Case description: Develop a web application as Minimum Viable Product

Deadline: 19 December 2025, 23:59 pm

1. Introduction

For this group assignment, you are required to develop a web application as Minimum Viable Product (MVP)— a version of the software with just enough features to be usable by early users who can then provide feedback for future product development. You do not need to develop a fully operational application. Instead, **focus on developing the most unique functionalities** of your application to ensure it can be tested effectively.

This year, the MVP must be developed in collaboration with an external partner, either a company (as a spinoff project) or an entrepreneur (as a startup project). Development should follow the Agile methodology, meaning you work iteratively in short development cycles (“sprints”). You are required to complete at least three sprints, though three to four sprints are strongly recommended. **After each sprint** (except the last one), **you will present your progress** (models, user stories, User Interface, web-application) **to gather and record feedback directly from your partner.**

Please note that it is **not the goal** of this project to develop a complete and launch-ready **SaaS, PaaS, or large-scale platform**. Instead, focus on building a **Minimum Viable Product (MVP)** that is:

- **functional**: covers the core features needed to validate the concept,
- **scalable**: designed so that it could later grow and benefit from network and scaling effects, and
- **validated**: tested with real feedback from your external partner.

You may either indicate your interest in one or more concepts proposed by an external partner of the Ghent University, or you may develop a concept based on a collaboration your group establish yourselves. Only one group member should complete the concept selection form below, indicating either your preferred concept(s) from the list, and/or the concept you intend to develop and the partner you will collaborate with:
https://docs.google.com/forms/d/e/1FAIpQLSfGZ3Vsa8hmVoLVgrq0E0guFnUVwB-WWVkRjMLcjOS8E3MfQA/viewform?usp=header

It is important that all groups sign a Technology/IP Transfer and Non-Compete Agreement with the collaborating entity. This ensures clarity about ownership of the developed MVP and prevents conflicts about future use of the concept. A suggested template for this agreement can be found here:
https://docs.google.com/document/d/18JSZaCqIUNcZ5zKtVDlYv0RjghrLo-n-KjUBPSn6VvU/edit?usp=sharing

This MVP needs to (at least) capture business entities (as will be explained in section 2) that need to be managed by your database implementation, the functional requirements (in section 3) which need to be supported by your system, and some algorithmic components (in section 4).

2. Business entities

The following lists is an indication of possible basic business entities that could be managed by your database implementation, together with possible fields.

- User / Account: Stores login info, roles (admin, user, partner), and contact details.
- Profile: Contains user-specific details such as preferences, bio, picture, and demographics.
- Product / Service / Listing: Represents the core item offered to users (e.g., trip, course, rental, post).
- Category / Tag: Organizes products or services into groups for easier browsing and filtering.
- Booking / Order / Reservation: Records a user's commitment to a service or product.
- Payment / Invoice: Tracks payment status, amounts, and billing details.
- Feedback / Review / Rating: Stores user evaluations of services, products, or other users.
- Message / Chat / Comment: Captures communication between users or with providers.
- Notification: Delivers system updates such as confirmations, reminders, or alerts.
- Organization / Partner / Provider: Represents external collaborators, companies, or service providers.
- Activity / Event / Task: Domain-specific entity for actions the user engages in (e.g., trips, lessons, deliveries).

3. Functional requirements

Your implementation needs to support the events and actions performed by users. Here are some general examples:

- Sign Up / Register: User creates an account. Setting a password is not required, a text box to enter the username is sufficient.
- Log In / Log Out: User starts or ends a session. Validating a password is not required, a text box to enter the username is sufficient
- Update Profile: User edits personal info or preferences.
- Browse / Search: User looks through listings, products, or services.
- View Item / Listing: User opens the details of a product, service, or activity.
- Create / Post / Upload: User adds a new item, listing, or piece of content.
- Edit / Delete: User modifies or removes their own content.
- Add to Cart / Wishlist / Favorites: User saves an item for later.
- Book / Order / Reserve: User commits to a product or service.
- Pay / Confirm Payment: User completes a transaction.
- Cancel / Modify Booking: User changes or cancels a prior action.
- Rate / Review / Comment: User gives feedback on a product, service, or another user.
- Send / Receive Message: User communicates with another user or provider.
- Receive Notification: User is informed of a system event (confirmation, reminder, alert).
- Log Activity / Attendance: User participates in or completes an activity/event.

