Bring ideas to life
VIA University College

Mushroom++

mpp

IoT group: Daria Maria Popa (293087), Natali Munk-Jakobsen (293132) and Mihai Anghelus (293169)

Data group: Levente Nagy (293115), Audrius Sauciunas (293156), Samantha Nettesheim (293089), Shaoyu Liu (294404)

Android group: Kristóf Lénárd (293110), Bogdan Mezei (293137), Uldis Alksnis (293168)

Supervisors: Erland Ketil Larsen, Astrid Hanghøj and Kasper Knop Rasmussen

VIA University College

131868 characters, of which 40000 images

Software Technology Engineering

4th Semester

4th June 2021

Table of content

Abstract

The goal of this project is to create a system that is able to satisfy the requirements of smaller-scale mushroom growers, and enable them to cultivate said mushrooms more effectively.

To achieve this, three subsystems were created – an IoT solution for real-time measurements and actuation, a central server for data warehousing and connectivity, and an Android app as frontend. These are connected by a WebSocket-based connection and a RESTful Web API, respectively.

In this report, besides the problem domain analysis, the technical aspects of this complex system will be detailed. Afterwards, the results will be summarized, and potential expansions and improvements will be detailed.

1 Introduction

Mihai Anghelus

People are starting to pay more attention to what they eat. Mushrooms, being low in fat and cholesterol have grown in demand because of it. They are rich in vitamins, minerals, fiber and high-quality protein and possess incredible health benefits. This caused their marked size to be on the rise (Industry Reports, Mushrooms, 2018).

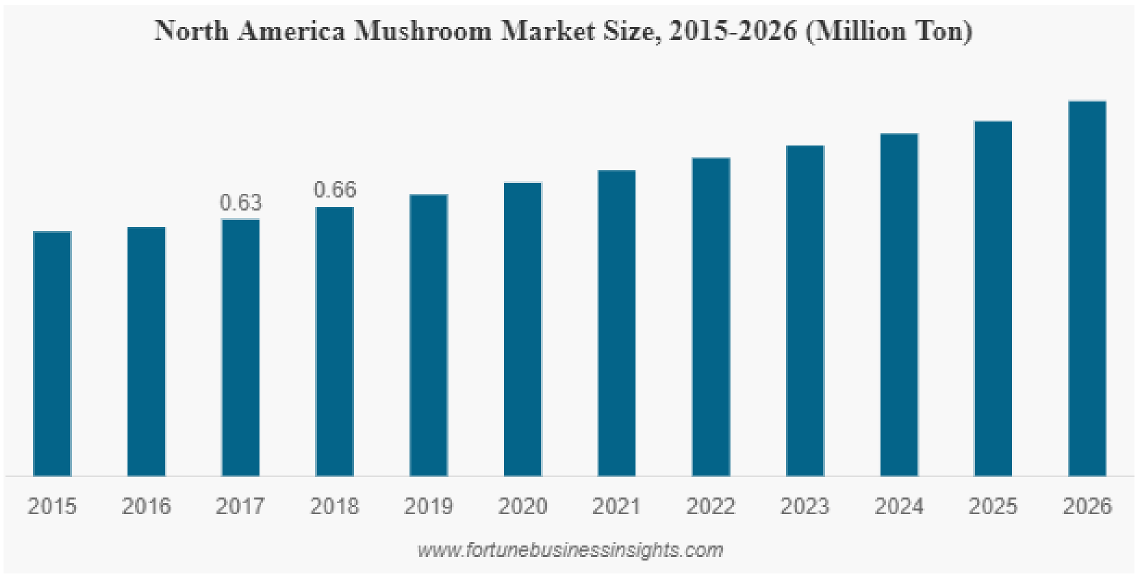


Image 1.1: Market share of mushrooms in the USA

While big companies have the necessary equipment to semi-automate the production of mushrooms, people who do this as a hobby or on a smaller scale lack easy access to it. Mushrooms require very specific precise growth conditions and have a small error margin for optimal growth. Without precise measurement tools that keep track of the humidity, light levels, and temperature at all times, cultivating mushrooms becomes a tedious task. This is why there is a growing market aimed towards helping the smaller mushroom growing enthusiasts (Industry Reports, Mushrooms, 2018).

Mushrooms can be grown both outdoors and indoors. Growing them outside is a more difficult task because there are a variety of uncontrollable factors, therefore many farmers grow them indoors, where the space can be better controlled and the variables have better consistency.

However, to this day, commercially available nonindustrial hardware and or software, tackling problems concerning mushroom cultivation are almost nonexistent (Industry Reports, Mushrooms, 2018).

2 Analysis

Levente Nagy

As written in the Project Description, the problem domain revolves around the obstacles concerning mushroom cultivations in smaller-scale environments. The following problems were recognized, not all of which are going to be solved in every aspect, as the toolset for this project is based on primarily software solutions and secondarily hardware solutions.

- Maintaining environmental properties for best mushroom growth, as mushrooms have lower tolerances towards the changing of some properties. (Chang and Miles, 2004)
- Managing and altering environmental properties for optimal mushroom growth according to the current growth stage and needs of the mushroom, as the optimal properties can heavily vary from the maturity and type of mushroom. (Chang and Miles, 2004)
- Identifying the optimal environmental properties and values during the cultivation process.
- Remotely monitoring the cultivation, as non-industrialized environments are more prone to impacts caused by the changing of the ambient environment.

Based on these problems, the focus was directed towards creating a centralized cloud-based service for collecting data of mushroom cultivation instances, making it possible to analyze and identify optimal environmental properties based on the success of past cultivations. To monitor and manage certain environmental properties, a hardware device also has to be introduced for each cultivation. This hardware would be responsible for both monitoring and management of environmental properties to ensure the wellbeing of the mushrooms.

2.1 Requirements

The following Functional Requirements were formulated to support a cloud-based solution, described in the previous subsection. The Non-Functional Requirements were established by the stakeholders.

2.1.1 Functional Requirements

Functional requirements were developed to capture what the system should perform.

Must have:

1. As a user I want to have a personal account, to distinguish myself and limit my access in the system.
2. As a user I want to be able to monitor the air temperature with my hardware.
3. As a user I want to be able to monitor the air humidity with my hardware.
4. As a user I want to be able to monitor the air carbon dioxide level with my hardware.
5. As a user I want to submit the status of my specimens associated with my hardware, to provide useful data for future analysis.
6. As a user I want all sensory data to be collected, to provide useful insights on my environmental conditions.
7. As a user I want the collected data to be accessible for me in a visualized way, where I can see the correlations between them.
8. As a user I want to be able to manage the air humidity with my hardware.
9. As a user I want the client application to work offline and persist data history.

Should have:

10. As a user I want to be able to keep a virtual diary of mushroom growing periods, so that I can keep track of mushroom growth.
11. As a user I want to be able to set up deadlines of things I must do, so that I could avoid potential growing failure.
12. As a user I want to be able to manage the air temperature with my hardware.
13. As a user I want to be able to manage the air carbon dioxide level with my hardware. - NOTE (May 19th): incompletable due to lack of hardware.
14. As a user I want to be able to monitor the light level with my hardware.
15. As a user I want to be able to manage the light level with my hardware.

Could have:

16. As a user I want to be able to monitor the soil humidity with my hardware. - NOTE (May 19th): incompletable due to lack of hardware.

17. As a user I want to be able to manage the soil humidity with my hardware. - NOTE (May 19th): incompletable due to lack of hardware.

Will not have:

18. As a user I want to be able monitor the soil temperature with my hardware.
19. As a user I want to be able to manage the soil temperature with my hardware.
20. As a user I want to be able to provide a clean environment for my farm, to avoid cross contamination.

2.1.2 Non-Functional Requirements

The non-functional requirements describe how the system should work.

Must have (IoT):

1. The interface to the sensors and actuators must be a AVR2560 MCU.
2. The AVR microcontroller must use FreeRTOS based application in C.
3. The hardware must use LoRaWAN to transmit and receive data.
4. Bridge application between the Loriot Network Server and the Database, must use web sockets.
5. The FreeRTOS application must use at least five tasks.
6. Some data in the FreeRTOS application must be used by more than one task.
7. The FreeRTOS application must use semaphores, mutexes and queues.
8. The FreeRTOS application must have some unit tests.
9. The Gateway application must be java based.

Must have (Data Engineering):

10. The database must utilize dimensional modelling.
11. The analysis of the data must be done using Microsoft Power BI.
12. Web services must be exposed for consumption in the frontend.

Must have (Interactive Media):

13. The frontend must be developed using the official Android framework.
14. The frontend must be developed with Java.
15. The frontend must follow the Google Material Design guidelines.
16. The frontend must be structured using an architectural pattern.

2.2 Use Cases

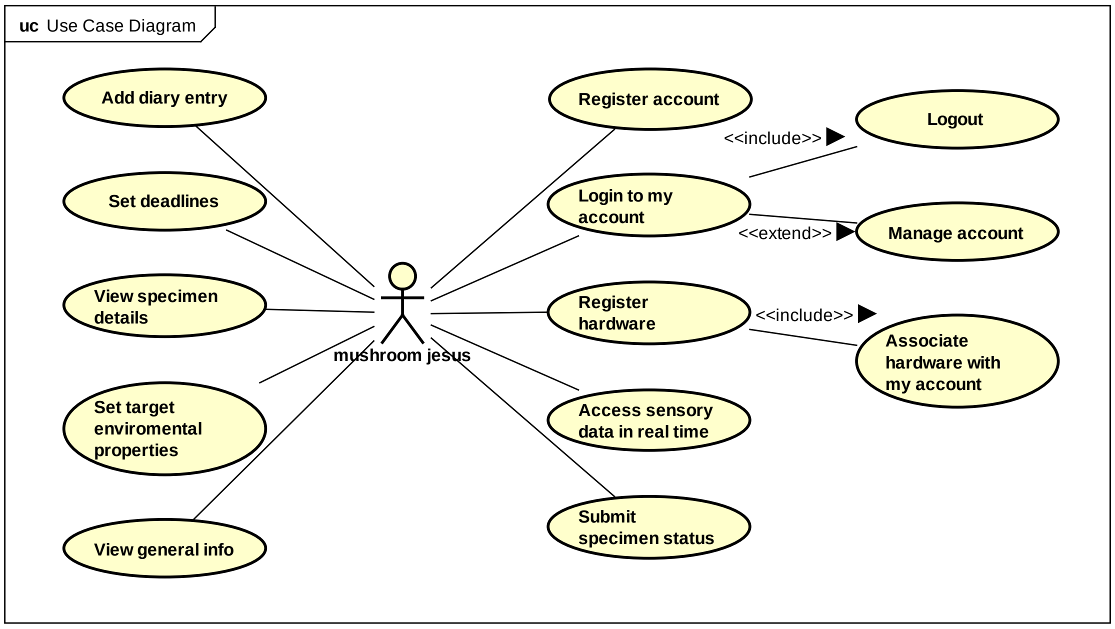


Figure 1 Use Case Diagram

The above diagram shows the use cases derived from the functional requirements. These were later developed into detailed use cases, with use case descriptions and activity diagrams.

Use case	Access sensory data in real time
Summary	The user could view real-time environmental data passed from sensor
Pre-condition	The user has already registered and set up the hardware
Post-condition	System presents related data to the user
Base sequence	1.User chooses to view sensory data. 2.User views all sensory data(air temperature, air humidity, air carbon dioxide, light level, etc)
Exception	Sensory data not able to be visualized due to technical errors
Note	All the data is real-time, so the user won't have to do anything to view changing data.

2.3 Domain Model

The following domain model was established, to support the Use Cases and Functional requirements described previously.

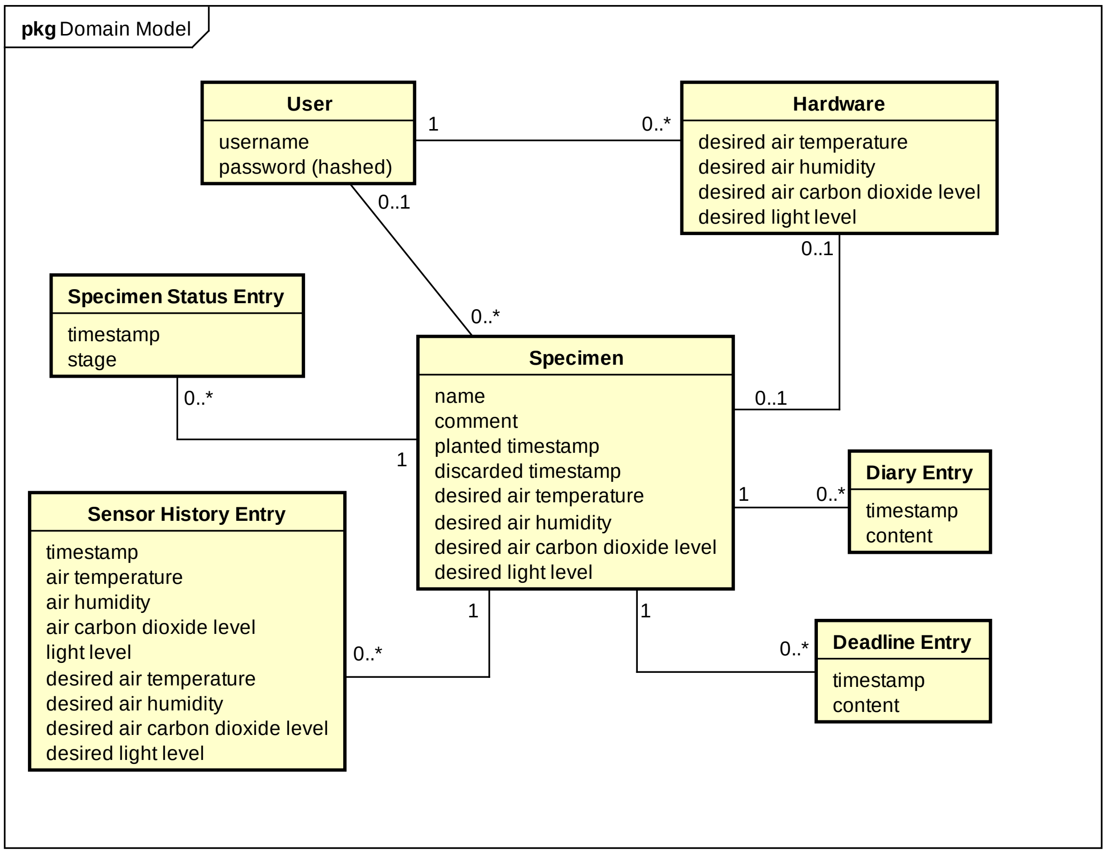


Figure 2 Domain Model

Most of the domain objects are visibly derived from the Use Cases and the Functional Requirements, although the differentiation between the hardware and specimen is not evident. The hardware object represents the physical hardware, deployed by the end user, and the specimen represents the abstraction of a cultivation instance. Both entities can exist without each other, as a hardware can exist without having an active mushroom cultivation associated with it. As well as a specimen can remain in the system as an archive entry of a past cultivation, without having a physical hardware. Nevertheless, when a specimen is having an active cultivation with an assigned hardware, the two objects are coupled in a one-to-one relationship.

3 Design

3.1 System Architecture

Levente Nagy

The following system architecture was defined first, to get a general understanding and outline of the components supporting the project. Most of the non-functional requirements were influencing, or even limiting the design choices, regardless, the following design patterns and architectural properties were heavily prioritized:

- Chain of responsibility pattern
- Logical and physical separation between components for easy maintainability
- The clean architecture dependency pattern

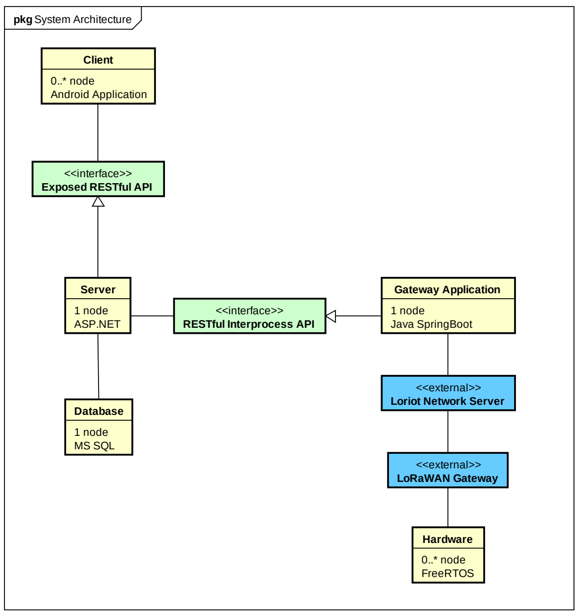


Figure 3 – System architecture

To clarify some of the design decisions, the chain of responsibility pattern is starting with the ASP.NET Web API Server, as this is the brains of the system, handling the business logic, also this is the only node having access to the source [dbo] database. (From now on, this document is going to reference to this node as Server or Web API Server)

The Gateway Application was heavily influenced by non-functional requirement number 9, as this node could have been a class library, used by the Server, instead of a separate entity. This design resulted in a bit higher complexity, but also resulted in physical separation, for ease of use.

Due to the nature of the development process, it was a high priority task to clarify and define the joins of the 3 main parts of the system:

- Client

- Business Logic & Data (includes the Server and the Database)
- Hardware (includes the Gateway and the Hardware itself)

The joins are the two RESTful API interfaces, defined between these entities, reducing the friction between various parts, while helping to follow the Clean Architecture Dependency Pattern.

By this diagram, it is not evident, that functional requirement number 7 is supported in the Database node, as it also contains an Enterprise Data Warehouse, for supporting the analyzation of the collected data with Power BI reports.

The currently developed system follows the previously established Design, with the PBI reports being published as a read only document.

The following alternative designs were made but were not developed due to the hardware and/or enterprise software requirements.

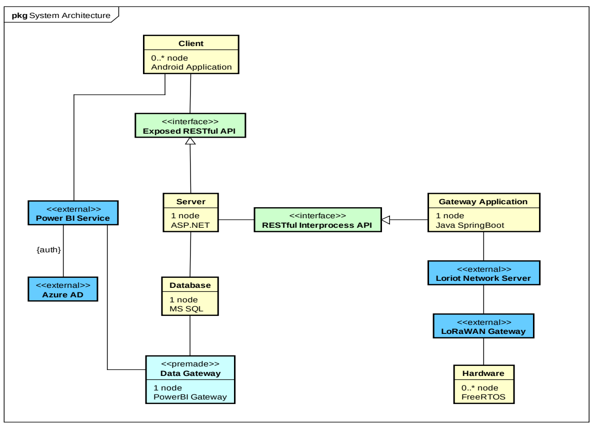


Figure 3 PBI Cloud Service

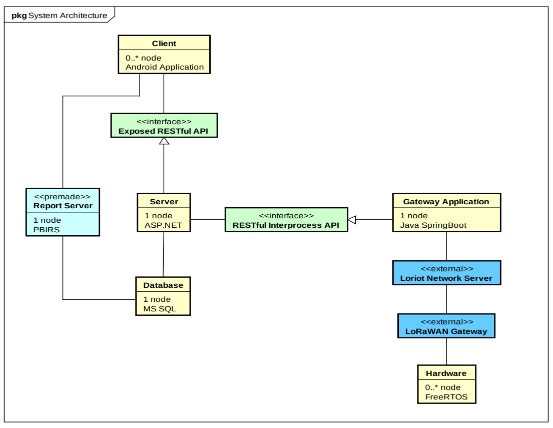


Figure 5 PBI Report Server

These two designs both support the live access of the PBI reports. Figure 3 uses Microsoft's cloud Azure services, which are not accessible freely, thus this design, was dropped and unused. Figure uses a PBI report server which starts out as an x86-64, windows only product, with high hardware demands. Due to these factors, this design was also left unexplored, as the team lacks a hardware which satisfies the said requirements.

3.2 Persistence

Samantha Nettesheim, Shaoyu Liu and Audrius Sauciunas

The entire system contains one or more sensors in the mushroom cultivation house. The database modeling is done here to facilitate retrieving, handling, and sending data, and avoid problems caused by data overloading. With the database modeling design, the relevant logic concerning the persistent storage of the entire system for data management is provided. Microsoft SQL Server service is chosen as the common tool for members to access data. Data Manipulation Language (DML) and Data Query Language (DQL) are used to query within schema objects and obtain data.

3.2.1 Conceptual Model

A conceptual model is developed to present an overall picture of the system by recognizing the business objects involved. Entities such as user, hardware, specimen, sensor entry and status entry make up the database prototype. The relationships with zero cardinality between each entity can be found in Figure 4.

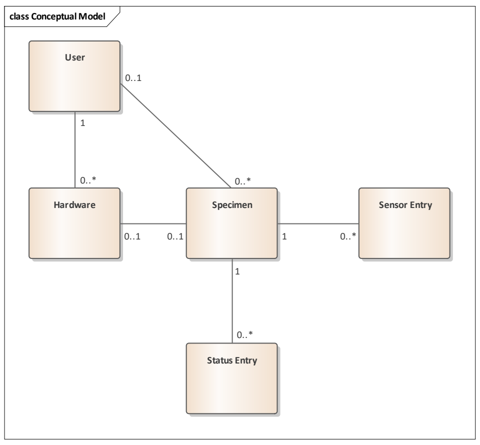


Figure 4 Conceptual Model

3.2.2 Logical Model

The logical model is a detailed version of the conceptual model. A logical model is made to enrich a conceptual model by defining the columns in each entity and introducing both operational and transactional entities. Figure 5, illustrates the logical model of the database.

To meet requirement number 1, the basic personal information of a user is obtained. In this case, the user's username, and password. In addition, the user's permission levels for a small number of other users, for instance, the developer and administrator, to allow various levels of access.

To fulfill functional requirements 2, 3 and 4, the details of hardware and environmental sensors including the equipment itself, target values and detected values are stored. In this case, desired environmental values, the sensor entries (in our case by minute) and real time environmental values for each timestamp are taken into consideration.

To fulfill functional requirements 5 and 10, the information of mushrooms by certain categories is retained. In this case, all mushrooms that are relevant to the system are present by species and moreover, by genus. Records for mushroom growing stages are utilized to track the cultivation lifecycle period.

The specimen details are tracked by timestamp (in minutes). In this case, the specimen details include planted date, assumed discarded date, desired target environment as static values, real-time environmental values by minute and timestamps as the smallest time unit in the database.

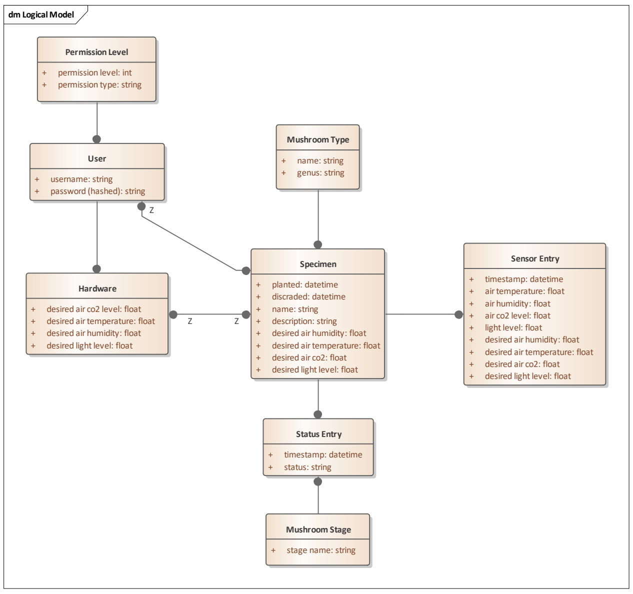


Figure 5 Logical Model

3.2.3 Physical Model

The physical model represents the actual design blueprint of the database. Based on the logical model, the physical model is created by including all data types, lengths, whether an entity is nullable, measures, entity relationships, etc. to implement the database. Since a physical model represents how data should be structured and related in the real database it is important to consider the restrictions and conventions of the database system, in which the database is created.

