12.7.2020



Time Management

Team 5

Christopher Gusenbauer

Projekt  
dokumentation

Inhaltsverzeichnis

[1. Einleitung 3](#_Toc40889930)

[2. Umgesetzte Anforderungen 3](#_Toc40889931)

[3. Überblick über das System aus Benutzersicht 3](#_Toc40889932)

[4. Überblick über das System aus Entwicklersicht 3](#_Toc40889933)

[4.1. Entwurf 3](#_Toc40889934)

[4.1.1. Überblick über das System 3](#_Toc40889935)

[4.1.2. Wichtige Design Entscheidungen 3](#_Toc40889936)

[4.2. Implementierung 3](#_Toc40889937)

[4.3. Code Qualität 3](#_Toc40889938)

[4.4. Testen 3](#_Toc40889939)

[5. Installationsanleitung 4](#_Toc40889940)

Versionshistorie

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datum | Ersteller | Änderungen  Mehr Details siehe Github |
| Release 1 | 31. März 2020 | Christopher Gusenbauer | Old Team Release |
| Release 2 | 11. Mai 2020 | Christopher Gusenbauer | First Solo Release  Grundfunktionalität, GUI, etc.. |
| Release 3 | 28. Juni 2020 | Christopher Gusenbauer | Big Feature Update  Tags, Statistik, Tracking, Filtern/Sortieren,Color Coding, Drag to Delete, Executable, .. |
| Final Release | 14. Juli 2020 | Christopher Gusenbauer | Final Polishing |

Tabelle : Versionshistorie

# Einleitung

*In diesem Projekt wurde ein Aufgabenplaner als Desktopapp implementiert. Mit diesem Programm („Tasky“) kann man Aufgaben verwalten, kategorisieren, filtern, Zeittracking betreiben, diese Daten sichern und importieren und Statistiken ansehen.  
  
Aufgrund der besonderen Situation des Wegfallens von zwei Teammitgliedern wurde das Projekt allein von Christopher Gusenbauer entwickelt und konzipiert. Der finale Code enthält 0% des Codes der ausgeschiedenen Mitglieder.   
  
Deswegen wurde ebenfalls vereinbart, die Anforderungen etwas abzuändern und das Programm von Projektmanagement mehr Richtung Taskmanagement zu entwickeln. Das wurde umgesetzt.*

# Umgesetzte Anforderungen

*Welche der Anforderungen haben Sie umgesetzt, wer aus dem Team war verantwortlich und welches Stundenausmaß ist damit verknüpft*

*Related issues*

*User Requirement #2394: TB-01*

*User Requirement #2395: TB-02*

*User Requirement #2396: TB-03*

*User Requirement #2401: TM-01*

*User Requirement #2403: TM-04*

*User Requirement #2401: TM-01*

*User Requirement #2403: TM-04*

*User Requirement #2978: TB-06: Aktivitäten können mit beliebig vielen Tags versehen werden*

*User Requirement #2979: TM-02: Zeittracking in der Applikation*

*User Requirement #2981: TF-05: Grafische Auswertung von Aktivitäten*

*User Requirement #2982: TF-01/02/03 Filtern und Sortieren der Aktivitäten*

*User Requirement #2983: TB-Neu-1: Intuitive Markierung von Aktivitäten als erledigt*

*User Requirement #2984: TB-Neu-2: Anpassung der Eigenschaften einer Aktivität*

*User Requirement #2986: TB-Neu-3: Applikation als ausführbares Programm zur Verfügung stellen.*

