MATRIX

ONLINE MATHEMATICS STUDYING PLATFORM

Matrix is a web platform aimed to help students achieve better result during high school by centralising a base of knowledge in the field of mathematics. The service can also be referred as a tool for students, providing powerful mechanisms for handling high school related mathematics problems.

5/13/2013

CONTENTS

1. INTRODUCTION

Matrix is a web platform aimed to help students achieve better result during high school by centralising a base of knowledge in the field of mathematics. The service can also be referred as a tool for students, providing powerful mechanisms for handling high school related mathematics problems. These mechanisms include but are not limited to[1]: online working on mathematics problems in workspaces, online storage which can be accessed with save and load actions, powerful mathematics library and easy-to-use Mathcad-like user experience, forum and discussions section where students can post mathematics problems and get relevant advice and an automated system for suggesting solutions based on previous user knowledge.

We chose to build a web platform rather than a desktop application because this tool must be widely accessible and not depend on the operating system or computer capabilities of the user. We aim to create a centralized knowledge base, self-sustained and continuously growing, and to keep it accessible for all the students that may need it.

Matrix is designed in two sections: the Workspace and the Forum. In the Workspace, the user can perform mathematical task and calculus, while in the Forum they can access the community and the knowledge base.

The platform is open-source and can be found on Github. Suggestions, feedback and contributions are welcome.

USE CASES AND SCENARIOS

The main use of the platform is for finding solutions to problems that would normally require time or that cannot be solved right away because their solving is not obvious yet. This kind of problems may be found in the national exams training sets, Mathematics Olympiad subjects and training textbooks, and even homework with higher difficulty level. The 'user' in these cases is the student who needs to find a solution to one of these problems as soon as possible and with minimum effort.

WORKFLOW

The general workflow is described as follows: the user has to solve a problem. They access the application, log in with their Facebook account and create a new workspace. This step may be optional, but is recommended. Then they switch to the Forum section of the platform and search for similar problems[2]. One they have identified their problem with a similar one already posted and solved on the platform, they are provided with a solving path, consisting in easy steps to follow in order to achieve the desired result. Please note that the platform does not provide the full solution, but only guidance for achieving it. Next, they switch back to the Workspace section of the platform and perform the calculus that was previously indicated in the solving path. Finally, the user can save their workspace for later use.

[1] Please note that not all these features and implemented. The status of completion for each feature will be specified in detail in the following sections.
[2] The search function will be defined in the following sections.

2. TECHNICAL DETAILS

The platform is a web application and will be explained in detail in the following sections. The implementation was designed to support live events and interactions with the user and provide a powerful and intuitive user experience, but also high performance. The technical details will be delivered in two main groups: server-side and client-side.

SERVER-SIDE

The server is currently hosted on could service provided by Amazon via Windows Azure. The server is power by node.js, which is a platform build on Chrome's JavaScript runtime. The code is entirely JavaScript; all the network optimisations are made by node.js. More about node.js can be found at their official website: Node.js.

The server uses the Express framework for node.js, making it easier to handle requests and deliver responses, set routes and configure the server. Express is a web application framework for node.js, and further details can be found on their official website: Express.js.

HTML ENGINE

The HTML is generated using Jade. The pages are dynamically updated, depending on the user context, session and platform status. In order to achieve this, variables are introduced in Jade templates which are parsed at rendering and converted into HTML pages. One example of such variables can be found below.

```
- if (!user)
      li: a(href='/auth/facebook')
            small
                  | Log in
                  span.fui-man-16.icon.space-before
- else
      li
            a.profile(href='#')
                  small #{user.name}
                  img.photo(src=user.photo)
            ul.profile-settings
                  li: a(href='#')
                        small
                              span.fui-settings-16.menu-icon
                              | Settings
                  li: a(href='/logout?redirect=forum')
                        small
                              span.fui-cross-16.menu-icon
                              | Logout
```

In the code above the `user` variable is passed to the Jade parser are contains information about the current logged user. If the information exists, then the user is logged and their name and profile picture are displayed. Otherwise, a link to the authentification route is placed.

Jade templating engine is based on indentation such that any element with a certain indentation level will be nested below the first element above it with the immediate lower indentation.