The physical model also includes a couple extra fields like surrogate keys and foreign keys, as well as a user_token and hardware_id fields to support operations concerning the LoRaWAN network. These fields were not included in the logical model, because they effectively have no value from the perspective of the business process.

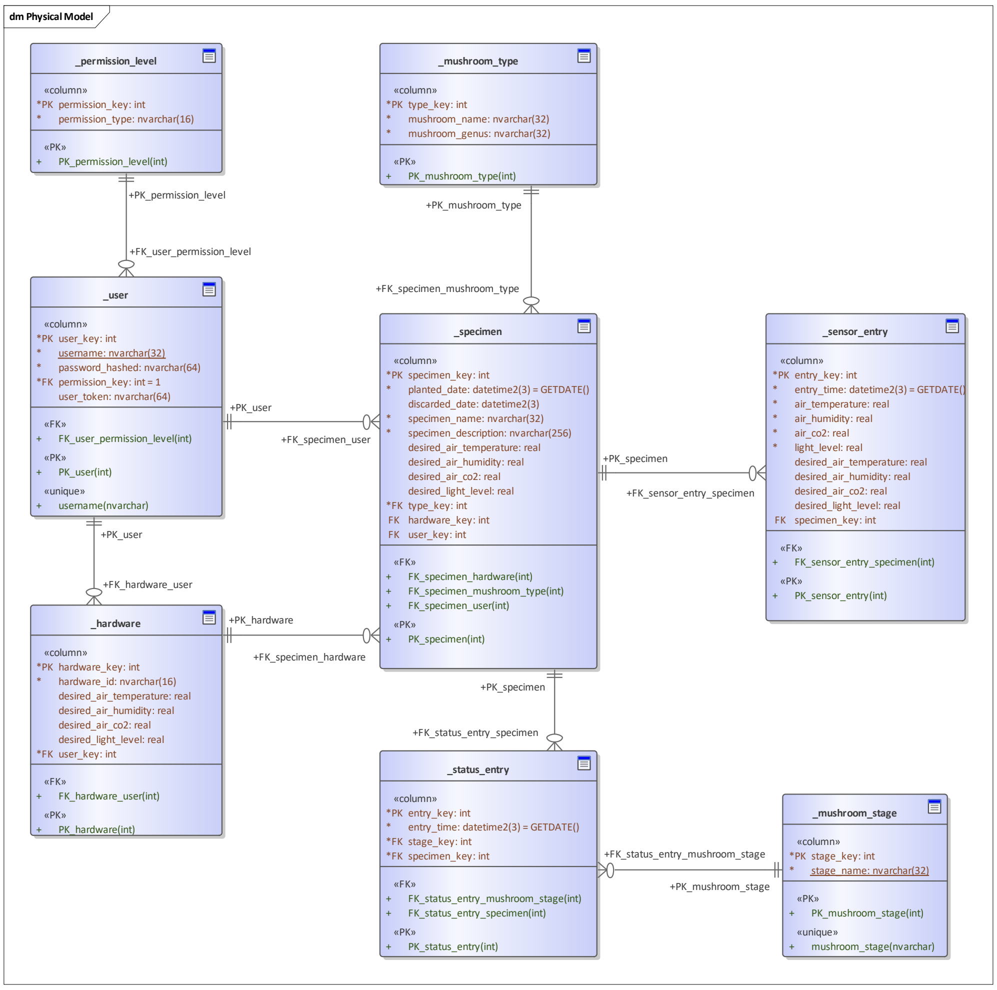


Figure 6 Physical Model

3.3 The APIs

Levente Nagy

As previously stated in the System Architecture section, there are two RESTful APIs in the system, connecting the various parts. The design of these APIs is crucial to the architecture, which began with the design of the unified JSON objects, to ensure coherent communication between various parts of the system.

The documentation of these APIs was created in Swagger, and the full detailed documentation of the APIs are presented in Appendix F.

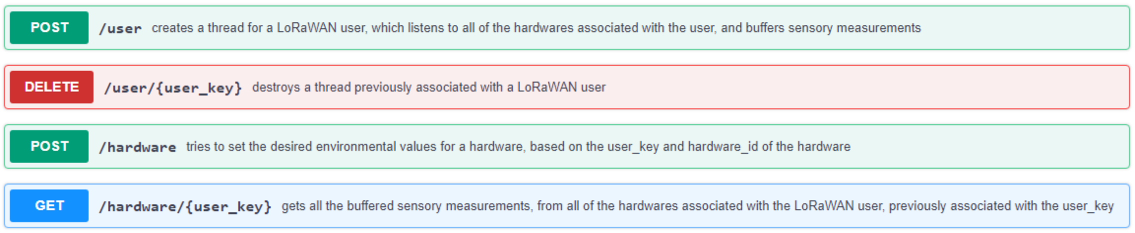
Five JSON objects were defined for communication between the client and the Web API: user, hardware, specimen, sensor_entry and status_entry.

After establishing the entities, which support the transmission of all the necessary data, the API endpoints seen in the Swagger documentation were defined (see appendix F). Most of the endpoints are CRUD operations for each model object. It is important to note that the fields in some of the models are read-only, nullable, write-only or completely unused in case of some of the operations. Three endpoints are obvious exceptions from the previously stated CRUD approach, these 3 GET methods are the defined/mushroom/stages, defined/mushroom/types and token endpoints. The “defined” endpoints are returning the current contents of the _mushroom_type and _mushroom_stage tables of the source database, as basically these values are acting as constraints for some of the other fields in the models.

Before the token endpoint is detailed in this document, it must be presented that most of the endpoints of the API use Bearer and Basic JWT Authorizations. In theory all the endpoints accept Basic Authorization formats, but it is a good practice to only retrieve the Bearer token using the Basic Authorization on the first login, and then use the Bearer token and Bearer Authorization for better security measures. The “token” endpoint is used to retrieve the Bearer JWT, with Basic Authorization. Alternatively, this endpoint can also be used to renew the Bearer token, using an already retrieved older token, with Bearer Authorization (this only makes sense, if the Server is configured to attach expiration dates to the JWTs).

The JSON objects used by the Gateway API, do match with the models used in the Web API, except for a few unused fields and a couple extra fields per object. (Further details are available in Appendix F)

Next, the endpoints of the Gateway API were defined by the IoT team (Figure 11).



POST `/user` creates a thread for a LoRaWAN user, which listens to all of the hardwares associated with the user, and buffers sensory measurements

DELETE `/user/{user_key}` destroys a thread previously associated with a LoRaWAN user

POST `/hardware` tries to set the desired environmental values for a hardware, based on the user_key and hardware_id of the hardware

GET `/hardware/{user_key}` gets all the buffered sensory measurements, from all of the hardwares associated with the LoRaWAN user, previously associated with the user_key

Figure 11 Gateway API Endpoints

The Gateway is responsible for connecting the hardware and the Web API Server. As the Gateway ended up being a separate entity, rather than a class library (previously discussed in the System Architecture section), it has a bit more complexity, as it is able to track multiple LoRaWAN users' hardware and collect/cache sensory measurements, from the said hardware. While it is also capable to send downlink messages to the hardware, setting the desired environmental values.

The Gateway API does not contain any kind of security measurements, as the communication between the Gateway and the Server happens behind a firewall, as an inter-process communication.

Although the Gateway is capable of some complex operations, it is not responsible for any business logic, in fact it is not dependent of anything else other than the LoRaWAN network and the hardware. Meaning, the Gateway is unaware of all the business processes like Specimens, Status Entries, the Database itself and more.

3.4 Web API Server

Levente Nagy

After the establishment of the APIs, the three departments (respectively Android, Data and IoT) could work on the distinct parts of the system independently, as the point of frictions are eliminated. Thus, the various parts are handled as standalone entities from now on in this document.

This section will detail the design of the Web API Server, and the business logic behind the system.

First the model classes were defined, which closely resemble the model classes defined in the APIs.

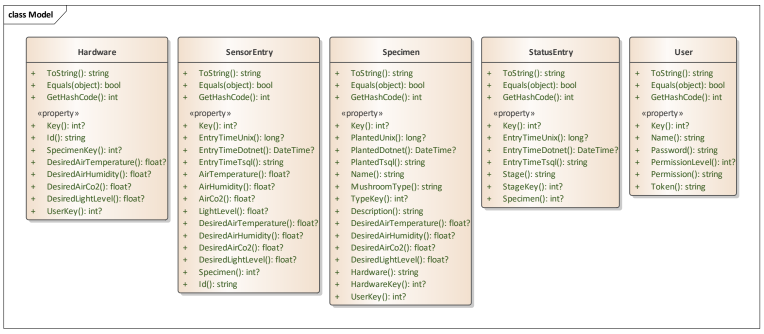


Figure 7 Web API Model Classes

Each class' properties have their own [JsonPropertyName("")] or [JsonIgnore] annotations to differentiate between properties which are only needed to transact internal logic, and properties which are needed to be sent over to other parts of the system. The last part to mention is the handling of the datetimes, as this system is a heterogenous system, the established unified datetime format used between parts is the Unix Epoch Millis format. For each class using a timestamp, the datetime is stored in the EntryTimeUnix property, and the EntryTimeDotnet and EntryTimeTsql properties convert to/from the Unix format. This is a convenient way to use these classes, as depending on the place where it is used (e.g., DAO objects) the correct property can be used to access the field, and the conversion happens seamlessly in the model class itself.

After the Model Classes were established, the Controllers were the next logical step, as they closely follow the API endpoints established in the previous The APIs.

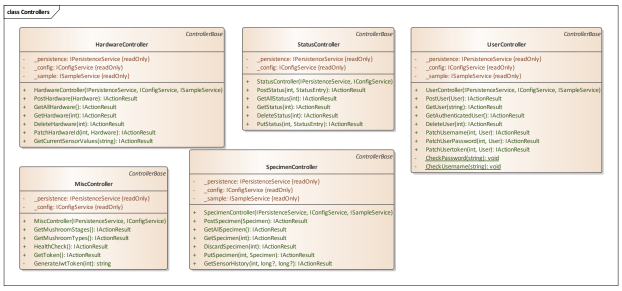

class Controllers
ControllerBase
HardwareController
- _persistence: IPersistenceService {readOnly}
- _config: IConfigService {readOnly}
- _sample: ISampleService {readOnly}
+ HardwareController(IPersistenceService, IConfigService, ISampleService)
+ PostHardware(Hardware): IActionResult
+ GetAllHardware(): IActionResult
+ GetHardware(int): IActionResult
+ DeleteHardware(int): IActionResult
+ PatchHardwareId(int, Hardware): IActionResult
+ GetCurrentSensorValues(string): IActionResult
ControllerBase
StatusController
- _persistence: IPersistenceService {readOnly}
- _config: IConfigService {readOnly}
+ StatusController(IPersistenceService, IConfigService)
+ PostStatus(int, StatusEntry): IActionResult
+ GetAllStatus(int): IActionResult
+ GetStatus(int): IActionResult
+ DeleteStatus(int): IActionResult
+ PutStatus(int, StatusEntry): IActionResult
ControllerBase
UserController
- _persistence: IPersistenceService {readOnly}
- _config: IConfigService {readOnly}
- _sample: ISampleService {readOnly}
+ UserController(IPersistenceService, IConfigService, ISampleService)
+ PostUser(User): IActionResult
+ GetUser(string): IActionResult
+ GetAuthenticatedUser(): IActionResult
+ DeleteUser(int): IActionResult
+ PatchUsername(int, User): IActionResult
+ PatchUserPassword(int, User): IActionResult
+ PatchUsertoken(int, User): IActionResult
- CheckPassword(string): void
- CheckUsername(string): void
ControllerBase
MiscController
- _persistence: IPersistenceService {readOnly}
- _config: IConfigService {readOnly}
+ MiscController(IPersistenceService, IConfigService)
+ GetMushroomStages(): IActionResult
+ GetMushroomTypes(): IActionResult
+ HealthCheck(): IActionResult
+ GetToken(): IActionResult
- GenerateJwtToken(int): string
ControllerBase
SpecimenController
- _persistence: IPersistenceService {readOnly}
- _config: IConfigService {readOnly}
- _sample: ISampleService {readOnly}
+ SpecimenController(IPersistenceService, IConfigService, ISampleService)
+ PostSpecimen(Specimen): IActionResult
+ GetAllSpecimen(): IActionResult
+ GetSpecimen(int): IActionResult
+ DiscardSpecimen(int): IActionResult
+ PutSpecimen(int, Specimen): IActionResult
+ GetSensorHistory(int, long?, long?): IActionResult

Figure 8 Web API Controller Classes

The controller classes use dependency injections for the upcoming services which are yet to be defined. Other than the methods responsible for the API endpoints, the UserController has two extra static methods which are responsible for checking the username and password constraints, as these constraints are easier to define here, utilizing regex, rather than in the database. The MiscController has a GenerateJwtToken method, which is responsible for generating the JWT token, containing the key of the user. The JWT token only contains the user key because the bearer authentication uses the database on premise, to establish the users' privileges, thus no further data is needed in the token, as all other information is queried, based on the said key. Finally, there is one extra endpoint in the MiscController, called HealthCheck. This method is not defined in the APIs, as it has no business value (its only responsibility is to return 200 Ok upon request), and its role concerns the deployment of the system, which this document will not discuss.

Next, the PersistenceService (DAO), SampleService (stands for sampling service, which is responsible for connecting to the Gateway and sampling the sensory measurements), ConfigService, LogService and JwtMiddleware classes were defined.

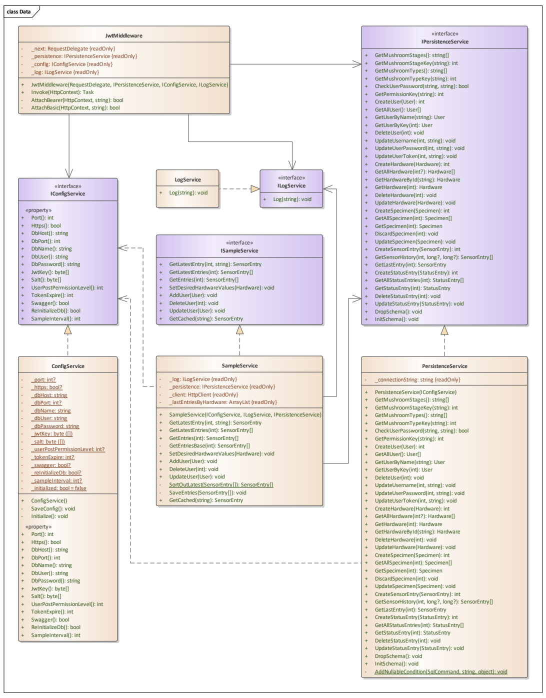

class Data
JwtMiddleware
- _next: RequestDelegate {readOnly}
- _persistence: IPersistenceService {readOnly}
- _config: IConfigService {readOnly}
- _log: ILogService {readOnly}
+ JwtMiddleware(RequestDelegate, IPersistenceService, IConfigService, ILogService)
+ Invoke(HttpContext): Task
- AttachBearer(HttpContext, string): bool
- AttachBasic(HttpContext, string): bool
«interface» IPersistenceService
+ GetMushroomStages(): string[]
+ GetMushroomStageKey(string): int
+ GetMushroomTypes(): string[]
+ GetMushroomTypeKey(string): int
+ CheckUserPassword(string, string): bool
+ GetPermissionKey(string): int
+ CreateUser(User): int
+ GetAllUser(): User[]
+ GetUserByName(string): User
+ GetUserByKey(int): User
+ DeleteUser(int): void
+ UpdateUsername(int, string): void
+ UpdateUserPassword(int, string): void
+ UpdateUserToken(int, string): void
+ CreateHardware(Hardware): int
+ GetAllHardware(int?): Hardware[]
+ GetHardwareById(string): Hardware
+ GetHardware(int): Hardware
+ DeleteHardware(int): void
+ UpdateHardware(Hardware): void
+ CreateSpecimen(Specimen): int
+ GetAllSpecimen(int): Specimen[]
+ GetSpecimen(int): Specimen
+ DiscardSpecimen(int): void
+ UpdateSpecimen(Specimen): void
+ CreateSensorEntry(SensorEntry): int
+ GetSensorHistory(int, long?, long?): SensorEntry[]
+ GetLastEntry(int): SensorEntry
+ CreateStatusEntry(StatusEntry): int
+ GetAllStatusEntries(int): StatusEntry[]
+ GetStatusEntry(int): StatusEntry
+ DeleteStatusEntry(int): void
+ UpdateStatusEntry(StatusEntry): void
+ DropSchema(): void
+ InitSchema(): void
LogService
+ Log(string): void
«interface» ILogService
+ Log(string): void
«interface» IConfigService
«property»
+ Port(): int
+ Https(): bool
+ DbHost(): string
+ DbPort(): int
+ DbName(): string
+ DbUser(): string
+ DbPassword(): string
+ JwtKey(): byte[]
+ Salt(): byte[]
+ UserPostPermissionLevel(): int
+ TokenExpire(): int
+ Swagger(): bool
+ ReInitializeDb(): bool
+ SampleInterval(): int
«interface» ISampleService
+ GetLatestEntry(int, string): SensorEntry
+ GetLatestEntries(int): SensorEntry[]
+ GetEntries(int): SensorEntry[]
+ SetDesiredHardwareValues(Hardware): void
+ AddUser(User): void
+ DeleteUser(int): void
+ UpdateUser(User): void
+ GetCached(string): SensorEntry
ConfigService
- _port: int?
- _https: bool?
- _dbHost: string
- _dbPort: int?
- _dbName: string
- _dbUser: string
- _dbPassword: string
- _jwtKey: byte ([])
- _salt: byte ([])
- _userPostPermissionLevel: int?
- _tokenExpire: int?
- _swagger: bool?
- _reInitializeDb: bool?
- _sampleInterval: int?
- _initialized: bool = false
+ ConfigService()
- SaveConfig(): void
- Initialize(): void
«property»
+ Port(): int
+ Https(): bool
+ DbHost(): string
+ DbPort(): int
+ DbName(): string
+ DbUser(): string
+ DbPassword(): string
+ JwtKey(): byte[]
+ Salt(): byte[]
+ UserPostPermissionLevel(): int
+ TokenExpire(): int
+ Swagger(): bool
+ ReInitializeDb(): bool
+ SampleInterval(): int
SampleService
- _log: ILogService {readOnly}
- _persistence: IPersistenceService {readOnly}
- _client: HttpClient {readOnly}
- _lastEntriesByHardware: ArrayList {readOnly}
+ SampleService(IConfigService, ILogService, IPersistenceService)
+ GetLatestEntry(int, string): SensorEntry
+ GetLatestEntries(int): SensorEntry[]
+ GetEntries(int): SensorEntry[]
- GetEntriesBase(int): SensorEntry[]
+ SetDesiredHardwareValues(Hardware): void
+ AddUser(User): void
+ DeleteUser(int): void
+ UpdateUser(User): void
- SortOutLatest(SensorEntry[]): SensorEntry[]
- SaveEntries(SensorEntry[]): void
+ GetCached(string): SensorEntry
PersistenceService
- _connectionString: string {readOnly}
+ PersistenceService(IConfigService)
+ GetMushroomStages(): string[]
+ GetMushroomStageKey(string): int
+ GetMushroomTypes(): string[]
+ GetMushroomTypeKey(string): int
+ CheckUserPassword(string, string): bool
+ GetPermissionKey(string): int
+ CreateUser(User): int
+ GetAllUser(): User[]
+ GetUserByName(string): User
+ GetUserByKey(int): User
+ DeleteUser(int): void
+ UpdateUsername(int, string): void
+ UpdateUserPassword(int, string): void
+ UpdateUserToken(int, string): void
+ CreateHardware(Hardware): int
+ GetAllHardware(int?): Hardware[]
+ GetHardware(int): Hardware
+ GetHardwareById(string): Hardware
+ DeleteHardware(int): void
+ UpdateHardware(Hardware): void
+ CreateSpecimen(Specimen): int
+ GetAllSpecimen(int): Specimen[]
+ GetSpecimen(int): Specimen
+ DiscardSpecimen(int): void
+ UpdateSpecimen(Specimen): void
+ CreateSensorEntry(SensorEntry): int
+ GetSensorHistory(int, long?, long?): SensorEntry[]
+ GetLastEntry(int): SensorEntry
+ CreateStatusEntry(StatusEntry): int
+ GetAllStatusEntries(int): StatusEntry[]
+ GetStatusEntry(int): StatusEntry
+ DeleteStatusEntry(int): void
+ UpdateStatusEntry(StatusEntry): void
+ DropSchema(): void
+ InitSchema(): void
- AddNullableCondition(SqlCommand, string, object): void

Figure 9 Web API Service and Middleware Classes

The JwtMiddleware class is part of the middleware pipeline, and responsible for attaching the "User" HttpContext to each call, containing the authenticated user. If there is no Authentication in the request, or the Authentication fails, then the "User" HttpContext stays null. The LogService is a simple logging service writing to the console, as well as appending to a log.txt file. This class mainly exists to help troubleshooting and logging in a live deployment environment. The ConfigService is responsible for reading and writing a config.json file, for easily configurable deployment. And finally, the PersistenceService contains all DAO methods which are needed to support the API endpoint operations. Configurable properties in ConfigService:

- "Port" port of the exposed API (if https is enabled, the https is on [port+1]).
- "Https" true/false enables or disables https on [port+1].
- "DbHost" host of the MSSQL server.
- "DbPort" port of the MSSQL server.
- "DbName" name of the database on the MSSQL server.
- "DbUser" username of the user used in MSSQL server (for mixed auth).
- "DbPassword" password of the user used in MSSQL server (for mixed auth).
- "JwtKey" 16 long cryptographically strong bytes, used to encrypt the JWT (it is randomly generated on initialization of the database).
- "Salt" 16 long cryptographically strong bytes, used to encrypt the passwords stored in the database (it is randomly generated on initialization of the database; WARNING: reinitializing, or losing the Salt, will result in the loss of all the stored passwords).
- "UserPostPermission" manages the ability of which users are having the privilege to register new users in the system. If set to 0, new users can be registered without Authentication, everything above 0 checks the Authorized User's permission level to be equal or larger than the setting.
- "TokenExpire" sets the expiration time of the generated JWTs in minutes, set to 0 for no expiration date.
- "Swagger" true/false enables or disables swashbuckler on /swagger/index.html path.

- “ReinitializeDb” true/false enables or disables the wiping, then initialization of the database on the next launch of the system.
- “SampleInterval” sets the sampling frequency in the SampleService, in minutes.