*Gesamtaufwand laut Redmine > 200 Stunden.*

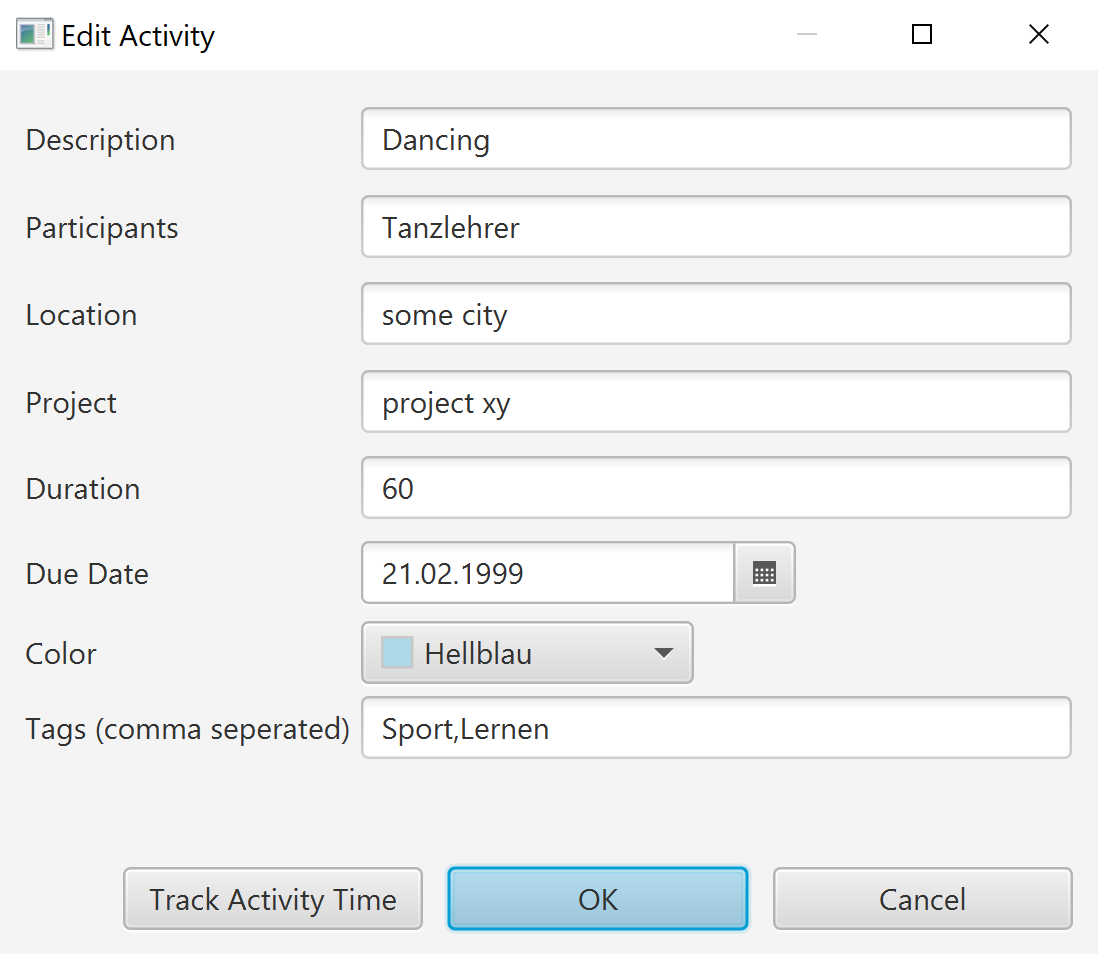
*Verantwortlich für Alle Requirements: Christopher Gusenbauer*

# Überblick über das System aus Benutzersicht

*Das ist Taskys Startoberfläche. Sie ist in vier Areale aufgeteilt. Oben befindet sich das Menü, über das die Daten von Tasky als XML gespeichert oder geladen werden können. Außerdem kann man dort die Statistikansichtöffnen. Auf der linken Seite befinden sich das Filtermenü und die Übersichtsansicht.*

