Scrum project TDAT2003

End report

For *Harmoni*

Version 1.0

The end report is in addition to other documentation, and will among other objectives, measure the process and if the process goals set earlier in the project were reached. The report will also give a summary and overview of the project.

Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description of change** | **Author** |
| 2020/01/24 | 1.0 | First revision | All |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Content

[1. Preface 4](#_Toc30766697)

[2. Introduction 4](#_Toc30766698)

[2.1 Purpose of the document 4](#_Toc30766699)

[2.2 Definitions and abbreviations 4](#_Toc30766700)

[2.3 References 5](#_Toc30766701)

[2.4 Content overview 5](#_Toc30766702)

[3. Task description 6](#_Toc30766703)

[4. Solution 7](#_Toc30766704)

[4.1 Backlog / User stories 7](#_Toc30766705)

[4.2 Sprint 1 7](#_Toc30766706)

[4.3 Sprint 2 8](#_Toc30766707)

[5. Implementation of the project 9](#_Toc30766708)

[5.1 Sprint Retrospective 9](#_Toc30766709)

[5.2 Sprint Review 10](#_Toc30766710)

[6. Further work 12](#_Toc30766711)

[7. Attachments 12](#_Toc30766712)

[7.1 GitLab WIKI 12](#_Toc30766713)

End report

# Preface

The project is part of the course TDAT2003 “Systemutvikling 2 med web-utvikling”, which again is part of the final year of the ITHINGDA bachelor programme, NTNU Trondheim. Its purpose is to educate us, the team members, in the SCRUM project methodology, as well as good design and programming routines for developing, and developing for the web in particular. We have learned a lot regarding both subjects, and all notice a difference from when we started the project a few weeks ago. When it comes to the official learning outcomes set for the project, we have developed our abilities greatly in both development and agile development and have our results to show for it. Performing user tests has also teached us the importance of never believing that we are completely done with a system, or the design of it. For this, we have several people to thank, but mostly our Scrum Master for giving us tips along the way, our Product Owner for cheering on our progress and reminding us of the important parts of the development, and our willing test subjects for pointing out all the little flaws.

# Introduction

This document is written to properly document our development process outside of just the models in our wiki.

## Purpose of the document

The report is written to assess the process of our development, and if the process goals set earlier in the project has been reached. It will also give a summary and overview of the project, to further understanding of how we performed our development of the product.

## Definitions and abbreviations

* API: A connection to other systems. Google Maps API let us use Google Maps in our system, but we only have access to the data they give us.
* Backend: Underlying systems communicating with database and give data to frontend
* Branching: Splitting the work into several parts and working on this part only. Your computer copies the system, and you work on this, for then to later *merge* the branch into the main branch (called master) later.
* CI: Continuous Integration: Every time a something is pushed (uploaded) to the VCS, tests are run automatically using the CI.
* Client: The part of the system the user communicates with, is stored locally in the user’s browser
* DB: Database
* Frontend: Visible systems, the parts presenting data to the user
* Fullstack: Both frontend and backend
* GDPR: General Data Protection Regulation: The European standard on personal data protection
* IDE: Integrated Development Editor: A fancy text editor that often has special tools that make development much easier
* ITHINGDA: Computer engineering bachelor programme on NTNU
* Merge conflicts: When a merge leads to a conflict, often due to developers working on the same files
* Merging: Combine the work of two branches into one
* MVP: Minimum Viable Product: The minimum required parts of a system needed to use it.
* MYSQL: A programming language for communicating with databases
* NTNU: Norges Teknisk-Naturvitenskapelige Universitet
* OWASP: Security guidelines for internet applications.
* REST: A set of rules to follow to make sure systems (especially client and server in this case) to communicate properly.
* Riders: Demands from artist to participate in event
* Scrum: Agile development process, defined in section 4.
  + Product owner: Our customer, the one who gives the vision document
  + Scrum Master: Responsible for the project, but not the development
* Server: The part of the system stored in a remote location, which the client communicates with to update.
* VCS: Version Control System: The system responsible for containing our files, keeping track of branches and merging these.