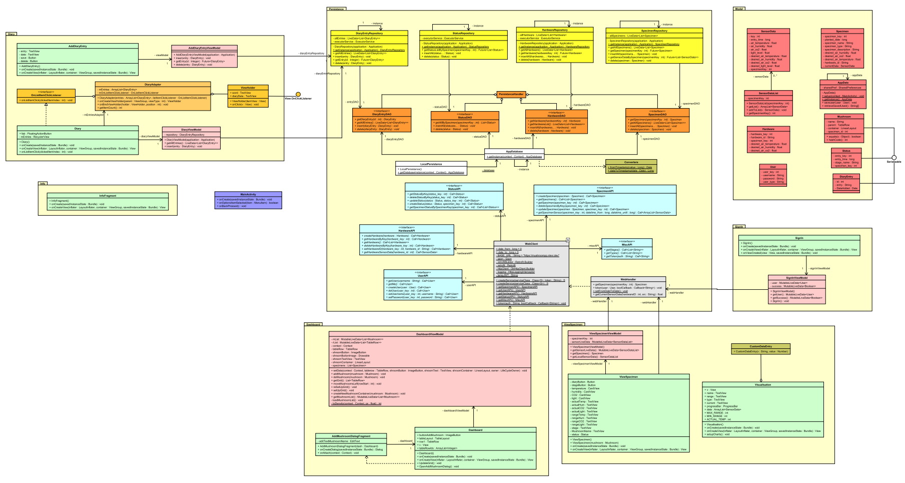
For further information regarding the business logic, refer to the OpenAPI specification of this API, as it has detailed descriptions for each method, field, entity and more. More specific information concerning any of the shown classes can be found in the upcoming Implementation section.

3.5 Android team

Bogdan Mezei, Uldis Alksnis and Kristóf Lénárd

3.5.1 General design and architecture.

The Android client application (used as “app” or client hereafter) was designed for only one purpose: to ensure that the user has easy access to those workings of the system that are necessary for them. Multiple architectural patterns were evaluated, however, the officially recommended architectural pattern (Android Developers, *Guide to app architecture*) is MVVM, therefore, this was used. This architecture ensured consistent user experience and appropriate software structure. The following class diagram shows the implementation of this architecture.



3.5.2 Specific designs.

Within the MVVM architecture, multiple design patterns were used, as seen from the class diagram. These were the Mediator, Observer, Adapter, Singleton and Proxy patterns. Also used as already existing implementations were the Builder and Thread Pool patterns.

The Mediator pattern was used in an architecturally significant location, as the WebHandler class. This class is responsible for the mediation of data between the WebClient class, which directly interacts with the Web API, and all classes that need significant changes in their data before connection. An example of this is the token generator, where the JSON Web Token is requested from the server. This requires a specific authorization; therefore, the Mediator pattern can be used to ensure that only properly formatted data is forwarded.

The Observer pattern was perhaps the most used in the system. This derives from the LiveData class of the Android framework, which provides an observable dataset in the viewmodel for consumption by the view. As MVVM was used, this was a vital component of the design of the views, since this pattern was used to ensure constant updates based on the changes in the data.

The Adapter class was used by the RecyclerView objects, as a mandatory part of the implementation of such objects. These objects require an adapter to be able to properly execute data binding, therefore, a custom implementation was used.

The Singleton pattern was also used in multiple locations, mostly with persistence classes such as the WebClient, which directly consumes the Web API, and the LocalPersistence, which implements the Room library, ensuring the local database's availability. These classes would normally not be duplicated, since re-generation would not only consume needless resources from the client device, but also from the API (in case of the WebClient), and might even cause data loss, therefore, the singleton pattern was the most proper to use.

The Proxy pattern was also used by the persistence handlers. These classes and interfaces were responsible for substituting commands to both the Web API and the SQLite database exposed by Room, ensuring that the other parts of the system would connect to a compatible placeholder for these external resources.

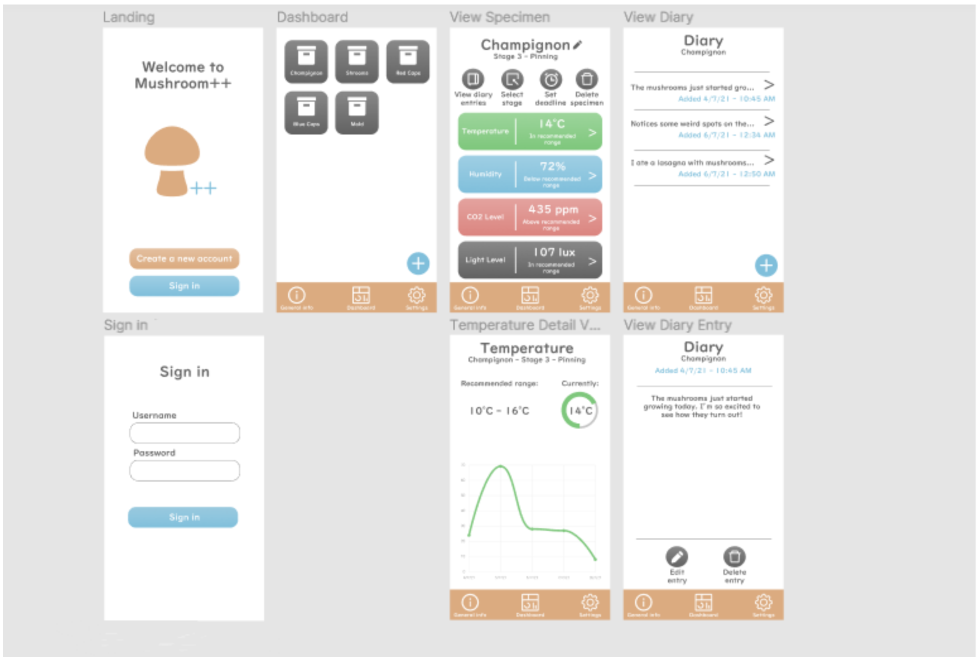
Other parts of the system mandated the usage of certain patterns as built into used frameworks and libraries. One such pattern was the builder pattern, used chiefly indirectly in the creation of the Retrofit client. This client can only be created by a builder exposed by the Retrofit object; therefore, it could not be avoided.

Another such pattern was the Thread Pool pattern as implemented by the Android operating system. The operating system has a special thread pool such as generating a UI thread, background (also called worker) threads, and possibly others. This is built into the Android system; therefore, it also was used.

3.5.3 UI design

During UI design, two main goals were defined. First, simplicity, to ensure that no detailed user guides are required to be able to use the system to the fullest. Second, accessibility, to ensure that users with certain disabilities (such as being sight-impaired) are also able to use the system without additional difficulties. Besides these goals, the main objective of course was to deliver a high-quality user experience in such a format.

UI designer Bogdan Mezei has been the principal force behind the initial creation of designs, with all three team members implementing these. These designs were simple yet effective, covering all functionalities that were to be implemented in the release version.



.

3.6 Enterprise Data Warehouse

Samantha Nettesheim, Shaoyu Liu and Audrius Sauciunas

3.6.1 Kimball's Approach

For the design of the data warehouse, the business dimensional lifestyle approach was utilized, also known as Kimball's approach. This approach proved to be useful for the various advantages it provides, such as fast construction, zero normalization, simplified querying and analysis, fast data retrieval, conformed dimensional structure and simplified system management.

First, the team utilized Kimball's 4-step dimensional design process to design and determine the design and structure of the warehouse. The first step in this process is to identify the business process, the operational activities performed by the organization.

3.6.1.1 Identifying the Business Process

MushroomPP needs a cultivation analysis with a strong focus on environmental variables and different periods of time.

The business process corresponds to rows in the data warehouse fact table. To meet functional requirements 5, 6 and 7 the project requires analysis of sensor entries that monitor environmental variables to provide analysis on the different mushroom cultivations.

3.6.1.2 Identifying the Grain

The second step in the dimensional design process is to declare the grain. The grain establishes the information that is represented in a single fact table row. Kimball's process recommends using the atomic grain, data that refers to the lowest level at which data is captured by a given business process. The lowest level of granularity for the grain achieves the highest level of detail and can withstand the assault of unpredictable user queries because the details can be grouped "any which way." (Kimball, et al., 2008) As requirement 6 calls for the collection of sensory data, the environmental data collected by sensor entries were chosen as the grain, as a fact row is created during the event of sensor entries capturing environmental data. The grain is numeric, additive and at the lowest possible granularity.

3.6.1.3 Identifying the Dimensions

The third step in Kimball’s dimensional design process is to identify the dimensions. A dimension is a structure that categorizes facts and measures to enable users to answer business questions. Dimensions describe the “who, what, where, when why and how” context of a measurement. The power of the data warehouse is proportional to the quality and depth of the dimension attributes, robust dimensions translate into robust querying and analysis capabilities. (Kimball, et al., 2008)

The dimensions chosen were based on the need to provide context for the business analysis of mushroom cultivation. Four dimensions were identified Specimen, Date, Age and Time. To fulfill functional requirement 10, a specimen dimension provides the ability to identify a mushroom by its cultivation instance. Dimension Date, Age and Time fulfill requirement 10, to provide the ability to track and compare mushroom growth over time.

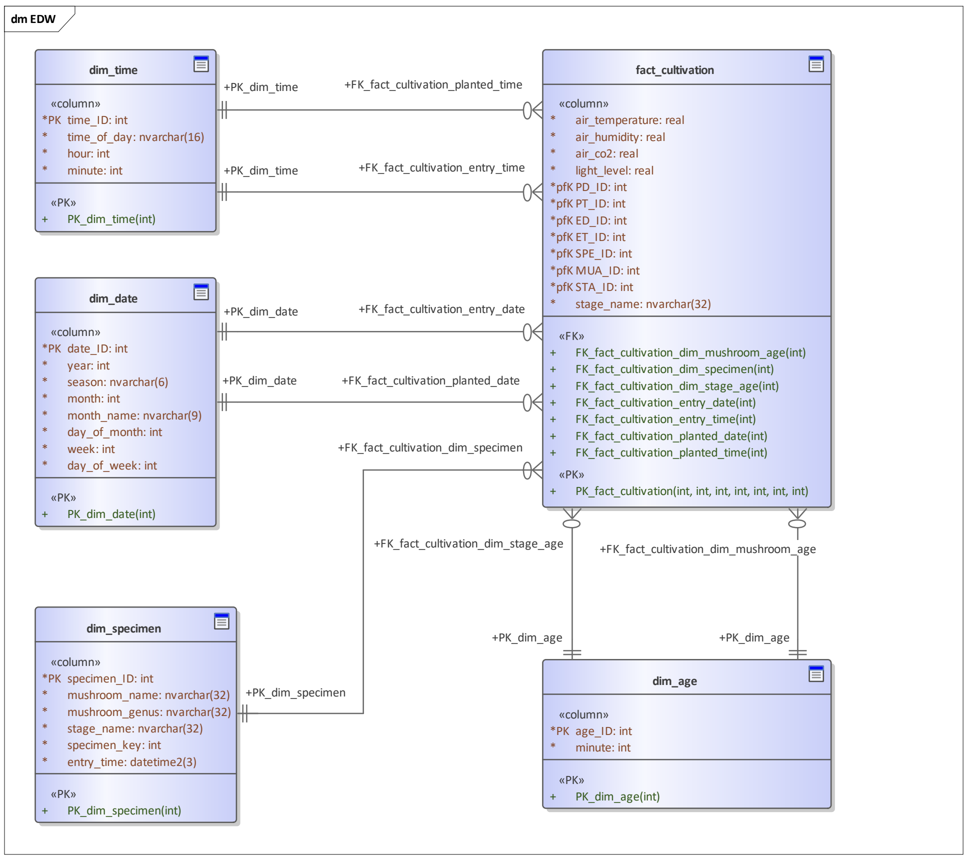

dm EDW
dim_time
«column»
*PK time_ID: int
* time_of_day: nvarchar(16)
* hour: int
* minute: int
«PK»
+ PK_dim_time(int)
dim_date
«column»
*PK date_ID: int
* year: int
* season: nvarchar(6)
* month: int
* month_name: nvarchar(9)
* day_of_month: int
* week: int
* day_of_week: int
«PK»
+ PK_dim_date(int)
dim_specimen
«column»
*PK specimen_ID: int
* mushroom_name: nvarchar(32)
* mushroom_genus: nvarchar(32)
* stage_name: nvarchar(32)
* specimen_key: int
* entry_time: datetime2(3)
«PK»
+ PK_dim_specimen(int)
fact_cultivation
«column»
* air_temperature: real
* air_humidity: real
* air_co2: real
* light_level: real
*pfK PD_ID: int
*pfK PT_ID: int
*pfK ED_ID: int
*pfK ET_ID: int
*pfK SPE_ID: int
*pfK MUA_ID: int
*pfK STA_ID: int
* stage_name: nvarchar(32)
«FK»
+ FK_fact_cultivation_dim_mushroom_age(int)
+ FK_fact_cultivation_dim_specimen(int)
+ FK_fact_cultivation_dim_stage_age(int)
+ FK_fact_cultivation_entry_date(int)
+ FK_fact_cultivation_entry_time(int)
+ FK_fact_cultivation_planted_date(int)
+ FK_fact_cultivation_planted_time(int)
«PK»
+ PK_fact_cultivation(int, int, int, int, int, int, int)
dim_age
«column»
*PK age_ID: int
* minute: int
«PK»
+ PK_dim_age(int)
+PK_dim_time
+FK_fact_cultivation_planted_time
+PK_dim_time
+FK_fact_cultivation_entry_time
+PK_dim_date
+FK_fact_cultivation_entry_date
+PK_dim_date
+FK_fact_cultivation_planted_date
+FK_fact_cultivation_dim_specimen
+PK_dim_specimen
+FK_fact_cultivation_dim_stage_age
+FK_fact_cultivation_dim_mushroom_age
+PK_dim_age
+PK_dim_age

Figure 10 Dimensions of the EDW

In dim_specimen, specimen_key is a primary key that is used as an identifying relationship in the Cultivation fact. This is obtained from MushroomPP.[dbo].specimen.

Mushroom_name and mushroom_genus identifies the species and genus of a mushroom. These two attributes are used to identify the environmental variables specific to species and genus. Stage_name and entry_time identifies the growth stage of a specimen. The definitions of the distinct stages are as follows:

Table 1 Mushroom Stage Definitions

Key	Name	Definition
1.	Inoculation	Mycelium is placed on a substrate induced to grow into it.
2.	Casing	Casing is a top-dressing applied to the spawn-run compost on which the mushrooms eventually form.
3.	Spawn Run	This period is also called the vegetative growth or incubation period
4.	Pinning	Development of mushroom initials
5.	Fruiting	The period during which the mushroom becomes harvestable
6.	Dead	The mushroom did not survive.

These attributes enable the business user to analyze environmental factors at different hierarchical levels by species, genus and stage, or different combinations of such attributes.

In dim_date, Date_ID is the surrogate key for this dimension and is used to identify dates in the fact. Year, month, week, day_of_week and day_of_month are used to identify the specific time of year. Season is used to be able to query by different seasons to see how cultivations are affected by them. Month_name is used to ease end user queries.

In dim_time, Time_ID is the surrogate key for this dimension and is used to identify daytime in the fact dimension. Hour and minute tell the exact time of the day, whereas Time_of_day is a rough representation of the day, for example evening or morning.

In dim_age, Age_ID is the surrogate key for this dimension and is used to identify age in the fact dimension. There is only one attribute called “minute” that tracks the age of a mushroom with 10-minute increments.

3.6.1.4 Identifying the Facts

The last step in Kimball's dimensional design process is to identify the facts. Facts store the performance measurements generated by the organization’s business events and activities. A “fact” refers to each performance measure. Typically, it is not possible to know the value of facts in advance, as they are variable. The fact’s valuation occurs at the time of the measurement event or activity, for example, when the sensor evaluates environmental variables. The fact’s grain is the business definition of the measurement event that produces the fact row. (Kimball, et al., 2008)

A single cultivation fact was identified. There is a need for analyzing sensor entries in the cultivation for future environmental property analysis. Sensor entries include the following: temperature; light level; CO2; humidity.

There is a relationship between specimen and fact to identify the type of specimen that is growing and its current stage to analyze environmental factors based on the specific mushroom growth variables.

3.6.2 Data Types

For string data types, the nvarchar type was used since it is the commonly used data type to use when size of the entries varies considerably. Nvarchar compared to varchar allows for UTF-16-character encoding which is beneficial if data is needed to be added from source systems with character sets not supported in UTF-8.

For all the primary keys and surrogate keys, int was used as it is usually used for such purposes, they boost performance, handle unknown or unapplicable conditions, track changes in dimension attribute values, are great for mapping to integrate disparate sources etc., int data type was also used for our time, date, and age dimensions.

3.6.3 ETL process

The ETL is the process by which data is extracted from the data source, optimized for analytics, and moved to the central host. Figure 20 illustrates the ETL process, the process can be described as pipeline in three phases. A staging and an enterprise data warehouse (edw) schema are created for the ETL process.

3.6.3.1 Extract

During the extraction, the desired data from the source database is identified, matched, and extracted to the staging area. The staging area is an intermediate zone used for processing data which sits between the data source and the edw. A complete description of the source to staging area mapping is available in the source-target mapping in Appendix E2.

Initially, a full extraction was performed on the source database. In the staging area, the quality of the data extracted is checked for errors or missing data. For example, the stage name, when extracted from the data source, is null. This provides an opportunity to validate extracted data before it is moved on to the data warehouse.

3.6.3.2 Transform

Data extracted from the source is raw and unusable in its original form. Therefore, it needs to be cleansed, mapped, and transformed. Transforming is the key step which adds value and changes data such that it can be meaningfully used in the data warehouse. During the transformation, data is deduplicated, combined, cleansed, standardized, verified, sorted and otherwise changed, all NULL values in the staging area are cleansed and repaired, attributes are formatted. One of the most major transformations in the data warehouse is key restructuration, otherwise known as the addition of surrogate keys. These keys boost the performance of a data warehouse, use less storage space and provide data abstraction. Data transformation is the most important part of ETL as it improves data integrity and ensures that data is fully compatible and ready to use. For instance, the stage name cannot be found and inserted directly from source database, so it is set to be NULL during extraction and assigned values after calculations when transforming.

In addition to this, some dimensions that could not be found from source system are loaded by hand, for instance, date and time dimensions.

3.6.3.3 Load

The last step in the ETL process is to load the newly transformed data into the data warehouse. During the loading, data from staging area is popped into the data warehouse. The data warehouse uses a full load, which means it dumps all the required data from the source to target system.

3.6.4 Power BI

The data in its final form can now be used for analysis by the business users. Functional requirement 10 establishes the need for visualizations. Due to limitations of the data set, correlations using visualization techniques were not implemented. The visualizations created using Power BI were expressed using various value encoding objects such as line charts, with a strong focus on ordinal scale. PowerBI was utilized for its robust data reporting and visualization features. To create reports for business users, a DirectQuery connection to the data warehouse was established. The DirectQuery is a live connection to the data warehouse which enables business users to access current data for analysis but limits the options for data manipulation.

When designing reports, it is imperative to take into consideration the cognitive biases of both the designer and business user. The PowerBI dashboard was designed to provide optimal representation of the data using value-encoding objects.

Natali Munk-Jakobsen

3.7 Embedded System

In the project, in order to design the system some physical components had to be taken into consideration. The Arduino ATMEGA2560 has been used as an Arduino core with additional components. These components are:

1. Temperature and Humidity Sensor
2. CO2 Sensor
3. Light Sensor
4. LoRaWAN transceiver
5. RC Servo

The sensors measure the temperature, humidity, light and CO2 level of the environment and the LoRaWAN transceiver handles sending and receiving data using long-range communication technologies. Two RC-Servo motors were used in the project to perform actions to change humidity and light levels.

Implementation of the embedded system is done by using the C programming language as the main technology. C language is preferred in embedded programming because it is compiled into raw binary executable which can be directly loaded into memory and executed. In addition, C provides optimized machine instructions for the given input and increases the performance of the embedded system. For these reasons, access to the hardware is both easier and faster.

Another technology used was FreeRTOS and its libraries. FreeRTOS Real-time operating system provides a safe, easy and reliable way of developing in-parallel running software. However, because the ATMEGA 2560 Arduino board has only one processor and does not support parallel execution, mutexes have been used in the project to ensure that sensors can carry out measurements in the given order and fulfil the requirements correctly, without entering deadlock or modifying data illegally.

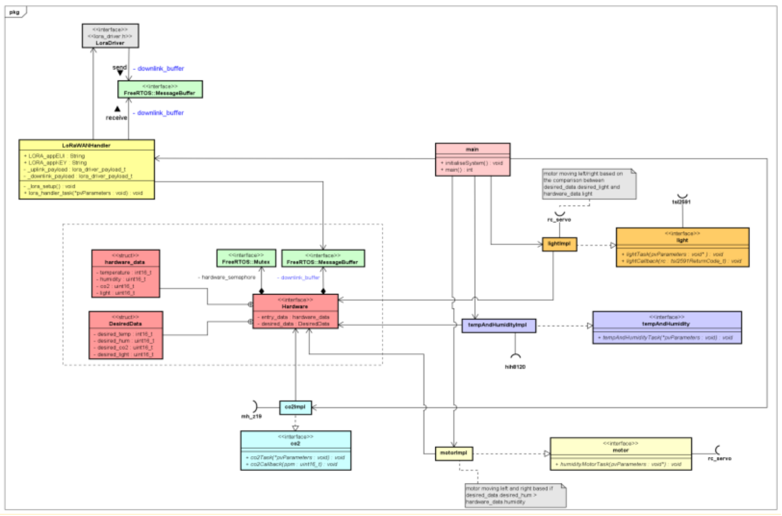


Image 3.1. Embedded System Class Diagram

The class diagram for the embedded system can be seen in Image 3.1. The main class is responsible for initializing the system and starting the task scheduler after creating the five different tasks. Each task represents and manages one of the sensors enumerated earlier.

In order to improve the code structure and architecture, each task is split into a header file and its implementation, which is also connected to the related driver (that is, the co2Impl has to use the mh_z19 driver for the co2 sensor). CO2, TempAndHumidity and Light tasks are responsible for measuring the data using sensors, while the motor task handles the motor movements based of the hardware's DesiredData.

LoRaWANHandler uses the LoRaWAN network in order to send and receive data. As the class diagram shows, a message buffer is also needed for the LoRaWANHandler in order to receive messages safely and store them. The 'hardware' is the system's representation of shared data, a simple architectural component for storing sensor information. This is represented as an individual component for better security and maintainability.