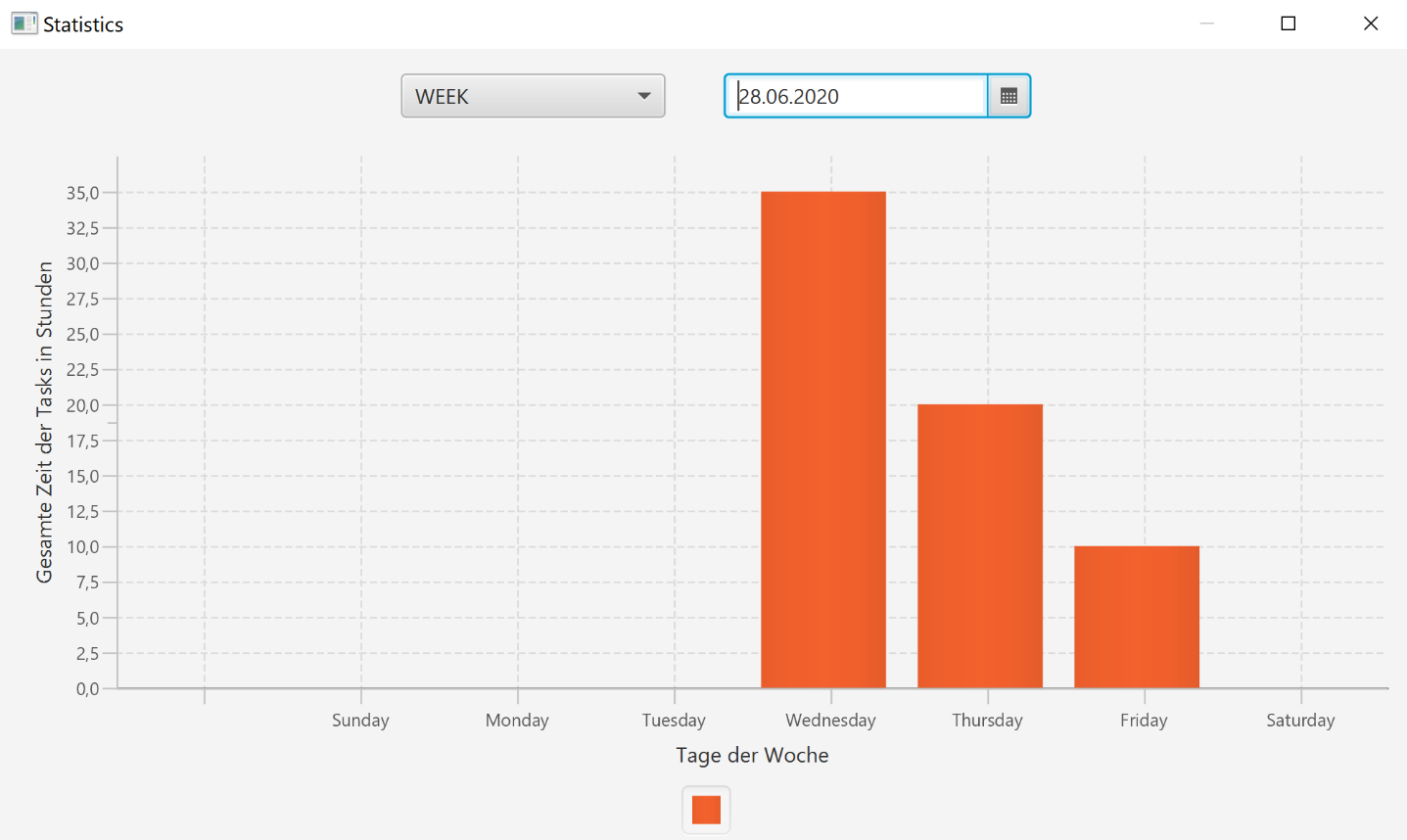
*In der Übersichtsansicht werden die Tasks tabularisch dargestellt und können mittels Klick sortiert werden und via Drag & Drop in ein eigens erscheinendes DeleteArea die Tasks löschen. In der Filteransicht kann nach textuellen Inhalten der Aktivitäten, nach einem Zeitraum des Fälligkeitsdatums und nach den inkludierten Tags gefiltert werden. Außerdem können die Suchkriterien auch zurückgesetzt werden.*

