Slovak Technical University

Faculty of Informatics and Information Technologies

Ilkovičova 2, 842 16 Bratislava 4

Team project

LOMON

Documentation for project management Academic year: 2021/2022

Team supervisor: Ing. Valach Alexander

Team members (team n. 9):

Bc. Balucha Ján
Bc. Bucko Dominik
Bc. Franczel Michal
Bc. Mačuga Peter
Bc. Villant Patrik
Bc. Zhan Soňa

List of abbreviations

API - Application Programming Interface
DDL - Data Definition Language
REST - Representational State Transfer
SDK - Software Development Kit
SQL - Structured Query Language

Table of content

1. Introduction

In this document we provide details about project management during the work on our team project - LOMON - in the academic year 2021/2022.

Document contains information about division of labour among team members, roles and responsibilities of each one. This is followed by description of management processes and methodology we adhere to while working on the project. We also include retrospectives for each of the sprints as well as the retrospective in global scope of the project.

2. Work distribution

Table 1 Work distribution on documentation for project management

	Dominik	Soňa	Patrik	Peter	Ján	Michal
Introduction	50%	50%	-	-	-	-
Roles of team members	16%	16%	16%	16%	16%	16%
Management	60%	-	-	40%	-	-
Sprint summary	-	30%	30%	-	10%	30%
Global retrospective	-	-	-	40%	60%	-
Methodology	16%	16%	16%	16%	16%	16%
Export jira issues	-	-	-	-	100%	-

Table 2 Work distribution on Engineering work

	Dominik	Soňa	Patrik	Peter	Ján	Michal
Introduction	85%	-	-	15%	-	-
Global goals	85%	-	-	15%	-	-
Database model	-	-	-	-	100%	-
Hardware	90%	-	-	10%	-	-
Backend	-	-	-	80%	20%	-
Web application	-	30%	30%	-	-	40%
UX testing	-	-	100%	-	-	-

Table 3 Work distribution on implementation

	Dominik	Soňa	Patrik	Peter	Ján	Michal	Weight
Firmware	80%	-	-	-	20%	-	0.15
Back-end server	10%	-	-	55%	35%	-	0.25
Database	-	-	-	40%	60%	-	0.1
Web application	-	40%	20%	-	-	40%	0.45
UX testing	-	-	100%	-	-	-	0.05

3. Roles of team members

Bc. Dominik Bucko

Team leader. Responsible for Firmware development.

Bc. Ján Balucha

Responsible for databases. He participated in creation of database models, backend implementation and also writing API specifications.

Bc. Peter Mačuga

Scrum master. Responsible for backend development. He participated in creation of database models, backend implementation and also writing API specifications.

Bc. Soňa Zhan

Responsible for front-end development. She participated in the design of the wireframes and also the implementation of the frontend for the web application.

Bc. Michal Franczel

Responsible for front-end development. He participated in the process of wireframe design and development of web applications.

Bc. Patrik Villant

Responsible for performing UX testing on a web application design prototype. He also participated in the creation of the front end and in the design of the mentioned web application.

4. Management

4.1 Development and planning management

For development management we use **Scrum**, which is a management framework used for managing small teams of 3 to 9 people. They work in short bursts of productive activity called **sprints**, which in our case last two weeks. Scrum is iterative and incremental in

nature.

The two main roles in Scrum are roles of Product Owner and Scrum Master.

Scrum Master is responsible for:

- Keeping track of progress
- Coming up with tasks
- Keeping up the morale and motivation of team members
- Making sure the team stays focused and on-track
- Handles conflicts within the team and possible obstacles

Scrum Product Owner is responsible for:

- Success of the project
- Maintaining high-level vision for the project
- Setting priorities for what needs to be done

Meetings each week allow for discussion about project needs, progress each team member has made and priorities for the project. We discuss our progress with our supervisor and present work that is either completed or stalled.