## References

* Vision document: The document created by the product owner to describe the requirements of the system, URL: <https://gitlab.stud.idi.ntnu.no/williaj/harmoni-team-7/blob/master/documentation/Visjonsdokument_2020.docx>, uploaded to GitLab 24.01.2020

## Content overview

This document contains a description of the task we were given (Section 3), our solution for the project (Section 4), how we implanted it and how this went (Section 5) and further work that could possibly be done to enhance the final product (Section 6). In Section 7 you will find a link to our wiki, containing documentation on the project.

# Task description

Our mission was to develop a web system for event planners to have an overview of their events, with information such as tickets, date and location, who will perform, crew who are working on the event and possible documentation. The system should also work as a contact book for the event planner. If possible, a part of the solution should include a page for the customers buying tickets, and for the artists getting information on the event, and how their requirements are being met for coming to the event. The system would primarily be used for musical events, such as festivals and concerts.

The employer for the project was Jørgen Moen Aune, contact information below:

Phone: 916 86 832

E-mail: [jorgemau@stud.ntnu.no](mailto:jorgemau@stud.ntnu.no)

The Scrum Master for the project was Alexander Holt:

E-mail: [alexander.holt@ntnu.no](mailto:alexander.holt@ntnu.no)

# Solution

**Methodology – Scrum**

During our project, we followed the agile process of scrum methodology. This consists of short sprints, with a planning period of, in our case, around eight hours, followed by around a week of trying to reach a sprint goal, and a sprint review and retrospective to see if we reached our goal, what we need to work more on for the next sprint and what we need to keep doing as well as we do. During this process we have contact with both a Scrum Master, who is responsible for the team making progress, and a Product Owner, who we show demos to make sure we stay on track making what he or she wants. This agile process of constantly receiving feedback makes it easier to respond to changes and new wishes from the Product Owner. In our first sprint, we learned feedback that we were not using the User Stories to our full advantage and comparing our product to them. This made it hard to know where we were in the process, and in the second sprint, we focused a great deal more on this.

**Use of literature and the Internet**

There has been little use of standard literature but asking questions and searching for information on forums such as StackOverflow.com has been a core part of the development process. Documentation pages for plugins and API’s we use has also been crucial. Previous web development projects have also proven important in this process, as looking on something we have already done ourselves make it much easier to apply it in a new setting than simply reading about it. We also used GitLab as our VCS and pushed to this regularly. Problems relating to this will come in the software section.

**Hardware**

We have used our own computers to host the system locally, and the school’s MYSQL database to store data relevant to the product. We also used a large TV in our office to show demos during the sprints.

**Software**

For development, we all used JetBrains WebStorm IDE, to easily synchronize experiences and make troubleshooting easier. Most of us were already using at least one JetBrains IDE, which made this the suitable choice. It’s integrated database connection system also made this a good choice. During VCS branch merge processes, the visual merge window was also of great help when we had merge conflicts. Using the GitLab VCS worked quite well, except for when we were setting up our CI system, where we had several conflicts with testing frameworks, importing modules and setting up the server within the test environment on GitLab’s CI runners. The «solution» to this was to test most of the server connections manually, and work thoroughly on validation in the visual parts of the system. The speed of this system also made it hard to test, when it could take five to ten minutes before we knew if the new configuration file we pushed would run. There is more information on our testing in the section for that.

## Backlog / User stories

Our backlog and user stories are detailed in our wiki (Section 7), and go into detail on every part of the system. They are listed with a priority, the story itself and acceptance criteria. Note that for the first sprint, the user stories were not detailed or good enough, but we changed this in sprint two.

## Sprint 1

Our main sprint goal for the first sprint was to create an MVP. This was not quite successful, with many “almost finished”. Here is the backlog for it:

* Create Database
* Fetch events from database
* Testing
* Display events in a collapsible list
* Filter / sort / search elements in event list
* Create layout
* File upload (Documentation)
* Map integration
* Form validation
* Switch between editable and static forms
* Create tickets
* Create performers with riders
* Create and assign crew
* Search for crew and performers
* Create Event, Rider, Performer, Crew and Ticket classes
* Cookies to keep user signed in
* Register new user in database
* Hash and salt
* Other

We came a long way with many of these, but things such as map integration, search, file upload and test setup took longer than expected. As mentioned earlier, we lost sight of both this list and the user stories, which made us focus on the main things for this sprint, which should have totalled an MVP. Instead, it ended up with a system that almost worked. Our sprint review uncovered this, and we used that day to plan sprint two better than we had the first one.