4. Algorithmic support

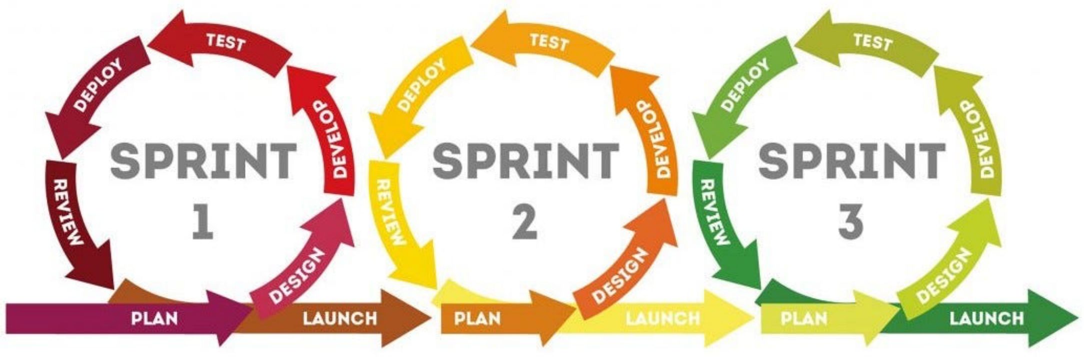
In addition to the basic functional requirements, we would like you to implement a more sophisticated algorithm that creates added value for users. How you choose to set this up is up to you, and innovative ideas will be rewarded. Nevertheless, to give you some direction we provide a couple of ideas:

- Priority scoring (what to show first): Learn from past user interactions (clicks, bookings) to estimate which items are most likely to be chosen, and show those at the top.
- Smart categorization: Automatically assign items or users into meaningful categories, either based on text descriptions or other attributes.
- Automatic grouping: If no categories exist, group similar items or feedback together into clusters so patterns become visible.
- Content-based recommendations: Suggest items that are similar to something the user has already viewed or chosen, based on item descriptions or features.
- Light personalization: Identify which types of items each user segment prefers (e.g., hiking, cultural tours) and recommend more of those.
- Anomaly detection: Flag suspicious or unusual activities (such as strange bookings or fake reviews) by finding data points that look very different from the rest.
- Dynamic pricing or quote helper: Suggest a reasonable price for an item or service by looking at key features (e.g., duration, season, category) while keeping it within sensible limits.
- Search ranking boost: Improve search results by combining multiple signals—such as relevance to the search terms, popularity, and recency—into one score.
- Feedback topic extractor: Analyze user reviews or feedback to detect the main themes and present them as summarized insights for the company or entrepreneur.

Your value-adding algorithm must be designed and implemented primarily by your team. You may use standard open-source libraries (e.g., scikit-learn, NumPy, pandas) for common utilities, but you may not depend on an external "black-box" service (e.g., hosted AI/ML or recommendation APIs) that performs the core ranking, prediction, classification, generation, or matching. External APIs may be used only for non-core tasks (e.g., maps, payments, email) and for data retrieval. If any external component is used, clearly document its role and ensure the algorithm's core logic (feature engineering, model training/inference, scoring, ranking) is implemented in your repository.