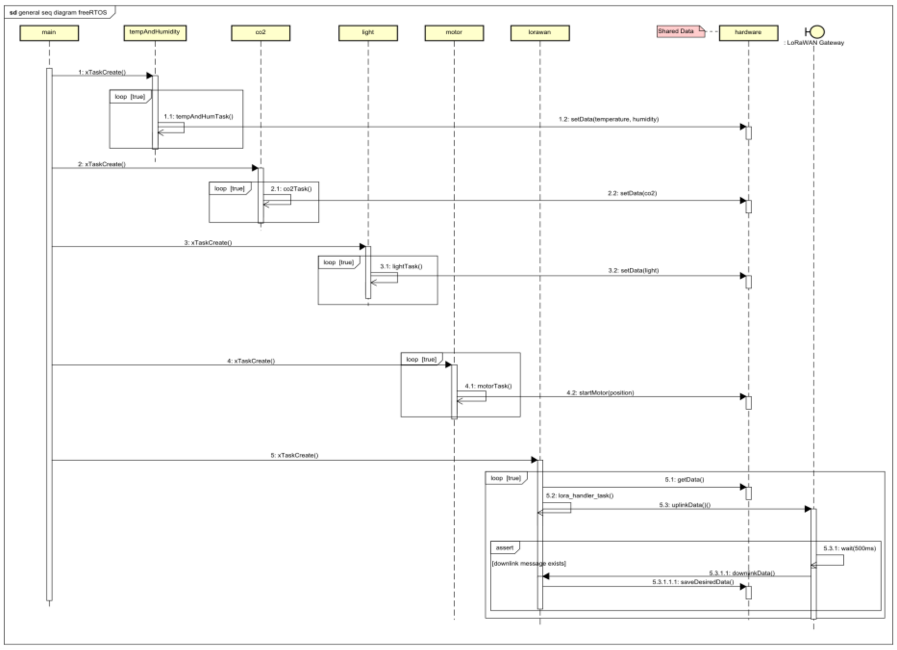


Image 3.2. Embedded System Sequence Diagram

The Embedded System Sequence Diagram (Image 3.2) shows the interaction between the main class and the five task implementation classes. The first step is the initialization of the drivers and the creation of tasks in the main class. Afterwards, the tasks run in a given order, perform the measurements and save the measured data in the shared data component, 'hardware'. The LoRaWAN task then collects information and sends it further, reading any new data and also persisting it in the 'hardware' element.

3.7.1 Gateway Application

The gateway application (Image 3.3.) is implemented using Java programming language. The Spring Boot framework and the Maven project management and comprehension tool technologies were also used for this part of the project.

The gateway application serves as a bridge between the embedded system and the database. It is responsible for handling the communication with the Web API and communicating with the embedded system using LoRaWAN Network. The RESTful Web API is used for establishing the communication with the data part and CRUD operations are handled using Spring Boot, which provides an easy approach to create Service and Controller classes.

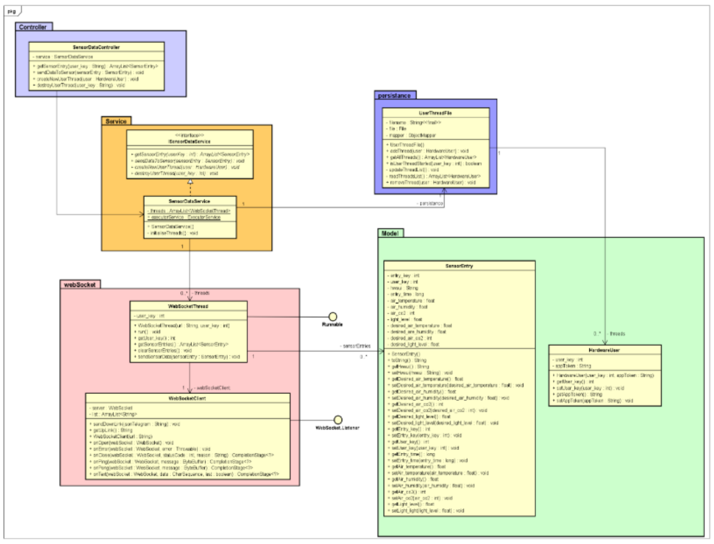


Image 3.3. Gateway Application Class Diagram

Two model classes (SensorEntry and HardwareUser) are created to hold measurements and user key-hardware relations. The persistence of the gateway application is provided using a simple text file that holds HardwareUser objects. The text file is used for getting a list of user key-hardware token combinations and creating a thread for each hardware when it is required.

The WebSocketClient class handles establishing WebSocket connections with the embedded system and sending and receiving data through the LoRaWAN network. Both downlink and uplink messages are prepared in the required data format and converted to JSON in order to be sent using the LoRaWAN network. The process of sending a downlink message can be seen in the sequence diagram (Image 3.4) as well as other interactions in the Gateway application.

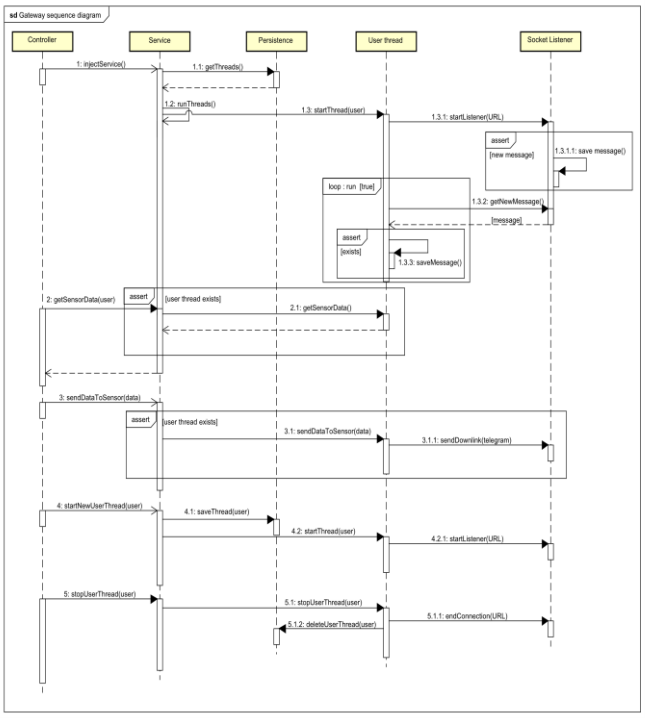


Image 3.4. Gateway Application Sequence Diagram

In order to improve code structure, certain design and architectural patterns have been used in Gateway Application. For creating the web service, the REST architectural style has also been used because of its well-defined constraints that enable the developer to write scalable Web Service Interface. As part of this style, HTTP is used as a communication protocol and HTTP methods GET and POST are used for retrieving and sending data.

The service layer pattern is used for keeping the application logic in a different layer, separated from the Controller class. The interface and implementation class in the service layer are annotated with @Service annotation and include methods for getting and sending data. Instead of implementing all functionality directly in the controller class, having a service layer was preferred in order to provide a well-structured code considering the separation of concerns principle and testability.

Another design pattern used in the Gateway application is dependency injection. Dependency Injection allows the creation of dependent objects outside of a class and provides those objects to a class in different ways. The SensorDataController class is autowired to SensorDataService class using the @Autowired annotation, which is used for dependency injection, and helps connect the collaborative beans in Spring Boot.

4 Implementation

4.1 Android team

Kristóf Lénárd

During the implementation of the designs for the Android client app, portability and user experience were prioritised to deliver a client application that is not only adaptable, but also easily usable. To achieve this, the implementation followed the recommended approaches by the Android developers.

Certain non-core libraries were used to facilitate development. These were the Retrofit library for Web communication, the Room library for local persistence, and the Anychart library for visualizations within the user interface.

The implementation of the code began with the development of core classes and interfaces. These were the MainActivity, which is the wrapper class for the UI, the web client connection, to ensure that the system is compatible with the Web API, and the model classes.

On opening the application, besides the mandatory preliminary setup (which is performed by the Android framework), the first method to run is the MainActivity's onCreate method. This sets up two core parts of the system: a part of the local persistence (the part responsible for storing user data locally), and the intrasystem navigation.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    AppData.setup(this);

    //Bottom Navigation Bar
    bottomNavigationView = (BottomNavigationView)
findViewById(R.id.bottomNavigationView);
    bottomNavigationView.setVisibility(View.INVISIBLE); //To not see navigation bar
on Sign In menu

    AppBarConfiguration appBarConfiguration = new
AppBarConfiguration.Builder(R.id.item_home, R.id.dashboard,
R.id.item_settings).build();
        NavController navController = Navigation.findNavController(this,
R.id.fragmentbox);
    NavigationUI.setupActionBarWithNavController(this, navController,
appBarConfiguration);
    NavigationUI.setupWithNavController(bottomNavigationView, navController);
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    }
```

After this, the SignIn fragment is loaded. Fragments were used in the implementation to facilitate modularity, as recommended. This fragment is used for the sole purpose of ensuring security. This is done by only enabling system access if the system was able to authenticate the user.

```
button.setOnClickListener(v -> {
    signInViewModel.getUser().getValue().setUsername(((EditText)
view.findViewById(R.id.editTextTextPersonName)).getText().toString());
    signInViewModel.getUser().getValue().setPassword(((EditText)
view.findViewById(R.id.editTextTextPersonName2)).getText().toString());
    signInViewModel.SignIn();
    final Observer<Boolean> observer = aBoolean -> {
        if(aBoolean) {
            System.out.println("Observer");

LocalPersistence.getDatabaseInstance(getActivity().getApplicationContext());
            BottomNavigationView bottomNavigationView =
view.getRootView().findViewById(R.id.bottomNavigationView);
            bottomNavigationView.setVisibility(View.VISIBLE); //Turns on Navigation
view
            NavController nav = Navigation.findNavController(view);
            nav.navigate(R.id.action_signIn_to_dashboard);
        }
    };
    signInViewModel.getSuccess().observe(getViewLifecycleOwner(), observer);
});
```

As seen in the above code, a Boolean variable stored in the viewmodel of this fragment is observed, and if the value of this variable (which is preset to false), is changed to true by the authentication method, the system navigation and access to the Room database are enabled. The application is then navigated to the dashboard, which is the main layout of the app.

```
public void addMushroom(Mushroom mushroom) {
    LinearLayout newShroomContainer = createNewMushroomContainer(mushroom);
    Log.i("INFO", mushroom.getName());
    Log.i("INFO", getMushroomList().getValue().size() + "");
    Log.i("INFO", "Current Grid Contains: " + getGrid().size());
    if (getMushroomList().getValue().size() % 3 != 0) {
        TableRow rowToAdd = getGrid().get(getGrid().size() - 1);
        rowToAdd.addView(newShroomContainer);
        mushroom.setParent(getGrid().get(getGrid().size() - 1));
        mushroom.setContainer(newShroomContainer);
        List<Mushroom> list = mList.getValue();
        list.add(mushroom);
        mList.setValue(list);
    } else {
        TableRow newRow = new TableRow(context);
        newRow.setId(ViewCompat.generateViewId());
        ViewGroup.LayoutParams tableRowLayout = tableRow.getLayoutParams();
        newRow.setLayoutParams(tableRowLayout);
        newRow.addView(newShroomContainer);
        mushroom.setParent(newRow);
        mushroom.setContainer(newShroomContainer);
        getGrid().add(newRow);
        getMushroomList().getValue().add(mushroom);
    }
}
```

The dashboard consists of two main unique parts: a custom table implementation and an option to add elements to this table. These elements all constitute individual mushroom specimens (with the corresponding hardware), and are retrieved from the Web API when this fragment is loaded. This is done to ensure up-to-date information on all specimens, which is a vital component of this system.

On tapping a mushroom, the navigation transacts to the main informational fragment – the ViewSpecimen. This fragment, acting as a major endpoint for individual specimens, displays sensor data and enables access to other functions relevant to any given specimen.

```
    private void setupCharts(){

      AnyChartView anyChartView1;
       anyChartView1 = v.findViewById(R.id.anychart_temp_graph);


      Cartesian cartesian = AnyChart.line();

      cartesian.animation(true);

      cartesian.padding(10d, 20d, 5d, 20d);

      cartesian.crosshair().enabled(true);
      cartesian.crosshair()
            .yLabel(true)
            // TODO ystroke
            .yStroke((Stroke) null, null, null, (String) null, (String) null);

      cartesian.tooltip().positionMode(TooltipPositionMode.POINT);
      ArrayList<DataEntry> seriesData = new ArrayList<>();

      switch (getArguments().getString("Type"))
      {
         case "Temperature":
         {
            cartesian.title("Temperature Monitor");
            cartesian.yAxis(0).title("Degrees C");
            cartesian.xAxis(0).labels().padding(5d, 5d, 5d, 5d);

            String pattern = "HH:mm";
            SimpleDateFormat format = new SimpleDateFormat(pattern);
            for (SensorData data : data)
            {
               Date date = new Date(data.getEntry_time());
               seriesData.add(new CustomDataEntry(format.format(date),
data.getAir_temperature()));
            }
            break;
         }
      Set set = Set.instantiate();
      set.data(seriesData);
      Mapping series1Mapping = set.mapAs("{ x: 'x', value: 'value' }");

      Line series1 = cartesian.line(series1Mapping);
      series1.name("Mushroom");
```

```
        series1.hovered().markers().enabled(true);
        series1.hovered().markers()
              .type(MarkerType.CIRCLE)
              .size(4d);
        series1.tooltip()
              .position("right")
              .anchor(Anchor.LEFT_CENTER)
              .offsetX(5d)
              .offsetY(5d);
        switch (getArguments().getString("Type"))
        {
           case "Temperature":
           {
              series1.stroke("green");
              break;
           }
        }
        cartesian.legend().enabled(true);
        cartesian.legend().fontSize(13d);
        cartesian.legend().padding(0d, 0d, 10d, 0d);

        anyChartView1.setChart(cartesian);
    }
}
```

One of the main achievements in the system is the visualizer, accessed by tapping a datapoint in the ViewSpecimen fragment for that specimen. This fragment displays current and historic data for this given hardware, with the addition of a graphical interface to enhance user experience. Originally, PowerBI reports were planned to be used, however, those reports cannot filter by specimen, therefore, a custom visualizer was used, derived from the Anychart library.

Besides these fragments, there is another user interface accessible from the navigation bar – the info fragment. This informational fragment is responsible for providing information on fungiculture to end users. This is a display-only fragment, with data added and updated by the system administrator.

These fragments, however, require support. There are two main classes that enable constant access to the data – the WebClient, responsible for the connections to the Web API, and the LocalPersistence, enabling access to the Room SQLite database. These both support the storage and retrieval of the data, to ensure that the most up-to-date information is displayed.

```
private static AppDatabase database;

private LocalPersistence() {

}

public static AppDatabase getDatabaseInstance(Context context) {
    if (database == null) {
        database = Room.databaseBuilder(context, AppDatabase.class,
"MushroomDatabase").build();
    }
    return database;
}
```

The LocalPersistence is structured as recommended in case of a Room database. It is a Singleton class, with the database instance being retrievable, enabling a common point of connection to the local database. This is necessary to avoid the readers-writers problem, and to ensure that the database contains information that is only overwritten at appropriate changes.

```
public static <S> S createService(Class<S> serviceClass, final String token) {
    httpClient.interceptors().clear();
    if (token != null) {
        httpClient.addInterceptor( chain -> {
            Request original = chain.request();
            Request.Builder builder1 = original.newBuilder()
                    .header("Authorization", "Bearer " + token);
            Request request = builder1.build();
            return chain.proceed(request);
        });
        httpClient.addInterceptor(logging);
        retrofitBuilder.client(httpClient.build());
        retrofit = retrofitBuilder.build();
    }
    return retrofit.create(serviceClass);
}

public static SpecimenAPI getSpecimenAPI()
{
    return specimenAPI;
}
public static UserAPI getUserAPI() {
    return userAPI;
}
public static HardwareAPI getHardwareAPI() {
    return hardwareAPI;
}
public static StatusAPI getStatusAPI() {
    return statusAPI;
}
public static MiscAPI getMiscAPI() {
    return miscAPI;
}
public static void token(String auth, final Callback<String> boolCallback)
{
    String s = "Basic " + Base64.getEncoder().encodeToString(auth.getBytes());
    Call<String> tokenCall = getMiscAPI().getToken(s);
    tokenCall.enqueue(new Callback<String>() {
        @Override
        public void onResponse(Call<String> call, Response<String> response) {
            if(response.body() != null) {
                tempJWT = response.body();
                specimenAPI = createService(SpecimenAPI.class, tempJWT);
                userAPI = createService(UserAPI.class, tempJWT);
                hardwareAPI = createService(HardwareAPI.class, tempJWT);
                statusAPI = createService(StatusAPI.class, tempJWT);
                boolCallback.onResponse(call, response);
            }
        }
        @Override
        public void onFailure(Call<String> call, Throwable t) {

        }
    });
}
```

The WebClient serves as the creator of the Retrofit client, and sets up the interfaces that directly connect to the Web API. These are static, to ensure a global point of access to the API. Security is provided here by a JSON Web Token, or JWT for short, as seen above (Auth0 Inc., *JSON Web Tokens*). This is an open standard (RFC 7519), standardizing a compact, URL-safe means of representing claims. Every such token has three components, separated by a '.' character. The first component, also called a header, encodes (in our case, with the Base64 encoder) the encoding algorithm and the token type of JWT. The second component, the payload, encodes information on such elements as the issuer, the expiration time, and the subject of the claims. The third component is the signature, which verifies the contents of the token and protects against tampering. These tokens are attached to the header of the HTTP call automatically, as seen in the createService method.

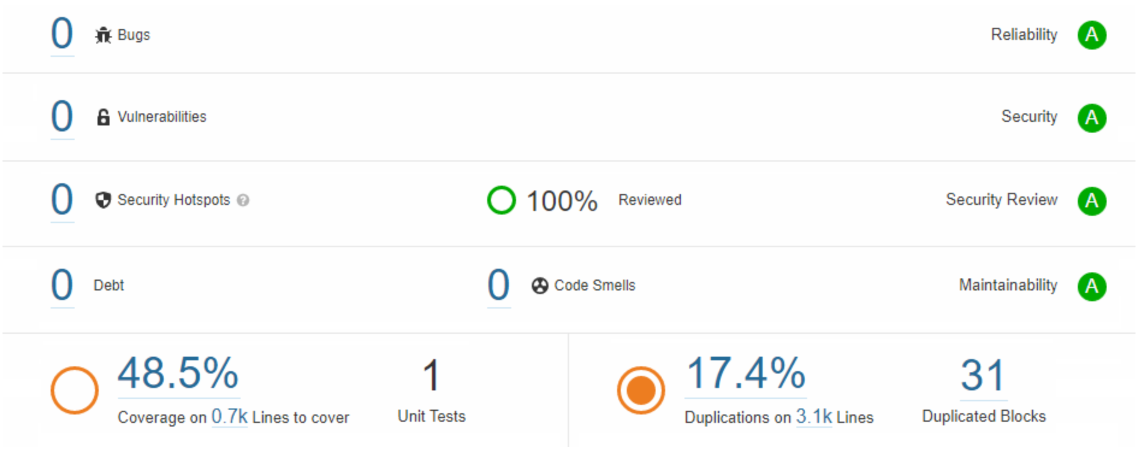
4.2 Web API Server

Levente Nagy

During the implementation process, the following key code quality aspects were prioritized: reliability; maintainability; testability; portability; reusability. Most of these traits were easy to follow, as the design of the code already heavily supports such code qualities.

Most of the code does not need extra explanations, because the API Documentation precisely describes the responsibilities of each endpoint, which the code does follow, with low cognitive complexity on every implemented method. Regardless, some of the non-evident parts will be discussed in this subsection.

Before the explanations, it must be said that the code was evaluated with SonarQube, and the following result were found satisfactory.



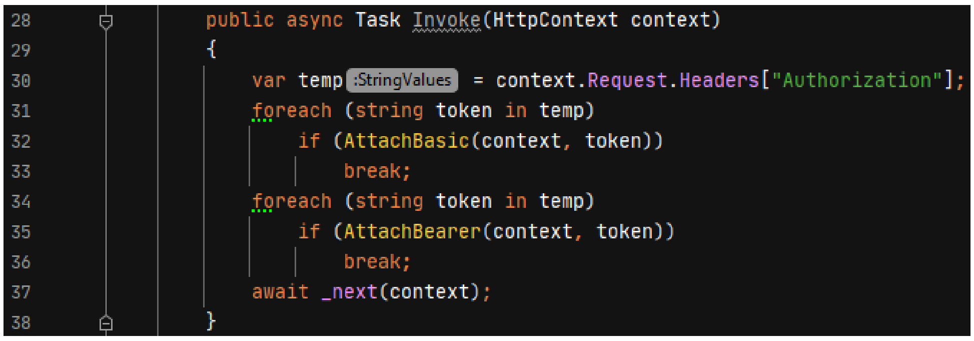
0 Bugs		Reliability A
0 Vulnerabilities		Security A
0 Security Hotspots	100% Reviewed	Security Review A
0 Debt	0 Code Smells	Maintainability A

48.5% Coverage on 0.7k Lines to cover — 1 Unit Tests — 17.4% Duplications on 3.1k Lines — 31 Duplicated Blocks

SonarQube helped in the recognition of some “Code Smells” which were all either fixed or marked as will not be fixed. Security Hotspots were also pointed out, which were concerning the usage of http, but in the current deployment pattern this security concern was marked as safe, as https gets applied outside of the Kestrel webserver’s scope. However, this must be marked as a serious security concern when deploying the application.

The test coverage shown in the SonarQube analysis will be discussed in the Test section of this document. Most of the duplicate code were not fixed, which can result in lower maintainability, and these duplicated lines mainly occur in the PersistenceService and the Controller classes. When authenticating and authorizing users in the controllers’ endpoint methods, logic could be refactored into the middleware pipeline to avoid duplicated fragments. Otherwise, each method in the PersistenceService opens and uses a separate database connection in its own scope, resulting in duplicates. This could be fixed with a method constructing new connections upon request, instead of using duplicate fragments.

There are some non-evident code fragments, starting with the middleware pipeline, more precisely the JwtMiddleware class. The responsibility of this class was discussed in the Web API Server.

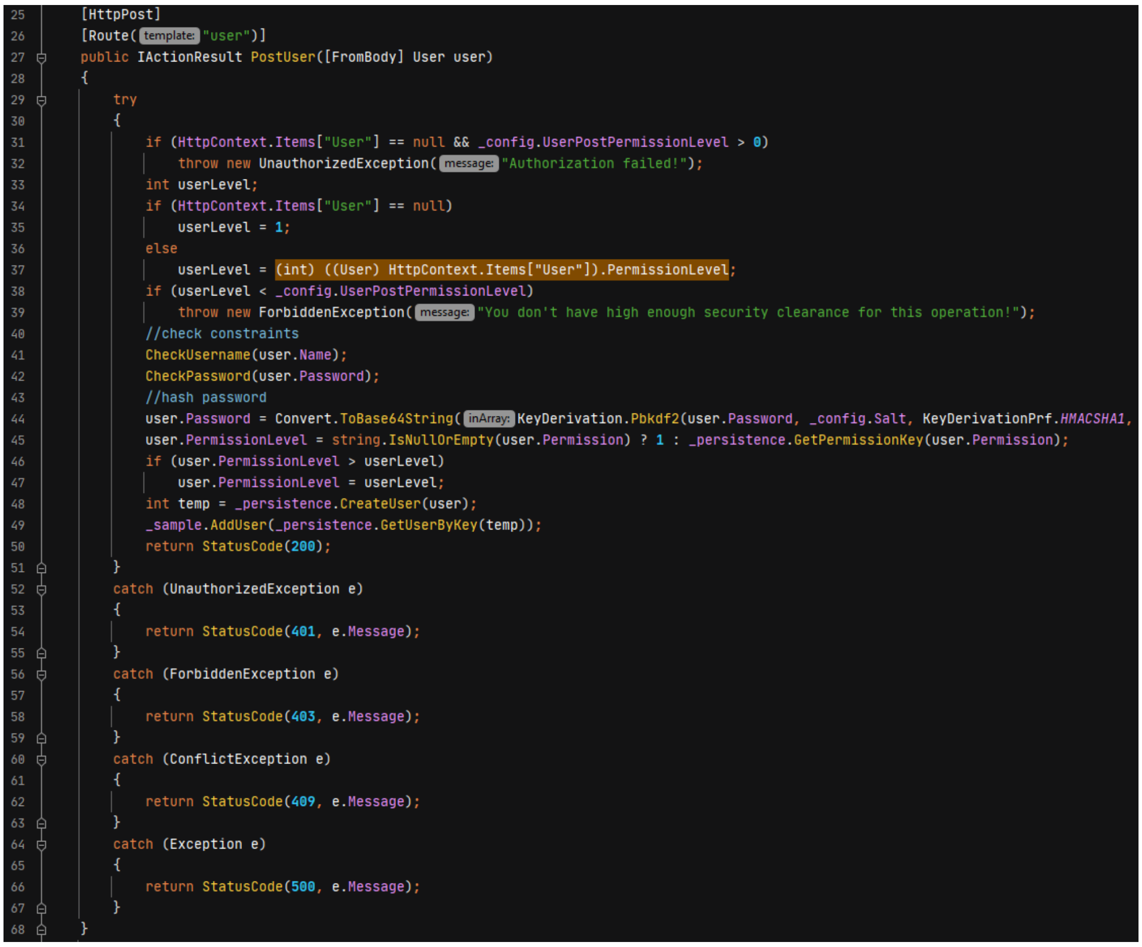


```
28  public async Task Invoke(HttpContext context)
29  {
30      var temp :StringValues = context.Request.Headers["Authorization"];
31      foreach (string token in temp)
32          if (AttachBasic(context, token))
33              break;
34      foreach (string token in temp)
35          if (AttachBearer(context, token))
36              break;
37      await _next(context);
38  }
```

This class first retrieves the attached “Authorization” headers of the request in line 30. Then loops thorough all the auth headers, as theoretically there can be more. During the first loop, it tries to find and authenticate the user using the basic format, then during the second loop it tries to do it with the bearer format. In each “Attach” method, on successful authentication the method returns true and attaches the authenticated user to the “User” HttpContext (this happens on line 63 and 84).

This also means that if there are multiple valid authentication headers in the request, it will prioritize the bearer ones, as that loop comes after the basic, and can overwrite the “User” HttpContext. If there are multiple valid authentication headers from the same format, it prioritizes the first occurring one, as each loop breaks on the first valid authentication. Then the request gets passed onto the next element down the middleware pipeline, and the HttpContext will contain the authenticated user (it will be null if there is no authenticated user).

The following snippet is the PostUser endpoint, which method also had the highest cognitive complexity rating during code analysis.



