example, if the user has bought a viewing package of five advertisements, he can redeem that package, either by viewing five different advertisements, or by viewing a single advertisement five different times.
- When someone registers to become a registered user, we check if the same username exists in the database, and if the e-mail he provides is valid. If one of the above conditions don't hold, the registration results in failure. We *deliberately* don't check if the e-mail he provides exists allready in the database, because throughout the project we needed to create multiple users to check our registration component (and other components), and maintaining different e-mail accounts for each user would be a very hard task.
- The default password that is created for every user registration is very strong, but it is stored unencrypted in our database.

# Chapter 6

# Useful links

- http://java.sun.com/developer/onlineTraining/tools/netbeans_part1/
- http://netbeans.org/kb/docs/websvc/rest.html
- http://netbeans.org/kb/docs/websvc/jax-ws.html
- http://soa.netbeans.org/soa/
- http://weblogs.java.net/blog/kalali/archive/2007/05/a_scenario_base.html
- http://cgi.di.uoa.gr/ sioutis/