## Sprint 2

The goal of the second and final sprint was to finalize the product, and of course create the MVP in the process. This is the backlog for this sprint:

* Correct verification of entered phone number
* Correctly load the phone number from database
* Correctly modify database
* Correct verification of profile picture
* Load profile picture from/to DB/file server
* "Correctly display profile picture in the user page and
* on the bottom left"
* verification of input fields
* correctly modify database
* Correct verification of location (Google API)
* Load map from Google API
* Register a ticket to an event (update DB)
* Show all tickets for an event (load from DB)
* edit/delete tickets for an event
* Fetch crew categories for organizer from DB
* Register crew to an event
* Set responsible crew for category for event
* Search for crew and add to event (^)
* Move delete button
* Add confirm dialog
* Correctly delete event from DB and update view
* Correctly register rider in DB
* Correctly load rider from DB
* Update information in DB about status and completion
* Correctly load information in DB
* Implement email sending button
* implement send email feature with email API
* Update relevant information in DB
* "Implement cancel button (replaces archive button when
* event is published)"
* Correctly update database
* Send email to all artists, crew, and buyers
* "Correctly sort documents into events and document
* categories"
* Ability to preview documents
* Ability to download documents
* Search/filter files by event
* Ability to choose document category when uploading
* Make uploading page prettier
* preview/download documents on event page (^)
* Ability to sort by date as default
* Sort by price, attendees, and location
* Filter events by published, archived, and under planning
* Filter events on search query (name or location)
* Add error report button and form
* Form validation
* Correctly register bug in DB
* Send email to developer team
* Fetch all artists for organizer from DB
* Display artists in list
* Create button and form for adding a new artist
* Validate form
* Upload formdata to database
* Fetch all crew for organizer from DB
* Display crew in list
* Create button and form for adding a new crew
* Validate form
* Upload formdata to database
* Ensure the entire site is responsive/mobile-friendly
* Set up CI with database creation
* DAO testing
* Service/Store testing
* Class diagram
* DB diagram
* Domain Diagram
* User stories
* User testing / wireframes
* Arcitecture
* Project structure
* Server services
* Documentation of source code
* Instalation and running instructions
* CI/Test documentation
* Security
* Report

As is clearly viewable here, we planned out the process of sprint two much more, and with more detailed backlog elements. These were then split up into issues and delegated among the team members. When we neared the end of the sprint, as this report was being written, most, if not all of these elements were done, with some of the documentation not completely updated. The sprint was successful, and we are proud to deliver the product.

# Implementation of the project

Our total goals were to develop a product that could work as an event planner information system and contact list. It should also contain the ability for artists to upload riders for specific events and for potential ticket buyers to buy tickets. Of these goals, only the last one was not met. Instead of implementing a system for the buyer that maybe worked, and losing time to perfect other systems, we chose to spend time on the parts of the system that related only to planning the event.

In the planning phase of our first sprint, we focused a great deal on the conceptual models, the visual wireframes. When these were done, we split up into a frontend and backend team to further create relevant models. In this process, creating user stories with well-defined acceptance requirements was not done well enough, which, as mentioned earlier, led to us losing focus in this sprint. Creating the MVP was not done during this sprint, but many of the features of it were implemented, including creating events, riders, crew and artists. A user page to add profile picture and edit contact information was also added. In the end, we believed that we had set too big a goal for the first sprint.

