# Express.js Exam – Let’s build Messenger

# Exam rules:

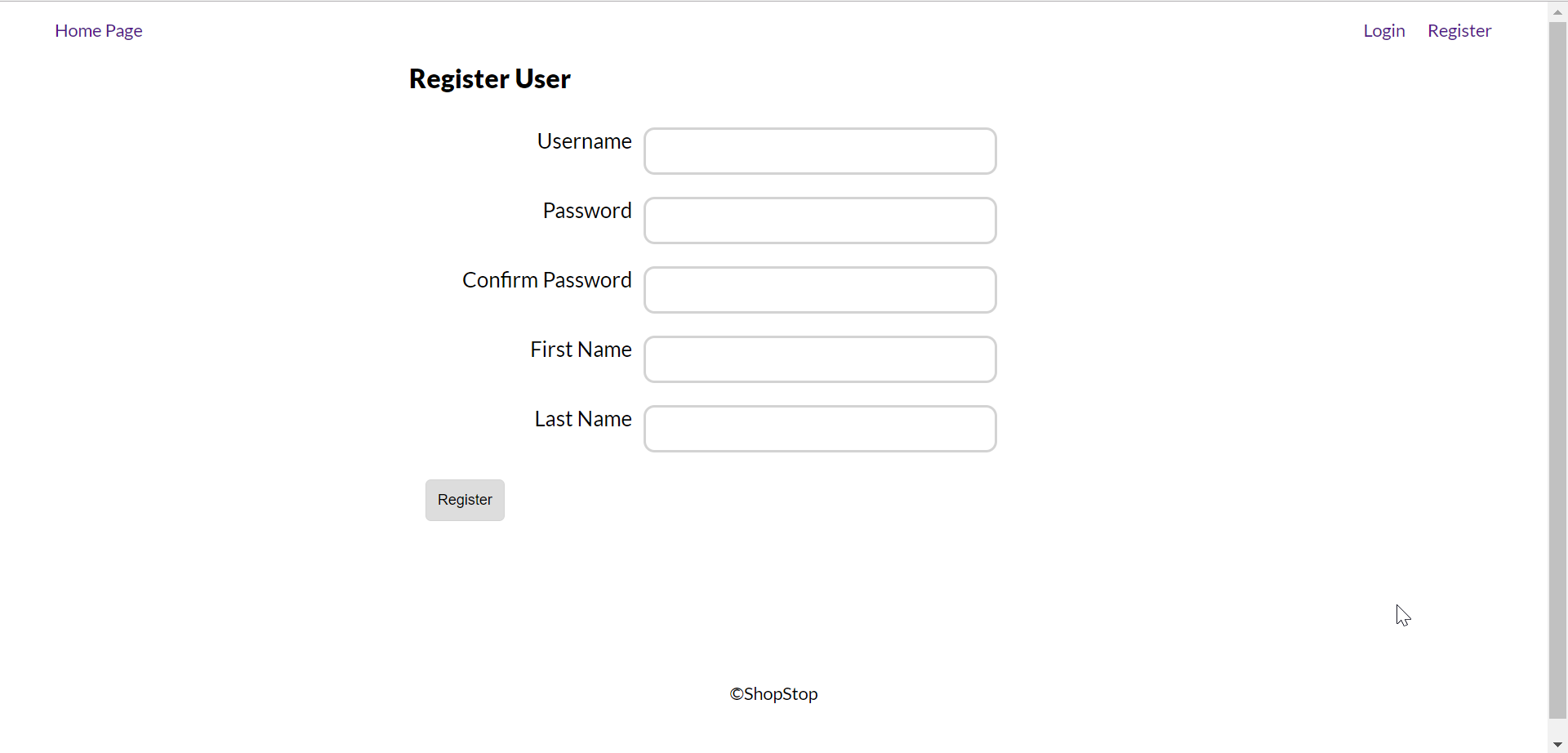
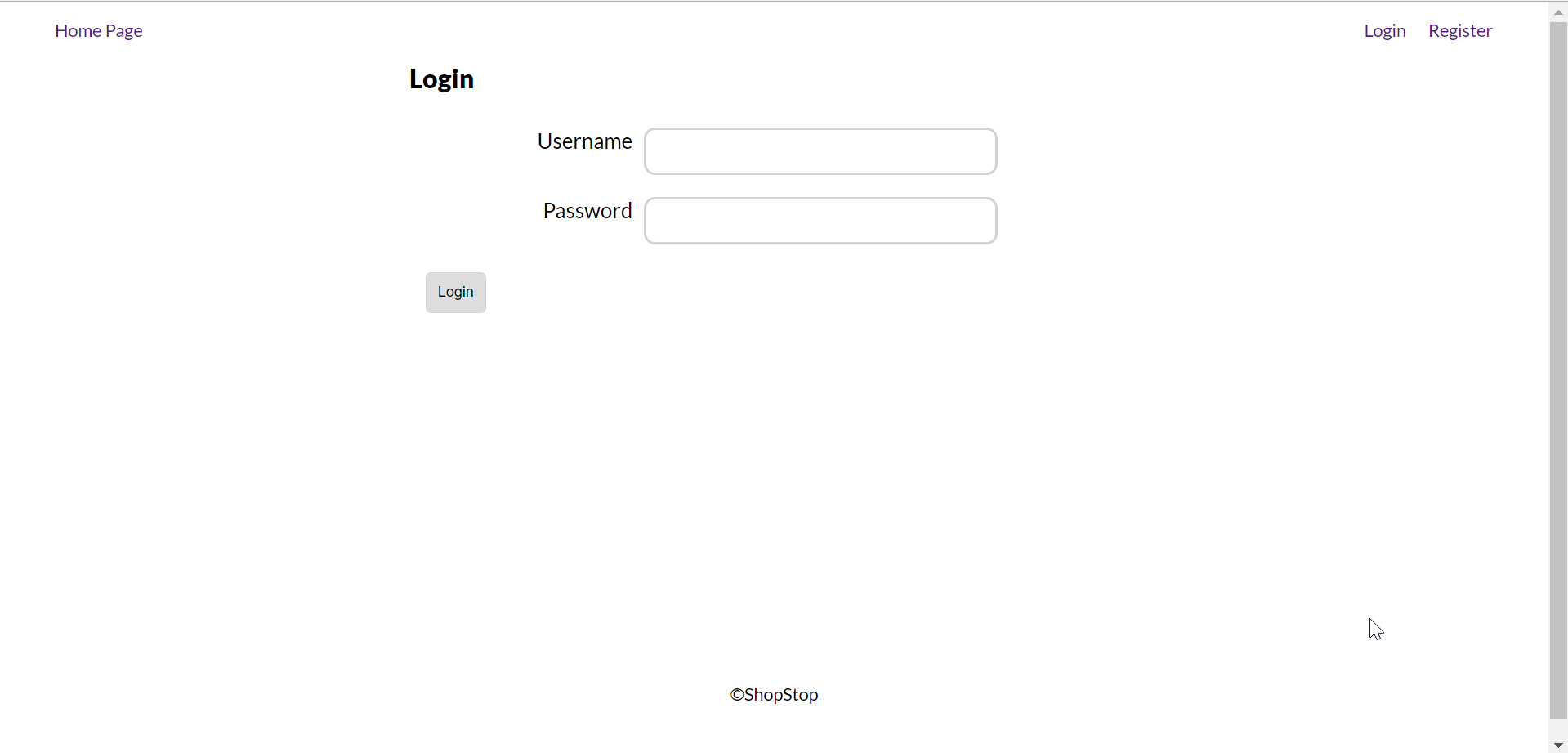
* You have 6 hours – from 16:00 to 22:00
* When you are ready, delete the *node\_modules* folder, make sure all dependencies are listed in the *package.json* file and submit your archived project at <https://softuni.bg/trainings/1642/expressjs-fundamentals-may-2017>
* There will be no breaks during the exam but you can go to the toilet or to breathe some fresh air outside **if you are alone**
* If you have any questions for the following description – ask the trainer, he is @ the back of the room
* Have fun! 

