TIE-23506 Web Software Development
TIE-23500 Web ohjelmointi

Spring 2017 course project

Online game store for JavaScript games

Table of contents

A quick step-by-step guide for the project

The projects are carried out in groups of **four**. Piazza can be used to search groups. You can post advertisements or read and reply to advertisements of others. When creating an advertisement, it might be good to tell:

- about your ambitions (e.g. would you like to get full points or just pass)
- possible time/place constraints (i.e. when and how often can you meet)
- about yourself and somehow describe the group mates you are searching for

After you have found your group, you should:

1. **Setup the repository:** Someone in the group creates a project in TUT's GitLab and adds all team members to it.
    a. **Name of project needs to start with "wsd17-"** (for example “wsd17-great-games”). No exceptions!
    b. **wsd-agent** needs to be added to the project as reporter
    c. project needs to be set to **private** (same steps as in Exercise round 1).
2. **Grouping events** for those students who need a group and those groups who need members
    a. On Monday 20.2. right after the lecture
    b. On Thursday 23.2. right after the lecture (*this is also the group registration deadline*)
3. **Register your group.** A group is considered **registered** when all the members of the group have been added to it in GitLab and wsd-agent is added as a reporter.
4. **Write and commit the project plan.** Write your project plan. Then commit and push it as the readme file of your project, called `README.md` to the root of your project. Plain text goes well, but you can also use markdown. If you want to use images in your project description you can add those to the project as well and link them in *markdown*.
5. **Create an issue in GitLab.** After you have your project plan finished, create an issue in GitLab where you request your plan to be reviewed and assign the issue to **wsd-agent**.
6. **Implement your online store and (at least) one JavaScript game.** Your application will eventually be deployed to Heroku, so you should read the Heroku Django tutorial before starting your implementation work. It should be noted that because Heroku provides only limited monthly resources for free, development and testing should be first performed locally (not on Heroku's remote servers). Otherwise, you may run out of your quota when you are supposed to deploy your project and when the course staff would grade what has been deployed.
    a. **Note** that if you get feedback on some specific aspect of your project plan and do not react accordingly in the final project, points will be deducted.
7. **Deploy** your online store to Heroku. Someone from your group needs to register to Heroku. *There will be a demo on Heroku on one of the lectures in spring.*
8. **Make your finished JavaScript game(s) available online.**
9. **Create a new issue** to your project in GitLab where you tell that the work is finished and assign that to to the person that gave you feedback on the project plan.

Deadlines

The following table describes the tasks groups must complete during the course and the deadlines tasks - will have to be completed by. The groups that start their work early on these tasks will find to workload to be easily manageable. Other groups will have a hard time completing them in time. So, for maximum results and minimum pain, start early! :-)

Description of the task to be completed	Time of the deadline
Group registration	Thursday 23.2.2017 midnight
Project Plan	Monday 6.3.2017 midnight
JavaScript games ready for distribution	Friday 31.3.2017 midnight
Test your store with other groups' games **[optional, but worth points]**	Sunday 16.4.2017
Final submission	Sunday 23.4.2017 midnight
Groups present their project work to course staff	24. - 28.4.2017

Group registrations and project plans are reviewed continuously as they come in and we encourage you start as soon as you can.