In the code example above, the lines

```
a.profile(href='#')
      small #{user.name}
      img.photo(src=user.photo)
```

indicate that an `a` HTML element holds a `small` HTML element and an `img` HTML element, and will be rendered into the following HTML code.

```
<a href='#' class='profile'>
      <small> #{user.name} </small>
      <img class='photo' src=user.photo />
</a>
```

Please note that `#{user.name}` and `user.photo` are variables and will be replaced by their actual values at rendering.

The Jade code is grouped into sepparate files called layouts. Every page or part of a page is described by such a layout. A page is than generated by combining these layouts into a view and rendering the view into HTML code.

For further information on Jade templating engine please refer to their official website: Jade.

DATABASE

The platform uses a non-SQL database, powered by Mondo DB. The database is hosted in cloud too, via MongoLab and Amazon Cloud Service. As compared to SQL databases, in non-SQL databases, the every record is an BSON encoded Object. The structure of such an object is detailed below.

```
{
      Key_1: value1,
      Key_2: value2,
      Key_3: {
            Nested_key1: value3,
            Nested_key2: value4
      }
}
```

The Object consists in pairs of keys and values. The values can be other Objects as well, as illustrated above. The example above is a standard JSON[3] Object. BSON is just a binary encoding over the standard JSON. All the records (also known as Documents) in a Mongo DB database are saved as such BSONs.

For database transactions another framework is used: mongoose. Mongoose is an elegant object modelling tool for Node.js and Mongo DB databases. Schemas are defined, which model each entity in the database, such a User or a Workspace. Using these Schemas, the server can query

[3] JSON stands for JavaScript Object Notation

the remote database and acquire or update information. An example of one of these Schemas is shown below.

```
var workspace = mongoose.Schema({
      _id: String,
      uid: String,
      name: String,
      data: String,
      lastupdate: Date
});
```

The code above is JavaScript and defines a `workspace` Schema containing five fields. The first four fields are of String type while the last one is a Date. From now on, every workspace saved in the database will present the same structure: an id, a name, a user id (the owner), content (data) and a date of the last modification. An example of a workspace stored in the database is given below.

```
{
    "_id": "5190d88fh36e2c42000001",
    "data": "place content here",
    "lastupdate": {
        "$date": "2013-05-12T21:44:19.535Z"
    },
    "name": "Title of the workspace",
    "uid": "7hf848966"
}
```

For database querying, mongoose provides a powerful yet intuitive API based on these previously defined Schemas.

The platform stores in the database two different entities, `user` and `workspace`. The workspace schema has already been posted above, containing five fields to describe and uniquely identify any of the saved workspaces. A workspace is identified by `_id` and `uid`. The `_id` field describes a unique id for the workspaces, generated at creation by mongoose, while the `uid` field describes the unique user identifier provided by Facebook on login.

The user Schema is defined as follows.

```
var user = mongoose.Schema({
      id: String,
      name: String,
      email: String,
      photo: String,
      type: {
            type: String, enum: ['admin', 'basic'],
            default: 'basic'
      }
});
```

A `user` is described by five fields: an unique `id` provided by Facebook, their `name`, also provided by Facebook, their `email` address – if avaliable, their `photo` retrieved from their Facebook profile, and a `type`, which is defined above as an enumeration of two possible values (admin and basic), with the default value beign set to basic. All the fields are of type String. Every user is identified only by their id.

Further details about Mongo DB and mongoose can be found on their official websites: Mongo DB, mongoose.

LOGIN AND USER ACCOUNT

The users can login using their Facebook account. This option was chosen because it avoids registration and account control. The authorisation process works with OAuth 2.0, which is an open protocol to allow secure authorisation methods for desktop, web and mobile application. This authorisation process is handled by Passport, a middleware for Node.js. Passport works with many authentification mechanisms, known as strategies. For this platform, a Facebook authentification strategy is used and defined as below.

```
passport.use(new FacebookStrategy({
  clientID: FACEBOOK_APP_ID,
  clientSecret: FACEBOOK_APP_SECRET,
  callbackURL: CALLBACK_URL,
  profileFields: ['id', 'displayName', 'photos', 'email'],
  },
  function (accessToken, refreshToken, profile, done) {

      /* body removed */
  }
));
```

The strategy defined above requires three compulsory fields: FACEBOOK_APP_ID, FACEBOOK_APP_SECRET and CALLBACK_URL. The first two are related to a Facebook Developer App which has been created in order to enable the login process. The third one will be explained in the following paragraph. Another field is specified in the strategy, profileFields, which describes what information will be retrieved from the user's Facebook profile after each successful login. In this case, the platform needs the user's id, name, profile picture, and email address, if available.

