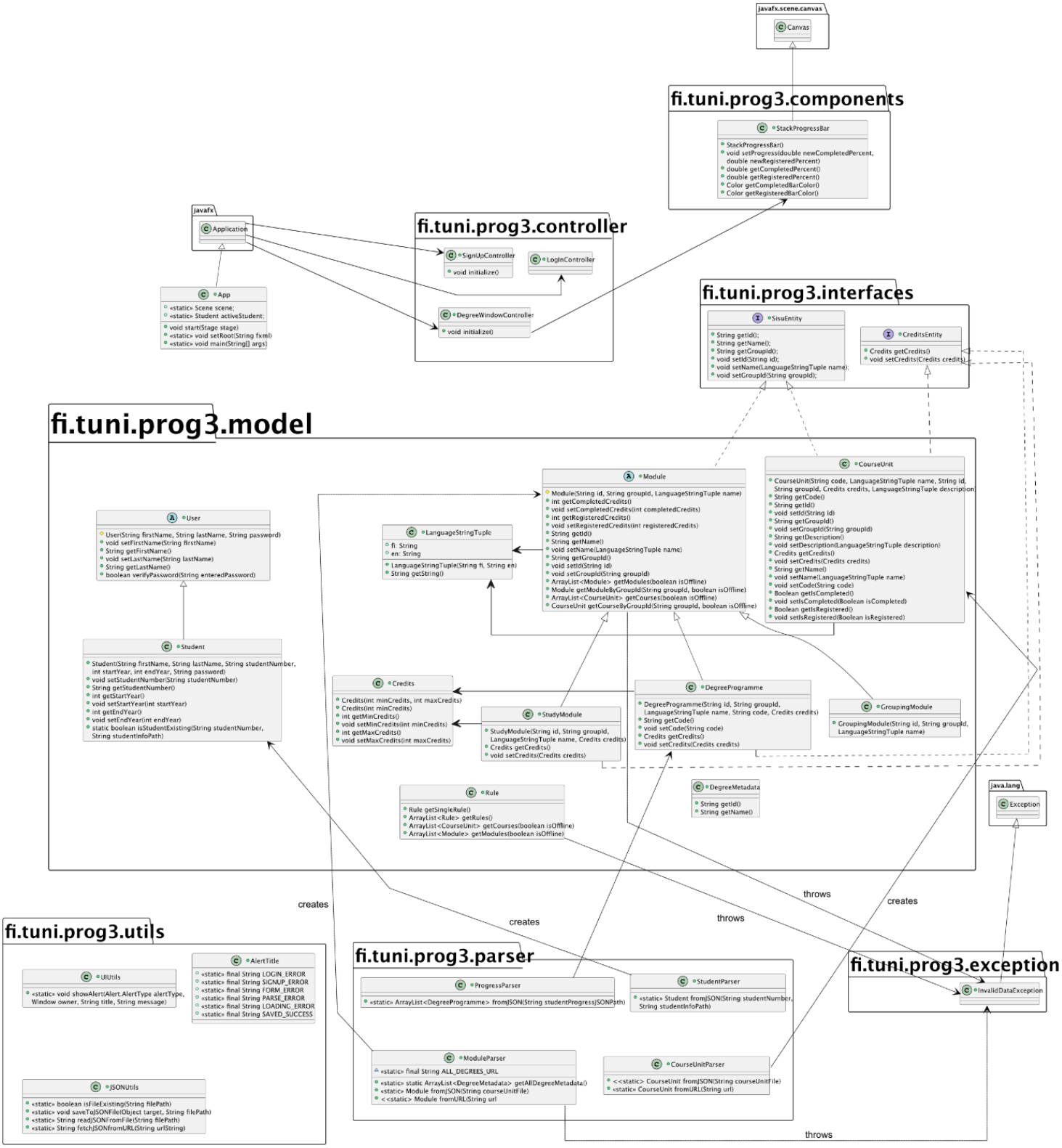
Project documentation: Sisu Unraveled

COMP.CS.140 Programming 3: Interfaces and Techniques

Duc-Anh Pham, anh.pham@tuni.fi
Hung-Anh Pham, hung.pham@tuni.fi
Quoc Nguyen, quoc.nguyen@tuni.fi

Class diagram



Key classes responsibilities

Components

- StackProgressBar (extends from Canvas): functionalities of the ProgressBar in the UI

Controllers

The controllers are the bridge between the user interface and the data models. Controllers take care of building UI elements and manipulating them in synchronization with the underlying data objects.