Project Topic and Implementation

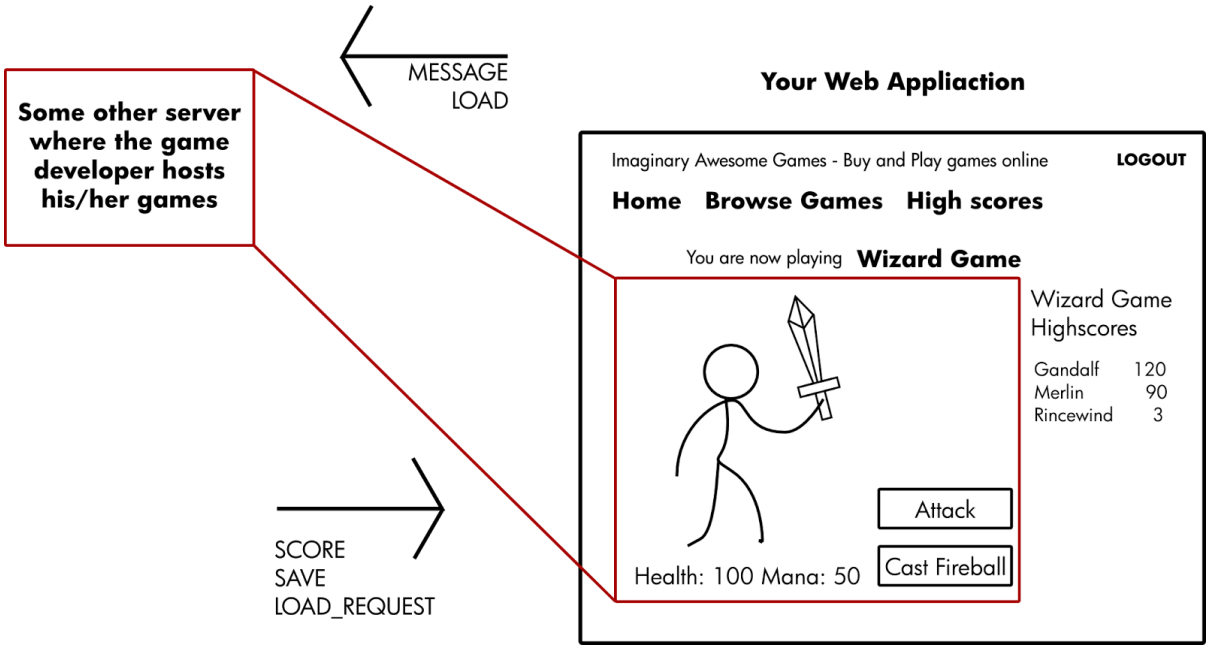
The topic for the course project is an **online game store for JavaScript games**. The service has two types of users: players and developers. Developers can add their games to the service and set prices for them. Players can buy games on the platform and then play purchased games online.

At the end, you deploy the project to Heroku. Game developers **do not** upload their JavaScript files of a game to the server. Instead, games are added to the game store by providing an URL to an HTML file that contains all assets and JavaScript links to JavaScript files.

The project has to be coded using the Django framework and has to include both client and server side code. On the client side, we strongly suggest using jQuery as a JavaScript library throughout the project. Other JavaScript libraries are allowed but not supported by the staff. Use of CSS libraries such as Bootstrap is allowed and recommended. Don't make your project a single-page-app (SPA), unless your team has extensive experience from traditional web applications and you would like to learn something new. The project has been designed so that the frontend can be

rather simple. Moreover, creating an SPA means you'd need to master techniques not covered during the exercises.

You should implement both server and client side code mostly by yourself and not compose the service from various third party Django apps. The motivation of this is that before using third party applications, it is good to understand the basics and be able to write simple services just by yourselves.



Picture 1. *Rough outline of the service (including interaction with the game). Note that this is just an example. You are free to decide on the style and layout.*

Groups will also design, implement and deploy to Heroku their own **JavaScript game(s)** that are capable of communicating with the game store using a protocol described in this document.

Game store service's functionality for players

Payment Service

Players pay for the games they want to play by using an external Simple Payments service. This service mimics an online bank payment service. Documentation and usage examples can be found on https://simplepayments.herokuapp.com

Game store service's functionality for game developers

The game developers have an inventory of games that are on sale on the site. New games are added to the inventory by giving a link to an URL of the game, which is an HTML file that should be displayed in an iframe to the player. The platform must support a simple message system, the games will send message events to the

parent document informing on the score of the game. The platform should have a global high score for each game, where top scores for each game is displayed.