```
25      [HttpPost]
26      [Route(template: "user")]
27      public IActionResult PostUser([FromBody] User user)
28      {
29          try
30          {
31              if (HttpContext.Items["User"] == null && _config.UserPostPermissionLevel > 0)
32                  throw new UnauthorizedException(message: "Authorization failed!");
33              int userLevel;
34              if (HttpContext.Items["User"] == null)
35                  userLevel = 1;
36              else
37                  userLevel = (int) ((User) HttpContext.Items["User"]).PermissionLevel;
38              if (userLevel < _config.UserPostPermissionLevel)
39                  throw new ForbiddenException(message: "You don't have high enough security clearance for this operation!");
40              //check constraints
41              CheckUsername(user.Name);
42              CheckPassword(user.Password);
43              //hash password
44              user.Password = Convert.ToBase64String(inArray: KeyDerivation.Pbkdf2(user.Password, _config.Salt, KeyDerivationPrf.HMACSHA1,
45              user.PermissionLevel = string.IsNullOrEmpty(user.Permission) ? 1 : _persistence.GetPermissionKey(user.Permission);
46              if (user.PermissionLevel > userLevel)
47                  user.PermissionLevel = userLevel;
48              int temp = _persistence.CreateUser(user);
49              _sample.AddUser(_persistence.GetUserByKey(temp));
50              return StatusCode(200);
51          }
52          catch (UnauthorizedException e)
53          {
54              return StatusCode(401, e.Message);
55          }
56          catch (ForbiddenException e)
57          {
58              return StatusCode(403, e.Message);
59          }
60          catch (ConflictException e)
61          {
62              return StatusCode(409, e.Message);
63          }
64          catch (Exception e)
65          {
66              return StatusCode(500, e.Message);
67          }
68      }
```

As it was previously established in the Web API Server, one of the configurable properties of the Server, is the “UserPostPermission” which defines, which user have the privilege to register new users to the system. On line 31, the evaluation of this rule happens, where the condition states that the user must be authenticated if the config property is set higher than 0. If the condition is not met, it throws an UnauthorizedException, with a proper error message detailing the issue.

From this point, this method has two possible states, in one which the rule allowed unauthenticated users to register new users, and the request does not contain an authenticated user, in the other case, there is an attached authenticated user in the HttpContext. If there is no authenticated user, in line 35 a “userLevel” local variable is defined as 1, which will serve as the user’s imaginary permission level which is nonexistent, as there is no authenticated user in the scenario. If there is an authenticated user, this userLevel, will be set to the authorized user’s level in line 37.

The IDE marks the nullable PermissionLevel casting on line 37 as a possible error, but in this scenario the concerned field is never going to be null, as it has been read from the database previously (in the JwtMiddleware) and the said field is not nullable in the database. In line 38 the evaluation of the required permission level happens, if the user has smaller permission level than the one defined in the config, a ForbiddenException is thrown, with a proper error message detailing the issue. In line 41 and 42, the Username and Password constraints are checked, if they are not met, ConflictExceptions are thrown in the called methods, with a proper error message detailing the issue.

On line 44 the password gets hashed (for the user which is yet to be posted) using the salt from the config, this ensures the security of the passwords as they are stored in a hashed form, negating the risk of leaking passwords from the database. On line 45, two things happen, first the user which is yet to be registered, is checked if the Permission field is null or not, if null, then the PermissionLevel is set to 1, which is the lowest, and the default value.

If the Permission is not null, the appropriate PermissionLevel of the user must be determined from the database. This happens because the PermissionLevel is an inner logic of the Server and is not visible from the end user’s perspective. And the PermissionLevel serves as the PK in the database for the _permission_level table. Then in line 46, the user which is yet to be posted is checked to have its PermissionLevel capped at the permission level smaller or equal to the authenticated user’s level. This happens to ensure that no user can gain higher privileges then their current one.

On line 48 the user is added to the database, and its generated key is returned and stored in a temp local field. This line can also throw ConflictExecptions, with a proper error message detailing the issue (e.g., username taken). Next in line 49 the user is also added to the Gateway Application, using the SampleService class. Finally, a proper response is returned, either with an empty body and 200 OK code, or a proper error code from a catch block, with the plain/text body of the error message.

This document will not detail further implementation solutions of the Web API Server, but the following implementation practices were heavily used:

- try-catch blocks, and exceptions
- default values for nullable fields when inserting to database

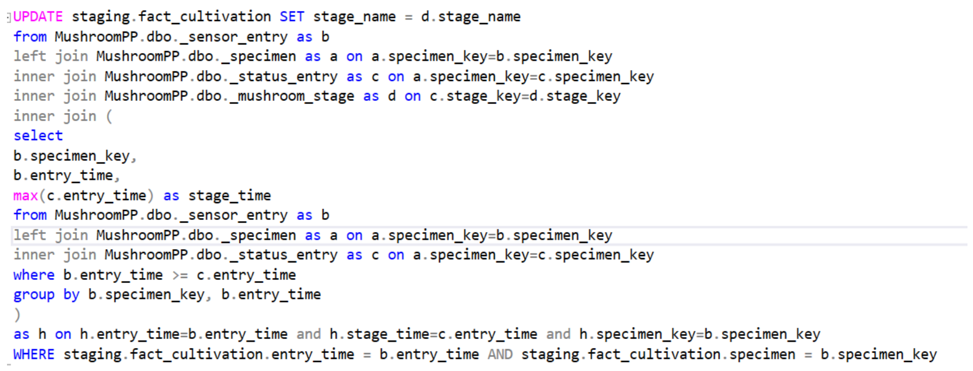
- checking for miscellaneous security concerns, like users trying to tamper with records which are not owned by them, etc.

4.3 Enterprise Data Warehouse

Samantha Nettesheim, Shaoyu Liu and Audrius Sauciunas

4.3.1 Transform

During the extraction, the stage name could not be inserted automatically as it results in a complicated calculating procedure. Therefore, the values were set to NULL, and were operated in transforming.



```
UPDATE staging.fact_cultivation SET stage_name = d.stage_name
from MushroomPP.dbo._sensor_entry as b
left join MushroomPP.dbo._specimen as a on a.specimen_key=b.specimen_key
inner join MushroomPP.dbo._status_entry as c on a.specimen_key=c.specimen_key
inner join MushroomPP.dbo._mushroom_stage as d on c.stage_key=d.stage_key
inner join (
select
b.specimen_key,
b.entry_time,
max(c.entry_time) as stage_time
from MushroomPP.dbo._sensor_entry as b
left join MushroomPP.dbo._specimen as a on a.specimen_key=b.specimen_key
inner join MushroomPP.dbo._status_entry as c on a.specimen_key=c.specimen_key
where b.entry_time >= c.entry_time
group by b.specimen_key, b.entry_time
)
as h on h.entry_time=b.entry_time and h.stage_time=c.entry_time and h.specimen_key=b.specimen_key
WHERE staging.fact_cultivation.entry_time = b.entry_time AND staging.fact_cultivation.specimen = b.specimen_key
```

4.3.2 Load

During the loading, a while loop generates and inserts every day from a given start date to an end date as a record to the date dimension. Division and modulus operations are used to calculate the days of the week. What must be mentioned here is that the “week” and “day of week” are not natural weeks and days of weeks as they will not have any usage in this system.

```
(DATEPART(day,@StartDate) -1) / 7 +1  as week,
(case when
DATEPART(day,@StartDate) % 7 =0 then 7
else
DATEPART(day,@StartDate) % 7
end
)as day_of_week
```

Similar to the date dimension, there is an age dimension by changing “day” to “minute”. The difference in handling the time dimension, is that the converting function uses hour and minute to convert to time_of_day.

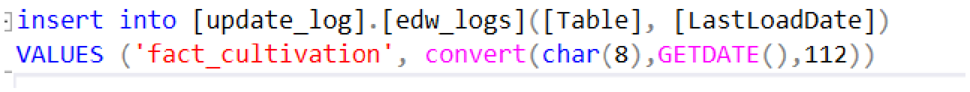
```
  CONVERT(nvarchar(100), DATEPART(hour,@startOfDay), 0)
+ ':'
+ CONVERT(nvarchar(100), DATEPART(minute,@startOfDay), 0) as time_of_day,
```

4.3.3 Incremental load

The system should be able to handle any kind of data changing for more flexible management. Under this situation, there has to be an incremental load script. Considering that the system is a real-time data detecting system, where environmental data is recorded in every minute, there is no access for the changing of past environmental data. With this premise the only possible change in the system is the addition of new records. Each time new data is added, the staging tables are re-extracted, implemented by selecting and inserting records existing in staging tables but not in edw tables. The concrete way here is to use “WHERE NOT EXISTS”.

```
where not exists (select 1 from (edw.fact_cultivation as i
inner join edw.dim_specimen as k on i.SPE_ID =k.spe_ID
inner join edw.dim_date as j on i.ED_ID = j.date_ID
inner join edw.dim_time as l on i.ET_ID = l.time_ID
)
where a.specimen = k.specimen_key
and datepart(year,a.entry_time) = j.year
and datepart(month,a.entry_time) = j.month
and datepart(year,a.entry_time) = j.year
and datepart(day,a.entry_time) = j.day_of_month
and datepart(hour,a.entry_time) = l.hour
and datepart(minute,a.entry_time) = l.minute
)
```

Additionally, to make changes easier to look up, a new table "edw_logs" with the attributes table name and updating date were created. Every time the incremental load script is executed, a record is inserted into this log table so that this history is easier to access.



```
insert into [update_log].[edw_logs]([Table], [LastLoadDate])
VALUES ('fact_cultivation', convert(char(8),GETDATE(),112))
```

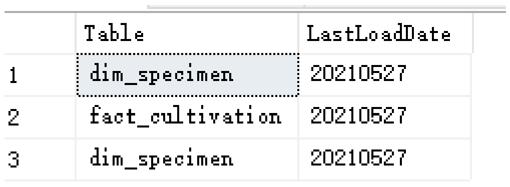


	Table	LastLoadDate
1	dim_specimen	20210527
2	fact_cultivation	20210527
3	dim_specimen	20210527

4.3.4 PowerBI

To visualize the data warehouse data by mushroom stage in order of growth stages, a new table Stage Order was added to the PowerBI data, which is managed by a relationship to the dimension specimen. The aim was geared towards simplicity for the end-user therefore the five most important visualizations were gathered. Focus was placed on clear and concise presentation of data with interdependent visualizations. Figure 21 illustrates the average mushroom age by stage using a doughnut chart, which is a recommended value-object to demonstrate part to whole type data.

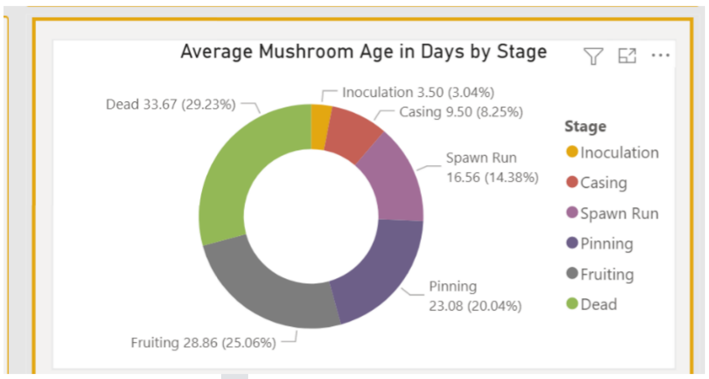


Figure 22 Mushroom Age by Stage

Line graphs were utilized to illustrate distribution of values and time variant data.

Figure 22 illustrates the average temperature in days for the Tuber Aestivum Specimen for the Inoculation Stage.

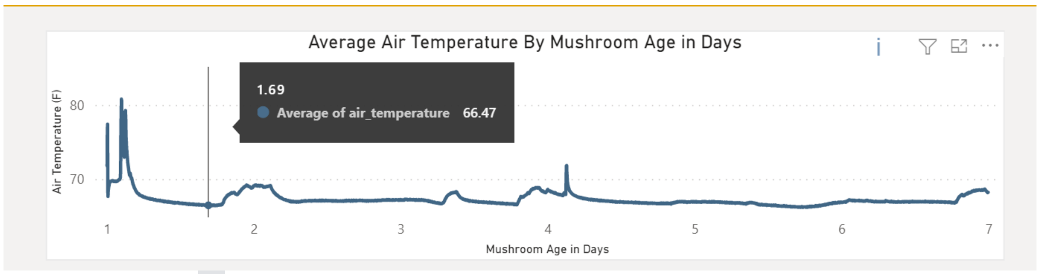
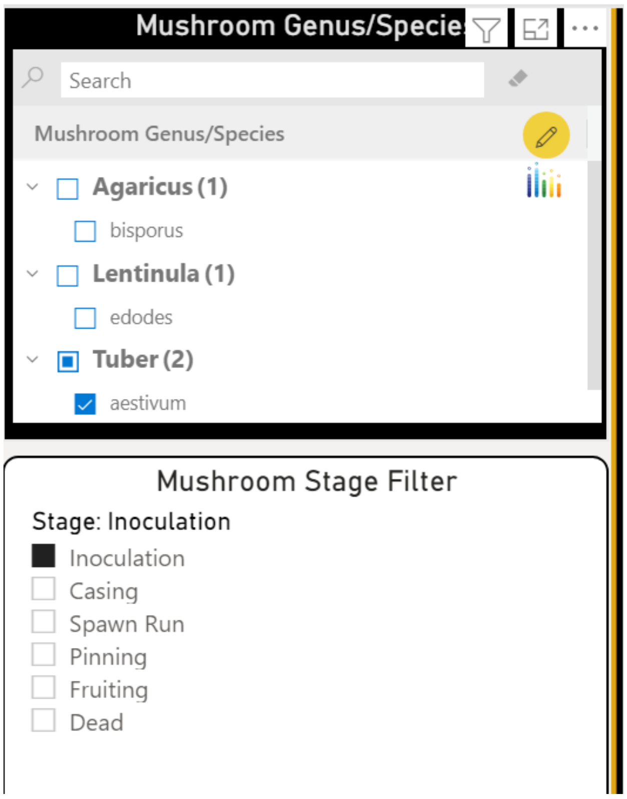


Figure 23 Time Variant for Inoculation Stage for Tuber Aestivum Specimen

Filters are visible on the dashboard with the selected Mushroom/ Genus/Species by hierarchy and Mushroom Stage checked so that the user can select and identify the mushroom genus/species and/or mushroom stage displayed on the dashboard.



Mushroom Genus/Specie

Search

Mushroom Genus/Species

- ☐ Agaricus (1)
  - ☐ bisporus
- ☐ Lentinula (1)
  - ☐ edodes
- ■ Tuber (2)
  - ☑ aestivum

Mushroom Stage Filter

Stage: Inoculation

- ■ Inoculation
- ☐ Casing
- ☐ Spawn Run
- ☐ Pinning
- ☐ Fruiting
- ☐ Dead

Figure 24 Dashboard Filter

Daria-Maria Popa

4.4 IoT Part

The IoT implementation is split into two main parts: the bridge gateway application and a FreeRTOS based embedded application. They are connected through the use of the Loriot Network Server, which handles the passing of messages between the two components.

Besides the ‘springframework’ libraries and dependencies used to set up and run the Web API, there were certain other more important libraries are used for the gateway server. The added libraries are chiefly connected to the efficient handling of the information sent to and received from the Loriot server. These libraries include: ‘fasterxml.jackson’ for managing the Json messages, java.util’s ExecutorService class for the thread pool which handles the user threads listening for new hardware information, ‘binary’ libraries from Apache for decoding and encoding Hex strings, and finally the ‘java.net.http.WebSocket’ for the WebSocket connection between the Gateway server and the Loriot server.

As described in the previous chapter, there are design patterns that have been implemented in order to achieve better maintainability, flexibility and separation of concerns for the Gateway Application.

The Service pattern has been simply represented using the java packages, the names respecting java conventions for easier-to-understand code. Because the gateway’s main role is to receive data from Loriot and send it forward to the Data side, the only needed service for the client is the Sensor data service (see Image 4.1).

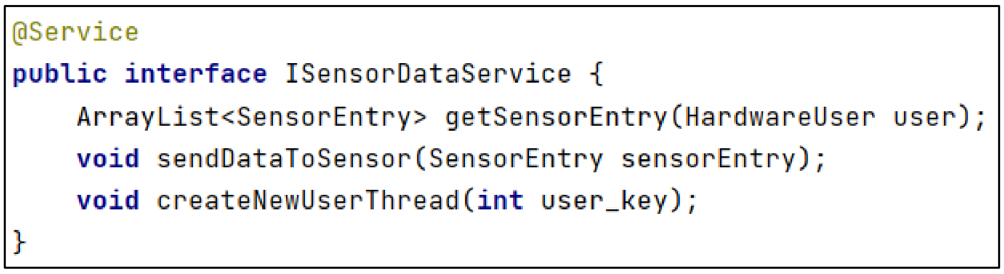
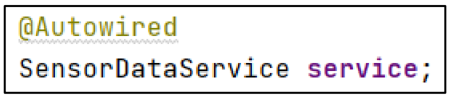

```
@Service
public interface ISensorDataService {
    ArrayList<SensorEntry> getSensorEntry(HardwareUser user);
    void sendDataToSensor(SensorEntry sensorEntry);
    void createNewUserThread(int user_key);
}
```

Image 4.1: Sensor service

The available operations are related to the ones in the controller: starting a new user thread to listen for information from Loriot, sending a message to a hardware equipment, and returning the received data from the user thread. The sensor data is saved to the database only after it reaches the Data server side, in order to keep the data persisting and modelling responsibilities separated from the data receiving and forwarding responsibilities.

Dependency injection pattern is implemented using Spring annotations, by declaring the 'ISensorDataService' interface as a '@Service' (see Image 4.1), and using '@Autowired' in the related 'SensorDataController' at instantiation (see Image 4.2). This improves the flexibility even further by eliminating the dependency of creating the Service from the one of using the Service.

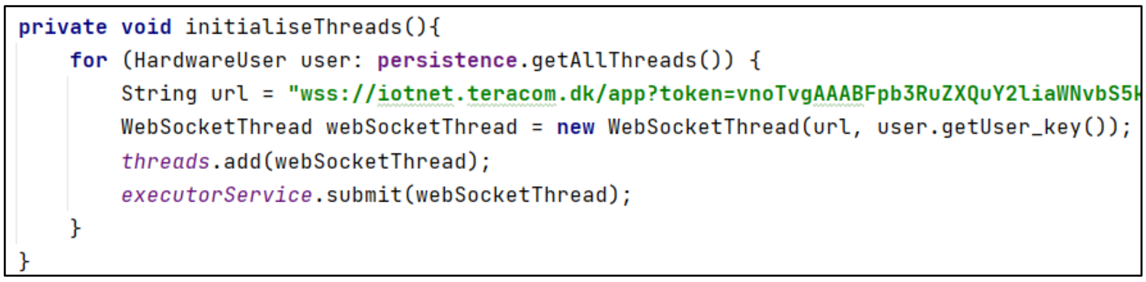


```
@Autowired
SensorDataService service;
```

Image 4.2: Dependency injection in sensor controller

Because the Gateway application is a Web server, REST is also used as an architectural style in combination with HTTP as the chosen RESTful communication protocol. The implementation of this style can be seen in the controller class, 'SensorDataController', through the stateless GET and POST HTTP operation as well as through the specific server-side implementation which indicates the separation of concerns aspect of the RESTful style.

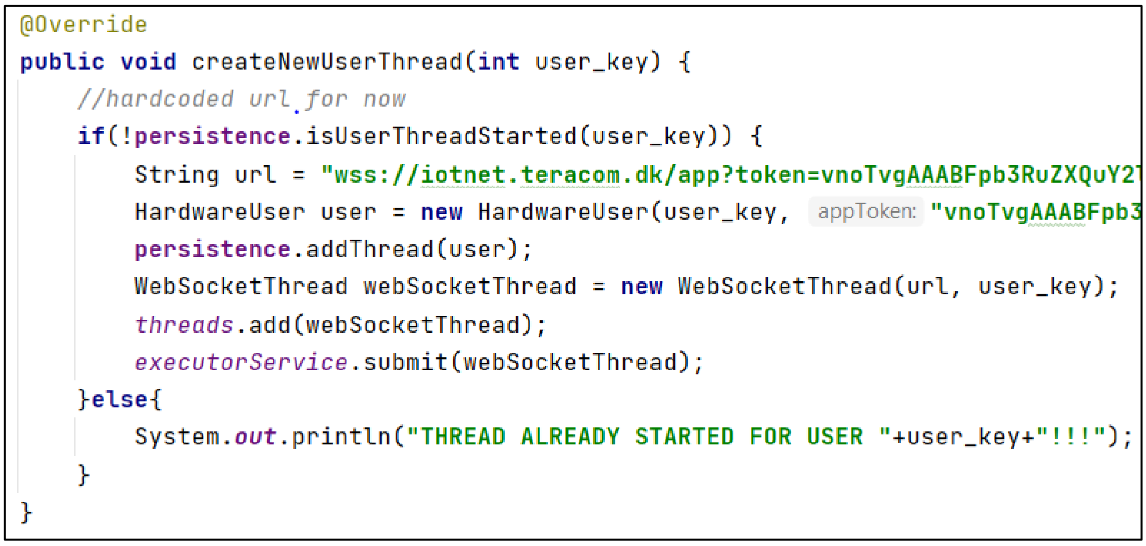
A local persistence file has also been implemented in order to save the already existing hardware user information, made up of the user's unique key and their Loriot application token. This is done in order to ensure that in case of the server failing and needing to be restarted, the user threads that were listening to information will simply be reinitialised (see Image 4.3), without the controller method for creating a new thread needing to be called multiple times again by the client. The 'SensorDataService' class is the one using the 'UserThreadFile' to initialise and manage the user threads.