## Route / (20 Points)

Get **familiar** with the given skeleton. It has functionality for **register**/**login** and **logout**.

Firstly type ‘**npm install’** in cmd to **install** all dependencies.

**Login and Register**:



Your task is to change the **index.pug** file in the **home** folder. If the user is anonymous show only a “**Welcome to Messenger**” head title. If the user is **logged** in show a **search** bar in order to find a user to **chat** with.

|  |
| --- |
| extends ../layout.pug  block append content  if user  form(method='get' action='/user/find')  input(type='text' name='username', placeholder='Search usernames...')  input(type='submit' value='Search')  else  h1.head-title Welcome to Messenger! |

Before we implement the **user/find** action, create a **Thread** model. A thread is a **chat** between two **users** so it should contain a username **array** and a date **created** on.

|  |
| --- |
| const mongoose = require('mongoose')  let threadSchema = new mongoose.Schema({  users: [{ type: mongoose.Schema.Types.String, required: true }],  dateCreated: { type: mongoose.Schema.Types.Date, default: Date.now }  }) |

Now implement the **user/find** action. Take the username of the user to chat with from the **query** string. After that **validate** if the **typed** in user is the same as the **logged** in. A user **cannot** chat with himself therefore an error must be **shown**.

|  |
| --- |
| if (currentUser === otherUser) {  return res.redirect('/?error=Cannot chat with yourself!')  } |

Now we have to use the **mongoose** models. Find the **first** user who matches the username from the **query** string. If no user matches **show** an error. After that find a matching thread **between** the two users. If such a thread does not **exist** create one and **save** it in the database. Also don’t forget to **push** the **id** of a user in the **otherUsers** array.

|  |
| --- |
| User.findOne({username: otherUser})  .then(user => {  if (!user) {  return res.redirect('/?error=User does not exist')  }  Thread.findOne({ users: { $all: [currentUser, otherUser] } })  .then(existingThread => {  if (!existingThread) {  Thread.create({ users: [currentUser, otherUser] })  .then(thread => {  user.otherUsers.push(req.user.\_id)  req.user.otherUsers.push(user.\_id)  Promise.all([user.save(), req.user.save()])  })  }  res.redirect(`/thread/${user.username}`)  })  }) |

Now we have to go back to the **index.pug** and add additional logic to the **rendered** html. **List** all of the **threads** that the currently logged in user has.

|  |
| --- |
| else  h1.head-title Welcome to Messenger!  .container  if users  each otherUser in users  .wrapper.row  h3  a(href='/thread/' + otherUser.username)= otherUser.username |

## Route /thread/{username} (20 Points)

Now it is time to create the **chat** room.

First let us create a **Message** model. A message between two users has **content**, **thread**, **user** (creator) and **dateCreated.** It also has additional properties that we will add **later**.

|  |
| --- |
| let messageSchema = mongoose.Schema({  content: { type: mongoose.Schema.Types.String, required: true },  user: { type: mongoose.Schema.Types.ObjectId, required: true, ref: 'User' },  thread: { type: mongoose.Schema.Types.ObjectId, required: true, ref: 'Thread' },  dateCreated: { type: mongoose.Schema.Types.Date, default: Date.now() }  }) |