We use **Jira** software for task management. Project needs to be broken down into smaller parts - Epics - due to its significant complexity. Each epic encompasses one part of the project, in our case that includes Frontend, Backend and Firmware. These Epics then contain tasks and subtasks (task in this sense is somewhat equivalent to User Stories), which are then fulfilled by one or more team members. Each task or subtask can exist in 3 states - To Do, In Progress and Done. Each task is also given a priority, which guides team members in deciding when to work on which part of the project.

4.2 Communication Management

Our team uses 2 platforms for communication regarding the project. We use Signal messenger for quick communication among team members and our supervisor, for quick

questions and synchronization regarding meetings.

We use Discord to communicate about specific topics regarding our project. We have dedicated messaging channels for different areas of work, for example frontend or backend development, firmware, etc.. When our meetings are not in-person, we also use Discord for remote meetings and collaboration using voice and video channels.

4.3 Version Management

We use GitLab for version control of each software component that we develop. Within the repository for each software component, versions are automatically created on each commit to master branch, which should be done primarily when merging a branch containing a new feature or bug fix to master branch. We can also create versions using tags, where we can enumerate specific versions using version naming convention.

4.4 Code Review Management

To mitigate issues that arise from mistakes individuals can make while writing the code, such as not adhering to code conventions or introducing bugs, we use code reviews. When a team member is done implementing a feature or bug fix, they create a merge request and request code review by some other member of the team. This person is then responsible for going over the changes and requesting revision when an issue is present. After all of the code is deemed by them as ready to merge, they merge it to the master branch and are therefore responsible for any problems this change may have caused. Team members are generally not allowed to merge their changes by themselves without an approval from any other team member.

4.5 Documentation Management

Documentation is created during the sprints and each team member records their tasks, progress, and results. As part of the finalization documentation, each member contributes particular parts, and the whole team checks that all steps have been followed and that the process is correct.

4.6 Test Management

Our product and its components are tested throughout the development. Some software components, such as the backend of our application server, are tested using automated tests on commit in continuous integration pipeline. Other components are tested by team members in non-automated manner, such as user interfaces and hardware devices. We also do user testing on wireframes before implementing frontend of our application.

4.7 Risk Management

In the context of this work, risks involve every unexpected event, both internal and external, which can occur throughout the duration of a project and which can affect the project in some negative manner. This part identifies potential risks, describes those with non-zero probabilities, analyses their sources, impacts, preventative measures to be used for their mitigation and measures, which will be executed if needed to lower their negative impact.

4.7.1 Bad time allocation for a task

Probability – High
Symptoms – Team member can't finish assigned task on time, or finishes too quickly
Impact - Not finishing tasks on time can have negative consequences on other tasks, which are dependent on it and can disrupt the timeline of the whole project. Because of completing tasks too quickly, however, team members can have too much unallocated time, which can result in them working ineffectively.
Preventative measures – Discussion with team members, assigning tasks according to individual member's capabilities
Negative impact mitigation – Team members with completed tasks can help team members, that are being late

4.7.2 Unfinished sprint tasks

Probability – Medium
Symptoms – Task, which was allocated to given sprint is still not completed at the end of a sprint
Impact – Task is moved to the next sprint
Preventative measures – Better planning, which is based upon individual competences and skills, better division of tasks into sprints
Negative impact mitigation – Help team members, that seem to be struggling

4.7.3 Web application failure

Probability – Low
Symptoms – Application shows error message

Impact – User can try again, but if he doesn't succeed he might give up on a given task. As a result, application doesn't work according to defined use case
Preventative measures – Code review and test application periodically
Negative impact mitigation – Release hotfix, which fixes given issues as soon as possible, especially if errors occurred in the production environment.

4.7.4 Server failure

Probability – Low
Symptoms – Server returns error messages, error messages reported in logs
Impact – Unresponsive application, missing data, corrupted data
Preventative measures – Code review and test server applications periodically
Negative impact mitigation – Release hotfix, which fixes given issues as soon as possible, especially if errors occurred in the production environment.

