Group project description, Software Design, fall 2024

In this project, each group will **design** and **implement** a software that retrieves, combines, and visualizes data from two individual (separate) services. You can freely choose whichever services you like, though it is recommended to pick an open one. A strict requirement is that the course personnel must be able to use your application as-is, without the need to, for example, create credentials for third-party services.

Example APIs:
- DigiTraffic, https://www.digitraffic.fi/en/
- The Finnish Meteorological Institute's open data, https://en.ilmatieteenlaitos.fi/open-data
- Statistics Finland, https://www.stat.fi/org/avoindata/index_en.html
- Avoindata.fi, https://www.avoindata.fi/en

All the above are from Finland, but using foreign ones are also accepted.

But what if you have made like a million of these applications on previous courses, and simply cannot come up with a good idea for the application?

In this project you do not need to come up with your own idea. In fact, **you are required not to**. Instead, you should provide the requirements of this assignment to an artificial intelligence (AI), and make it give you an idea for your work. If the idea cannot be implemented in practice within the time and resources available for the course work, you do not have to follow the AI's suggestion in its entirely. Using common sense is recommended.

You can use any AI tool to generate your idea (just remember to document what you used!), but a popular option is to use the free version of OpenAI's ChatGPT (https://openai.com/blog/chatgpt).

The requirements

Students are allowed and expected to use their (or AI's!) own creativity in specifying the functionalities and look of the application. The goal is that each group will discuss internally about what will be implemented and present their choices in the first meeting with the group's assigned teaching assistant (TA).

How AI (tools) are used in the design and implementation of the project are up to the group members to decide**. For design, the minimum requirement is the formalization of the application idea** (as described in previous chapter). **For implementation, the goal is that the group get some hands-on experience on how AI could be used.** An easy option is to try out, for example, Copilot (https://github.com/features/copilot), which has a free trial, and a free license for students (https://github.com/education/students), and works in, for example, VSCode (https://docs.github.com/en/copilot/managing-copilot/configure-personal-settings/installing-the-github-copilot-extension-in-your-environment).

No excessive AI use is required, but there is also no limitations on how much AI can be used for the completion of the project. You can be as innovative as you like. The use of AI tools is not graded – i.e., the tools you choose and how much you use them will not affect the points you get from the assignment, **but the documentation of your project must state which AI tools you used and how you used them**. For your own learning experience, a more in-depth look at the tools is *strongly* recommended, though for passing the course, it is not required.

Overall, the basic requirements are made of practical requirements and functional requirements.

Practical requirements:

- Sensible use of design patterns and other principles taught on the course.
- Documentation
- Readability of code – especially important if you use AI to generate your code**. You should be able to explain what your code does!**
- Use of version control
- Interface design and implementation
- Unit testing of crucial (core) components. A limited set of tests is accepted, the purpose is that you show that you have had experience in trying out unit testing with Java.
- The use of AI (tools) in design and implementation of the project.

Functional requirements:

1) The application must retrieve data from two separate third-party APIs (services).
2) The data from the services must be combined in a meaningful way and shown to the user:
   - In this case, "meaningful" can be freely interpreted. Innovative and even bizarre solutions can be accepted as long as they are functionally sound, and from a software design perspective valid.
   - The data visualization should include graphs, plots, maps, or other self-developed advanced visualization. Tables *can* be used, but they cannot be the only way of showing information. The purpose is to increase the complexity of the design of your application, the visualizations do not have to be especially impressive (i.e., the visual quality of the visualizations is not graded).
   - The user must be able to adjust the parameters of the visualization (e.g. date, time, location/coordinates, etc. depending on the data types used).
   - The user must be able to select what data is shown (e.g. customize results). In practice, this also means that the application must be able to show multiple types of data. Showing simple data (e.g. only a weather forecast) will not be accepted.
3) The user can save preferences for producing visualizations (e.g., date and location), and fetching those preferences will produce a visualization using the most recent data with the given parameters.
4) The design must be such that further data sources (e.g. another third-party API), or additional data from existing sources could be easily added.

The core components of the application must be written in Java. Using other programming languages for creating visualizations (or user interface) is allowed.