- LoginController & SignUpController: handle functionalities of the first window, where the student (user) can register for an account or sign in to the application.
- DegreeWindowController: functionalities of the second window, where the student can choose the degree programme and control their study progress, displayed in a progress bar.

Exception

- InvalidDataException (extends Exception): a custom-defined exception that is used in cases where the JSON data being parsed is invalid.

Interfaces

- CreditsEntity: defines credits-relevant methods for a StudyModule, DegreeProgramme or a CourseUnit.
- SisuEntity: defines methods relevant to properties of a general data object from Sisu (e.g. id, groupId, name).

Data models

Represents entities used in the application, storing data from either the Sisu API or modified & saved locally.

- CourseUnit: representing a course object
- Credits: representing a credit object, used in StudyModule, DegreeProgramme or CourseUnit.
- DegreeMetaData: representing metadata of a degree programme, which are fetched from the list of all Sisu's degree programmes.
- Rule: representing the "rule" property of a degree, a module, or a course, defined by the Sisu API.
- Module *(abstract)*: represents a general module.

- DegreeProgramme, GroupingModule, StudyModule (extends from Module): represents a module object that corresponds to a specific module type, each of which has its unique characteristics.
- User *(abstract)*: represents a user of the application that requires authentication.
- Student (extends User): represent a student that uses the application.

Parsers

Classes mainly responsible for parsing JSON data. Their public methods are all static.

- CourseUnitParser: parsing appropriate JSON files (either locally or from the Sisu API) into CourseUnit objects
- ModuleParser: parsing appropriate JSON files (either locally or from the Sisu API) into StudyModule, GroupingModule, or DegreeProgramme objects. It also parses all degrees' metadata from the Sisu API.
- ProgressParser: parses locally saved JSON files to get a student's current progress.
- StudentParser: parses locally saved JSON files to get a student's information, either for authentication or general personal info.