4.7.5 Hardware failure

Probability – Medium
Symptoms – Sensor, end device or gateway stopped working
Impact – Either sensor (or potentially whole end device) is unable to deliver reliable data, or it has to use different gateway because of gateway failure, in which case connection can be lost
Preventative measures– reliable devices, replacement sensors, end devices and gateways
Negative impact mitigation – replace failed device as soon as possible

5. Sprint summary

5.1 First sprint

In the first sprint, we as a team focused mainly on understanding the issues of our project. We studied some documents on the technology which we will use. We also performed an analysis of individual sensors that we will need for the implementation of the project. We considered several options and chose specific types for measuring physical quantities such as: humidity, temperature, shocks and atmospheric pressure. In this sprint, we defined roles of team members and also assigned other roles such as Scrum Master (Peter Mačuga) and Team Leader (Dominik Bucko). Other tasks we performed were deploying the team's website and defining organizational rules regarding the applications we will use and how we will use them, whether for communication, versioning, task assignment, etc. Following the agreement, we deployed these applications for our project. Specifically, these were applications such as: Jira for defining sprints and tasks for individual team members, Discord for communicating team members, Miro for diagram creation, Figma for creating application wireframes and we also created a document for defining methodologies for writing documentation, source codes etc.

- Dominik Bucko and Ján Balucha - analyzed individual sensors and chose suitable for our project
- Michal Franczel and Soňa Zhan - created team’s website and deployed it
- Peter Mačuga and Patrik Villant - deployed the applications for communication, versioning etc. and created document for methodologies

We managed to cover the vast majority of the set goals, but some tasks are more extensive, and therefore the results may differ for some members of the team, for example, it is the study of documents.

Finally, we managed the sprint to a large extent successfully and all tasks were performed as needed. The study of individual documents was solved on an individual basis.

5.2 Second sprint

In the second sprint, we determined the basic but important parts of the application from all aspects: firmware, backend, and frontend, with main focus on the implementation and design of these parts:

- Peter Mačuga created goals and tasks for the second sprint in Jira and finished the Django setup.

- Peter Mačuga together with Ján Balucha designed the database, created database models, and prepared API specifications.
- Ján Balucha created a Logo for the application.
- Ján Balucha and Dominik Bucko prepared the application for the TP Cup.
- Dominik Bucko created a flow diagram for the firmware update and focused on the firmware implementation.
- Dominik Bucko started working with sensors and implemented the measurement of components/sensor BME680 to collect data on humidity, temperature, and atmospheric pressure.
- Michal Franczel, Patrik Villant, Soňa Zhan started designing and prototyping wireframes in Figma to demonstrate the basic idea of the admin panel.
- Michal Franczel, Patrik Villant, Soňa Zhan iterated wireframes and categorized components into atoms, molecules, and pages.
- Michal Franczel, Patrik Villant, Soňa Zhan started with the implementation of atomic and molecular components.

The tasks in the second sprint were reasonable and manageable. Team members completed assigned tasks in time, held regular meetings, and helped each other as needed.

5.3 Third sprint

In the third sprint, we mainly focused on implementation of backend, frontend and API specification and UX testing:

- Patrik Villant created prototype, UX test scenarios and UXTweak test cases.
- Dominik Bucko set up a virtual machine, Raspberry Pi for LoRaWAN servers.
- Peter Mačuga and Ján Balucha finished first phase API specification.
- Peter Mačuga, Ján Balucha, Dominik Bucko discussed back-end implementation details.
- Peter Mačuga and Ján Balucha finished first phase backend implementation.
- Peter Mačuga and Ján Balucha finished endpoints for devices, tags and locations
- Soňa Zhan and Michal Franczel created several subpages of the web application such as Login page, Dashboard and Device management

UX testing is prepared, we are only waiting for respondents to be tested using the UXtweak website. As for frontend, a significant part is already completed, but the implementation of two subpages are missing, which we expect to complete in the next sprint. The backend is

moving in the right direction, we completed fundamental parts that we will use in future development.

We managed to complete the majority of the planned tasks in the third sprint, but there were some that took longer to complete than we initially thought, particularly some implementation tasks on frontend as well as backend.

5.4 Fourth sprint

In the fourth sprint, mainly focus was to deploy backend and frontend to school VM and begin to use CI / CD to automate deploying changes onto our shared development environment. To do so we created subdomains for each part of the web application. Last but not least we finalize project documentation.

