Process Report – Cinema Web Application

Martin Krisko (240314)

Miroslav Fratric (240160)

August 2018

IT-SEP-X-A18

Table of Contents

1. Group policy

To make our group functional we had to set up some basic policies in order to finish our project in time.

- Group members are obligated to attend everyday meetings where will be discussed what we achieved previous day and what we will work on this day
- Group members are obligated to work on the project minimal 8 hours per day
- If the group would find necessary, each group member will have to work during weekends in order to hand in working project till deadline.
- We will work together at one place as much as possible to ensure the information sharing is efficient.
- Our last day of coding is 19.8.2018

2. Process workflow

Our workflow was decided on the first day of work 6.8.2018. From beginning we wanted to use SCRUM like approach. In this approach we would user Sprint structure with two days Sprints with daily meetings and requirements with time estimations. So, we created our first functional requirements but when we tried to estimate how long will each requirement take we came to conclusion that we don't have enough information and experience in programming languages and frameworks we want to use. This means that our estimations were so inaccurate that creating meaningful Sprint structure and time estimations was not possible. None of us ever worked with Angular 2+, we both worked with Angular1.6. which is different. Also, none of us worked with ASP.NET API.

Therefore, we had to come up with new approach for our workflow. After some research we decided to combine Waterfall model like approach part of SCRUM approach.

From Waterfall model we took first four phases. These phases are:

- System – functional and non-functional requirements
- Analysis – here we created use cases.
- Design – for creating architecture of the application
- Coding – implementation of our use cases and requirements to the code with usage of architecture from Design phase

From SCRUM we took:

- Daily stand up – to inform each other what we accomplished previous day and what we will work on today.

We included new phase to Waterfall model:

- Documentation – writing documentation for our project. This include Project Report, Process report and any other documents we need in order to hand in the project

Combination of these approaches will hopefully grant us possibility to easily adjust to new situations but still hold some sort of structure through the daily stand ups. We are aware that this type of approach could lead to chaos, but we are confident that because we are only two-member group we can avoid it by frequent communication during project development.

2.1. How our workflow will work

At the start of each day (9:00) we will have the stand up meeting. Here we will inform each other what is done and what needs to be fixed. After we fulfil this meeting we then look into our requirements which we divided into four categories:

- Must have requirements – these needs to be finished so our web application could work.
- Should have requirements – requirements which should be implemented into the web application in order to give users basic functionalities
- Could have requirements – these requirements further develop users functionalities and improve user experience in our web application
- Nice to have requirements – these requirements are there if we would somehow finish all the previous requirements in time and want to push our web application to production like state.

We also have non-functional requirements which needs to be fulfilled in first days of development to enable us to build our web application.

We structured our requirements in order from top priority where each requirement could be fulfilled if the previous requirement is implemented. This requirements structure enables

us to have defined workflow for tasks. So, when we finish one requirement we know which requirement will follow.

When we decide that one requirement is completed we take next requirement and separate it to smaller tasks which are divided between both members. We couldn't make these smaller tasks from requirement earlier, because we are lacking the knowledge and experience.

Requirement is considered as completed once all task within the requirement are finished and locally tested.

2.2. What we want to achieve with our own workflow

As stated above we felt like we couldn't use SCRUM workflow effectively because we couldn't make reasonable time estimation. Also, as we were new at these technologies we needed to work together as much as possible to understand what is happening in the code.

We hoped to have more agile system which would allow us to effectively learn the technology and work on our project at the same time. Also, short period of the project and fact that we are only two-member group, pushed us into the conclusion that we don't need more sophisticated system which would be hard to maintain during what we predicted would be chaotic development.

So instead of trying to avoid chaotic workflow we tried to embrace it with additional structure of requirements and stand ups we were confident that this approach would be the best for our specific situation.

3. Day One 6.8.2018

This day we had a presentation in VIA Common House, so our work started at 12:00. This day was selected to start on the first phase of Waterfall which is System.

