Peer review

Does the project use a consistent coding style?

Yes, the names of methods and variables are consistent. For example if some class has something to do with mails, the methods and variables have “something”mail most of the times, which makes it clear that they belong to that class.

Is the code reusable?

Some parts of the model allows for good re-usage of existing code. Adding new account types could reuse a lot of code. By simply adding a new enum in the interface Account, the code in the class AccountBuilder would be heavily reusable.

Is it easy to maintain?

There are some good examples of code that is easy to maintain. For example if Gmail changes the way that their emails are sent there only needs to be changes done to a few places in the code, which handles sending and retrieving of emails.

Can we easily add/remove functionality?

Additional functionality could be added easily, due to the many interfaces used throughout the project. Removing functionality is a more difficult question. The difficulty depends heavily on what you want to remove. One well designed area is how the controller package is implemented with a hierarchy of controllers, with “bigger” ones on top. For example, the settings menu of the application has its own controller hierarchy. This makes adding or removing features easier, as they divided into clear sections with their own responsibilities.

Are design patterns used?

There are some design patterns used in the program. The model is using the BuilderPattern, which is used to prevent the need for using constructors with many parameters; this makes the creation of objects easier to understand. There is also an implementation of the MVC architectural pattern where the model does not have any dependency on things outside of the model.

Is the code documented?

Many functions and classes are not documented which makes reviewing and following the code harder. For example, DatabaseService has several methods, while only a few of them are documented.

Are proper names used?

We believe that the names of most classes, functions and variables have been properly assigned as they describe the usage quite well. The method createAccountButtonAction(), in the class AddAccountController, has good naming for both the method itself, and for its local variables.

Is the design modular? Are there any unnecessary dependencies?

It is rather obvious that the group have not ran any type of code inspector. The project contains about 26 unused imports which creates unnecessary dependencies.

Does the code use proper abstractions?

No abstract classes has been used in the project. If these are needed, we are not entirely sure. Interfaces are widely used throughout the whole project. Although, many methods in these interfaces are simply mentioned in the implementing classes, but they do not contain any functionality. These methods are also never being used. This violates the “Interface Segregation Principle”.

Is the code well tested?

The code is tested with 51% line coverage, some packages are tested more than others which is fine. From what is described as the model package in the SDD, although not clear in the code itself, these parts of the code are tested with close to 100% line coverage. For the Controller component, there exists some framework used to create tests. The classes in the Controller component have been tested to some extent, though there seems to exist some inconsistency with the tests or some race conditions in the code, causing them to fail occasionally.

Are there any security problems, are there any performance issues?

The application uses accounts, which uses usernames and passwords to authenticate users. Passwords are stored in a database in plain text, meaning that anyone with access to the database file(s) can see and retrieve the values as normal strings. This is a security issue as sensitive data should not be so easily available -- better would be to first salt and hash the passwords. The current implementation of the application, however, requires the passwords to be in plain text form. As the functionality for actually sending mails doesn’t work yet, we couldn’t fully investigate the performance of the application. From testing the application as-is, no performance issues could be found.

Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts?

It is pretty hard to understand what part of the code does what, because of a lack of a good folder structure. It is hard to identify the model of the project. However, there is a controller folder, so they are easy to find. The in-code documentation is lacking overall which makes the code hard to understand, unless one reads the documentation in the SDD. The project follows the MVC structure suggested by Oracle for using JavaFX with FXML files. There is not a clear model package though, which makes it difficult to follow the MVC structure. The model is isolated from the other parts though, which is good.

Can the design or code be improved? Are there better solutions?

The analysis tool in IntelliJ found 564 warnings, 23 errors, and 217 typos, many of which could be easily corrected. Correcting this would improve the code. There are also classes that implement interfaces and their methods, but the methods doesn’t contain any functionality. Removing unused methods would be a design improvement and help follow the interface segregation principle.