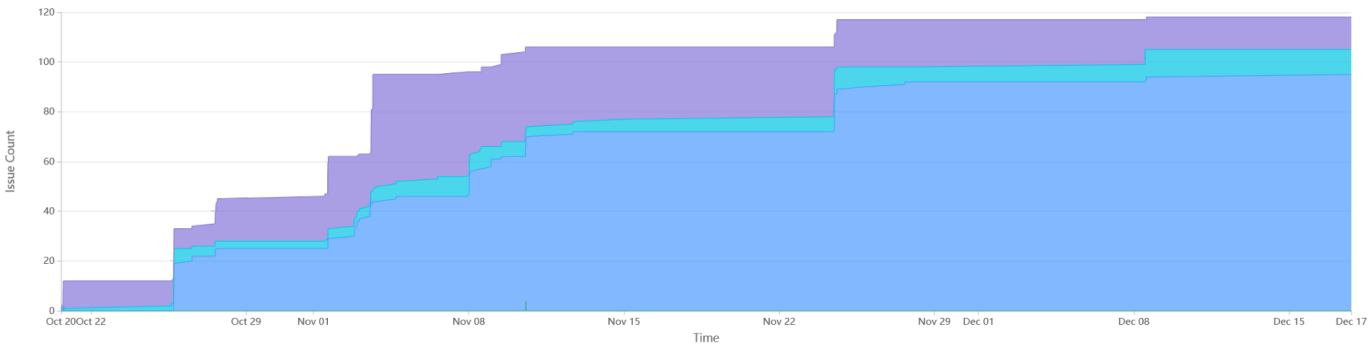
- Patrik Villant finalizes and evaluates UX testing of the application.
- Dominik Bucko implemented a Class A LoRaWAN device in the firmware of our development board, with over-the-air activation (OTAA) capability and sensor data transmission.
- Dominik Bucko set up LoRaWAN Gateway based on Raspberry Pi, installed a packet concentrator, Chirpstack application and network server.
- Peter Mačuga specified API endpoints for commands and firmware pages.
- Ján Balucha and Peter Mačuga implemented commands and firmware endpoints in backend
- Ján Balucha implemented filtering objects on the backend.
- Michal Franczel and Soňa Zhan created subpages for commands and firmware management.
- Michal Franczel and Dominik Bucko set up CI/CD pipelines for automated deployment of our services during development

Once again, we were successful in completing the majority of the tasks for sprint no. 4. Primarily, we were finally able to make progress on the firmware for our end devices, as we have finally had all the necessary hardware available. We also successfully deployed all of our services onto a virtual machine development environment, with CI/CD set up to automatically deploy any major changes.

6. Global retrospective

Compared to the first sprint, the tasks in the second and third sprint were much more time demanding. Overall, we did not correctly estimate the time required in the third sprint and due to other school duties we did not complete all the tasks, however we completed all important fundamental tasks for further development. The time and the status of the tasks

are shown in the following figure. The fourth sprint consisted of fewer tasks, but more complex and time consuming, which didn't boost graph as much as previous.



7. Winter semester summary

During the winter semester, we have managed to design and implement a significant part of our project. We have completed the majority of work on our web application, which will serve as a primary means of interaction of users with our system. We have also implemented a major part of the backend of our application, with defined API specification and database models. These two parts are integrated together and currently work with generated (mock) data, to simulate the final working application. We also managed to automate deployment of our services to a remote environment, which in our case is a virtual machine provided by school. All major changes that are merged in our frontend or backend application are automatically reflected in the environment on our virtual machine.

Late into the semester we have also managed to make progress on firmware for our devices. We have implemented a Class A LoRaWAN device, with capability for over-the-air activation (OTAA), which sends data measured by attached sensors to our application server. We have also set up a LoRaWAN gateway, running packet concentrator, network server and application server.