The authorisation process consists in three steps. In the first step, the user clicks on a link on the main page of the platform. The link activates the /auth/facebook route (see Routing, the next section for details on this route), which internally sends an OAuth request to Facebook. In the next step, the user is redirected to Facebook, where he has to gain permissions to the Facebook Developer App that enables the login. Permissions include basic profile fields and the profile picture. If the user is not logged into Facebook, they have to login first. The last step takes place after the user gained permissions to the Facebook app. In this step, they are redirected back to the platform, on /auth/facebook/callback route (which must be set in the CALLBACK_URL field when defining the authentification strategy). Here, the server receives data from Facebook, consisting in the profile fields specified in the strategy. The server also receives an accessToken than can be used for further operations with Facebook such as posting on wall.

ROUTING

The server is configured to support both POST and GET requests from clients. Several routes have been created to serve the main functionalities of the platform. In Node.js, such a route is a function handles a HTTP request. The function receives as parameters the request and a reference to the response, both encoded as JSON objects. A basic route function should look like this:

```
var route_name = function (req, res) {
      /* process the request, do something with the response */
}
```

In this example a new route, with the name route_name, is defined as a function with an empty body. The server has seven routes, all of them listed in the table below.

Route	Method	Description
/	GET	Renders the index page
/forum	GET	Renders the forum page
/sync	GET	Syncs the workspaces with the server
/load	GET	Loads a workspaces
/auth/facebook	GET	Login with Facebook
/auth/facebook/callback	GET	Returning after login; processes user data
/logout	GET	Logs the user out

A typical example of code for the / route is poste below.

```
var index = function(req, res){
      res.render('index', {
         title: 'Matrix',
         user: req.session.passport.user
      });
}
```

In this example, the route handles requests for the index page, and unconditionally renders the index view[4], passing an object to it. The object consists in two fields, title and user. The first annotates the page title while the second holds information for the current logged user, if the user is logged. These fields can be then user as variables in the specified layout *(please go back to 'HTML engine' section for details)*.

THE / AND /FORUM ROUTES

The index route, also referred as the / route, renders the index view. The /forum route renders the forum view. These routes are defined in separate files and loaded into the main server script using the following code.

```
routes.index = require('./routes/index');
routes.forum = require('./routes/forum');
```

The require() function is a standard JavaScript function that loads the result of a script into the left-hand side variable. In the routes Object contains fields for every route in the server. Next, the server is configured to accept GET requests from clients on these routes:

[4] A view is represented by a Jade file that describes a layout

```
app.get('/', routes.index);
app.get('/forum', routes.forum);
```

Here, `app` is a variable that holds the whole application (server) and is instantiated as follows:

```
var app = express();
```

The `.get()` method is part of the Express API for Node.js.

THE `/AUTH/FACEBOOK` AND `/AUTH/FACEBOOK/CALLBACK` ROUTES

These two routes are not handled by the server itself, but by the Passport middleware. The server has been configured to accept requests on these routes, but no function was built for any of them. Instead, the Passport middleware provides an API that has been used for these two routes for handling authentification requests. The code for the server configuration is as below.

```
app.get('/auth/facebook', passport.authenticate('facebook'));
app.get('/auth/facebook/callback',
  passport.authenticate('facebook', { successRedirect: '/',
                                      failureRedirect: '/' }));
```

This code passes the request handling responsibility for these two routes to the Passport middleware. Passport will ensure that the authentification process goes as planned.

THE `/SYNC` ROUTE

The `/sync` route does not render any view, but is used for live user – database transactions. The platform enables the user to edit and save the workspaces in real-time, without refreshing the page. In order to achieve this, a client-side script[5] initiates an AJAX call to this route on the server, requesting the update or the creation of a new workspace. The route receives data by `GET` method. The data encoded into a JSON object with the following structure.

```
{ wid: WORKSPACE_ID,
  uid: USER_ID,
  name: WORKSPACE_NAME,
  data: WORKSPACE_CONTENT
}
```

This structure is sent to the `/sync` route, which processes it. The acquisition of this information is achieved by the following code. The `req` variable holds the request information.

```
var uid = req.query['uid'];
var wid = req.query['wid'];
var data = req.query['data'];
var name = req.query['name'];
```

First, the authenticity of the user is verified by comparing the user id sent with the user id stored into the current session on server.

```
if (uid === req.session.passport.user.id) {
      /* user is ok */
} else {
```

[5] Please refer to the appropriate section for more details on client-side scripts.

```
      */ user is an intruder */
}
```
Afterwards, other checks are performed and the database is queried for the required information. If the workspace id provided is found in the database, the corresponding workspace will be updated (the owner of the workspace is checked as well). Otherwise, a new workspace is created and the new workspace id is returned to the client for storing and further updates.