5. Agile development and feedback

Because you are working with a real partner, it is essential to work in an agile way. This means you develop your MVP step by step based on user stories, in at least 3 short sprints of 1–2 weeks each. At the end of each sprint, you should present your progress (user stories, UI and/or webapplication) and gather feedback (the review step in the image below) from both your partner (mandatory to record at least 2 feedback sessions with your partner in audo or video) and potential users (for example, friends or family). This way, you can quickly adapt your design and focus on what brings the most value (must-haves instead of 'nice-to-haves').



To guide your work, keep the following in mind:

- In the first sprint start with a prototype: Before building the web application in Flask, create a simple user interface prototype in a tool like Figma, Adobe XD or MarvelApp, or use the AI pototype developer Lovable. Test it with a few potential users (e.g., family and friends) to check if it is clear and intuitive. Only then move on to implementation.
- Keep it simple: Do not waste time on general security features such as complex logins or password management. Focus on the core functionalities that are unique to your MVP.
- Prioritize usability: Make sure the interface is clean and easy to use. A simple, working MVP is more valuable than a complex system nobody understands.
- To gether feedback on the web application, you can launch the prototype via PythonAnywhere.
- Think ahead: Even if you don't build it now, consider how your application might scale in the future when more users, interactions or transactions.

6. Timeline and contact

Below is a suggested timeline for the project, including deadlines, classes and feedback seminars.

Week	Date (time)	Content	Location
2	3th Oct (12:00)	Deadline: Indicate external partner preference via form[1]	
3		Sign IP agreement + create ERD model	
4		Map ERD model to DDL	
5	21 Oct (10:00 – 13:00)	Create database + Design User Interface (UI) + Feedback	1.1 S2 De Sterre
6	28 Oct (11:30 – 12:45)	Introduction to Flask & Agile	Aud Picard
7 - 9		MVP development	
10	25 Nov (10:00 – 13:00)	Feedback project	1.1 S2 De Sterre
11	2 Dec (11:30 – 12:45)	Flask best practices seminar	Aud Picard
12	9 Dec (10:00 – 13:00)	Feedback project	1.1 S2 De Sterre

[1] https://docs.google.com/forms/d/e/1FAIpQLSfGZ3Vsa8hmVoLVgrq0E0guFnUVwB-WWVkRjMLcjOS8E3MfQA/viewform?usp=header

13	19 Dec (23:59)	Deadline: Project + Peer review	

Contact:

- All questions related to software installation and use (Supabase, Figma, MarvelApp, draw.io, Flask), please use the discussion page of A&D on Ufora.
- Each group can book one meeting of 20 minutes for additional feedback on the ERD model and database design, from week 3 – week 9 (untill 20/11) via following link: Book time with Thomas Derave
- For other questions or issues, you can mail to thomas.derave@ugent.be

7. Deliverables

Each team member must register in their group's repository through GitHub Classroom (Links will be announced in class and on Ufora in week 6). The first team member to click on the link will create a new team (e.g., group5) and a corresponding repository. The other team members can then join this team and will be added as collaborators on the repository. Make sure that each team member is subscribed.

In the repository, please ensure that you include the following:

- Flask app
- Models: ERD & DDL
- Screenshots/images of UI (prototype)
- Final version of the User Stories
- Database dumb (as backup)
- Readme file with:
  - Extra information how to install/use the app (if required)
  - Link to UI prototype (if used)
  - Link to Kanban board (if used)
  - Link to audio/video recording of feedback sessions with partner
  - Other links/info

8. Individual assignment

Already passed Algorithms and Data Structures, but not Database Systems: Develop a database based on an ERD model for a digital marketplace of your choice. The database can be basic, and there is no need to consider messages between users, payment systems, user login, or other more complex data requirements.

Already passed Database Systems, but not Algorithms and Data Structures: You must create a small digital marketplace application with Flask that connects to an existing MySQL database.

Working student with an individual assignment for both Algorithms and Data Structures as well as Database Systems: Same assignment as described above, but with a focus on second-hand sales of a specific type of goods (e.g., CDs). No need for messages between users, payment systems, user login, or other complex functionalities. Only a small algorithm is required.