8. Methodology

8.1 Methodology of code formatting

8.1.1 Python

The entire code is written in English. The following code writing conventions are defined for the Python programming language:

- Constants are declared with capital letters.
- Classes will be declared using PascalCase.
- Variable names are declared using lowercase nouns.
- Function names are declared using lowercase verbs.
- For multi word names, these words are separated by the _ character

Comments that apply to one line of code are defined on the same line as the described code. The comment begins with a # character followed by a space, followed by a comment in English

Multi-line comments are also written in English. They begin and end with ''' characters, with these characters at the beginning and end on separate lines. These comments appear above the described block of code.

8.1.2 C or C++

- The same rules apply to function and variable names as for the Python programming language.
- Functions are defined so that the type, name and parameters are separated by a space and the body of the function will start on a new line
- Each comment will be written in English and will appear above the part of the code it describes. The comment is written between the characters "/ *" and "* /", these characters standing on separate lines. Comments begin with a "*" on each new line.

8.1.3 TypeScript

- To declare type names and enum values, we use the so-called PascalCase.
- We define functions, property names and local variables using camelCase.
- For functions, the character {, which indicates the beginning of the body of the

function, is on the line where the name and parameters of the function are located, and the character}, which indicates the end of the function, is at the very end on a new line.

- Each commentary will be written in English and will precede the part of the code it describes. The comment is written between the characters "/ *" and "* /", these characters standing on separate lines. Comments begin with a "*" on each new line.

8.2 Methodology of writing comments in code

Comments in the code will be in English. Each comment is placed in front of the object it describes. The comment is written between the characters "/ *" and "* /", these characters standing on separate lines. Comments begin with a "*" on each new line.
Example:

```
/*
* comment
* comment
*/
```

8.3 Methodology of writing documentation

We will use Microsoft Office software to write the documentation and it will be written in English.

Styles used in documentation:

- **Normal text:**

  Font: Times New Roman

  Size: 12pt

  Color: black

  Font type: normal

  Line spacing: 1,5 line

  Spacing before and after: 0b 6b

  Alignment: justify

  Indentation: special for 1. line

- **Title 1:**

  Font: Times New Roman

  Size: 18pt

  Color: black

  Font type: bold

  Line spacing: 1,5 line

Spacing before and after: 24b 0b

Alignment: left

Indentation: none

- **Title 2:**

  Font: Times New Roman

  Size: 16pt

  Color: black

  Font type: bold

  Line spacing: 1,5 line

  Spacing before and after: 10b 0b

  Alignment: left

  Indentation: none

- **Title 3:**

  Font: Times New Roman

  Size: 14pt

  Color: black

  Font type: bold

  Line spacing: 1,5 line

  Spacing before and after: 10b 0b

  Alignment: left

  Indentation: none

- **Title 4:**

  Font: Times New Roman

  Size: 12pt

  Color: black

  Font type: bold, italic

  Line spacing: 1,5 line

  Spacing before and after: 10b 0b

  Alignment: left

  Indentation: none

- **Figure and graph titles:**

  Font: Times New Roman

Size: 10pt

Color: black

Font type: normal

Line spacing: single

Spacing before and after: 6b 10b

Alignment: centered

Indentation: none

Object formats in text:

- **Chapter:**

  Every chapter is Title 1 and begins at new page

- **List:**

  Heading numbering represents a number and a period. For next level headings, the numbering is similar. Example: "1.", "2.1.", "3.2.1."

  In the case of a numbered list, enter a number, a period, and a parenthesis. Example: "1.)"

  In the case of an unnumbered list, the first level is recorded as a dot and the second level as a hyphen. Example: "●", "-"

- **Figure:**

  Figures are numbered

  Each image contains a description below it

  There are references to pictures in the text

- **Table:**

  Tables are numbered

  Each table contains a description below it

  There are references to tables in the text

- **Links, sources and literature:**

  Each source must be listed according to ISO 690

8.4 Methodology of communication

For communication within the team, we decided to use the Signal and Discord tools. The Signal will be used mainly for communication with the entire team and the project leader. Discord will allow communication within the created channels. These channels help us maintain clear communication related to individual areas of the project. At the same time,

Discord also offers voice channels, which we can use for debugging but also for pair programming.