During the rest of the day we were creating functional requirements and after some research on non-functional requirements we created them as well. Reasons why we choose these technologies are stated in the Project Report. At the end of the day we come up with these functional and non-functional requirements

3.1. Functional requirements

Must have:

- Database of users, movies and parking
- Registration of users
- Login for users
- Users can book a movie with seat

Should have:

- Two roles Administrator and user
- Calculation of price for tickets
- Administrator can create a movie
- Administrator can change or delete a movie
- Administrator can change or delete users
- Users can book a parking place

Could have:

- User can see their current and past bookings
- Administrator can see history of all data
- Administrator can see free and occupied parking spaces
- Administrator can remove user from a parking place
- JWT authentication

Nice to have:

- PDF generation
- Gateway for payments

3.2. Non-functional requirements

- Back end written in C#
- .NET core framework
- Entity Framework
- Angular 6 front end

- Compatible with Google Chrome version 68
- RESTful web services

4. Day Two 7.8.2018

We meet up at 9:00 to work on next phase of Waterfall which is Analysis. In this phase we created our use cases. To create use cases, we took our functional requirements and define use cases for each requirement. During process of creating the use cases we deepen our knowledge of our web application.

Once use cases were created at 12:00 we finish Analysis part of our customized Waterfall model. Now we moved to another phase which is Design. Here we implemented our non-functional requirements. We had to make this part together so both of us understand each part of our system architecture.

We started with back end because we needed back end working in order to set up front end correctly. First, we downloaded .NET core framework 2.1., then we run a Visual Studio and created new project with .NET Core web application template. Then we choose Angular template. But when we checked the version of Angular it was on version 5 and we wanted to work with Angular 6. We found procedure how to update the version, but it hadn't work for us. After two hours of trying to update the version we decided to delete the template files for an Angular and created standalone front-end application with Angular 6.

We kept the WEB API template because it worked as it should. Now in order for our back end to work we installed nugget package with entity framework and created dummy database on which we tried queries and basic functionalities, so we can decide to keep it or find something else. Controlling the database was good so we decide to keep it.

Now we created Models in the WEP.API/Models and dbcontext file which created tables for our database from models we created. Then we created HTTP Get method in our movieController which took all movies inside the database and send it. So, at the end of the day we were able to get data from the database and present them on the server URL with api/movie.

5. Day Three 8.8.2018

We start our meeting at 9:00 plan for the rest of the day was to finish integration of front end.

For generating standalone front end, we need to install AngularCLI globally and Node.js. After we made it work we created simple service which would call our server API and retrieve the data from the database. Before we could do that, we needed to go through tutorial for Angular. After we finished the tutorial we were able to create the service with all the implementation in the app.module.ts. But when we tried to make the call to the server we get CORS error which is security measure for browser, so it doesn't allow to call from localhost to another localhost which was our case. After some research we implemented simple proxy which solve this issue. With CORS solved we were able to receive the data from server.

With front-end connected we finished the Design phase and move to code implementation.

Because we still have time this day we moved to our first functional requirement which is Database of users, movies and parking.

Task for Database of users, movies and parking

- BE: Create proper model for Movie object and add it to dbcontext
- BE: Create model for User object and add it to dbcontext
- BE: Create model for ParkingPlace and add it to dbcontext
- BE: Create simple controllers for movies, parking and users
- FE: Test if we can call different controller from the front-end

Because we already knew how to create tables from our Design phase we were able divide tasks between ourselves and finish this requirement quickly and with fulfilling all the task we end the day.

6. Day Four 9.8.2018

During the stand up in the morning we were able to quickly cover what we worked on previous day (because we worked together) and took another requirement which is Registration of users. We divide it to following tasks:

- BE: Add HTTP Post method to the user controller
- FE: Update user service with post method for registration
- FE: Generate registration component
- FE: Create helper function for handling errors
- FE: Implement form validator for register form
- FE: implement PrimeNG
- FE: Create the HTML elements to present the form.
- Test that we are sending correct data and data are saved in the database

We divided tasks between ourselves and started implementing these tasks. Adding the HTTP method was easy because we used the tutorial. Updating the user service was straight forward as well so we haven't had an issue same goes to generating the registration component because this is handled by AngularCLI.