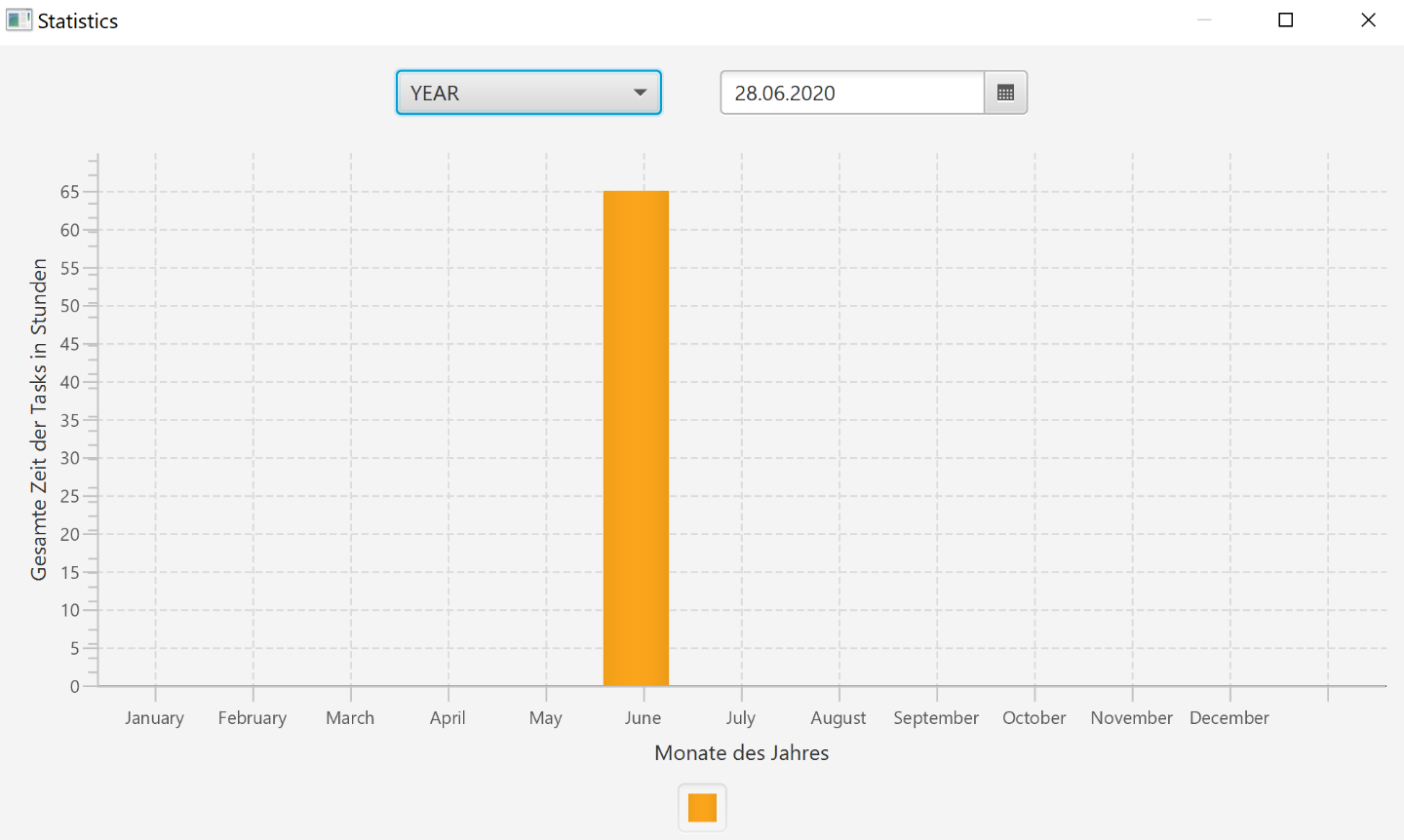
*Rechts sehen wir eine Detailansicht der jeweilig ausgewählten Aktivität und ihrer Attribute. Mit den Buttons „New“, „Edit“, „Delete“ wird die jeweilige Option für die ausgewählte Aktivität ausgeführt. Mit dem Edit oder New Button gelangt man in folgende Ansicht:*



*Hier kann eine Aktivität verändert werden. Zusätzlich zu den „normalen“ Eigenschaften gibt es auch besonderheiten. Ein Due Date wird mittels einem Date Picker gewählt. Mit einem Color-Picker kann man die Farbe auswählen, in der die Aktivität in der tabularischen Ansicht dargestellt wird. Es können mehrere Tags angegeben werden, welche automatisch gefiltert und getrennt werden und in der Filteransicht als Drop-Down verfügbar sind. Mit einmaligen Klick auf „Track Activity Time“ wird die Duration Eigenschaft im Sekundentakt hochgezählt, bis der Button erneut geklickt wird.*

*Auf der folgenden Seite ist noch die Statistikansicht zu sehen, welche bei gewähltem Zeitpunkt entweder die verbrauchte Zeit aller Aktivitäten pro Monat eines Jahres oder Tag einer Woche (des Zeitpunkts) anzeigt.*