The second sprint was a bit more intensive, but instead of focusing on visuals and technical requirements in the planning stage, this time we set up proper user stories with proper acceptance requirements. This greatly sped up the production process, along with us working more overtime to finish the product in time. When it comes to reaching our final goal during this sprint, the answer is almost yes. The only part of the main requirements that we did not manage to implement was to create a landing page for buyers of tickets. This could be implemented in a future update of the product. A requirement set by the product owner around the middle of sprint two was to implement GDPR, which we did by set up a privacy policy, and the ability to delete and edit all contact information registered in the system. This, along with the main goals – except for the buyers’ page, lead us to believe that the final product is almost done. There are of course always things do be done on a project like this, but we feel like we have done the job we set out to do.

## Sprint Retrospective

Unfortunately, by the time this report was to be delivered, we had only performed one sprint retrospective, since the second one was planned for the Monday after. During the one retrospective we did, we piled a bunch of post-its with “could-be-better-on” notes, and “should-keep-doing” notes. Both parts had recurring themes. The first category talked about focus in planning phase, especially regarding user stories and actual requirements. We should also have had more communication during the workday to know what everyone works on. Despite this, we had few merge issues. When it came to the things that went well in the first sprint, many mentioned the team spirit, good branching and motivation of the team members. During this sprint, we had split up quite clearly in the middle, with five members doing backend work, and four doing frontend work. This led to the grievances mentioned above. In the second sprint, we decided to do more fullstack work, which also was quite natural, as we were tying the parts together in this sprint.

We did an ad-hoc retrospective for the second sprint during the writing of this report, and some of the things that were said during that was that the user stories were much better planned this time around. On the other hand, we spent quite a lot of time outside of user stories, with testing, documentation and “general” issues. This means that we probably could have spent even more time defining the user stories. Related to this, we also ended up spending a lot of time on small tasks that probably weren’t that important, which means that the priority of tasks within user stories could have been organized better. We decided to do small stand-ups after lunch, as well as updating the burndown chart publicly at the end of the day, which led to people knowing a lot more about what was being done, and what was done. It was also mentioned that we got better at branching in the second sprint, which is a good thing, knowing that none of us were very experienced with the use of them in the start of the project. Everyone was working all the time, and the time we spent on taking breaks kept motivation and efficiency at a high level.

## Sprint Review

We have already talked about the goal fulfilment in relation to the solitary sprints, so this section will focus only on the project. The plan was to create an application where event planners and artists had an overview of events they were part of. In our view, this was successful. We will now go through the list of requirements from the vision document and discuss whether these single parts were successful.

**Login system**

**Register User:** This is done by entering your or your company’s name, an email and a password. Once these meet the requirements set in the relevant user story, the user can be registered. A verification of the email is not provided at this time. This could have been set up but was not deemed important enough to divert time away from other systems. In this part of the system, the user must accept the privacy policy, which meets the requirements set by the GDPR standard.

**Change password:** This is possible through the user information page and meets the user story acceptance requirements.

**Log in:** You can log in with a registered user and is met with an error if you enter a wrong username/password combination.

The complete login system is almost complete, with the email verification being the only thing that’s missing. It is not mentioned in the vision document but would be a requirement should the system go into actual production and use.

**Event system**

**Register Events:** The user is met with a page where they can enter information related to the event in any order they want, the system accepts empty values for all pieces of the form except for the name. This part of the system also includes a part where the user can register artists, crew and upload documentation for events.

**Delete/Archive events:** If you choose to delete an event, all information about the event is deleted in the database, so that we adhere to GDPR. Archiving an event makes the user unable to edit it, but the event still exists, and the user can see it, and un-archive it if they wish to use it again. We also added to ability to cancel an event if it is “published”, as in potential buyers can see the event.

**Edit Event:** The edit and register event page is the same, so that when you register an event, you will be sent to the edit screen of that event, and it is already created.

**Map integration:** We have included a Google Maps API integration to show the location of the event. This has been set up with a combination of Google Maps and OpenStreetMap, where the only downside is that the Google Maps viewing of the system has not been paid for, which means the map is for development. This should be set up properly if the system was going into production.

**Add tickets to event:** You can add tickets in compliance with user story acceptance requirements, with a name, price, amount available and other necessary information.