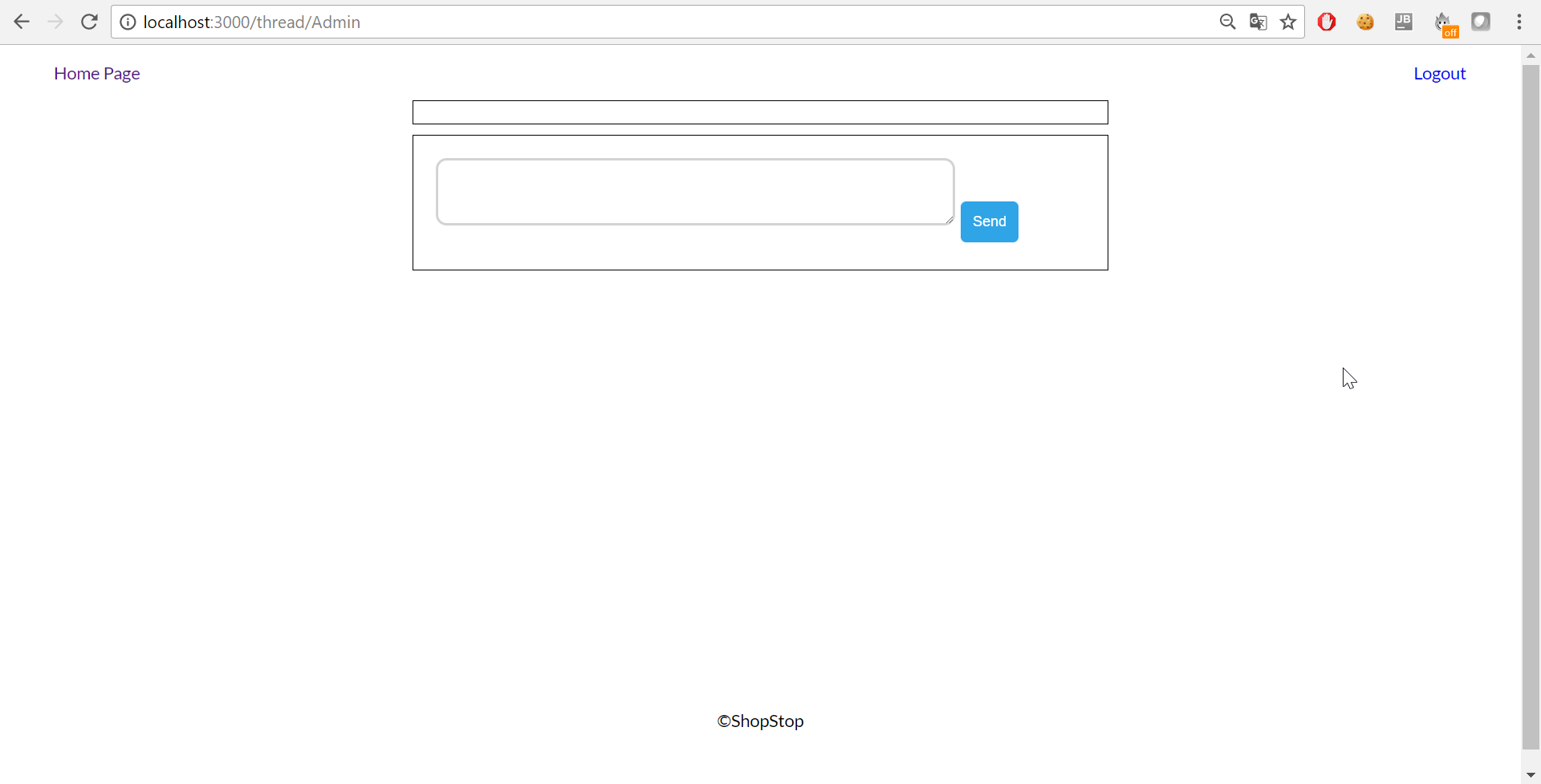
Create a route to **/threads/:username** and a **threads-controller** which has a **chatRoom** with **get** and **post** methods. At the **get** method we will retrieve the Thread **between** the two users and all of the **messages** connected to the given thread (sorted by **dateCreated** ascending). Add **error** validations when needed.

|  |
| --- |
| chatRoom: {  get: (req, res) => {  let currentUser = req.user.username  let otherUser = req.params.username  Thread.findOne({  users: { $all: [currentUser, otherUser] }  }).then(currentThread => {  if (!currentThread) {  return res.redirect('/?error=Thread no longer exists')  }  let data = { currentThread } // context to send to pug view  Message.find({ thread: currentThread.\_id })  .sort({ dateCreated: 1 })  .populate('user')  .then(messages => {  data.messages = messages  res.render('thread/chat-room', data)  })  })  }  } |

After that we should **create** out **final** pug view which has the following **html** at first:

|  |
| --- |
| extends ../layout  block content  .container.chat-wrapper  .wrapper.chat  if messages  each message in messages  - positionClass= message.user.\_id.equals(user.\_id)? 'right' : 'left'  .thread-message(class=positionClass)  span= message.content  .clear  .wrapper  form(method='post')  .form-group  textarea#chat-form.input-field(name='content')= newMessage  input.btn.btn-primary(type='submit' value='Send') |

Be **careful** with the pug **indentation**. If everything is correct you should **display** the following view:



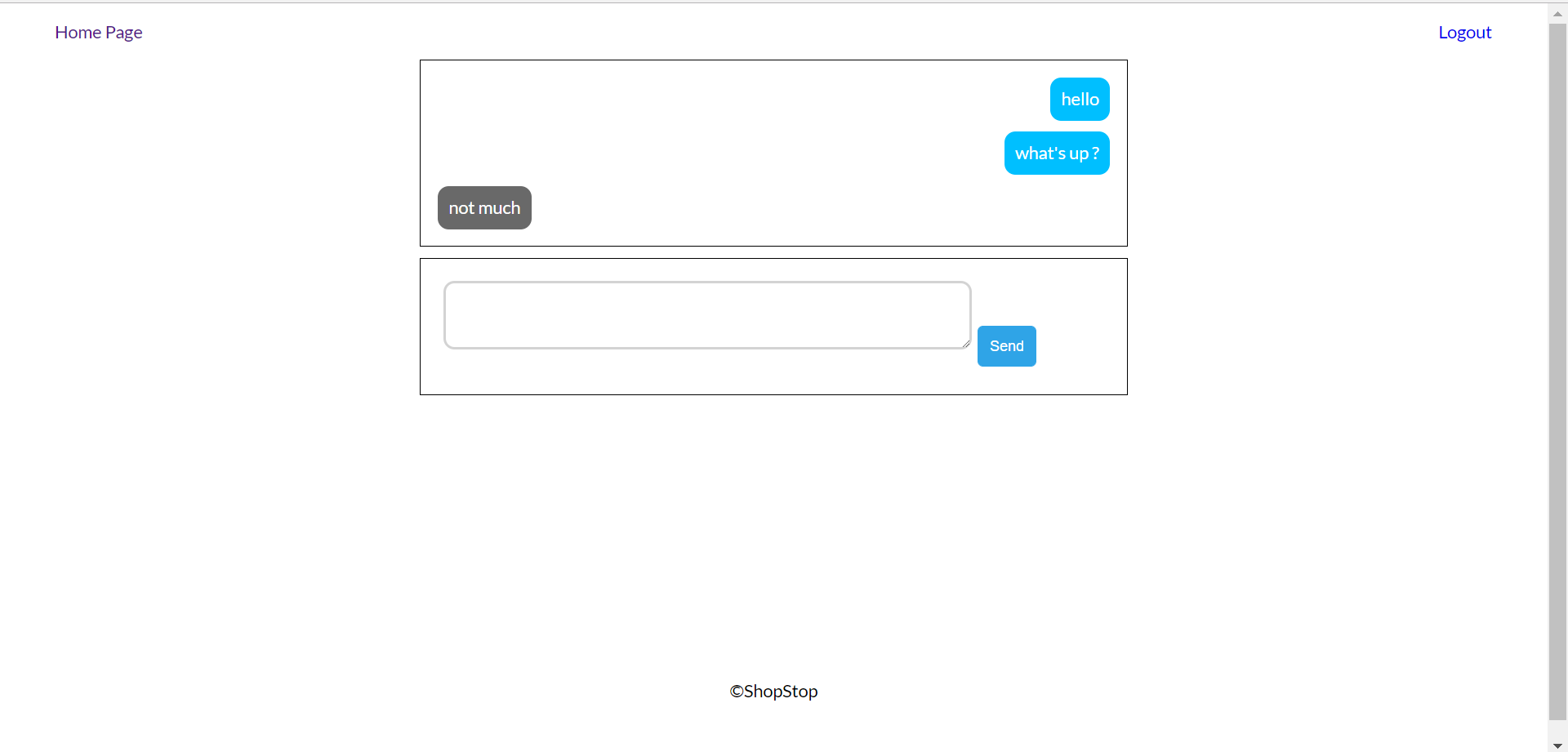
## Add messages form (20 Points)

The currently logged user can **send** messages in a given thread. Go back to the messages schema and add **validation** for the **content** of a message. A message shouldn’t be more than 1000 **symbols** long.

After that implement the **post** method in the chat room to load the current **thread** again and **add** the message to the database. After a **successful** message creation **redirect** to the thread again. If an **error** happens (such as message content is more than 100 symbols), send an **appropriate** response to the user.

|  |
| --- |
| let messageData = {  thread: currentThread.\_id,  user: req.user.\_id,  content: content  }  Message.create(messageData)  .then(m => {  res.redirect(`/thread/${otherUser}`)  })  .catch(err => {  res.redirect(  `/thread/${otherUser}?error=${err.errors.content.message}`  )  }) |

If everything goes according to plan you should see all of your messages listed with a **blue** background (try to log in as the other user to see the difference).



## Allow users to send links and images (15 Points)

To make our messenger app more **user friendly** we should add additional functionality for messages that are **links** and **images**. If a message starts with “**http**” or “**https**” it should be **clickable** and if it ends with “**jpg**”, “**jpeg**” or “**png**” we should **show** the actual image.