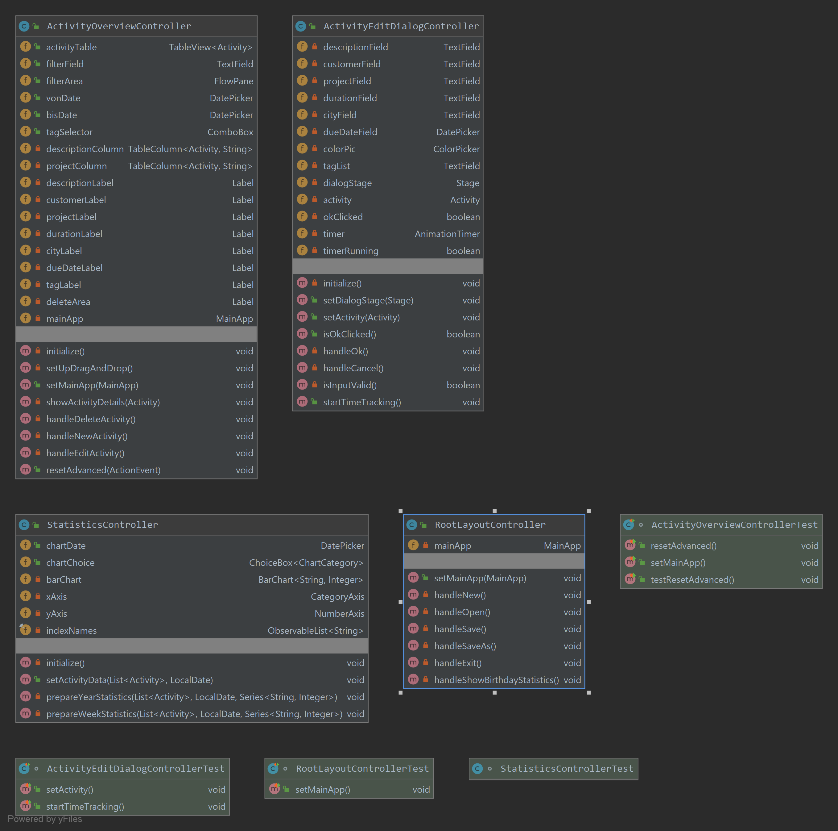
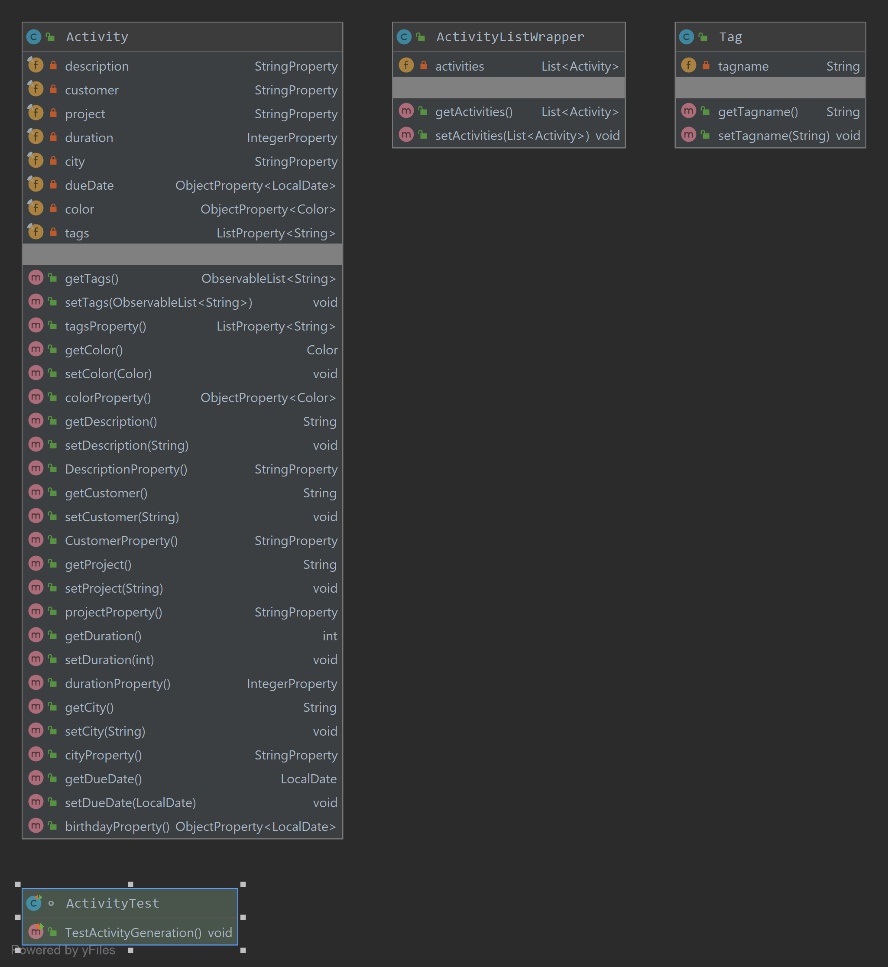