Web applications *might* be accepted, as long as the core functionality (e.g., the back-end) of the application is implemented in Java. But remember, that the application must be easily compiled, run and tested *locally* (on the course personnels' computers). Solutions that require online hosting are not allowed.

You can freely choose any framework you like, but your work should show your own contribution. Simply using a single framework with "a few lines of own code" for implementing the whole application will not be accepted.

Grading

1. Prototype: 0-1p

Everyone who makes a submission *on time* and *according to instructions* will get 1 point.

2. Mid-term submission: 0-2p
   Evaluation criteria: submitted on time, self-evaluation, some actual code exists (work done on actual product based on prototype)
3. Design (document): 0-3p
   1. High level description of the design 0-1
   2. Describing components and their responsibilities: 0-1
   3. Interfaces (internal): 0-0.5
   4. Components' internal structure and functions: 0-0.5
   5. **Your documentation should describe what design patterns and architectures you have used, and why (goals and reasoning).**
4. Final submission (software): 0-7p
   1. Functionality: 0 - 2
   2. Interfaces: 1
   3. Use of design patterns and other principles taught on course: 2
   4. Instructions for compiling and running the application. Details of the environment, including software and library versions used in testing and implementation. The instructions should be clear enough for TAs to test your application *easily*. 0.5
   5. Quality of documentation, Readability of code, Other possible factors: 1
   6. Unit tests for crucial components of the application 0.5
   7. The documentation *clearly* states how AI tools were used in the design and implementation of the application: fail/pass the project, no points awarded.
   8. Work submitted on time**: late submissions will be rejected**. If you have an *extremely very good reason* for late submission, contact you TA or course personnel. Last minute requests for extra time will almost certainly result in a failure! You have been warned.

5. Peer review: 0-2p (for reviewers)

Note that there is no requirement for excessive testing. Simple unit tests made for the core components of your application that show that you tried out unit testing are enough. You are not required to, for example, write extensive test suites for your application, but it is up to you to guarantee the working condition of your application. **The course personnel will not fix any bugs or issues found in your application. An application that does not compile, crashes, or does not function as specified in your documentation (specifications) faces a risk of severe point reductions or rejection**. Documenting your testing approach and any known issues might save points in case something goes wrong with your work.

And finally, always remember that documentation is not made for you, it is made for others. Do not expect that everyone knows everything you do about your software!

Deadlines and timeline:

Submission deadline for Prototype: 29.9. Prototype evaluation. Two items to be submitted:

1) **Software prototype.** The purpose is to get the group to start thinking about implementation and to illustrate how the group has understood the requirements (helps the discussions with the TA). There are several acceptable options for delivery:

Option 1: A well-thought sketch of the UI, showcasing all the requirements, including transitions between selections (using, e.g., Figma or some other method of achieving a high-quality sketch)

Option 2: A rough implementation of the UI, with something that ”functions and moves”. Actual code, but made with minimal effort. This can be something that is thrown away, and the actual implementation of application can be started from scratch!

Option 3: Something between Options 1 & 2: quickly made implementation of UI elements, not necessarily any functionality, but clearly showing how requirements would be implemented.

2) **A design document**, describing the structure and most important components and interfaces of the software. No official template will be given, it is advised to include a figure or diagram (or couple) of some kind.

The documentation should also include (tentative) description of what approaches (design patterns, architectures, etc.) you are planning to use. You can update your choices later, if they turn out to be less than optimal.

Idea of what APIs (and data) you are planning to use. It is recommended to try out the APIs to see what they can and cannot do, but no actual implementation is required in the prototype phase.

You are not required to include documentation of your AI tool usage until final submission, but it is *strongly recommended* to keep internal documentation of what you did so that you do not forget it!

30.9. – 4.10. Group meetings with TAs, a demo of the prototype and discussion on group's understanding and plans for implementing requirements, discussion on documentation.

Submission deadline for Mid-term: 27.10. Mid-term evaluation. Two items to be submitted:

1. Software. At this point some of the functionalities should be properly implemented.
2. An updated design documentation. The applicability of the original design should have been assessed based on implemented functions. The design documentation must include a self-evaluation of the proposed solution and suggestions of required changes (how to refactor the software).