THE /LOAD ROUTE

This route serves two functions. Depending on the parameters received from the client, it returns all the workspaces owned by the user, or only the workspace with a certain id. The structure of the data sent to this route depends on the situation and can be one of the two shown below.

```
1. { uid: USER_ID }
2. { uid: USER_ID, wid: WORKSPACE_ID }
```

In the first case, the route receives only a user id and, after comparing it with the id stored in the session for the current user, the route will return a list of workspaces representing all the saved workspaces owned by the user with the id `USER_ID`.

In the second case, the route receives a user id and a workspace id, performs the user authenticity checks, performs the workspaces ownership check, and if all of the above said are correct, returns only the workspace with the id `WORKSPACE_ID` owed by the user with the id `USER_ID`.

In addition, if the workspace or the user does not exist, or the workspace is not owned by the user with the specified user id, an object containing certain error codes and descriptions is returned to the client.

THE /LOGOUT ROUTE

The `/logout` route does not perform any checks, but only deletes the current session. Moreover, the logout route must receive an `redirect` parameter from the client, and will redirect the client to the desired URL. The whole logout route function is posted below.

```
var logout = function (req, res) {
      req.logout();
      res.render(req.query['redirect'], { title: 'Matrix', user: false
});
```

The `request` variable contains a `.logout()` method that deletes the current session.

The last three routes are also registered with the server using the following code.

```
app.get('/logout', routes.logout);
app.get('/sync', routes.sync);
app.get('/load', routes.load);
```

CLIENT-SIDE

The client-side consists in the scripts and resources loaded in order to achieve the user experience. The interface is intuitive and easy to use, created using some powerful tools and frameworks. All the scripts that handle the user interactions are JavaScript.

STYLING

The user interface has been styled using two major frameworks: Bootstrap from Twitter and Flat UI (derived from Bootstrap). Both are in their majority pure CSS rules, with jQuery for visual effects.

The styles have been organised into separate files depending on their use. There is a main `style.css` file holding general rules, while two other files, `index-style.css` and `forum-style.css` describe the styling of the two sections of the platform. There is another file, `matrix.css`, containing only rules related to the mathematics elements.

The matrices and all the mathematics elements in the pages were created using pure CSS rules. In the example below is shown the rule used to achieve the matrix design.

```
table.matrix {
      border: 1px solid black;
      border-radius: 6px !important;
      border-top: none;
      border-bottom: none;
      border-collapse: separate;
      border-spacing: 3px;
      padding: 3px;
      margin: 10px;
      display: inline-table;
}
```

Some visual effects have been implemented using jQuery.

For further information about Bootstrap and Flat UI, please visit their official websites.

LOCAL STORAGE

The platform is designed to be used by switching between two sections, Workspace and Forum, so the user must be able to keep his work in the Workspace even they leave the Workspace section. All the information about the workspace is saved in the local storage of the browser using the JavaScript API. The example below sets and loads an item from the local strage.

```
localStorage.setItem(itemName, value);
localStorage.loadItem(itemName);
```

Similar calls have been used to store information about the workspaces.

USER INTERACTIONS

User interactions are handled using JavaScript and jQuery. The real-time events are using AJAX calls to communicate with the server. There are there AJAX calls that may fire during the use of the platform. The cases in which one of these calls may fire are: the user saves a workspace, the user loads a workspace and the user requires seeing the list of saved workspaces. In the example below, there is shown the AJAX call for loading a workspace.

```
$.ajax({
      type: 'GET',
      url: '/load',
      data: user_data,
      success: function (data) {
            /* do something with the data */
      },
      error: function(err) {
            console.error(err);
      }
});
```

MATHEMATICS AND COMPUTATION

The mathematical calculus and computations are currently handled by a client-side script. The platform supports simple matrix related operations such as adding, subtracting, multiplying matrices, computing the determinant and the transpose. In the future, the platform will be able to perform more mathematical calculus. For more details on further development please refer to the appropriate section.