Similarly to submitting high scores, the service should accept save/load game state messages. The save message can contain arbitrary JSON formatted data containing a state of a game and save it to the database. Similarly, if there is a saved game state for that game and that particular player, the service should send a message to the game, so the state can be loaded and the player can continue playing from their previous state.

Protocol for service-game communication

Game and the game service communicate with `window.postMessage`. All the messages must contain a `messageType` attribute, which can be one of six things:

- **`SCORE`**
  - sent from the game to the service, informing of a new score submission
  - the message must also contain `score` attribute
- **`SAVE`**
  - Sent from the game to the service, the service should now store the sent game state
  - the message must also contain `gameState` attribute, containing game specific state information
  - You can assume that the `gameState` is serializable by JSON.stringify
- **`LOAD_REQUEST`**
  - Sent from the game to the service, requesting that a game state (if there is one saved) is sent from the service to the game
  - The service will either respond with `LOAD` or `ERROR`
- **`LOAD`**
  - Sent from the service to the game
  - Must contain `gameState` attribute, which has the game specific state to be loaded
- **`ERROR`**
  - Sent from the service to the game
  - Must contain `info` attribute, which contains textual information to be relayed to the user on what went wrong
- **`SETTING`**
  - Sent from the game to the service once the game finishes loading
  - Must contain `options` attribute, which tells game specific configuration to the service. This is mainly used to adjust the layout in the service by providing a desired resolution in *pixels*, see examples for details.

If you plan to divert from this protocol, you need to justify this in you project plan and explain why the given protocol is not suitable for you.

Sample JSON messages

Example message SCORE from game to the service

```
var message = {
   messageType: "SCORE",
   score: 500.0 // Float
};
```

Example message SAVE from game to the service

```
var message =  {
   messageType: "SAVE",
   gameState: {
         playerItems: [
               "Sword",
                "Wizard Hat"
         ],
         score: 506.0 // Float
   }
};
```

Example message LOAD_REQUEST from game to the service

```
var message = {
   messageType: "LOAD_REQUEST"
};
```

Example message LOAD from service to the game

```
var message = {
   messageType: "LOAD",
   gameState: {
         playerItems: [
               "Sword",
                "Wizard Hat"
         ],
         score: 506.0 // Float
   }
};
```

Example message ERROR from service to the game

```
var message =  {
   messageType: "ERROR",
   info: "Gamestate could not be loaded"
};
```

Example message SETTING from service to the game

```
var message =  {
   messageType: "SETTING",
   options: {
           "width": 400, //Integer
         "height": 300 //Integer
         }
};
```

Example game

An example “game” is available in the seitti2017-repository in the GitLab, in file example-game/example_game.html . There you can see an overly simple example of how a game could implement the some of the protocol.

Note About Security

The simple messaging system used by the **game** platform can easily be tampered with JavaScript (e.g. to achieve higher scores on games/modify the saved game state). You **do not** have to implement any measures to protect against this.

Student projects have often had problems with authorization and authentication or they could have allowed the user to e.g. tamper with data coming back from the payment service or do script injections to items shown to other users. You **have to** try to prevent these.

Think about how you (or someone else) could attack your software... We will :)

Note About Software

The course is a web **software** development course. Every group member has to contribute in the software - meaning that it is not enough to concentrate on HTML/CSS only. Ideally everyone should be given a possibility to try and learn all the topics. Remember also that the course project is a learning opportunity, so remember to leave your comfort zone and try new things. If there is a guru in you group, do pair programming with them and gain a skill that will benefit you in the future.

Requirements for the online service and games

Generic requirements

- Valid CSS and HTML (use a validator... we will)
    - We also encourage the use of linting services for checking the JavaScript code (http://jshint.com/ and http://www.jslint.com/)
- The service should work on modern browsers, especially Firefox or Chrome which conform to standards rather well

- Code should be commented well, with useful comments
- Write your own code. Although in real life you might use an existing django app for some parts of your project, beware of “externalizing” all aspects of the project. We will only grade the part you wrote.