Current channels for communication:

- #general - The main channel for general project and team stuff
- #documentations - Documentation communication channel
- #jira - A channel for things related to the management tool
- #firmware - Questions, information, and everything to do with firmware development
- #app_server - Questions, information, and everything to do with server development
- #database - A channel for database and database-focused discussion
- #team_web - Everything about the project website
- # merge-request - Notifications for merge requests mediated by a discord bot
- #app_web - Questions, information, and everything related to web interface development

8.5 Methodology of version control

We use the git version system for version management and we use the GitLab tool to back up individual repositories for the project.

8.5.1 Terms used

- Branch - A branch with the project code, or part of it.
- Commit - Upload new changes to the repository.
- Merge - The merging of two branches.
- Conflict - If there are different codes in different branches in the same file in the same place, a conflict will occur during the “merge” action, which the programmer must resolve manually.
- Pull-Request - The developer's "request" for approval to add new changes to the requested branch. If the pull-request is open, it is possible to perform code-review and possible adjustments in the form of further commits before the branch merges with another branch.
- Rebase - If new changes are added to main branch A after creating auxiliary branch B from the main branch, a rebase must be performed on the auxiliary branch. This means that all new commits from the main branch are first applied to the second branch (where the original commits were temporarily removed) and then the flood

commits from branch B. are applied. which occur during the rebase action.

8.5.2 Branching

The project uses the “Git Flow” methodology and thus two main branches are created, while any task represents another branch which, after successful completion, must be merged into one of the main branches - develop.

- main - The main project branch, which contains the production code of the project and thus only its verified and functional parts. Only the person responsible for versioning has the right to add new functionalities to the main branch and is responsible for the correctness of the merged files.
- develop - The secondary main branch of the project. It is the branch from which auxiliary branches are created for individual tasks and into which these auxiliary branches are merged. Commits must not be created directly into this branch as well as into the master branch. Each additional functionality must first be implemented in an auxiliary branch, which can be merged into the develop branch only after a successful code review. After each merging of the auxiliary branch, the latest version of the application in this branch must be executable.
- auxiliary branches - Each developer will create a auxiliary branch to implement the new functionality, in which he will create commits and which as a whole will represent the new functionality, this branch will be named feature- *. If you need to fix the bug as soon as possible, a branch named hotfix- * can be created directly from the main or develop branch. An auxiliary branch called release- * can be used to create a new production version. If a developer wants to merge an auxiliary branch, he must follow certain principles - specified below. Only one developer works in one auxiliary branch at a time.

In order to merge the auxiliary branch into the parent branch, the code must be sufficiently and clearly commented, checked in the form of a code review and tested. If everything goes well, the auxiliary branch can be merged into the parent branch.

8.5.3 Pull-request

- Each pull-request branch is subject to the following rules:
- There are no errors in the branch and the functionality of the branch is tested.

- The branch is compilable and the compilation does not report any errors or warnings.
- The rules of the branch to which the child branch is added are parent.
- Each pull-request must be approved in the form of a code review.
- Each pull-request must be approved by at least 2 developers.

A pull-request is a developer's request to check its code that it wants to merge into a parent branch. The check must be performed by 2 developers in addition to the applicant. The developer takes a task from the board that represents the review and approval of the branch requesting the pull-request, performs the check, and if the check is successful, allows the branch to be merged. If the check is not successful, it will notify the developer who worked on the branch of the found errors and deficiencies, who must eliminate the errors.

8.5.4 Commit

Commit contains all new changes that have occurred in the repository. All commits created by the developer are stored locally until the developer decides to upload the commits to remote storage, in our case GitLab. Each commit contains a message in the form of a list of changes in functionality and a short description of them. The message must be short, concise and it must be clear what changes the developer has made.

Commit name rules:

- It starts with a capital letter
- English is used.
- The message has a maximum of 50 characters
- The message does not end with a period
- The message starts with a noun (Add instead of Added / Adding / ..)

For each project created in git, it is necessary to create a .gitignore file so that we do not upload unnecessary files created by the IDE, operating system and SDK to gitlab.

We will create such a file according to the system and language of the project we are working on.