# Überblick über das System aus Entwicklersicht

## Entwurf

### Überblick über das System

*UML Diagramm mit Erläuterungen*

*Verwendete Design Muster (z.B. Model-View-Controller)*

*Das MVC Design Muster wurde leicht abgewandelt gewählt. Die View entspricht den FXML Dateien, das Model & der Controller (Package Controller) sind aber vermischter als üblich, da die App sehr kompakt auf nur einer Hauptseite dargestellt werden muss. Des weiteren gibt es eine Main Klasse die die Orchestrierung der App übernimmt.*

### Wichtige Design Entscheidungen

*Beschreibung der 3-5 wichtigsten Design Entscheidungen nach folgenden Schema*

*Entscheidung: Kompakte Gui mit möglichst wenigen Fenstern*

*Begründung: ein Taskmanager sollte so einfach wie möglich sein und ein Benutzer sehr schnell die gewünschte Funktionalität nützen können*

*Alternativen, die in Betracht gezogen wurden: Aufteilung auf verschiedene Fenster*

*Annahmen: bequemer für Nutzer*

*Konsequenzen: Kompakte Gui mit möglichst wenigen Fenstern*

*Entscheidung: JAXB statt moderneren Libraries*

*Begründung: JAXB ist ausgereifter, vollständiger und besser dokumentiert*

*Alternativen, die in Betracht gezogen wurden: SAX, DOM, StaX*

*Annahmen: keine bessere Alternative verfübar*

*Konsequenzen: Migration der etwas veralteten JAXB Library Usage auf neue Modulare Java Version*

*Entscheidung: händische Enduser Tests*

*Begründung: wenige verfügbare GUI-Testing Frameworks, starke Kopplung mit der Logik wegen des kompakten EinFensterGui – Aufbaus, TestFX Dokumentation mangelhaft…*

*Alternativen, die in Betracht gezogen wurden: TestFX, JUNIT allein, JemmyFX, selber Framework schreiben*

*Annahmen: kaum Alternativen/Doku verfügbar*

*Konsequenzen: händisches Testen*

*Entscheidung: Color Coding umsetzen Ja/Nein*

*Begründung: Ja, weil einfach ein wichtiges und praktisches Feature, auch wenn es viel Aufwand verursacht*

*Alternativen, die in Betracht gezogen wurden: Color nur im Edit Dialog anzeigen*

*Annahmen: umsetzbar*

*Konsequenzen: großer Zeitaufwand, eigene Recherche, Stackoverflow Posts etc.*

*Entscheidung: Wie Tags Umsetzen*

*Begründung: Als einfache komma-separierte Liste, da simpel und gut zu benutzen*

*Alternativen, die in Betracht gezogen wurden: Als eine Art Container mit Löschbutton, verworfen weil für Benutzer komplizierter*

*Annahmen: umsetzbar*

*Konsequenzen: hin und her Parsen notwendig*

## Implementierung

Das Projekt baut auf dem JavaFX Maven Archetype aus. Benutzte Technologien sind hauptsächlich Java und JavaFX. Das Projekt ist in die MainApp Klasse, eine Launcher Klasse und drei Packages gegliedert:

*Dataclasses* – beinhaltet Datenobjekte die repräsentiert werden

*Util* – Diverse Hilfsklassen wie Enums & Dateparser

*Controller* – beinhaltet die Steuernde Logik, die Controller

Im *Ressources* Ordner finden sich die Views (fxml) und die App-Settings als Property-File, wie auch Styling-Related Content.