28.10. – 1.11. Group meetings with TAs. Discussing and giving a demo of the current version of the software. Making plans for implementing the remaining requirements. This is a point to particularly discuss: what should be changed in the structure of the software.

Submission deadline for Final submission: **1.12.** Final submission. Two items to be submitted:

1. Software. Everything has been implemented.
2. An updated, final version of the design document. One specific issue to document is how the software has been refactored during the implementation process, and how refactoring has succeeded. Another one, is what AI tools you used, and how you used them.

Final submission includes submitting these both to your TA, and to the group who is reviewing your work as part of peer review! Final submission is not considered done if you have not given your work to your peer group.

2.12. – 6.12. Meetings with the TA. Demo and discussion, particularly about what went smoothly and where did the group find challenges. If you have already received your peer-review from another group, you can also discuss these with your TA, but this is not required.

Submission deadline: 2.12. – 6.12. Peer review.

Each group is assigned another group whose code and documentation they will review and comment. Reviewers will get points. **The peer review deadline is midnight before your final TA meeting**, but be considerate of your fellow students and try to finish the review rather sooner than later.

General guidelines for submissions:

All submissions (to a specific deliverable, i.e., proto, midterm or final) will be made to the groups' Git repositories and commits relevant for the deliverable should be marked with tags such as "Proto deliverable, data retrieval", or with some other clear tag message. This will help the TAs' job of checking that all submissions have been made.

You can use the provided GitLab repositories or GitHub, whichever works best for you. If you are planning to use something more exotic, ask your TA if it is OK. It is up to each TA to decide if something else is accepted. Note that only git-based versioning is accepted. **Using private repositories is accepted, but remember to invite your TA.**

Groups are required to write a clear README file, noting the environment required to run the software, and what files need to be executed and for which purpose. The instructions must be so that the TA is able to download all necessary libraries and compile and execute the code on his/her own computer. For example: "As an environment we use a ready installed VSCode with University's VPN… For compiling the application, run mvn package on the root directory."

In the meetings, the group will present the documentation required for the submission, give a demo of their implementation, and discuss their solutions.

Remember to have all documentation and code/commits ready to show in the meeting.

And finally, a brief list of what's allowed, and what's not:

You…

- …want to use an existing application that seems to fit the given specs? - Not allowed. Getting caught means severe penalties.

- …got a new idea based on a work you have done on a previous course, at work, or as a hobby, but it requires further design and implementation to make it functional? - Sounds good, go ahead.

- … have partial code snippets or small modules that could be used to implement required functionalities (e.g. retrieving data from an API XYZ)? - Use them! But remember to document where you got them.

- … want to use an open data source, which only provides data as a downloadable dataset (e.g. .csv), and not by means of an API? - Ask the course personnel, great ideas will not only be accepted, but encouraged!

- … found an AI tool that generates documentation? - Who likes to write documentation anyway? Feel free to use any tools you come up with, just make sure that the produced documentation matches what was required by the assignment. And of course, document what was the tool you used.

- … would rather not use all that JavaFX stuff for making your user interface? – You can use any programming language for your UI (e.g. HTML/JavaScript) as long as the core components are implemented in Java.

- … think that Java's days for making desktop applications is over and you want to try your skills in implementing a back-end for a web application? – As long as the requirements of the assignment are otherwise fulfilled, go ahead! Remember that you also need to implement a user interface.

- … have used a framework, HTML, or some other solution for making an awesome user interface, but it seems to be quite challenging to write unit tests for it? – Writing unit tests only for the Java components is enough. The important thing is to show the course personnel that you have tried out making unit tests. The tests do not have to be complex or cover all possible scenarios.

- … figured a way to use AI to make the entire group assignment all the way from design to implementation and testing? - That's excellent! Just remember to document how you did it. Also, your documentation should *clearly* show that *all members* of your group know how it was done. **It is up to you to proof that everyone participated in the work**!

- … are not sure if you can use your idea to complete the group assignment? - Ask the course personnel. Innovative and even bizarre ideas might be accepted!