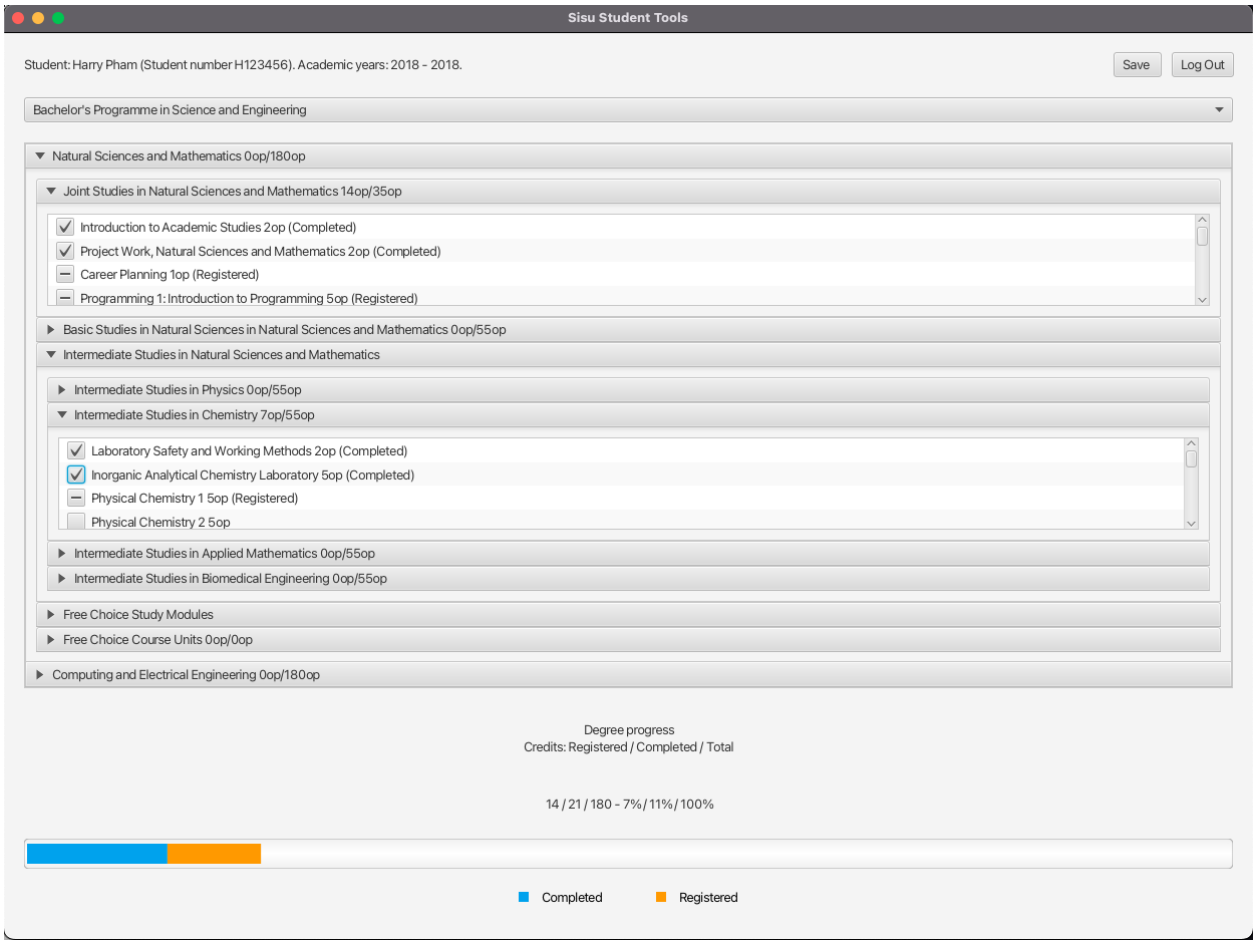
Utils

Helper functions that are used in many places throughout the application. Their public methods are all static.

- JSONUtils: provides methods for reading JSON files, stringifying and saving JSON, and checking if a certain file exists.
- UIUtils: provides simple UI manipulation function. RIght now it only has one method to show an UI alert.
- AlertErrorTitle: defines specific error messages of the application, which are shown to the user in UI alerts.

Project functionality & extra features

The application fulfills all requirements set out in the project specifications. Students can register accounts in the application, each account can “study” multiple degree programmes. At the top of the window, students can select a degree programme to view once at a time. Students can view the degree structures divided into submodules in the middle accordion, register for a course by clicking once on the checkbox, and clicking twice will complete that course. The progress of the student in the selected degree is shown in the bottom progress bar. The study progress can be saved by clicking the “Save” button on the top right corner.



All degree data is fetched from the Sisu API, which requires Internet connection. Students' accounts information are saved locally, as well as the students’ progress.

Course requirements (up to grade 3)

- The program compiles
- The program contains one dialog window (e.g. the initial dialog). This application contains 2 windows, one of them has 2 tabs.

- The program is able to show the selected study program structure in the main window
- A graphical user interface has been fully implemented by the team (the simple readily provided JavaFX project is not used)
- The program gets the degree structures from the Sisu API
- The program works i.e. the degree shown can be changed and the progress situation of the student's degree is shown.
- The program handles errors in file handling. UI alerts are shown to the user.
- Unit tests have been implemented for the program (both "normal" JUnit tests and TestFX).

Extra features (up to grade 5)

- When the user hovers the mouse cursor on a course's checkbox, additional information (course' outcomes) is shown on the UI.
- A student can sign up & switch between different accounts
- A student can securely log in to their accounts with student ID and password
- The student's study progress is shown in respect to the degree structure: not only the whole degree's progress, but the progress is also shown in each submodule.
- The application differentiates between "registered" courses and "completed" courses. The student can click on the checkbox once to register, twice to complete a course. The progress bar also shows both the amount & percentage registered and completed credits.
- Prompts and alerts are used to make sure users know if errors occurred, or if something important happened. For example, an error alert is shown if the program tries to fetch data without the Internet, or when the user saves their study progress.

Other than that, we also believe that we have designed a very intuitive and easy-to-use UI.