Signs of Quality

- Reusability
- Modularity
- Versatile use of Django’s features
- Sensible URL scheme
- Security (basic security can be circumvented, e.g. acquiring other user’s session/credentials, privilege escalation, injection attacks etc. **TEST ENOUGH**)
- Going beyond the basics
- Crash & idiot proof (**TEST ENOUGH**)

Functional Requirements and Grading

Please note that in order to get max points, the solution needs to be well done, not just something resembling a solution. Where minimum points have been listed, you will get minimum points for a working solution that fulfills requirements. In general, working solution is worth half of the maximum points in order to get full points the implemented feature needs to be of excellent quality.

Mandatory functionality

Minimum functional requirements (mandatory)

- Register as a player and developer
- As a developer: add games to their inventory, see list of game sales
- As a player: buy games, play games, see game high scores and record their score to it

Authentication (mandatory, 100-200 points):

- Login, logout and register (both as player or developer). See documentation about extending the User model https://docs.djangoproject.com/en/1.10/topics/auth/customizing/#extending-the-existing-user-model
- Email validation is not required for the minimum points but is required to get more than 100 points. For dealing with email in Django see https://docs.djangoproject.com/en/1.8/topics/email/#email-backends *You do not need to configure a real SMTP-server, using Django's Console Backend is enough for full points.*
- Use Django auth

Basic player functionalities (mandatory, 100-300 points):

- Buy games, payment is handled by a mockup payment service:
- Play games. See also game/service interaction
- Security restrictions, e.g. player is only allowed to play the games they’ve purchased
- Also consider how your players will find games (are they in a category, is there a search functionality?)

Basic developer functionalities (mandatory 100-200 points):

- Add a game (URL) and set price for that game and manage that game (remove, modify)
- Basic game inventory and sales statistics (how many of the developers' games have been bought and when)
- Security restrictions, e.g. developers are only allowed to modify/add/etc. their own games, developer can only add games to their own inventory, etc.

Game/service interaction (mandatory 100-200 points):

- When player has finished playing a game (or presses submit score), the game sends a postMessage to the parent window containing the current score. This score must be recorded to the player's scores and to the global high score list for that game. See section on Game Developer Information for details.
- Messages from service to the game must be implemented as well

Quality of Work (mandatory 100-200 points)

- Quality of code (structure of the application, comments)
- Purposeful use of framework (Don't-Repeat-Yourself principle, Model-View-Template separation of concerns)
- User experience (styling, interaction)
- Meaningful testing

Non-functional requirements (mandatory 100-200 points)

- Project plan (part of final grading, max. 50 points)
- Overall documentation, demo, teamwork, and project management as seen from the history of your GitLab project (and possible other sources that you submit in your final report)

Own JavaScript game(s) (mandatory 100-300 points)

- Develop a (simple) game in JavaScript that communicates with the service (the game has to use at least these three service features: high score, save, load)
- The game needs to be deployed somewhere. Note that games may be possible to deploy as static content as many of them do not require backend - other than the game store (If you'd like to deploy games to heroku as well, it can be in it’s own project in GitLab. If this is the case, please share this project to wsd-agent and link the project from your main repository)
- Note that it does not need to be the greatest game ever, going a little beyond the given example test game is enough to get half the points. For full points, we expect to see something interesting.

Functionality that earns your group extra points

Save/load and resolution feature (0-100 points):

- The service supports saving and loading for games with the simple message protocol described in Game Developer Information

3rd party login (0-100 points)

- Allow OpenID, Gmail or Facebook login to your system. This is the only feature where you are supposed to use third party Django apps in your service.