Implementing the form validators was the hardest part because at this time we did not know how to implement it properly but after some research we managed to make it work.

PrimeNG was a little bit tricky because their get started tutorial was not very helpful but after we spend few hours with combined forces we made it work.

Creating the HTML was really fast we just implement the components from PrimeNG.

Even though everything was going according to plan we have not anticipated another CORS problem. The issue was that POST is not a simple method and it requires pre-flight check therefore we had to search for a solution. We were not able to make it work to the end of the day.

7. Day Five 10.8.2018

During the stand up we decided that only one of us should focus the CORS issue and the other should start working on next Requirement. The next requirement in line was Login for a user. At this point we did not need a working registration because we just add user into the database, so we can work on Login. We divided this as follows

- BE: Create new controller for authentication (so we can later implement JWT authentication)

- FE: Generate component and add logic and form for logging in + html
- FE: Update user service with login method
- FE: implement data service which will hold the logged in user
- Test that we can Login and then we can store the user in the front end

In a first hour we find a way how to enable CORS for whole application on the back end and with that fix issue from previous day. We add service into Startup.cs which handles all CORS call and enabling any method and any header, later that day we added any credentials just to be certain that we won't have CORS issue ever again. With this implemented we could test the functionality of registration and close the Registration requirement.

Regarding the Login requirement, after we create AuthController we found and issue because we wanted to create on HTTP Get method. Issue with this is that HTTP Get can't send objects in body. Therefore, we had to find a way to send them. After some research we found out that we can add parameters to our call and then take the login and password from it. After we figured this out we were able to finish rest of the task and after we tested that Login is working we closed this requirement.

Because we still had time we separated next requirement to tasks. This requirement was to enable user to book a movie with seats. There we split this requirement to tasks as follow:

- BE: Create controller which will process this movie reservation
- BE: Save the new data into the DB
- Create models for movie registration on FE and BE
- FE: Create component for movie reservation
- FE: Create service for handling movie reservation
- FE: Create component for seeing all movies
- FE: Find way to navigate properly in the FE and send parameters
- FE: Navigate from selected movie to the reservation movie component

Till the end of the day we were able to create the Movie Reservation Controller and start implementing logic to this controller. Simultaneously we created component for movie

on the FE which displayed all the movies we had stored in the database. With these we finished the day.

8. Day Six 13.8.2018

During the stand up meeting we presented what we were able to finish at Friday and went through the tasks we establish at Friday. At this point we figure out that this requirement is much more complicated then we anticipate last work day.

We had an issue with storing data to the database properly because the reference for second object was missing after we recall the function. One of us was delegated to fix this issue. Another issue was how to properly navigate in Front end. So, we need to do some research on this.

Fixing the saving data to the database took almost all day. The issue was looping reference in the database, problem was that it had not shown any error. We fixed it with help of former colleague from internship. With this we make the BE up and running as supposed to.

On the front-end side we were able to find how to pass parameter through router and in combination with data service we implemented proper routing to movie reservation component with getting all necessary data for reservation to work.

To find solutions for this issue, fix it and then debug took whole day but at least we were able to overcome probably the biggest challenge so far.

9. Day Seven 14.8.2018

During the stand up we once again went through the issues of other day and check again if everything is working properly. After this check was completed we were continuing with the rest of the tasks.

The rest of the tasks was not an issue because we already had experience with this kind of tasks so we were able to go to another requirement: Two roles Administrator and User. We then split it to tasks

- Update model of user to have role in BE and FE
- FE: Split dashboard for User, Visitor and Administrator

It took us little time to implement these changes but when we add if condition to the html to object from our data service we find out that the data service is slower then loading of the html therefore we have to find solution how to get user consistently and on time. For this we add new task:

- FE: find better way for storing user

We experiment a little bit with what we know but we could not find reliable solution. After some searching we start to store the current user into the local storage and still used data service for Boolean is logged in. Combination of these two approaches resulted in best result so we rewrite FE to store user to local storage upon login and deleting him upon log out.