Every matrix is stored in page as a hierarchy of HTML elements, encoding all information needed. An example of such a matrix stored in the DOM is shown below.

```
<table id="m1" class="matrix" nrows="2" ncols="3">
   <tbody>
      <tr>
         <td class="holder" contenteditable="true">1</td>
         <td class="holder" contenteditable="true">2</td>
         <td class="holder" contenteditable="true">3</td>
      </tr>
      <tr>
         <td class="holder" contenteditable="true">4</td>
         <td class="holder" contenteditable="true">5</td>
         <td class="holder" contenteditable="true">6</td>
      </tr>
   </tbody>
</table>
```

The matrix is encoded as a `table` HTML element, with a unique `id` and matching the CSS rule for the class `matrix`. The table also encodes the number of rows and columns of the matrix. Each `td` of this table holds a number from the matrix, and also matches the CSS rule for class `holder`. This class helps identifying elements on which certain filters must be applied. Moreover, each `td` is set with the `contenteditable` attribute, allowing the user easily to edit the numbers in the matrix, without having to insert a new `input` element in the page. This pseudo-input element must be filtered and certain keys must be disabled. In this direction, jQuery is used to identify all `holder` class elements. Next, a function is bind to the `keydown` event for all the `holder` class elements, which will filter the user input. The code below illustrates the filtering function.

```
$('.number, .holder').on('keydown', function (event) {

      /* prevents the enter key */
      if (event.keyCode == 13) {
            event.preventDefault();
            return false;
      }

      // Allow: backspace, delete, tab, escape, and enter
      if (  event.keyCode == 46 ||
            event.keyCode == 8 ||
            event.keyCode == 9 ||
            event.keyCode == 27 ||
            event.keyCode == 13 ||
             // Allow: Ctrl+A
            (event.keyCode == 65 && event.ctrlKey === true) ||
             // Allow: home, end, left, right
            (event.keyCode >= 35 && event.keyCode <= 39) ||
            (event.keyCode == 109 && $(this).text().length == 0)) {

                  // let it happen, don't do anything
                   return;
      }
      else
      {
            // Ensure that it is a number and stop the keypress
            if (  event.shiftKey ||
                  (event.keyCode < 48 || event.keyCode > 57) &&
                  (event.keyCode < 96 || event.keyCode > 105 )) {
                        event.preventDefault();
            }
      }
});
```

In order to create these matrices as HTML elements, two functions have been implemented; allowing the application to both convert JSON encoded matrices into HTML elements and the reverse.

The structure of a JSON object describing a matrix is as below.

```
{
      nrows: Number,
      ncols: Number,
      numbers: []
}
```

The `nrows` and `ncols` fields store the number of rows and columns of the matrix while the `numbers` field is a bi-dimensional array of numbers representing the content of the matrix.

A matrix element is then added to a sequence. A sequence is basically a line on the screen, supporting only one mathematical operation. Once a new matrix is added to the workspace, it is automatically inserted into a new sequence. If the user decides to operate on that matrix, the result will be added to the same sequence and the sequence will be considered completed. The completion of a certain sequence is stated by the presence of the `completed` class. Below are posted examples of completed and not yet completed sequences.

```
<div class="sequence">
   <table id="m3" class="matrix" nrows="1" ncols="1">...</table>
</div>

<div class="sequence sum completed">
   <table id="m3" class="matrix" nrows="1" ncols="1">...</table>
   <div class="operator">+</div>
   <table id="m5" class="matrix" nrows="1" ncols="1">...</table>
   <div class="operator">=</div>
   <table id="m4" class="matrix result" nrows="1" ncols="1">...</table>
</div>
```

Every sequence contains information about the operation that has been performed on the matrices nested under it. In the example above it has been performed a summation between two matrices. When the result of the calculus has been delivered to the script, it creates a new matrix holding the result and adds it to the same sequence. Finally it completes the sequence by adding the `completed` class. Nonetheless, operators have been added for the sake of presentation.

If the user still wants to perform a mathematical operation on a matrix nested under an already complete sequence, the script will clone the matrix and add the copy to a new sequence, where the process goes on as usual.

In the case of summation, subtraction or multiplication, the user has to select and right-click the first operand and to choose the desired operation from the context menu that will appear. After selecting an operation, the script will automatically add the second operand with the correct dimensions, depending on the operation.