RESTful API (0-100 points)

- Design and Implement some RESTful API to the service
- E.g. showing available games, high scores, showing sales for game developers (remember authentication)

Mobile Friendly (0-50 points)

- Attention is paid to usability on both traditional computers and mobile devices (smart phones/tablets)
- It works with devices with varying screen width and is usable with touch based devices (see e.g. http://en.wikipedia.org/wiki/Responsive_web_design)

Social media sharing (0-50 points)

- Enable sharing games in some social media site (Facebook, Twitter, Google+, etc.)
- Focus on the metadata, so that the shared game is "advertised" well (e.g. instead of just containing a link to the service, the shared items should have a sensible description and an image)

Testing your service with other groups' games (0-200 points)

- Use the games other groups' games, choosing the games from a list when the list becomes available
- Give feedback for the game and it's functioning with your online game store. Make sure your feedback is constructive and professional in tone and in substance.
- Feedback will be given using the games list using a mechanism on it, when the list becomes available later.
- High quality of feedback and more than 2 tested programs lead to maximum points.

Grading

In the grading of previously listed features, correctness, security, coding style, usability, documentation, and your own tests will all be considered. Generally all members of the group receive the same grade from the project but exceptions can be made if the workload isn't distributed roughly equally. Exceptions can also be suggested by the group if someone has done more than their fair share of the work.

Tentative grade limits:

- Grade +1 1000 points (minimum to pass the project)
- Grade +2 1300 points
- Grade +3 1700 points

The exact grade limits will be decided after all the projects have been delivered.

Requirements for Different Stages

Project Plan

General description of what you are doing and how you are doing that (what kinds of views, models are needed), how they relate to each other, and what is the implementation order and timetable.

In the project plan your group will at least tell the following:

- students in your group (names, email addresses, student ids)

- name of your group
- what of the listed features you plan to implement
- are there some extra features not listed in the project description what you plan to implement?
- for each feature, how you are going to implement it
- information on how you plan on working on the project (will you meet face-to-face regularly, will use some project management tools, etc.)

Keep your project plan updated during the project!

JavaScript games ready for distribution

Design and implement a JavaScript game(s) that fulfills the requirements. Before the deadline, make the game available online, so that everyone can access it. Use Heroku and SSL. Heroku has easy-to-use wildcard SSL certificate installed on their base domain (*.herokuapp.com) so it will secure your the sub-domains automatically. As long as the the game supports ssl, other deployment methods are also possible.

Then register your game using an online form:
https://docs.google.com/forms/d/1MNFhaldF_woZ49URLYMvzoX1i1y_ThFBGCdDA1TVKcQ

Remember to document your game and to keep this documentation updated!

Test your store with other groups' games

Coming soon! :-)

Final Submission

When submitting your project,
1) create an issue and assign that to **to the same person that gave you feedback on the original project plan**
2) add the following information to your Readme.md (you can also link to a separate report from your readme):
- Your names and student IDs
- What features you implemented and how much points you would like to give to yourself from those? Where do you feel that you were successful and where you had most problems. Give sufficient details, this will influence the non-functional points awarded.
- How did you divide the work between the team members - who did what?
- Instructions how to use your application and **link to Heroku** where it is deployed.
- If a specific account/password (e.g. game developer) is required to try out and test some aspects of the work, please provide the details.

Project Demonstrations

- 30 min demo with course personnel

- whole group required to be present
- Times for the demonstrations will booked using Doodle or other suitable system. This will announced and updated here to this document.

Getting help

If your group is having problems with the project and you have already tried to handle the problems yourselves, there is help available from the course staff in the following ways:

1. You can use the channel #courseproject in course Slack https://seitti.slack.com/messages/courseproject/
2. TA Mikko Nurminen has weekly office hours at the same time as the course's Kooditorio sessions have been:
    a. Mondays 10 - 12
    b. Thursdays 14-16

    These hours are dedicated to helping groups. Mikko's office is in TF105. Just come knock on the door during the hours or walk right in if the door is open!

Good luck and happy developing!