With this requirement completed we then moved to next requirement which is Calculation of price tickets. The task created from this are follow:

- FE: update movie reservation to calculate total price from price of one ticket and number of tickets booked.
- FE: update the html of the movie reservation.

Till the end of the day we were able to implement this logic and html changes, so we closed this requirement.

10. Day Eight 15.8.2018

After stand up and ensuring that what we did previous day works we start working on next requirement: Administrator can create movie.

- BE: create method in movie controller which handles creating movie
- BE: Store movie to the database
- FE: Create admin component for creating a movie
- FE: Update movie service
- FE: implement form to handle creation of the movie
- FE: implement calendar to handle date
- Test if movie was created

We already had a experience with all of the tasks so we were able to complete them rather quickly but during finishing of the tasks, we figured out that it would be nice to implement PrimeNG calendar with time picker. We fought that it would be just simple implementation of component. Unfortunately, we had issue first with getting the date from the calendar. We were stuck on this for several hours.

Once the calendar was fixed we found out that JavaScript date is not compatible with C# DateTime object. After discussion we decided to convert the JavaScript date to millisecond in UTC format. Implementation of this on the front-end was quick but we had issue on the back end because convert DateTime to milliseconds and vice versa is much more complicated than we thought. But after searching on the internet we found a relatively easy way to implement this.

Once we were able to send milliseconds and convert them to dates we could test the requirement and after it close this requirement.

11. Day Nine 16.8.2018

After mandatory stand up we took another one which was Administrator can change or delete movie:

- BE: Create change and delete method in the movie controller
- FE: Modify movie service to support delete and put method
- FE: Modify movies in admin dashboard to show delete button

In this task we first tried implement delete and put method luckily, we did not run into any issues, so completion of these tasks was fast.

We had a lot of time left therefore we move to another one which was Administrator can change or delete users. We created following tasks.

- BE: Implement change and delete method in the user controller
- FE: Modify the user service to handle put and delete method
- FE: Modify users in admin dashboard to show delete and modify button
- FE: Create new user detail component which will handle changing the user

Because we did same operations earlier this day we were able to finish them both in one day.

We still had some time left this day, so we picked up some debugging we found during our development, mostly just small changes in the JavaScript files and in html.

12. Day Ten 17.8.2018

After mandatory stand up meeting we pick next requirement: Users can book a parking place. We created fallowing tasks.

- FE: implement parking reservation into the movie reservation component
- FE: Modify movie reservation model to include parking
- BE: Add logic in the movie reservation controller to handle parking reservation
- BE: Modify Movie reservation model
- Test if data were stored correctly

Changes on front-end were really quick to implement because we became quite proficient in adjusting front end. On the back end it was a little bit harder to keep track of reference in the database. But after few debugging sessions we were able to make it work.

Once we fulfil all tasks and we had three hours left we decided to make a little clean-up of the code and fix additional bugs we found during our development as well as comment our code so it's more readable specially in Startup.cs in the back end. Reason why we choose to do it in this time is that we finish all Must have and Should have functionalities.

At the end of this work day we decided to work through the weekend because our last day of coding was at 19.8.2018 and we still have plenty of small bugs and we wanted to implement some requirements from the Could have category as well.

13. Day eleven 18.8.2018

This morning stand up was quite different from the previous one. Because we decided that each of us will pick one requirement from could have category, finish this requirement and then he will continue debugging till Sunday night.

First pick was requirement Administrator can remove user from parking place. We created follow tasks:

- BE: Create remove parking method in HTTP Delete
- FE: Update parking component
- FE: Update parking service to reflect new functionality

Implementation of this requirement was quite quick, which allows us to focus more on the debugging.