```
private void initialiseThreads(){
    for (HardwareUser user: persistence.getAllThreads()) {
        String url = "wss://iotnet.teracom.dk/app?token=vnoTvgAAABFpb3RuZXQuY2liaWNvbS5
        WebSocketThread webSocketThread = new WebSocketThread(url, user.getUser_key());
        threads.add(webSocketThread);
        executorService.submit(webSocketThread);
    }
}
```

Image 4.3: Initialising saved user threads

The Gateway application works by following a couple of steps. Firstly, when the server is started, the service class retrieves any saved user information from the persistence, creates 'WebSocketThread' objects with it, and adds them to the thread pool to be run (see Image 4.4). Every 'WebSocketThread' object also has a 'WebSocketClient' attribute, therefore, when a thread is created, the web socket connection to Loriot is also established using the user's application token to create the URL.

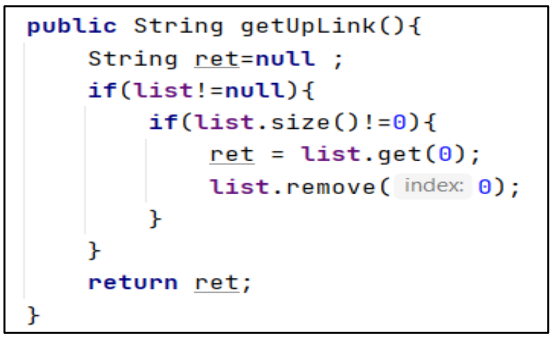


```
@Override
public void createNewUserThread(int user_key) {
    //hardcoded url for now
    if(!persistence.isUserThreadStarted(user_key)) {
        String url = "wss://iotnet.teracom.dk/app?token=vnoTvgAAABFpb3RuZXQuY2
        HardwareUser user = new HardwareUser(user_key,  appToken: "vnoTvgAAABFpb3
        persistence.addThread(user);
        WebSocketThread webSocketThread = new WebSocketThread(url, user_key);
        threads.add(webSocketThread);
        executorService.submit(webSocketThread);
    }else{
        System.out.println("THREAD ALREADY STARTED FOR USER "+user_key+"!!!");
    }
}
```

Image 4.4: Creating a new user Thread in service and saving it to persistence

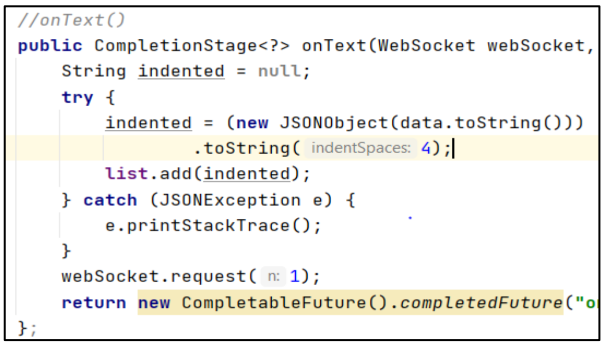
Because of the project constraints with using the Loriot Network, only one URL was available to use, therefore this was hardcoded in the methods, but the 'HardwareUser' class was created having in mind the future of the project and having multiple users with different tokens. The thread object then runs forever, periodically asking the socket for any new information, wrapping it into a SensorEntry class, and saving it to a local ArrayList.

The 'WebSocketClient' implements the java.net 'WebSocket.Listenet' interface and its methods (see image 3.??). When new information is sent through the Loriot server, it is heard in the 'onText' method (see Image 4.6) and saved into a local List of String objects, which is accessed through the 'getUpLink' method by the 'WebSocketThread' (see Image 4.5). After the information is sent to the thread class, the list is emptied to avoid duplicate data being sent to the client.



```
public String getUpLink(){
    String ret=null ;
    if(list!=null){
        if(list.size()!=0){
            ret = list.get(0);
            list.remove( index: 0);
        }
    }
    return ret;
}
```

Image 4.5: Retrieving uplink from the socket client to the socket thread



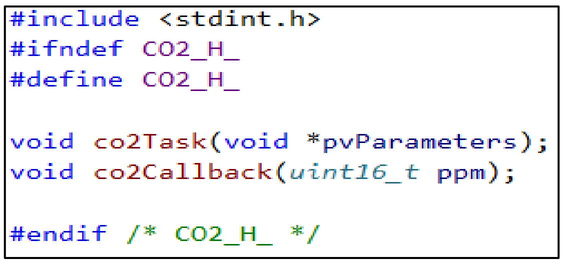
```
//onText()
public CompletionStage<?> onText(WebSocket webSocket,
    String indented = null;
    try {
        indented = (new JSONObject(data.toString()))
                .toString( indentSpaces: 4);
        list.add(indented);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    webSocket.request( n: 1);
    return new CompletableFuture().completedFuture("o
};
```

Image 4.6: Hearing new uplinks in the socket client

At any time while the Gateway server is running, any of the three controller methods in the 'SensorDataController' can be called by the client. Each will forward the responsibility to the related service class method. The 'createNewUserThread' method will save the information to the 48ersistence file and then run the new thread, while the 'getSensorEntry' and 'sendDataToSensor' will either ask the thread with the specified user key to return the information saved in its local SensorEntry list, or create a jsonTelegram with the specified info and call the 'sendDownlink' in the socket class, therefore sending information to the wanted hardware.

The embedded program uses FreeRTOS libraries for the creation and scheduling of tasks. Extra driver libraries were included in order to retrieve sensor data and connect to the LoRaWAN network. The drivers used in this project are 'mh_z19', 'lora_driver', 'rc_servo', 'tsl2591' and 'hih8120' for the CO2, LoRaWAN transreceiver, servo motor, light, and temperature and humidity components respectively. 'ATMEGA_FreeRTOS' and 'stdio', respectively 'stdio_driver' libraries are also used throughout the whole program for the scheduling and printing of messages to the console.

The general architecture has been implemented such that each task has its own header file and C class implementation, with the header file defining the task and callback (where necessary) methods headers, and the C classes including the respective '.h' files and implementing the methods.

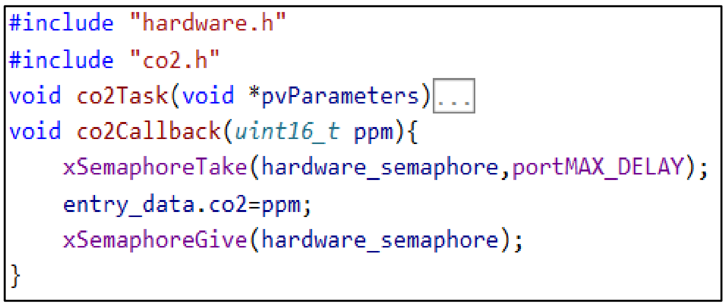


```c
#include <stdint.h>
#ifndef CO2_H_
#define CO2_H_

void co2Task(void *pvParameters);
void co2Callback(uint16_t ppm);

#endif /* CO2_H_ */
```

Image 4.7: 'co2.h' snippet

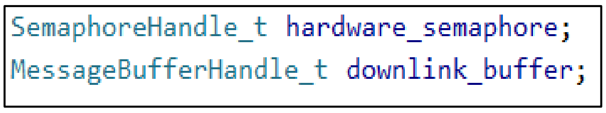


```c
#include "hardware.h"
#include "co2.h"
void co2Task(void *pvParameters)...
void co2Callback(uint16_t ppm){
    xSemaphoreTake(hardware_semaphore,portMAX_DELAY);
    entry_data.co2=ppm;
    xSemaphoreGive(hardware_semaphore);
}
```

Image 4.8: 'co2.c' implementation snippet

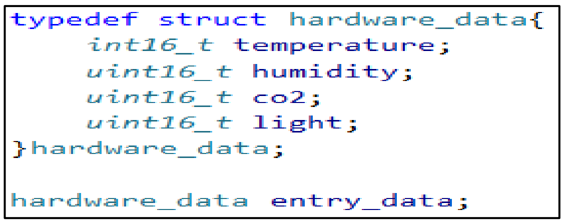
For storing and accessing information from all the sensors, the 'hardware.h' header file has been implemented, containing the struct definitions for both the 'entry_data' collected from sensors, and the 'desired_data' received through downlink and used by the motor to decide when and how it should move.

Because the same instance of the struct ought to be used by all tasks, for simplicity and better memory usage, there is no 'hardware.c' implementation, the structs are instantiated in the header (see Image 4.9) right after their definition. The Mutex Semaphore used by all the tasks (see Image 4.8), as well as the Message Buffer used by the LoRaWAN tasks to receive information sent by the Gateway are also defined in the 'hardware.h' file (see Image 4.10).



```
SemaphoreHandle_t hardware_semaphore;
MessageBufferHandle_t downlink_buffer;
```

Image 4.9: Semaphore and Message Buffer definition in ‘hardware.h’



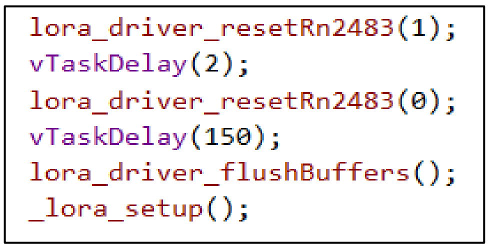
```
typedef struct hardware_data{
     int16_t temperature;
     uint16_t humidity;
     uint16_t co2;
     uint16_t light;
}hardware_data;

hardware_data entry_data;
```

Image 4.10: ‘entry_data’ struct definition and instantiation in ‘hardware.h’

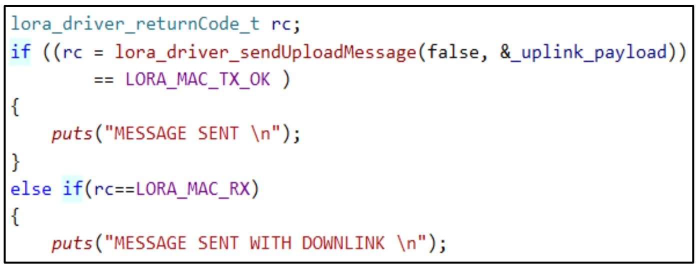
The system and tasks initialization are done in the main class, ‘main.c’. This follows a specific order so as to ensure the correct operation of the embedded system, with the system and sensor initialization first, then the task and semaphore creation, and finally the task scheduler being started, right before the infinite while loop which ensures the program never returns from there. The system initialization is done by calling the ‘initialiseSystem’ method, which also handles setting the ports for the LEDs and creating and linking the downlink buffer to the LoRa driver.

In order to join the LoRaWAN network server, the ‘LoRaWANHandler’ task must first reset and then go through the OTAA setup, which is done by calling the ‘_lora_setup’ method (see Image 4.11). The task will try to join the network ten times before yielding if denied.



```
lora_driver_resetRn2483(1);
vTaskDelay(2);
lora_driver_resetRn2483(0);
vTaskDelay(150);
lora_driver_flushBuffers();
_lora_setup();
```

Image 4.11: lora task uplink & downlink

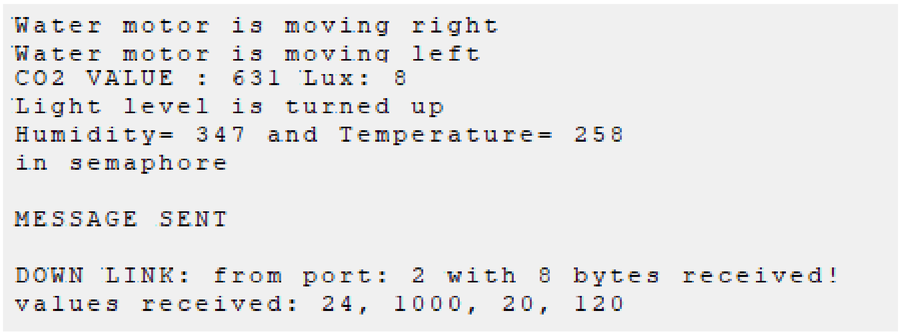


```
lora_driver_returnCode_t rc;
if ((rc = lora_driver_sendUploadMessage(false, &_uplink_payload))
        == LORA_MAC_TX_OK )
{
    puts("MESSAGE SENT \n");
}
else if(rc==LORA_MAC_RX)
{
    puts("MESSAGE SENT WITH DOWNLINK \n");
```

Image 4.12: lora setup inside task

Afterwards, if the connection is ‘accepted’, the uplink and downlink payloads are given lengths and values, and the infinite for loop is started. In the loop the task waits a certain time before constructing the uplink payload bytes with the ‘entry_data’ information saved by the sensor tasks in the ‘hardware.h’ file.

The message is then sent and if successful a small 500ms window of time opens when a downlink message can be received in the message buffer, its data decoded and saved in the ‘desired_data’ struct of the shared data component, ‘hardware.h’. The existence of a downlink message is indicated by the LoRaWAN response code (see Image 4.12). This cycle repeats infinitely together with the other four tasks resulting in a loop of measuring values, motors moving if needed, and LoRaWAN task sending and receiving information (see Images 4.13 and 4.14).

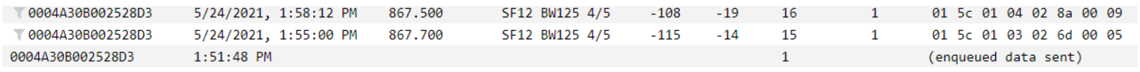


```
Water motor is moving right
Water motor is moving left
CO2 VALUE : 631 Lux: 8
Light level is turned up
Humidity= 347 and Temperature= 258
in semaphore

MESSAGE SENT

DOWN LINK: from port: 2 with 8 bytes received!
values received: 24, 1000, 20, 120
```

Image 4.13: Hterm capture



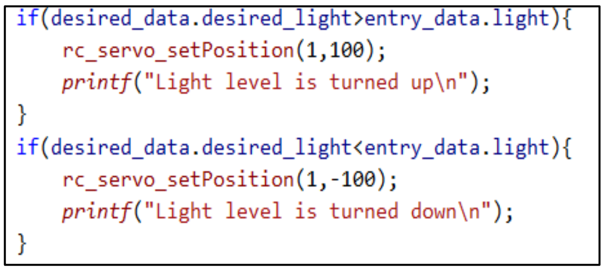
```
0004A30B002528D3   5/24/2021, 1:58:12 PM   867.500   SF12 BW125 4/5   -108   -19   16   1   01 5c 01 04 02 8a 00 09
0004A30B002528D3   5/24/2021, 1:55:00 PM   867.700   SF12 BW125 4/5   -115   -14   15   1   01 5c 01 03 02 6d 00 05
0004A30B002528D3   1:51:48 PM                                                      1       (enqueued data sent)
```

Image 4.14: Loriot | LoRaWAN gateway server capture

All of the taskwork that deals with saving or retrieving shared data from 'hardware.h' is done inside the mutex semaphore in order to protect said data (see Image 4.8). The mutex semaphore has been chosen because it is better at protecting shared data due to its mutual exclusion between tasks through the use of the priority inheritance mechanism.

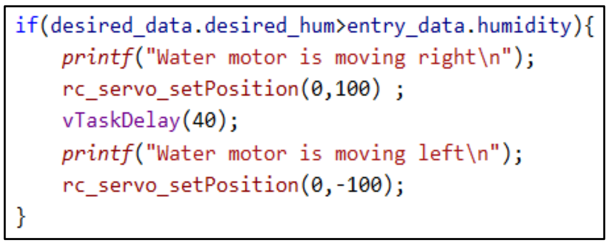
The tasks are also each given a priority, with LoRaWANHandler having the highest (priority 3), then the watering motor task (priority 2), then all the other sensor tasks (priority 1). The priority helps further synchronize the system and ensure that some of the most important functions (such as sending an uplink) are valued as such by the scheduler.

The motor task in 'motor.c' handles one of the two motors, namely the watering one, starting only if the measured humidity is lower than the desired one (see Image 4.16). Because it releases colder water into an enclosed space, this motor affects the air temperature too. The second motor, connected to the lighting system, is part of the light task in 'light.c' class and moves left or right, turning the light off or on, based on how the measured light level compares to the desired light level (see Image 4.15). These two ensure that the humidity and light levels always stay around certain values specified by the user.



```
if(desired_data.desired_light>entry_data.light){
    rc_servo_setPosition(1,100);
    printf("Light level is turned up\n");
}
if(desired_data.desired_light<entry_data.light){
    rc_servo_setPosition(1,-100);
    printf("Light level is turned down\n");
}
```

Image 4.15: Light motor



```
if(desired_data.desired_hum>entry_data.humidity){
    printf("Water motor is moving right\n");
    rc_servo_setPosition(0,100) ;
    vTaskDelay(40);
    printf("Water motor is moving left\n");
    rc_servo_setPosition(0,-100);
}
```

Image 4.16: Watering motor

5 Test

Levente Nagy

5.1 Web API Server

The most fragile part of the Server Application is the PersistenceService class, which has a Unit Test included in the source code, to ensure the correct working of the DAO and Database. Further black box testing was performed on the Web API endpoints themself, using the Swashbuckle swagger interface of the server, ensuring the correct working of the Controller classes and the JwtMiddleware class.

No other unit tests were performed on either the Model classes or other Service classes. The reason for the lack of these tests is the pure simplicity of the model classes, the LogService and ConfigService and the required dependency of the SampleService. Additionally, the SampleService class performed without an issued during integration testing.

5.2 Enterprise Data Warehouse

After the integration testing, the testing of the data warehouse was performed during the test run of the system. The initial ETL process was executed on a set of dummy data first, filling and initializing the fact and dimension tables. Afterwards, while the test run of the system was producing measurement entries from the hardware, the process of the incremental load was tested, with no errors.

Daria-Maria Popa

5.3 IoT Part

For the testing of the IoT system, both white box and black box technologies have been used in order to ensure a complete and thorough testing of all the requirements and features.

For the embedded system the Unit Testing was achieved using the 'gtest' library from Google and mocking the macro functions from the 'FreeRTOS' and driver libraries using the FFF-Framework.

The Gateway server application was Unit Tested using the ZOMBIES Framework and Black Box tested through System tests, Non-functional tests and Regression testing after major updates.

The whole IoT system was functionally tested for the user requirements of the 'Mushroom+' system which are directly related to the Embedded and Gateway application. This was represented using test scenarios, test cases, and a requirement traceability matrix. The ZOMBIES Framework was used for the functional testing as well.

5.3.1 IoT Test Specifications

A. **Test Scenarios**

<u>Gateway Application:</u>

1. Starting a listening thread
2. Returning data to the user
3. Forwarding information to the Loriot Network Server
4. Stopping a listening thread

<u>FreeRTOS based application:</u>

5. Monitoring air temperature and humidity
6. Monitoring air carbon dioxide level
7. Monitoring light level
8. Sending message through LoRaWAN
9. Receiving message through LoRaWAN
10. Managing air humidity and temperature
11. Managing light level

B. **Test Cases** (for the complete list see Appendix – IoT Test Cases)

Test Scenario #2 – Returning data to the user

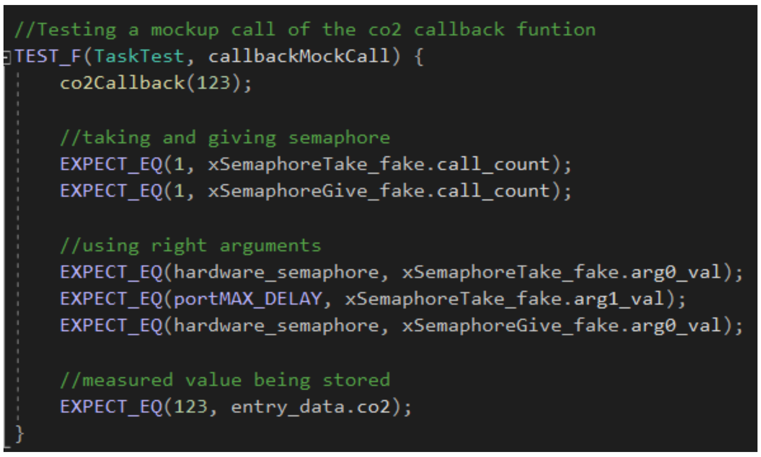
Test case ID	Test case Description	Test Steps	Test Data	Expected Results	Actual Results	Grade

TC21	Requesting data for an existing user	1. Call the specific GET controller method with the wanted user information	user_key = 155 user_token = Vdja6a7dSj	Data returned in a list format and local list cleared to avoid duplicate data transmission	As expected	Pass
TC22	Requesting data for an existing user when no data was yet received	1. Call the specific GET controller method with the wanted user information	user_key = 156 user_token = Vdja6a7dSj	Data returned as an empty list to avoid null pointer exceptions	As expected	Pass
TC23	Requesting data for a new user	1. Call the specific GET controller method with the new user information	user_key = 155 user_token = Vdja6a7dSj	Empty list returned as there is no such user thread started yet and related terminal message shown	As expected	Pass

Test Scenario #6 – Monitoring air carbon dioxide level

Test case ID	Test case Description	Test Steps	Test Data	Expected Results	Actual Results	Grade
TC61	Measuring air co2 level with sensor and task set up correctly	1.Start the main program for the FreeRTOS application	Related mh_z19 driver setup functions called correctly	The measured air co2 level will be shown and the value saved in the shared data periodically without affecting other tasks	As expected	Pass
TC62	Measuring air co2 level with sensor set up incorrectly	1.Start the main program for the FreeRTOS application	Related mh_z19 driver setup functions called incorrectly / not called	Error messages for initialising the sensor shown in the terminal	As expected	Pass

The black box functional testing for the embedded system provides a simple and efficient way requirements testing, while the Unit Testing focuses on the ensuring the correct implementation of code (see Images 5.1 and 5.2).



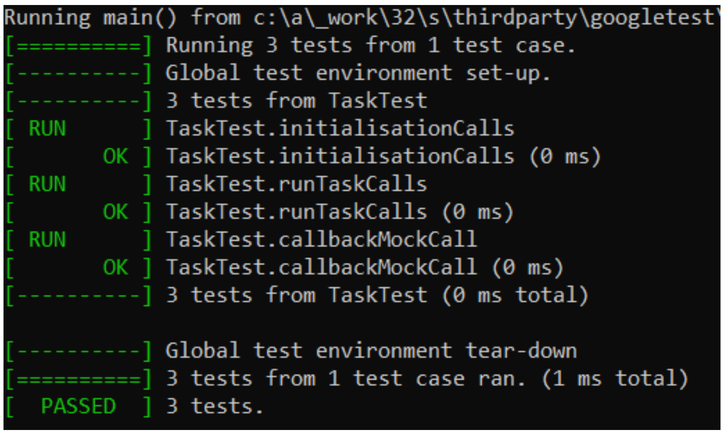
```
//Testing a mockup call of the co2 callback funtion
TEST_F(TaskTest, callbackMockCall) {
    co2Callback(123);