Create an additional **utility** in “utils” folder to check if a message content is a **link** or is an **image**.

|  |
| --- |
| module.exports = {  isLink: (message) => {  return message.startsWith('http') || message.startsWith('https')  },  isImage: (message) => {  return message.endsWith('jpg') || message.endsWith('jpeg') || message.endsWith('png')  }  } |

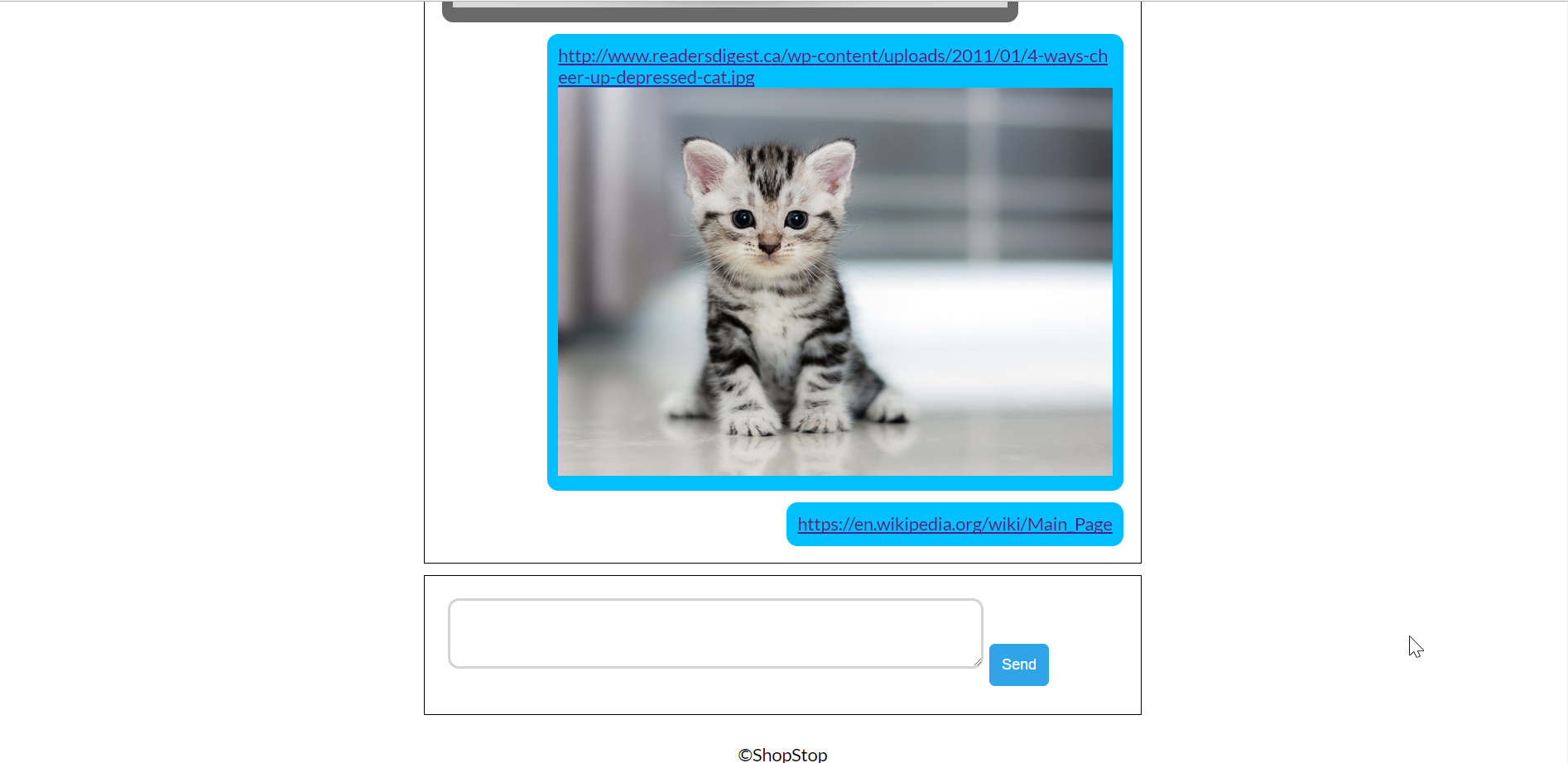
After that go back to the **get** method in the thread controller and **iterate** through the messages that MongoDb returns and **add** an additional property to each message depending on if it is a link/image.

|  |
| --- |
| for (let msg of messages) {  if (messageChecker.isLink(msg.content)) {  msg.isLink = true  }  if (messageChecker.isImage(msg.content)) {  msg.isImage = true  }  } |

And finally change the **chat-room.pug** view so it can display the link/image properly

|  |
| --- |
| if message.isLink  a(href=message.content target='\_blank')= message.content  if message.isImage  img.message-img(src=message.content)  else  span= message.content |

If everything goes well your thread should have **clickable** links and **display** images.