Second pick was requirement JWT authentication. This requirement was always one of our priorities and during process of development we were looking for good opportunity to start implementing it. Unfortunately, time pressure forced us to postpone it to the end of the development, because we were well aware of time this implementation requires. But this way we finished all our major requirements for a functionality of the web application and we still had opportunity to finish JWT. Tasks we created:

- BE: implement token generation
- BE: Set up JWT authentication
- BE: Test BE token generation and Authentication
- FE: Create service which handles authentication
- FE: Create function to intercept the HTTP calls and input token into the header
- FE: Create proper http request to handle the authentication
- BE: Adjust controllers so the API endpoint is called only if the request is from authenticated user.
- Test the functionality

This was a huge task for such a short period of time especially when only one of us could work on it. During implementation of these task we will extensively use tutorials and examples. Implementation of token generation were completed with the help of tutorial. This functionality was tested and works properly. Set up JWT authentication was quite challenging task but after some searching we start using Basic Authentication in the header. These two first tasks were created with the request from postman and it seemed like

everything is working. Proper set up the back end took all day but at the end of the day it seemed fine.

14. Day twelve 19.8.2018

As decided in the group policies it was our last coding day. What we do not finish this day we will have to write to the documentation as an unfulfilled requirement. One of us was finishing the debugging of issues that could be fixed until the deadline. The second had tasks from JWT authentication.

Front end for the authentication was quite hard. We found several tutorials how to approach sending the headers with basic authentication, once we set up everything for the actual API call our request did not pass the pre-flight check from to CORS. After long debugging we were able to send the request but it somehow lost a header with the authentication parameters. We tried to debug it but after some time we discussed it and decided to stop working on JWT because even if we would send the correct request and get the correct response without any further debugging we would not have time to implement the actual JWT authentication on BE and FE in proper way.

At the same time, we finish last bug which could be fixed before deadline. With two hours left we have decided to start another phase which is Documentation. In the following two hours we start to update our diagrams to reflect current state of code.

15. Day thirteen 20.8.2018

During the morning stand up we decided to once again split the remaining work into parts. We figured it will be better for one of us to focus on the writing of the project report and the other one will provide him with updated versions of diagrams and will work on user guide.

However, while working on the user guide we found out that there are still some issues that would not require lot of work to fix. With this in mind we decided that if we have some spare time, or we just need a little break from the report and diagrams, we would fix those issues.

Once the user guide was finished we asked one of our friends to give us a feedback. While the text seemed to be good there was one thing that was pointed out to us, the green background. We noted this information and decided that we will put this on the list of issues that do not require a lot of time to fix.

16. Day fourteen 21.8.2018

After a brief morning update on the status of the documents we decided to keep on working in the same way, one person on the project report the other one would read through it and look if there is anything missing or would need to be rephrased, afterwards he would focus on updating diagrams.

By the end of the day we updated all diagrams and we still had some time to fix the small issues:

- Green background
- No message for user when trying to book a seat but the show is sold out
- Inconsistent text on the page (different html elements)

17. Day fifteen 22.8.2018

Because we were creating the project report together there was not much to present during the stand up meeting. There was still a lot of work to do on the Project Report, but we couldn't wait with the Process report any longer. So, one of us continued work on Project Report and the other one took Process Report.

In Project Report we have finished Abstract, Requirements, Analysis and started working on Design.

In a Process Report we had to put together information from hand written notes we created during our stand up meetings. These notes were really just a bullet points, so we had to decipher what was intended by the bullet points and then sum it up.

Till the end of the day we were working on these documentations and made a good progress mainly because we extended our working hours. For this day we both worked for 11 hours.

18. Day sixteen 23.08.2018

This stand up took a while because each of us had to read more than 40 pages of documentation. Unfortunately, we found out during our review of the documentation that we are using outdated template for a project report. Therefore, we had to update our documentation as well as create several other documentations which will be then included into the Appendixes. These documents were code structure and tests.

We got ourselves into the mess in order to fix this problem we decided that one of us will continue working on Project report which still has some blank spots, but it had to be updated as mention above.

The other member will then finish the Process Report, currently we were in day Seven of our report so there was still a lot of work.

Because we still have other documentation to write we decided that the first one to finish his report will then pick up the code structure and test document. When the other will finish his report as well we will split remaining tasks in order to finish the documentations till the rest of the day.