Division of work

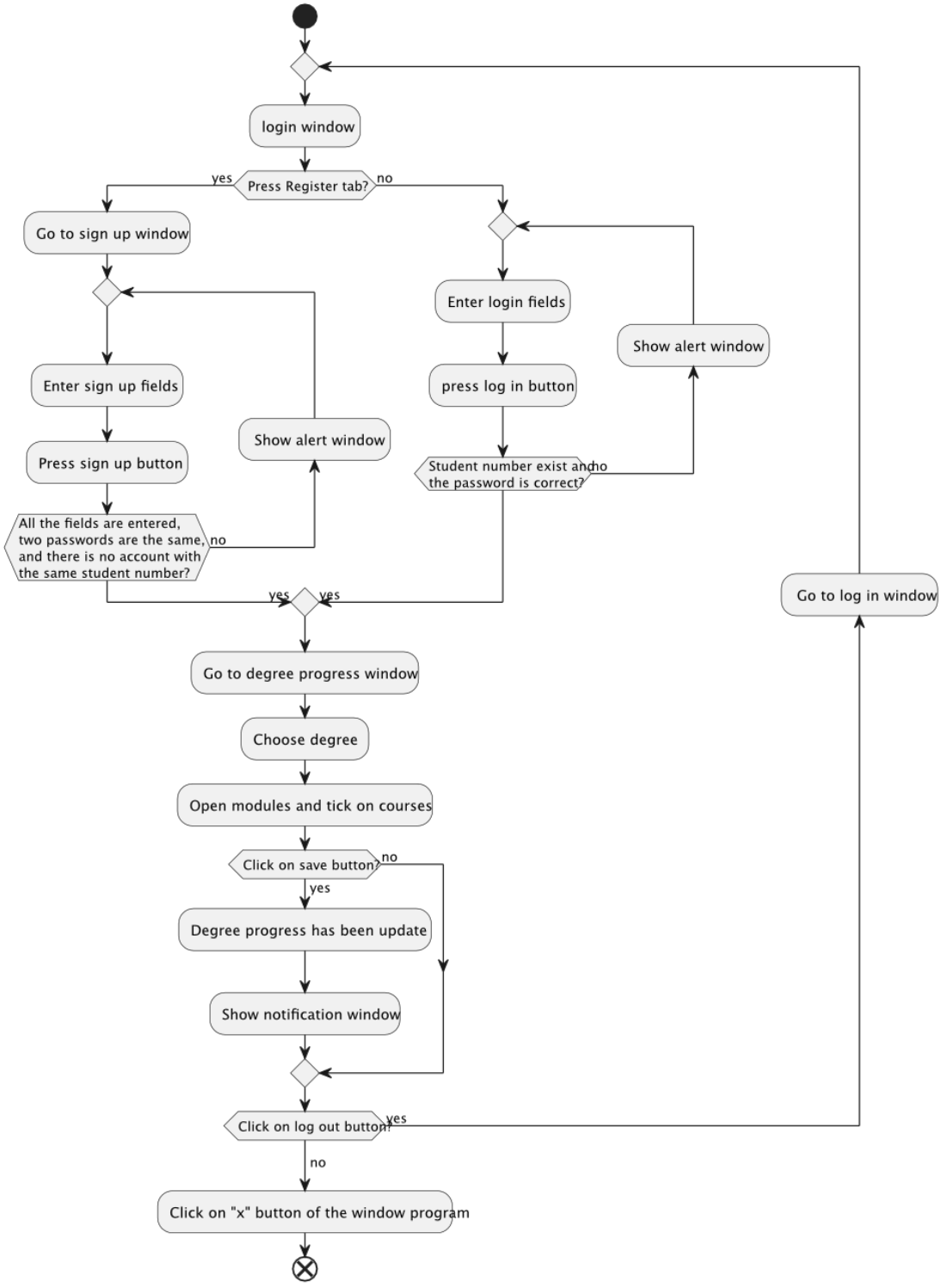
We do not divide concretely who does what from the beginning. Instead, we try to work in a Scum-like process. We have a product backlog on Trello where tasks are defined and assigned to members, and we have weekly meetings where we update each other on the completed, ongoing and to-do tasks. We think that it's important for all members to do different things in the project, therefore we do not pick tasks based solely on an individual's strong points either.

We use branches and merge requests to manage our parallel work. In order for code to be merged to the main branch, at least another member of the team must review the code. This increases teamwork and ensures code quality.

With constant collaboration & communication between team members, we feel the work has been equally divided among us, and each member has contributed greatly (and equally) to the project.

User manual

Below is an interaction diagram which illustrates how user interactions will influence the program behavior. We believe the UI is intuitive enough that a step-by-step instructions manual is unnecessary.



Bugs and missing features

There are currently no bugs that we know of, and we believe we have implemented all required features, with many extra ones. One thing to note when using the UI: as a degree programme and its modules & courses are shown in one accordion (collapsing menus), its content can be difficult to see if you have multiple opened modules. If that happens, try closing some opened modules to make more space for the content.

The project was developed using VSCode. We suggest you use it to grade our submissions as well, just to make sure there are no conflicts between different IDEs.