## Allow users to block other users (15 Points)

If someone does not **like** you – they should be able to **block** you. **Disable** the form for **blocked** users but leave the chat history (if any). Also – allow users to **remove** blocks.

First order of business is to **extend** the **usersSchema** to have an additional property **blockedUsers**.

Blocking should be done at **home/index.pug** and we should change the view.

|  |
| --- |
| if users  each otherUser in users  .wrapper.row  h3  a(href='/thread/' + otherUser.username)= otherUser.username  if user.blockedUsers.indexOf(otherUser.\_id) !== -1  a(href='/user/' + otherUser.\_id + '/unblock') Unblock  else  a(href='/user/' + otherUser.\_id + '/block') Block |

Create the routes **/user/:userId/block** and **/user/:userId/unblock** and in the **users-controller** add **two** new methods that would **block** and **unblock** the given user by id. Although here comes a very common problem in non-relational databases which is **extending** our schemas to work with **new** properties. If there are users who do not own a property **blockedUsers** array we have to add it **manually**.

|  |
| --- |
| block: (req, res) => {  let userId = req.params.userId  if (!req.user.blockedUsers) {  req.user.**blockedUsers** = []  }  req.user.blockedUsers.push(userId)  req.user.save().then(() => res.redirect('/'))  } |

Implement the **unblock** yourself.

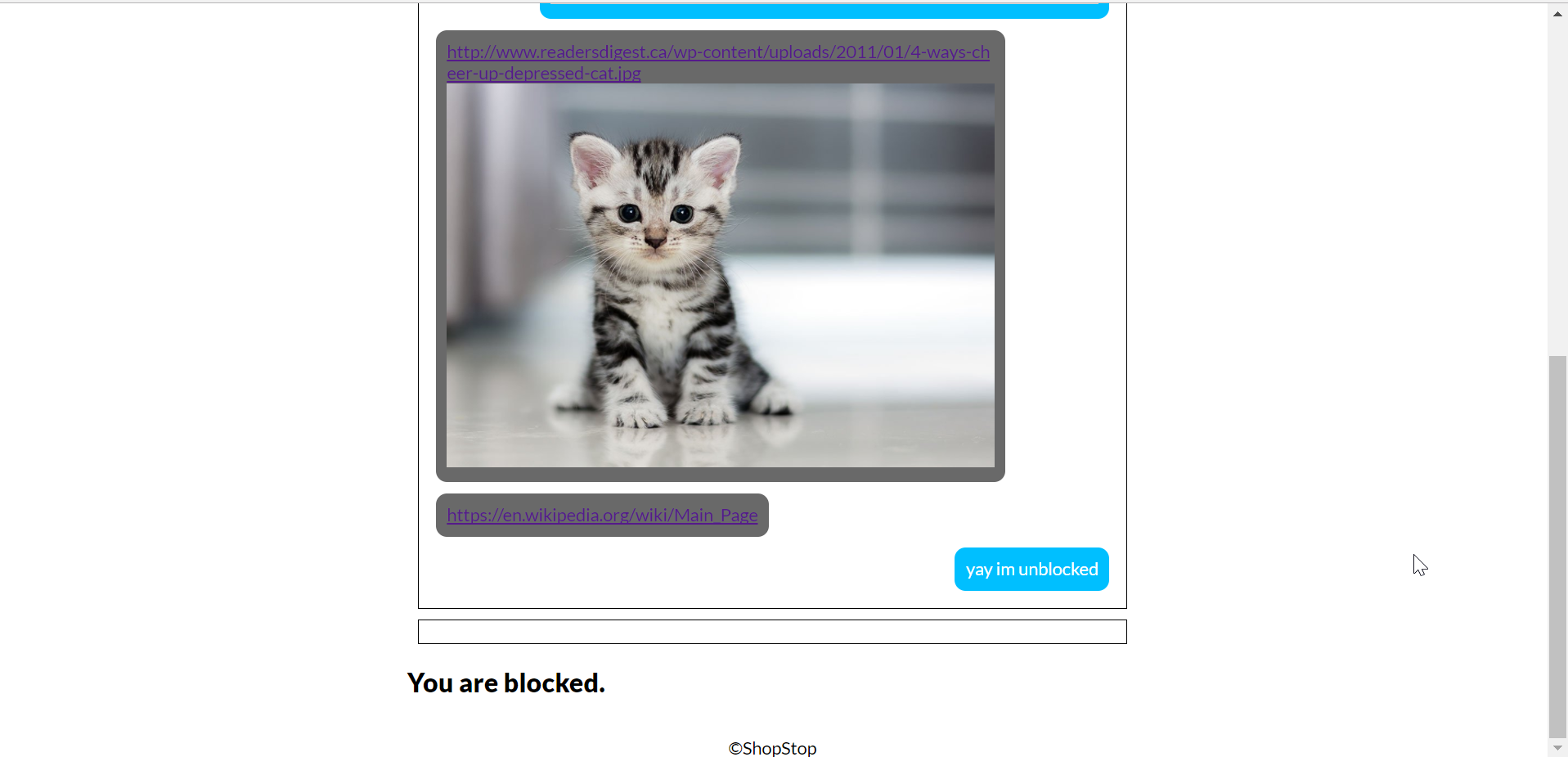
After that return to the **get** method of the **threads-controllers** and check if the second user is **blocked**. If he is blocked add an **additional** boolean property.