**Ausgewählte kleine Codebeispiele**

*Wie generell die Input/Fehlerkontrolle funktioniert*

private boolean isInputValid() {  
 StringBuilder errorMessage = new StringBuilder("");  
  
 if (descriptionField.getText() == null || descriptionField.getText().length() == 0) {  
 errorMessage.append("No valid Description!\n");  
 }

Wie das Zeittracken funktioniert

@FXML  
public void startTimeTracking() {  
 if (timerRunning) {  
 timer.stop();  
 timerRunning = false;  
 } else {  
 final long startTime = System.*currentTimeMillis*();  
 timer = new AnimationTimer() {  
 @Override  
 public void start() {  
 super.start();  
 timerRunning = true;  
 }  
  
 @Override  
 public void handle(final long now) {  
 final long elapsedMillis = System.*currentTimeMillis*() - startTime;  
 durationField.setText(Long.*toString*(elapsedMillis / 1000));  
 }  
 };  
 timer.start();  
 }  
}

Wie der Aufbau des Drag and Drop Features funktioniert

activityTable.setOnDragDetected((final MouseEvent event) -> {  
 // drag was detected, start drag-and-drop gesture  
 final Activity selected = activityTable.getSelectionModel().getSelectedItem();  
 if (selected != null) {  
  
 final Dragboard db = activityTable.startDragAndDrop(TransferMode.*ANY*);  
 final ClipboardContent content = new ClipboardContent();  
 content.putString(selected.getDescription());  
 db.setContent(content);  
 event.consume();  
 }  
 deleteArea.setStyle("-fx-background-color:#34c3eb;");  
 final FadeTransition fadeIn = new FadeTransition(  
 Duration.*millis*(500)  
 );  
 fadeIn.setNode(deleteArea);  
 fadeIn.setFromValue(0.0);  
 fadeIn.setToValue(1.0);  
 fadeIn.setCycleCount(1);  
 fadeIn.setAutoReverse(false);  
 fadeIn.playFromStart();  
 deleteArea.setVisible(true);  
 } //drag  
);

Wie die aktuelle Woche herausgefunden wird und ein DatenArray für die Chart erstellt wird

private void prepareWeekStatistics(final List<Activity> activitys, final LocalDate date, final XYChart.Series<String, Integer> series) {  
 String[] days;  
 int[] dayOfWeekCounter = new int[7];  
 final DayOfWeek firstDayOfWeek = WeekFields.*of*(Locale.*getDefault*()).getFirstDayOfWeek();  
 final DayOfWeek lastDayOfWeek = DayOfWeek.*of*((firstDayOfWeek.getValue() + 5) % DayOfWeek.*values*().length + 1);  
 activitys.forEach(p -> {  
 final LocalDate firstDate = date.with(TemporalAdjusters.*previousOrSame*(firstDayOfWeek)); // first day  
 final LocalDate lastDate = date.with(TemporalAdjusters.*nextOrSame*(lastDayOfWeek));  
 final LocalDate currentDate = p.getDueDate();  
 if (currentDate.isEqual(date) || currentDate.isAfter(firstDate) && currentDate.isBefore(lastDate)) {  
 final int day = currentDate.getDayOfWeek().getValue() - 1;  
 dayOfWeekCounter[day] = dayOfWeekCounter[day] + (p.getDuration() / 60);  
 }  
 });

*wie der FilePath gesetzt wird mithilfe von Java Properties*

public void setActivityFilePath(final File file) {  
 final Properties prefs = new Properties();  
  
 try (InputStream in = MainApp.class.getResourceAsStream("app.properties");) {  
 prefs.load(in);  
 } catch (IOException e) {  
 System.*out*.println("No properties file exists");  
 }  
 if (file != null) {  
 prefs.put("filePath", file.getPath());  
  
 // Update the stage title.  
 primaryStage.setTitle("TaskyCompanion - " + file.getName());  
 } else {  
 prefs.remove("filePath");  
  
 // Update the stage title.  
 primaryStage.setTitle("TaskyCompanion");  
 }  
 try {  
 prefs.store(new FileOutputStream("app.properties"), "Appproperties");  
 } catch (IOException e) {  
 System.*out*.println("No properties file exists");  
 }

*Wie die Daten/Objekte dann wirklich abgespeichert werden über JAXB*

public void saveActivityDataToFile(final File file) {  
 try {  
 final JAXBContext context = JAXBContext  
 .*newInstance*(ActivityListWrapper.class);  
 final Marshaller m = context.createMarshaller();  
 m.setProperty(Marshaller.*JAXB\_FORMATTED\_OUTPUT*, true);  
  
 // Wrapping our activity data.  
 final ActivityListWrapper wrapper = new ActivityListWrapper();  
 wrapper.setActivities(activityData);  
  
 // Marshalling and saving XML to the file.  
 m.marshal(wrapper, file);  
  