**List of crew members:** The event edit screen includes a tab to create new crew members and search for existing ones. Here you can see who are responsible for different categories, add categories and easily see contact information. You can also see your contact list with crew members on another page, to see those who aren’t necessarily part of an event.

**Calendar:** This has not been done, but the ability to sort by date on the artist main overview of events was deemed enough for the product owner.

The event system is almost done from a developer’s perspective. The only part that needs to change in production is to pay for a Google API license key and add a calendar. You could possibly also add the possibility of uploading contracts or other documents to crew members, but at this point these have to be uploaded as general documentation.

**Booking system**

**Connect artists to events:** Same as crew, artists can be registered from the event edit page. You can search for artists in your contact book or create new artists. Here you can also upload contracts and send an invitation to the event.

**Upload contract**: You can both upload contracts on the general documentation page, and on the individual artist. These can then be sent to the artist in an email generated by the system, along with a link to see the artists relevant information for the event they are invited to. The system does not include the option for the artist to digitally sign the contract, but this could be implemented later.

**Register riders:** The organizer can register riders as text or upload them as documents. If he or she takes the time to enter the text, they can be updated with a status text and a checkmark marking completion. The artist can also do this from the link sent to them in their invitation to the event. In the vision document, the product owner said that one should be able to choose riders from a list of available riders, but after talks with him, this system was deemed a better solution. E-mail communication would have to be established to communicate about riders at this point, but a possible solution to this is implementing a messaging system from the artist’s login page to the organizer of the event. The vision document included the ability for artists to edit riders, but at this point, they can only delete and create new ones. A communications interface could fix this, maybe along with a confirm dialog for both sides when the other one edits a rider.

**Customer system**

The customer system could have included buying tickets, searching for events and a social aspect of asking questions. This was not implemented.

**Non-functional requirements**

**Unit testing:** Unit testing of server side is implemented. The client does not have implemented tests, because of issues with running the tests in connection with our node system and various issues with static variables. This could have been done better by spending more time before the project started looking up how to implement client-side unit testing and set up the whole project in a way that the tests can be run both locally and in GitLab CI.

**Database:** All data is saved through NTNU’s MYSQL database and connected through REST-services.

**Web browser:** The system runs on both Chrome and Firefox, as required. It does not run on Internet Explorer and Microsoft Edge.

**Response time:** The response time is well under one second, but has not been tested with many users, since the system is hosted locally. This would have to be tested before deployment.

**User Experience:** To make the best possible user experience, we have had contact with both our product owner to ensure the product is made the way users would want to use it, and friends and family to test the system with well defined user tests. These are documented on our wiki.

**Security:** We have implemented hashing/salting of passwords, and token verification of logged in users. More information on security can be found on our wiki. The system is tested against the required OWASP rules.

**VCS:** All code is saved on the schools GitLab.

**End report:** Is this document and contains all necessary information.

**Reflection note:** Is delivered individually by each team member on the team.

**Wiki:** Can be found here: <https://gitlab.stud.idi.ntnu.no/williaj/harmoni-team-7>

In total, the project covers most of the requirements, both functional and non-functional, except for proper testing. This has instead been tested mostly through use of the system and safeguarded through good validation systems. Other than that, we are lacking the customer system, an event calendar and the requirement to validate your email before you can login to your account. These are all deemed low or no priority, except for testing, which means we have created the product in accordance with the goals but should have focused more on setting up and investigating how to set up a good testing environment.

# Further work

As is already written above, the points that need further work are these:

* Proper tests for the client-side of the application
* Customer system
* Validating email before login
* Adding a calendar for better overview of events
* Adding a Google API license key to the Google Maps integration
* In-app communication between artist and organizer. Maybe also between crew?
* A login method for crew could also be a feature where volunteers could see their responsibilities

# Attachments

## GitLab WIKI

<https://gitlab.stud.idi.ntnu.no/williaj/harmoni-team-7>

This is the link to our GitLab wiki. It contains the following:

* User stories
* Domain model
* User tests and conceptual models (wireframes)
* System architecture
* Project structure
* Class diagram
* Server-services
* Database model
* Source code
* Installation and running
* CI and testing
* Security