    //taking and giving semaphore
    EXPECT_EQ(1, xSemaphoreTake_fake.call_count);
    EXPECT_EQ(1, xSemaphoreGive_fake.call_count);

    //using right arguments
    EXPECT_EQ(hardware_semaphore, xSemaphoreTake_fake.arg0_val);
    EXPECT_EQ(portMAX_DELAY, xSemaphoreTake_fake.arg1_val);
    EXPECT_EQ(hardware_semaphore, xSemaphoreGive_fake.arg0_val);

    //measured value being stored
    EXPECT_EQ(123, entry_data.co2);
}
```

Image 5.1: Unit testing snippet for co2 task



```
Running main() from c:\a\_work\32\s\thirdparty\googletest\
[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from TaskTest
[ RUN      ] TaskTest.initialisationCalls
[       OK ] TaskTest.initialisationCalls (0 ms)
[ RUN      ] TaskTest.runTaskCalls
[       OK ] TaskTest.runTaskCalls (0 ms)
[ RUN      ] TaskTest.callbackMockCall
[       OK ] TaskTest.callbackMockCall (0 ms)
[----------] 3 tests from TaskTest (0 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran. (1 ms total)
[  PASSED  ] 3 tests.
```

Image 5.2: Unit testing result for co2 task

C. **Requirement Traceability Matrix** (for the IoT related/dependent requirements)

Requirement nr.	Requirement name	Test case ID & Grade	Final Grade
2	As a user I want to be able to monitor the air temperature with my hardware	TC51 – Pass TC52 – Pass	Pass
3	As a user I want to be able to monitor the air humidity with my hardware	TC51 – Pass TC52 – Pass	Pass
4	As a user I want to be able to monitor the air carbon dioxide level with my hardware	TC61 – Pass TC62 – Pass	Pass
6	As a user I want all sensory data to be collected, to provide useful insights on my environmental conditions	TC11 – Pass TC12 – Pass TC13 – Pass TC21 – Pass TC22 – Pass TC23 – Pass TC31 – Pass TC32 – Pass TC33 – Pass	Pass

		TC41 – Pass TC42 – Pass TC43 – Pass TC81 – Pass TC82 – Pass TC83 – Pass	
8	As a user I want to be able to manage the air humidity with my hardware	TC91 – Pass TC92 – Pass TC93 – Pass TC101 – Pass TC102 – Pass TC103 – Pass	Pass
12	As a user I want to be able to manage the air temperature with my hardware.	TC91 – Pass TC92 – Pass TC93 – Pass TC101 – Pass TC102 – Pass TC103 – Pass	Pass
14	As a user I want to be able to monitor the light level with my hardware	TC71 – Pass TC72 – Pass	Pass
15	As a user I want to be able to manage the light level with my hardware	TC91 – Pass TC92 – Pass TC93 – Pass TC111 – Pass TC112 – Pass TC113 – Pass	Pass

Test case results:

- Number of test cases: 30
- Number of passed test cases: 30

Functional testing results:

- Number of related requirements: 8
  - Of which:
    - Must have: 5
    - Should have: 3
- Number of passed requirements: 8

5.4 Android team

Uldis Alksnis, Bogdan Mezei and Kristóf Lénárd

For the Android team, three types of testing were used: white-box, gray-box and black-box testing.

During connection development (that is, the development of the code connecting the application and the API), white-box testing was used. This was done to ensure that the connection between the API (developed by the Data team) and the client application (developed by the Android team) matched, and to further verify the feasibility of a common model.

Before deployment, gray-box testing was used. There the running code was connected to a debugger, and various actions were attempted, to be able to evaluate how the current stack and heap changes, and from those, derive whether the system works as intended. Most of this testing was done to attempt to filter any issues that the system may have, especially in areas that would have been difficult or inappropriate to test otherwise.

During deployment and the final pre-release verification, black-box testing was used. This was performed by group members not belonging to the Android team, and were used to ensure that user inputs cannot interfere with system operations. Another use was to verify whether the Android team’s acceptance and functional testing was correct.

6 Results and Discussion

Natali Munk-Jakobsen

All essential functional and non-functional requirements related to the IoT part have been implemented and integrated according to the planned project timeline and use case descriptions.

- Sensors work properly and can measure CO2, humidity, temperature and light values correctly.
- Measured data can be sent to the central server successfully.
- The LoRaWAN transceiver can transmit and receive data using Loriot Network.
- Light level, humidity and air temperature can be managed by the user.
- Motors work as intended when user input is received.
- Unit testing and black box testing has been completed and passed.
- Integration with the central server has been successfully completed.

Kristóf Lénárd

The Android sub-group was able to achieve a satisfactory result. The system was implemented, with all intended tasks finished. The tasks that remain unfinished are due to external factors, and the finished tasks are meeting the acceptance criteria set during the project planning. The outcome of the project is deemed satisfactory.

7 Conclusions

Daria-Maria Popa

The IoT system works overall completely as intended, measuring values and sending them over the Loriot Network and then further to the central server reliably, achieving the user's need of fast real-time data about the conditions that their mushrooms are in.

From the requirements' perspective, all of the must and should have requirements have been fulfilled successfully, both the ones related to the embedded side of the system, and the ones related to the Gateway Bridge application. The implemented architecture of the system is also in concordance with the desired outcome described in the design chapter, and provides high levels of maintainability, reliability, and scalability.

However, as mentioned in the previous chapter, improvements could be made for the 'could have' requirements, and to the way these could be implemented in the system to provide even more useful information for the mushroom farm owners.

All in all, the IoT system was created following the users' requirements, with the goal of managing the indoor climate of mushroom farms.

8 Project future

Daria-Maria Popa

Despite fulfilling almost all functional and non-functional requirements and providing a sufficient architecture, the system will always be open for improvement in order to be more usable and preferable.

At the present, there are only two simple motors managing the desired light and humidity levels for the farm. With the required hardware components, managing light and humidity levels could be improved in the future version of the project with a more complex watering and lighting system. In addition, soil humidity, temperature level and CO2 level could be added as new system features with an upgrade to the number of actuators the hardware could support.

Since only one URL was given for Loriot Network, the current version of the system works as a single user system with one hardcoded URL. Nevertheless, the system has been designed and implemented in order to be used by multiple users with different tokens considering the future of the project, therefore offering great manageability and portability.

As a near future change, the local user thread persistence in the Bridge application could be expanded to handle the latest received information, ensuring no data loss in case of very unlikely gateway server failures. The embedded system could also undergo unit tests for all of the not unit tested tasks, to confirm that all features do work as intended, not only from the black box testing perspective.

Kristóf Lénárd

There are many things that the Android team could do. There are additional features, that were either unachievable (due to external and extenuating circumstances, such as not having enough hardware components for certain requirements), or predetermined to not be included in this release. There are, of course, other things besides this that could be improved.

For example, while the Android part of the project is infinitely scalable, there are quality improvements that could make the software run even better, or perhaps support, with some features missing, some older versions of the Android operating system. All of these, however, are extra features.

9 Sources of information

Fortunebusinessinsights.com. 2020. Mushroom Market Size, Share, Growth - Industry Analysis 2026. [online] Available at: https://www.fortunebusinessinsights.com/industry-reports/mushroom-market-100197 [Accessed 1 June 2021].

Android Developers, *Android Studio - Android Platform/API Version Distribution*. Available at: <https://developer.android.com/> [Accessed 2 Jun. 2021].

Anon, 2021. *Guide to app architecture* [online] Available at: <https://developer.android.com/jetpack/guide#recommended-app-arch> [Accessed 2 Jun. 2021].

Auth0 Inc., 2021. [online] JSON Web Tokens. Available at: <https://jwt.io/> [Accessed 2 Jun. 2021].

Eolss.net. 2019. [online] Available at: https://www.eolss.net/Sample-Chapters/C17/E6-58-06-03.pdf [Accessed 1 June 2021].

10 Appendices

- Appendix A – Project Description
- Appendix B – Hardware and Gateway Installation Guide
- Appendix C1, C2 and C3 – Source code.
- Appendix D1, D2 and D3 – Source code documentation.
- Appendix E1, E2 and E3 – Diagrams with exported images.
- Appendix F – Swagger documentation
- Appendix G – Group Contract.
- Appendix H – Process documentation.

Bring ideas to life
VIA University College

Mushroom++ Process Report

IoT group: Daria Maria Popa (293087), Natali Munk-Jakobsen (293132) and Mihai Anghelus (293169)

Data group: Levente Nagy (293115), Audrius Sauciunas (293156), Samantha Nettesheim (293089), Shaoyu Liu (294404)

Android group: Kristóf Lénárd (293110), Bogdan Mezei (293137), Uldis Alksnis (293168)

Supervisors: Erland Ketil Larsen, Astrid Hanghøj and Kasper Knop Rasmussen

VIA University College

49125 characters

Software Technology Engineering

4th Semester

4th June 2021

Version: June, 2021

Table of content

1 Introduction

During this project, two methodologies were used to execute tasks and proceed toward the predefined goal. These were Agile Unified Process (AUP) and the Scrum framework. Unified Process consists of four distinct phases: Inception, Elaboration, Construction and Transition.

During the first phase, Inception, the team defined the base technical requirements of the project, agreed on a common interpretation of facts concerning both the problem domain and the proposed solution, and established the core non-technical requirements, such as solution functionality, targeted process and timeline, and feasibility. This phase ended on the 17th of March 2021, beginning the next phase.

During the second phase, Elaboration, the team established the groundwork for all future work. Common design elements, such as the connectivity between different parts of the system, and general architecture were established, with supporting materials such as descriptions, core documentation and a working proof of concept.
During the second half of this phase (from the 7th of April 2021), the Scrum framework was also used to support the development. This was due to a variety of factors which will be further detailed in the Project Execution section.

For Scrum, Levente Nagy served as Product Owner and Kristóf Lénárd served as Scrum Master. All other team members served as members of the development team. This team was organized into smaller departments, consisting of three or four members, working on a distinct part of the system, led by the agile role of technical leads. Scrum was organized regularly, with the usual set of meetings, ie. sprint planning, daily scrum, sprint review and sprint retrospective. Further details can be found in the Project Execution section. Scrum was used, as previously mentioned, during the second half of the Elaboration phase and during the Construction phase.

During the Construction phase, the main focus was on the development of the software code, along with all required supplementary materials such as diagrams, reports, and other documentation. This was the longest phase of the project, stretching from the 28th of April all the way to the 2nd of June, with more frequent working days than before.

On the 3rd of June 2021, the project reached Transition phase. This phase was the shortest, even though, in theory, it could possibly be indefinitely long, since this phase consists of post-release support. This was also when the system was deployed to its currently final Production Release iteration- This phase also included finalization of supplementary materials such as user guides, and the final hand-in of the project.

2 Group Description

2.1 Android team

The Android team had three group members by the end of the project: developer Uldis Alksnis from Latvia, developer Bogdan Mezei from Romania and technical lead and Scrum master Kristóf Lénárd from Hungary.

In our group, tasks were divided fairly evenly, with all members working on coding, Kristóf working on the process report, along with handling the compilation of materials for the hand-in, while Uldis and Bogdan worked on the project report and technical materials.

We all had some experience, of course, this being our fourth project at VIA. This we feel helped us in many ways. One was that we have had more experience with software engineering and development in general, enabling more individual work, which is one of the core tenets of both Scrum and AUP. We have also known more about the issues we might have ran into during the project, which helped us in estimating what we need, and what we can accomplish in a given timeframe. Both of these were vital to the successful completion of the project.

In the group, we have emphasized working with the other philosophies of the Agile Unified Process, besides the aforementioned individual work.
We have not based our work on huge amounts of documentation, but planned connectivity and other things only in advance, the ones where substantial modification would have needed the other teams to adjust as well. Obviously, by using Scrum, we have worked in an agile context. We, from start to finish, prioritized (after completing general architectural duties) working on the most high-value requirements, only adding others after we had known that what we needed earlier worked.

All of these were core issues that we considered as such; therefore, we feel that our groupwork was successful in this project.

2.2 Data team

The Data group consists of four members, Audrius Sauciunas (Lithuania), Levente Nagy (Hungary), Samantha Nettesheim (United States) and Shaoyu Liu (China). The group was formed based on previous collaborations over the last three semesters.
The cultural differences at times were apparent but we worked through our differences throughout the project period to maintain a healthy work environment.

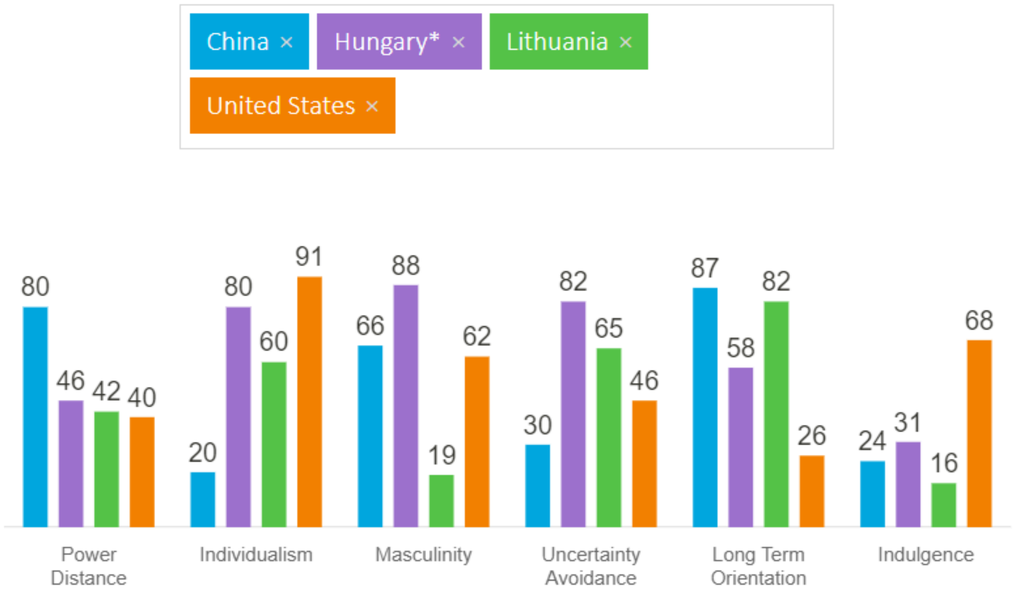
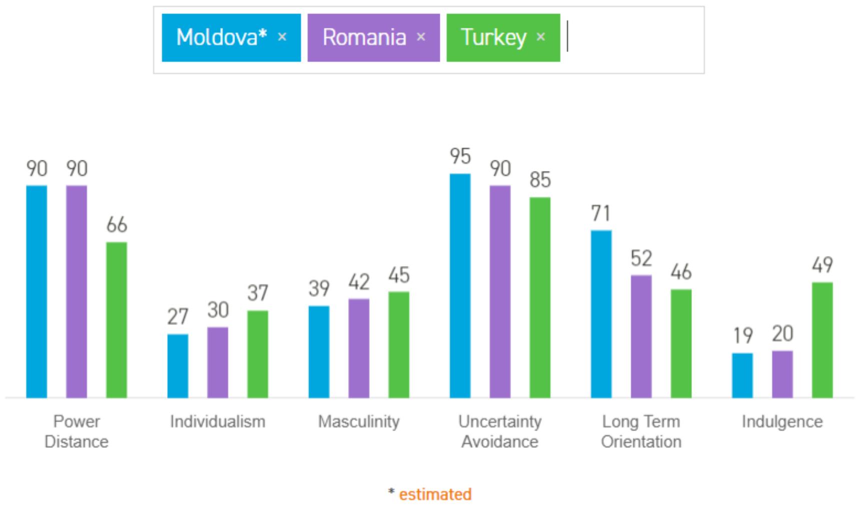


Figure 1 Hofstede's Cultural Dimension (Hofstede, n.d.)

The Hofstede's cultural dimensions provide us a reminder of how our cultural backgrounds influence our behavior and interpretations of other's behavior. As the group had not experienced working together in this specific constellation, the adjustment to each other's habits was a challenge. In addition, previous disappointments in group work and with each other influenced our feelings towards one another. This was a disadvantage and our initial progress suffered. We experienced a few weeks of disconnect during the design and implementation which was resolved through communication and over the course of time. Fortunately, the group started the project design and implementations phase early so we were able to mostly recover from our initial missteps. Group members were encouraged to assert themselves about their feelings and concerns. While some members were more reticent to discuss such matters than others, as the project progressed, members found confidence in each other and themselves which provided for a positive work environment.

2.3 IoT team

The IoT team for this semester project is formed of three people: Daria, Natali and Mihai. We were all lucky to have known and worked together before, which aided with the speed of the team integration and the general organization.



All three members come from different countries, which brought a lot of diverse views and experiences to the team, a fact which we considered really valuable. Even with Natali being from Turkey, Mihai from Moldova, and Daria from Romania, countries with unique and different cultures, we all shared similar views on teamwork and task prioritization.

The Hofstede profiles for our countries had very similar values, with a high power distance, low individualism and masculinity, high uncertainty avoidance, and a medium long term orientation (see Image 2.1). The indulgence for Turkey was the only noticeably higher value than for the other two countries (49 versus 19 / 20), but we all personally agreed with a high level of indulgence. For the power distance and masculinity, we also went against the values of our countries, choosing teamwork as equals and quality of life over goals and competition.

Our personal colour profiles were mostly composed of blue and green, with little red or yellow, which also indicates our common preference for good communication and teamwork, striving to work as equals and avoid conflicts by talking through ideas and listening to everyone equally, things which were very true for our team and made the project work a really pleasant thing to do.

Based on our country values and personal profiles, we therefore created a team which held as main values: communication, unity, equality, and freedom of expression and speech. We worked together as teammates but also as friends, shared everything equally and fairly, and managed to achieve all our goals in due time.

3 Project Initiation

Before any group starts a project, there must be something that generates the group, a metaphorical force that gets the members of the group together. Of course, in our case, one aspect of this was the fact that the project is a mandatory part of our studies here at VIA. There were other factors, though.

At the beginning, some of us have tentatively formed two groups, each consisting of only a few people, expecting a similar project as we have experienced in previous years. This, as it later turned out, was not the case. When the project description meeting was held and the full scope of the project detailed, we had known that we needed more people. Soon after a group was generated – the original group of seven. This was still too few, so we sought to attract more people, ones that we knew we could trust to accomplish their tasks. Thus, through friends and contacts, we came into contact with another small group. This group then joined us, bringing the count of our members to eleven. While this number would later be reduced to ten by the unfortunate departure of Eva Nikolaeva, this was still a number that was sufficient – and thus, the group came to be.

So now we had a group. We asked the question: “What next?”. The answer was obvious – we need to have something as our project. For that we needed an idea, and so we sat down (unfortunately, due to the corona-situation, not in person, but online, which we kept on using through the whole project). We have discussed many ideas, iterating through the ones that we thought of as interesting or otherwise appropriate. The results of this can be seen in Appendix C. In the end, we reduced the final round of ideas to two, and voted on the better one – thus, we selected our topic.

We then knew what should be our next step – forming a consensual understanding of the problem, and detailing our way of solving it.

4 Project Description

The first challenge when writing the project description came from the fact that most of us were unfamiliar with the problem domain, that is, the processes and inner workings of fungiculture. This problem was tackled by intense research, and with the most experienced member of our group, Audrius Sauciunas, working to secure more data from personnel who were then working in the industry. This gave us a significant amount of data on which we could base our background description. This helped us define the problems within the entire problem domain more precisely, and have allowed us to select a realistic goal.

Many of the following parts of the project description phase were derived from this information, as per usual procedure. These sections were no less important, however, since, for example, the supporting questions of the problem statement, formed the basis of our functional requirements later.
These parts were also critical procedurally. Before, we have worked on either single-issue problem resolutions (such as the decision of the idea), or on heavily factual texts (such as the background description). These sections, however, taken all that and added the problem of balancing the derivation of certain elements from others, and negotiating which aspects should we focus on. Both of these are critical to the process of software development. Therefore, we have, with the benefit of hindsight, have seen this as the final rehearsal of these before starting the more technical work on the project. During these, all members demonstrated such conduct and integrity that it was obvious to all members what to expect if and when the group again had similar meetings. We have aimed at having similar discussion in both procedure and conduct for the entire remainder of the project, a goal that we hope we have achieved.

This phase was, however, perhaps the most important in forming a preliminary way of meetings. During this phase, we have agreed on how to hold meetings, both with regards to meeting issues and procedures, and the conduct of the person chairing the meeting. These we have sustained during the rest of the project. Some of these matters derive from the group contract – however, some are merely procedural, which occasionally required adjustment.

The following sections, however, were more technical in nature. For example, the time schedule, while without a doubt important, was a highly factual and technical document, with perhaps only one thing remarkable with regards to the process. This was the fact that the group showed admirable restraint and appropriacy in discussing these issues. There was nearly no time wasted, which proved to be of advantage later, as we have continued to show these traits in later procedural discussion, which have allowed us to spend more time on, and focus better on, the development of the project instead of having to spend additional time on resolving minor issues. We were still able to solve all issues – but the time and energy not expended on these were of great use in the project.

All in all, however, we view this phase as highly influential in conducting this project, and highly successful in enabling further work on the project.

5 Project Execution

5.1 General work

After getting the feedback on the project description document, and finalizing it based on that, we began working in earnest on the first phase of the project, that is, the analysis section. This, and the first part of the Design section belonged to the first half of the Elaboration phase of AUP, involving four disciplines: the main discipline of Modeling, and the supporting disciplines of Configuration Management, Project Management, and Environment.

Our execution here was very simple: we held regular meetings on every working day (ie. on Wednesdays), and talked about the things we needed.
First and foremost was the definition of the requirements, both functional and not. These we derived from two sources by consensus – the functional ones from the project description document, chiefly the Problem Statement and Definition of Purpose sections, while the non-functional ones were derived from the predefined mandatory requirements document.
From these, further documents were derived: the use cases, the supporting documentation for said use cases (such as activity diagrams and test cases), and other materials that composed the analysis of the problem domain.
All of these, however, had one and only one goal: to help define a document that can both be non-technical, and be the core of the technical implementation. This was none other than the domain model, which was the last document created for the Model discipline of AUP.
As this discipline Model, according to the official description of Agile Unified Process, is defined as “The goal of this discipline is to understand the business of the organization, the problem domain being addressed by the project, and to identify a viable solution to address the problem domain.”, the domain model and the other aforementioned documents fulfil this purpose. Therefore, we concluded that this discipline was completed

for the time being, and the team could move on to the next main discipline – Implementation.
Here is where I ask others to step in, and help me detail what the reflections of each team are in this phase, where we, while not working together on day-to-day issues, still relied on each other to form a coherent system.

5.2 Android team

The Implementation discipline began after the Model discipline was finished, and lasted for the second half of the Elaboration phase and the entirety of the Construction phase, running concurrently with the Test and, occasionally, the Deployment discipline.

This part was, naturally, the largest – this is where we had to develop the system, test it, and deploy the final version of it to gain documents that we needed for the hand-in.
To facilitate this, and support the AUP, we have employed another framework, both in-group and in-team: Scrum.
This, of course, had numerous advantages: experience, a clear and concise methodology of working, and a good way of tracking the supporting disciplines of the AUP.

The implementation of Scrum followed the constructed framework. Before each sprint, the project backlog was reviewed by the Scrum master and the product owner, and constructed or reconstructed in such a way that ensured continuous refinement of tasks, and facilitated the selection of sprint goals.
This was followed by a sprint planning meeting. During this meeting, the teams were assessed individually, with any member of the group able to ask questions and add comments to anything that was decided, with these being explicitly recommended in tasks involving multiple departments. After the tasks have been decided and estimated for all teams, the group clarified the acceptance criteria for that given sprint. When this, too, was concluded, the group voted to accept these resolutions as the sprint plan.

The daily scrum meetings were individual for each team. Usually, they started at 10, with the Android team's meeting taking place first (this was so that the Android team could start working while the Android team's technical leader Kristóf Lénárd, who was also Scrum master, could hold the meetings for the other teams). During this meeting, three things were usually discussed: the completed work, the work that the team members planned on doing that day, and whether there were any impediments to progress, either from a technical or a process standpoint. Occasionally, "after-party" meetings were conducted later, to discuss a problem in more detail.

At the end of the sprint, two meetings were conducted as a whole group, usually directly following each other. These were, of course, the sprint review and sprint retrospective meetings. The sprint review meeting, as usual, focused on the completed and incomplete tasks. We discussed the tasks that were completed, how they impacted both currently existing and future work and how they contributed to the sprint goal. Incomplete work was also discussed, along with reasons of delay, and how that impacted the future work. Finally, the product owner, as customer representative, made suggestions with regards to future work, which in turn impacted the next sprint planning meeting.
The other meeting was the sprint retrospective. Here we discussed what we thought of the past sprint, and what issues we had with the process. Sometimes, there were no issues, other times, however, the team resolved to have a quick brainstorming session to try and find solutions to any aforementioned issues. Usually these lasted for about 30 minutes with discussion, at the end of which the group resolved to adopt one or more of the proposed solutions. These were then employed by the teams, enabling better cross-team work.

The Android team feels that these methods were effective at driving the development forward. We feel that the group has avoided micromanagement, yet still was able to work together on their goals, even though the group was essentially working in three smaller pieces. We feel that out of the methods we are familiar with, this one was perhaps the most balanced with regards to working together in three small teams.

We also think that with these frameworks and methodologies, we have been able to work together efficiently, and were able to execute the tasks that we planned to. Therefore, we think this project was successful not only in a technical, but also in a procedural sense.