 // Save the file path to the registry.  
 setActivityFilePath(file);  
 } catch (JAXBException e) { // catches ANY exception  
 final Alert alert = new Alert(Alert.AlertType.*ERROR*);  
 alert.setTitle("Error");  
 alert.setHeaderText("Could not save data");  
 alert.setContentText("Could not save data to file:\n" + file.getPath());  
  
 alert.showAndWait();  
 }

**Abhängigkeiten**

OpenJFX

Apache Commons

JUNIT

Log4J

Jakarta.xml.bind

com.sun.xml.bind

Maven Shade Plugin

**Ressourcen**

Als Hilfestellung wurden sehr viele verschiedene Websiten und Guides benützt, z.B.:

Stackoverflow, JavaFX CSS Reference Documentation, Marco Jacobs JavaFX Blog Guides und ein Base Project, Baldung, Oracle Documentation, Github Wikis, draeger-it Blogs, IntelliJ Doku, diverse Subreddits wie /r/java /r/javahelp und /r/programming

## Code Qualität

*Beschreibung der Verwendung von PMD, Beschreibung der Findings und welche davon behoben wurden*

*Viele Findings, aber viele davon falsche Treffer wegen JavaFX, teils sinnlose Findings und einige deren Benefit im Vergleich zum zusätzlich benötigten Zeitaufwand es nicht wert waren, bzw. die unlösbar sind. z. B. meine FilterLogik, da viele Filter auf einmal filtern ist die Komplexität einfach recht hoch.*

*In Fällen wo die Implementationszeit verdreifacht werden würde, wurde ebenfalls nichts geändert, da der Benefit hier für mich nicht wirtschaftlich erscheint.*

*85 Findings wurden behoben.*

## Testen

Es wurde immenser Aufwand betrieben um sinnvolle Tests für JavaFX zu schreiben, wobei die Schlussfolgerung aber darauf hinauslief, dass es nicht mit dem derzeitig im Web verfügbaren Wissenstand möglich ist, tatsächlich nützliche, über absolute Basictests hinausgehende Tests zu schreiben. Aufgrund der Tatsache, dass es sich um eine GUI-App handelt und praktisch keine alleinige „Model-Logik“ verfügbar ist, sondern hauptsächlich Controller und GUI-Logik ist mit JUNIT alleine nur eine sehr geringe Anzahl an sinnvollen Tests schreibbar. Es wurde intensive Recherche betrieben, eine Lösung zu finden. TestFX <https://github.com/TestFX/TestFX> wäre eine mögliche Lösung. Leider existiert vermutlich aufgrund der geringen Popularität von JavaFX im Internet kaum Dokumentation oder Anleitungen, die über das reine Setup von TestFX hinausgehen. Die meisten Anleitungen sind bereits Outdated. Schlussendlich hat sich auch einfach herausgestellt, das der Aufbau meiner App für Code-Tests nicht gut geeignet ist, und tatsächliche GUI-Tests hier fast ein must-have wären, weil aufgrund einiger Anforderungen von Spezialfällen eine teils starke Verbindung der Klassen notwendig war.  
  
Um etwaige Fehler identifizieren zu können, wurde deswegen versucht sehr genau mit Enduser Tests, also manuellen GUI-Tests und dem Ausprobieren von Szenarien das Programm zu testen. Mit dieser Methode wurden einige Fehler entdeckt und ausgebessert, wie beispielswiese ein Bug mit der Generation von Tags

# Installationsanleitung

*Benötigt: Maven, Java, je nach Ausführmethode OpenJFX oder ein JDK mit OpenJFX inkludiert, z.B. Liberica JDK*

*Via IDE: Rechtsklick auf Launcher.java, Ausführen*

# Via Maven:

mvn clean javafx:run

# Via JLINK/RunScript:

Mvn javafx:jlink

Im Projektdirectoryroot in der Shell:  
target/image/bin/java -m at.gusenbauer.taskycompanion/at.gusenbauer.taskycompanion.Launcher

In der aktuellsten Projektversion wird auch ein .bat Script generiert, welches automatisch die App ausführt.

Dieses generierte Image ist Plattformabhängig und muss je OS neu generiert werden, im Github Repository inkludiert sind die Windows Dateien.

Abschließende Worte

Ich möchte mich für die Kulanz und den netten Umgang mit der Situation das ich unverschuldet plötzlich zur Sologruppe geworden bin vielmals bedanken. Ich werde mein Projekt wahrscheinlich als Open-Source Projekt selber noch später weiterentwickeln.