|  |
| --- |
| User.findOne({ username: otherUser }).then(secondUser => {  if (!secondUser) {  return res.redirect('/?error=User no longer exists')  }  if (!secondUser.blockedUsers) {  secondUser.blockedUsers = []  secondUser.save()  } else {  if (secondUser.blockedUsers.indexOf(req.user.\_id) !== -1) {  data.**blocked** = true  }  }  }) |

And finally change the **chat-room.pug** to prevent **blocked** users from **sending** images.

|  |
| --- |
| .wrapper  if blocked  h2 You are blocked.  else  form(method='post')  .form-group  textarea#chat-form.input-field(name='content')= newMessage  input.btn.btn-primary(type='submit' value='Send') |

**Blocked** users should see the **following** view:



## Add likes (10 Points)

The **final** addition to our app is to **allow** users to like other messages (like the Facebook Messenger reactions). Show some sort of indication that someone **liked** your message. Also – add option do **remove** the like.

Extend the message **schema** to have **isLiked**(Boolean) and **likedUser**(ObjectId) properties.

Create the last two routes **/message/:messageId/like** and **/message/:messageId/unlike.** After that create a **message-controller** and implement a **toggle** function which **switches** the value of the properties given above.

|  |
| --- |
| module.exports = {  like: (req, res) => {  **toggle**(req, res, 'like')  },  unlike: (req, res) => {  **toggle**(req, res, 'unlike')  }  } |

After that all you have to do is change the **chat-room.pug** for the final time (Here is the full view):

|  |
| --- |
| extends ../layout  block content  .container.chat-wrapper  .wrapper.chat  if messages  each message in messages  - positionClass = message.user.\_id.equals(user.\_id)? 'right' : 'left'  - likedClass = message.isLiked? 'liked': ''  .thread-message(class=positionClass + ' ' + likedClass)  if message.isLink  a(href=message.content target='\_blank')= message.content  if message.hasImage  img.message-img(src=message.content)  else  span= message.content  if !message.user.\_id.equals(user.\_id)  if !message.isLiked  form(action='/message/' + message.\_id + '/like' method='get')  input(type='submit', value='like')  if message.isLiked && message.likedUser.equals(user.\_id)  form(action='/message/' + message.\_id + '/unlike' method='get')  input(type='submit', value='unlike')  .clear  .wrapper  if blocked  h2 You are blocked.  else  form(method='post')  .form-group  textarea#chat-form.input-field(name='content')= newMessage  input.btn.btn-primary(type='submit' value='Send') |

**OTHER REQUIREMENTS**

* Use Node.js as a web server
* Use MongoDB as a database storage
* Use Express.js as a routing framework
* You may use whatever frameworks you like