5.3 Data team

The data group followed AUP and Scrum for the project methodology. Kristof Lenard acted as Scrum Master for the entire group and Levente Nagy was the Product Owner as well as the team lead for the data group. The data group held daily scrum meetings to discuss our progress and challenges from the previous day as well as the intended work for the current day. The data tasks were assigned on a volunteer basis, tasks were executed together, in sub-groups or independently. Due to challenges that the group faced during the elaboration phase, executing tasks independently proved inefficient. The data group had to reassess the work established during the elaboration phase which led to in-depth discussions of the knowledge obtained from the Data Analytics Infrastructure (DAI) class. Upon implementation of the data warehouse, we continued to face challenges and it was necessary to work very closely with each other to complete these tasks. We continually met with the data supervisor to receive feedback on our new design. At times, we could not progress until we had received satisfactory communication from our supervisor.

The AUP/Scrum process proved effective for the data group but only because we started the design process early and we worked well together to recover from our mistakes. The group members were in constant communication and helped each other through design or coding challenges with detailed explanations for members when needed. We continually experienced challenges with our knowledge of the DAI material and sought explanations from the supervisor. Although, the supervisor was instrumental to the process, she could not advise the group that our design or implementation were always correct.

The results of the data aspects of the project fell short in some respects. Changes to the data warehouse resulted in destroying the PowerBI reports from which we were unable to recover. Therefore, dedicated efforts from the team to create useful visualizations with a limited data set were wasted and we cannot present our true abilities. However, we all worked on this effort equally, so the wound is spread amongst the members.

5.4 IoT team

For the execution of the project our main helping tool was Azure DevOps, where we listed all the requirements for the sprint and kept track of their status throughout the sprint duration. Although the SCRUM meetings were held together, all individual teams had daily short meetings for checking up on the progress, including our IoT team.

For most of the sprints, the IoT work was split into three major areas: the gateway application, the embedded application, and the documentation of changes or new features. The last sprint was especially aimed towards adding finishing touches to the code and documentation, finishing testing and preparing for deployment.

The sprint tasks were split into individual smaller tasks for each member to complete. Occasionally, especially for the first couple of sprints, essential tasks like setting up the hardware for the first time, were handled by meeting up and working together. Even when we were not working together physically, the semester project work days were done via Discord, so we could talk through the process, share the screen, and figure out any problems or questions together.

For the first 2-3 sprints our work focused on the embedded side and gateway separately and their related requirements, such as measuring values and ensuring local persistence in the embedded side, and setting up the Loriot connection and preparing for receiving data in the gateway. Afterwards, the work continued to be a cycle of sending and receiving information between the two sides, implementing new features, testing, fixing (with supervisor help sometimes) and deploying the changes.

After the connection between the embedded and gateway server was ensured, the other connection between the Data server and Gateway server was also continuously and thoroughly tested in collaboration with the Data team after any changes.

All in all, this workflow proved successful in implementing all the requirements, passing the tests, and constructing the documentation along the way, and therefore having the IoT system working as intended ready to deploy in time.

6 Personal Reflections

6.1 Data team

Audrius Sauciunas

This was the third semester marked by covid-19. This semester I finally felt like the online teaching didn't affect me negatively at all study-wise. Consequently, I had a great semester during which I managed to gain a tremendous amount of knowledge about the different subjects. At first, for our SEP project I chose to be in the IoT group, but after realizing that I have some sort of passion for data analytics and data warehousing I chose to switch the teams for the first and last time, ending up in the data team.

For the very first time during the 4 semesters, I was very content with my SEP team. Previously having experienced enormous amounts of stress due to team members lacking motivation, this semester I felt like everyone was finally on the same track. I enjoyed every single day we worked together, which in result helped us achieve better results as a team. Every part of the project was done with passion and care. It felt a little bit strange at first to see so much organization as a group of 4 as well as a team of 12. It wouldn't have been as well done if not for the outstanding work of the team leads.

We got our semester project theme just by brainstorming. I was very happy with the thematic as I've been thinking about growing mushrooms for a long time for bio fabrication. We all liked the idea and as such all worked together trying to research the topic. I had the honor to make some sort of an interview with people who are growing mushrooms, something I've never done before. I took this idea of interviewing others after our innovation weeks, after this we managed to shape our project in different ways. This experience helped me open my eyes on how important it is to connect with people that might be using your project later on instead of just playing the guessing game.

While working on the data warehouse, there were many things that weren't clear at first, but after having a dialogue with my teammates, though constant helping with explaining the different concepts and ideas I felt like it helped me solidify my existing knowledge about data warehousing. I realized that this is the field where a part of my passion lies and finally realized after so long what path as a software engineer, I would like to take.

All things considered; I am very pleased with working in this team. This semester was the most important to me so far, where I finally realized what I would like to do as a software engineer. I am happy that this semester project gave me so much insight on not only what it means to work together as a big group but also what it means to be responsible for your specific part, as the whole project depends on it.

Samantha Nettesheim

I chose to work on the data team because IOT and Android was not a viable option as I live in Aarhus and I don't have an Android phone. I now know how to use the emulator in Android Studio well enough but I faced so many challenges with it during the semester that it I did not feel that the Android team was an option. I generally like data, I enjoyed having Astrid as a teacher and supervisor. She is very understanding and empathetic and gets excited about the material and motivates by her own enthusiasm.

I feel disconnected from the other two disciplines, IOT and Android, since I did not actively participate in the coding for those in SEP. At least with Android we created our own projects but with IOT; the distance between completing class exercises and creating a project using the knowledge attained from class is enormous and I now lack that experience.

I enjoyed working with the four data group members. All the team members were dedicated and worked well together. Levente made for a valuable team leader, he was very open to concerns or ideas from the group. He has a high level of coding knowledge, but he was able to explain concepts without being condescending and at a level that is understandable for me. Kristhof as Scrum Master was also indispensable, he and Levente really led the project as a cohesive unit. I was able to express concerns or vent to him whenever I needed. He provided a lot of insight and help with technical and process related questions throughout the project. I feel that the efforts made by the team were exceptional and I found the experience and knowledge that I attained to be invaluable.

Shaoyu Liu

This is another semester that we have to study and work at home due to the current situation. In the beginning, I thought it'll be a tough time again, but somehow having a lot of similar experience makes it kind of easier for us to accept and adjust our cooperation. Although there're still inconveniences and obstacles during the online studying, I recovered quickly to my track.

As for the process of the project period, I'll give a high rating towards our data teamwork together. Though in the beginning I felt like I didn't fully immerse into it since the team is new to me, we had much better interaction as time went. And different from the last few semesters, I also have an even better working environment in which I can express my deep ideas more. Except for more experience of online working, I think another important reason is also the whole team's high motivation, acceptability and seriousness towards our project. Therefore, on this point, I think every group member has contributed to the group as much as possible. There's something else I have to mention here is that we implement our scrum and scrum meetings very well not only in my 4-person data group but also in the 12-person whole group. When we're executing our meetings, we make it clear that what problems we countered, what needs to be finished before a certain date and how we can do better based on the past working to get accessible result after each separate working period. It's also every member's hard work and fully communication that makes the result more understandable with better performance.

It's not easy to imagine how things could smoothly work through a large group with so many members, but during the process everyone's positivity turns it to be reliable for me. I would say I enjoy such a working environment that every single day during the project period seems to be new and challenging, and during every period we work together trying to solve different problems that continuously occur as the project progresses. During elaboration part we try to make use cases clearer. When designing data warehouse, we had plenty of attempts and finally with more and more ideas come up as well as supervisor's helpful suggestions we made the mature version. During data handling (mostly ETL) we found potential problems in the database and data warehouse modeling and went back to resolve them, which to some extent also thanks to scrum’s non-linearity.

Overall, I really enjoy working in this team with satisfying environments. From our scrum master, project owner to every member in the devolving team, everyone is willing to answer questions and offer enough help, which also makes the process go well.

Levente Nagy

This was our 3rd semester affected by the covid19 pandemic, which influenced our work moral in a positive way in the sense that all of us were adjusted to this kind of environment. Since last semester, some students were merged into our class, thus I had groupmates whom I have not had any kind of interactions before in person. And it is a bit weird that I worked with people whom I only know by their voice.

Other than it being weird, this semester we finally managed to do our project with the proper utilization of Agile methodology and Scrum framework, which was a relief as everyone was actively contributing. This resulted in an end product which covers most of what we have planned in the beginning to produce. The contribution of everyone on the team is visible since I did not need to stay up late on any of the project days to finish somebody else's task.

I was sceptical in the beginning as I have never worked in a team the magnitude of 10 people, but the used tech lead agile role which we utilized to ensure coherent workflow, performed perfectly in my opinion.

I enjoyed the dynamics and interactions between team members as I was contributing as both a tech lead and product owner on the data team. Because the data module connects both the client and hardware, I had to collaborate with mainly the other tech leads (other than my teammates), from the other two teams, ensuring that we have a coherent system. This collaboration happened surprisingly smoothly, after we (the tech leads) got to an understanding of the general system and the friction points of the system. We were able to mediate the findings to our respective teams, thus we were able to work as team of 10 people.

Personally, I worked the most on the Web API Server application, and the design of the general System Architecture, but I also contributed to the enterprise data warehousing component of our system. From time to time, I also helped the android and IoT team as well. To mention some examples, I made small adjustments to the gateway application to ensure the correct working behavior of the system, I also helped the android team utilizing JWT and HTTP Authorization, etc.

Afterall, I am proud of the Web API Server, as I really poured every bit of knowledge into it, and I ensured quality work with external tools, such as SonarQube (detailed in the Implementation section of the Project Report). Afterall I am more than confident that I can defend every part of the Design, as well as the Implementation of the Server Application.

My confidence is also based on the fact that I spent days of extra work (which is not even included in the Project Report, as it was not a mandatory requirement), deploying the Server and building a complete infrastructure for it, including reverse proxies which provide HTTPS certs, SHH tunnels, auto health checks, DNS providers, and many more, thus I know the system better than my own backyard.

6.2 IoT team

Personal reflection - Daria

This semester was completely different from all the other projects so far and therefore brought forward a couple of new challenges. Although suddenly working together with so many people was a bit scary in the beginning, with good cooperation, organization, and especially a lot of communication, it proved to be really pleasant to do.

I believe working in specialization gave us a lot more time to work on the specific parts of the project and therefore at the end we had a really good and polished project to hand in which I am happy about.

The group formation was done before the start of the project period and it was overall quite a simple process, we were all quick to decide on specialization teams and start the ideation process. The mushroom farm system idea came from an overall agreement of using the sensors to aid with farms/plants, and I think the mushroom farm was an especially interesting idea to work with because it added some special requirements for the system and pushed us to think even more about our design.

In order to handle the communication, we used multiple organizing and social platforms, most notably being Discord, Microsoft Teams and Azure DevOps. For me it was the first time using Azure and it worked perfectly for organizing task by sprints and checking the overall progress, so I would definitely like using it again.

Working in the IoT team, most of what we did for this semester was new and obviously a little challenging, but having physical components and seeing the code interact with the real world was extremely rewarding and I believe I have learned a lot about embedded and real time programming, and generally about the C programming language.

Overall, I think we managed to set realistic goals and organize ourselves really well to achieve them. I am proud of the end result and the general collaboration of the team, both in our IoT group and overall, and there is nothing I would have changed or done differently. I have learned a lot and it was a great practice for working in a real-world scenario and specialising on certain work areas.

Personal reflection - Natali Munk-Jakobsen

This semester project was more challenging for me compared to the other semesters because we had to work in a bigger group and I was not familiar with most of the group members. At the beginning of the semester, I was afraid of having conflicts in the group since we were not very experienced in working in big groups. However, it all turned out better than I expected. All group members were open to communicate and solve problems, therefore we worked in a pleasant environment, knowing that all members are reasonable, responsible and complete their tasks on time.

In the beginning I was a little bit worried about working in the IoT part, because C programming was completely new for me, but now I'm really glad that I challenged myself and became a part of the IoT team, because I had the opportunity to learn more about embedded systems and hardware components in the project.

As IoT team members we already knew each other and our working styles because we worked together in former semesters. Therefore, from the very beginning we managed to cooperate perfectly with each other and we did not have any problem in terms of communicating and sharing tasks.

Our scrum meetings were more professional and well-organized this semester. I think our scrum master organized everything very well and managed the process successfully. Although we did not have the chance to have physical meetings and met only on online platforms, those meetings were efficient and beneficial. I believe everybody contributed and shared their ideas freely and we managed to communicate and cooperate without any problem.

In conclusion, I enjoyed working on this project with my teammates. I am glad that we managed to fulfil project requirements and had a satisfying product in the end. I learned a lot of new technical knowledge about embedded systems and C programming and had the experience of working with a group of 10 people. I am sure all these valuable experiences will be very useful for me in the future.

Personal reflection - Mihai

This semester was a lot better than I expected. My first thought when I heard that we need groups of 9-12 persons was that it would be difficult to cooperate with each other. It was actually the opposite. Part of what made this work was constant communication and good coordination.

For the first half of the semester, we all worked together on the same thing. After brainstorming ideas for the project, we chose the one to present by voting. The workload was then divided between members, so each had something to do.

Around the middle of the semester, we were divided into 3 teams that focus on a specialization. We decided our roles when we formed the team, so we did not encounter any problems here. I ended up working with my group members from the previous semester. Knowing each other's strengths and weaknesses, we did not encounter many problems.

We worked on the IoT section of the project. Although it started off a bit frustrating, after we managed to make it work, it was quite satisfying seeing the hardware work. A big downside to it was the long waiting time needed for testing. In order to properly test if the code was working properly, we had to leave it running for a couple of hours.

The SCRUM meetings were really well done. From the big ones at the beginning of sprints to the short daily ones. The scrum master had everything prepared, asked important questions and made sure we were on the right track.

In conclusion, I am happy with how things ended up. We managed to set realistic requirements and complete all of them. I experienced working in a bigger group than my previous project, which I believe will be valuable during my internship next semester.

6.3 Android team

Personal Reflection - Bogdan

This semester was a challenging one overall, from all the new technologies we had to use and being part of a much bigger semester group. But because of these reasons I also had the opportunity to learn a lot more and take responsibility in working and time management.

This semester the project idea was more restricted than other semesters, having to develop a system that uses IoT sensors and displays the readings on an Android app makes it so the implementation is almost the same across teams but we can choose the context. We chose to create a mushroom cultivation monitoring system because we had some great ideas on how we want the application to look like and we could make full use of the sensors.

We created our group by combining different groups from the previous semester and I think it went great as everyone got to be in the specialization they wanted. The biggest difference from previous semesters was the size of the group. Having 12 people in one team proved to be an organizational challenge in the beginning which we solved by communicating more, having the tech lead role in each specialization group and making full use of organizational tools. For example, for me it was the first time I have worked on a project while organizing the sprints in Azure DevOps and it proved to be such a powerful tool that allowed everyone to know what the others were doing. I enjoyed very much working in this group as everyone was very friendly and helpful and we could trust each other to get the work done while meeting deadlines.

Overall, starting the work on the project was a bit slow at first, especially when combining the work from different specialization groups, because we did not have the most efficient cooperation there were a lot of small fixes and tweaks, we had to make when combining the work from everyone. However, we quickly adapted and solved all these issues and, in the end, I believe we created a system we can all be proud of.

Personal reflections – Kristóf Lénárd

In my opinion, while it certainly wasn't easy, the process of the project was still well-executed.

I'm fairly sure that others will talk about some of the more fact-based things. Things such as the group contract, its impact on our process, our methodology, the general process of the execution of this project. I want to do something different, give a perspective on the management of this project as Scrum master.

I have seen Scrum in action. I have worked with the Hungarian firm Nemesys Games, who used Scrum. Who better to serve as Scrum master than a person who has seen it in action, who has seen it work in a real-life context? So, I volunteered. The group accepted. It was all said and done there, and I became the Scrum master. I knew it would not be easy. It was the biggest group I had worked with, both here at VIA and before. The project, so varied in many things, was perhaps the most complex. But we persevered. We all did, and here is our result. There were many obstacles in the way, and what we have is far from perfect. This is something that is not necessarily bad. The blood, sweat and tears that this team – that any team – would pour into a project is a necessary part of any project that is successful. Sometimes, it seemed meaningless, that no matter how much we work, it would not matter in the end. That we would fail. Looking back at that, I wonder how we got so far. We have accomplished things that sometimes seemed impossible. We have worked separately yet together so that our project may become whole. Did it work? Did we manage to make it work? Looking at it, I see all the mistakes we made. All the small and big problems we had, all the hours we spent on hammering all this into shape. What we have is not perfect – there are many things not done, perhaps even more small, yet-unseen mistakes. The key is, it never had to be perfect. I knew that from start and yet on some level, I do not want to accept it. That all this work, the combined work of such a great team, perhaps the best I even worked with, and still, it's not enough. That there are still things left unfinished. Accepting this was a hard thing to do. It always was so for me. But that is part of our journey. Part of becoming an engineer is learning to accept such things – and then disregarding that acceptance, and trying, desperately trying to make sure that there is nothing left to do. It never is so, and it probably never will be.

Looking at this team, I see a group that is, in my opinion, united in purpose. When I look at them, I still see the differences between them. I see points of friction; I see things waiting to erupt one day. This could, possibly be enough to destroy a group like ours. There is one thing that gives me hope against this – and that is the group itself. We had been through many troubles during these last few months. We have learnt what makes us work better together – and what we need to avoid. When I look at this group, I couldn't be gladder – and for me, as Scrum master, I feel that this is, perhaps, the greatest thing I could achieve. This group has surpassed my expectations. All members are great workers, and I feel that they will all become great software engineers at some point. They sometimes needed motivation or arbitration. They needed guidance in both technical issues and process. Once they learned what and how to do certain things, there was nothing that could stop them. That I had managed to make it work, that we were able to collaborate so effectively – this is such a feeling that I cannot and as matter of fact, do not want to describe. I always loved the feeling of accomplishment. Whenever I solved a problem in software, whether a bug or a logical problem, I felt this feeling. This is why I loved it so much. That is why I wanted to become a software engineer. That is why I came here. I have not felt this before just in my capacity as Scrum master. Now, I feel this very thing – and I know in my heart that while we had, and if we work together again, will always have some difficulties, we will be able to overcome them. That whatever stands in our way – we can beat it. We can and we did in this project. For me, that alone would be enough to call this project a success. There are many other things why I could say that we succeeded – but this is why I will remember our success. That no matter what life threw at us – we persevered, and we succeeded. And that is all that matters in the end.

7 Supervision

7.1 Android team

The Android team was completely satisfied with the supervision. We, personally, have not needed to work with the supervisors a lot, except for a few questions about implementing certain things and clarifications to other small matters, but all of these questions were answered in a timely and helpful manner.

The other times we met with the supervisors, during feedback meetings, we received helpful information, tips and other guidance that helped us create, even though some of the suggested things were merely not scoped for that release, instead of not being thought of. However, one thing I must highlight is the amount of preplanned feedback we received, which was much greater than the previous two semesters. We feel that this is a welcome improvement.

Communication with the supervisors was easy – we could just send an email, or ask them in a Zoom chat, and while this was not as fast and quick as the in-person method (which is, unfortunately, still unusable for some of us), it was still fast enough for basically all things that we could need.

All in all, while we have not had much contact with our supervisors, when we had, their help was greatly appreciated.

7.2 Data team

The supervisor for the data team was Astrid Hanghøj. The team utilized the supervisor frequently to clarify misunderstandings about data warehousing concepts. Due to the initial mistakes that were made in the design of the star schema, the group relied heavily on feedback from the supervisor in during the entire project period. we discussed included Junk and Bridge dimensions in the staging, business keys in the stage fact cultivation, and for the data warehouse process. We were pleased with the interactions between the supervisor and the data group, the feedback was crucial to the success of the project.

8 Conclusions

At the conclusion of the project, we discussed whether we felt that the project was successful, both from a project and a process standpoint. We felt that both the project and process were successful. We have worked together efficiently and purposefully, and have been able to achieve our goals. While our original plans for working were not always flawless, they were adjusted, including by sprint retrospective meetings (see appendix A for further details). Therefore, we have no further major recommendations regarding the process.

Appendices

- Appendix A – Scrum log book
- Appendix B – Azure DevOps log
- Appendix C – Ideation document