This day was really long both of us worked for 10 hours but at the end we were able to finish all documentation, even though the hustle of last few days could affect the quality of these documents

19. Result

At the end of our process we were able to calculate how long we spend on each task and requirement which we were not able to do from beginning. Time is calculated as total amount of time we put into them. We don't subtract brakes and eating because we were not measuring this.

Name of the phase or requirement	Number of hours

System phase	10
Analysis phase	6
Design	22
Coding – Database of user, movie, parking	4
Coding – Registration of user	19
Coding – Log in	11
Coding – User can book a movie	20
Coding – Two roles Administrator and User	12
Coding – Calculating price of tickets	4
Coding – Administrator can create movie	15
Coding – Administrator can change or delete movie	6
Coding – Administrator can change or delete User	4
Debugging Day 9	6
Coding – User can book parking place	10
Debugging and code clean-up Day 10	6
Coding – Administrator can remove user from parking place	3
Coding - JWT authentication	14
Debugging Day 11-12	11
Documentation – updating diagrams	8
Documentation – User guide	4
Documentation – Project Report	30
Documentation – Process Report	16
Debugging Day 14	6
Documentation – Code structure	3
Documentation – Tests	4
Total time spend on Stand ups (15 * 15 min)	4

Total working hours approximately: 258

Our workflow has worked in certain degree. We were able to learn new technologies and develop the application in the same time. Daily stand ups and separating each

requirement into the tasks helped us to properly understand the code and find out what we have to focus on for each requirement.

With this workflow we were able to finish all requirements we consider curtail and finish the project with all basic functionalities and several more advanced one.

20. Retrospective

At this stage we will look back on our process and consider if it worked as we anticipated and if not, what we did wrong and how we would fix it.

First of all, we will look into our workflow because it could have been our biggest strength or our biggest weakness. When we look back on our development we still think that in first days it was great approach because we didn't know how to proceed. But after few days when we implemented all our Must have requirements and finally understand how to code properly SCRUM approach we intended from beginning would be better. It's hard to decide if we made mistake because we don't know if we would be able to have such a fast start.

Issues within our workflow. We fought that our approach would allow us more flexible development but in later stages when we start dividing tasks we found out that our approach is more rigid than SCRUM approach. It didn't allow us to properly track bug fixes and when we were going back to fix something from previous requirements we de facto violate our workflow because we didn't take this part into the consideration when we were creating this workflow.

Another issue was that by our workflow we had to wait till all tasks are completed to start working on another requirement. If we would take this into proper consideration in the beginning, we could most likely finish more requirements and we would have less bug fixes. On the other hand, it helped us to work together because we were more depending on each other, which lead to more unified code and deeper understanding of code itself.

This workflow forced us to violate it quite often because we didn't set up proper way how to handle issues. This is not how workflow should work in general but because we were working together we were able to figure out most issue outside of the set workflow.

If we look back, we have to admit that we should spend more time in Analysis part. We had vague diagrams and use cases from beginning which limited us in our development. If we would spend more time during the Analysis part, we could have spent less time on stand ups and we would prevent misunderstanding we have on some functionalities.

When we made basic time table we set end of the coding day to 19.8.2018 with following Documentation phases. We fought that four days would be sufficient to create proper documentation. It would probably work if we would keep proper notes from stand ups and write down every bug fix and issue we encounter during our development. But because we often get lost in the chaos and we were more concern about functionality we haven't do it properly. Also, we haven't updated our diagrams when we changed the code. This all cause that writing of the documentation took significant more time than we anticipated. To fix this we had to increase number of work hours for last two days to 10+ to catch up. This was foolish mistake we could avoid if we would chase functionality and implement the SCRUM approach.

So, if we sum up our process: We should have put more thoughts into the workflow and analysis part. It would ease up our development and we would be able to create better documentation. Probably we would be able to implement more functionalities at the end as well as avoid issue with updating the movie when we just forget to implement forget because we haven't include it into the tasks.