In case of determinant, the result is no longer a matrix and will be added as it is to the current sequence, after which the sequence will be considered completed. Moreover, the `determinant` class is added to the operand, forcing it to match the CSS rule for this class and adjusting the visual style of the matrix, straightening it's the borders. The CSS rule for the `determinant` class is posted below.

```
body table.determinant {
      border-radius: 0 !important;
}
```

Please note that this CSS Rule just overwrites the non-zero `border-radius` property of the original `matrix` rule. In addition, even if the operand is displayed as a determinant, it is still a matrix and further operations can be performed using it as operand. If another operation should be performed on this matrix, the matrix will be cloned into another sequence and the `determinant` class removed.

COMPUTATION

The calculus is performed by a client-side script[6] written in JavaScript. It currently offers support for simple matrix related operations only, but will be extended to other fields in mathematics soon. All the algorithms implemented so far are the naïve ones, strictly sticking to the definitions of the operations. For example, the determinant of a matrix is computed as sum by all permutations of the product of the matrix elements given by the current permutation. The transpose is computed by reversing lines with columns in `O(n2)`. The summation and subtraction are computed by going through all the elements of the two matrices, and the multiplication is based on its definition. The code computing the transpose is shown below.

```
var transposeMatrix = function (matrixData) {
    var result = new Object();
    result.ncols = matrixData.nrows;
    result.nrows = matrixData.ncols;
    result.numbers = [];
    var i = matrixData.ncols, j;
    while (i--) { j = matrixData.nrows;
      result.numbers[i] = [];
      while (j--) {
            result.numbers[i][j] = matrixData.numbers[j][i];
      }
    }
    return result;
};
```

3. HARDWARE AND SOFTWARE REQUIRMENTS

In order to be able to access the application, the user must have all of the following:

1. A personal computer
2. An internet connection
3. An webkit enabled browser (Chrome, Safari)

4. FUTURE DEVELOPMENT

The platform is currently in pre-alpha state, being continuously improved. Some major improvements and development is scheduled.

COMPATIBILITY FOR ALL BROWSERS

The platform highly uses the webkit rendering engine implemented into Chrome and Safari browsers. Trying to run the application in other browsers may fail because of the lack of support for some visual impact properties. Moreover Bootstrap and Flat UI, both used for styling have troubles because their CSS rules can't be successfully rendered in certain browsers as Internet Explorer. The platform should be accessible to all users regardless their browser preferences, so an update in this direction will be scheduled as soon as possible.

[6] But will be moved to server-side

In order to achieve this, we will use other prefixes than webkit, too, as well as research new ways to style the pages and avoid compatibility problems.

OPTIMISATIONS FOR REQUESTS

All the requests are currently sent using `GET`. This can be a problem if a large amount of data has to be sent, overloading the URL. Further development will include changing the request type to `POST` for the `/sync` route, where a large amount of data may occur. Moreover, new ways to encode the workspace information will be researched, as for now the whole HTML code of the workspace is sent for saving. In general, we are looking forward to improving the security of the data too.

PUBLIC AND PRIVATE WORKSPACES

A workspace is owned by only one user and can be access only by them. However in the future, workspaces will be equipped with permissions, and users can be added as contributors or spectators at a workspace. A schema similar to Google Drive's one will be used. In this way, users can share knowledge and contribute to workspaces just as one may contribute to an open-source project on Github.

ADDING SUPPORT FOR OTHER MATHEMATICAL FUNCTIONS

In the pre-alpha state, the platform only supports basic matrix related operations. However it will be extended to support other mathematical functions such as polynomial calculus, integrations and derivation, algebra, linear equation systems solving, trigonometry and more. For all these functions some APIs can be used such Wolfram Alpha. In addition, the currently implemented algorithms must be optimised.

FORUM SUPPORT

The Forum section will be the place where users exchange knowledge and share guidance in order to achieve solutions to problems. The forum will allow a user to search for similar problems to theirs and access relevant guidance. Moreover, the user can give feedback and share their progress with others, constantly growing the knowledge base.

This section will be designed using general forum structures, enabling chat, notifications, private messages, moderators and generally user hierarchy. In addition, the forum will be equipped with a badging system, rewarding active users. The awards can be posted on Facebook, in this way introducing a social component to the platform.

INTELLIGENT SEARCH

Ways of rigorously defining mathematical problems will be researched, and the results used to implement an intelligent search tool for the users to easily and efficiently finding similar problems to theirs.

BIBLIOGRAPHY

[1] Wikipedia

[2] Mongo DB official website

[3] Mongoose official website

[4] Passport official website