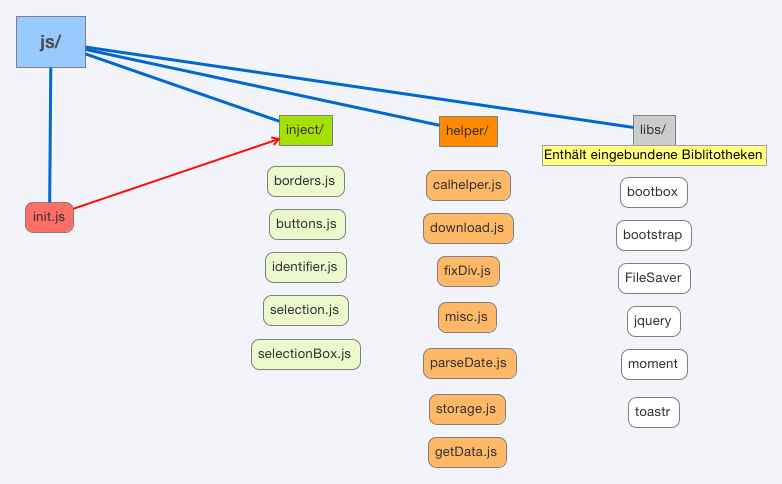
# 2. Review

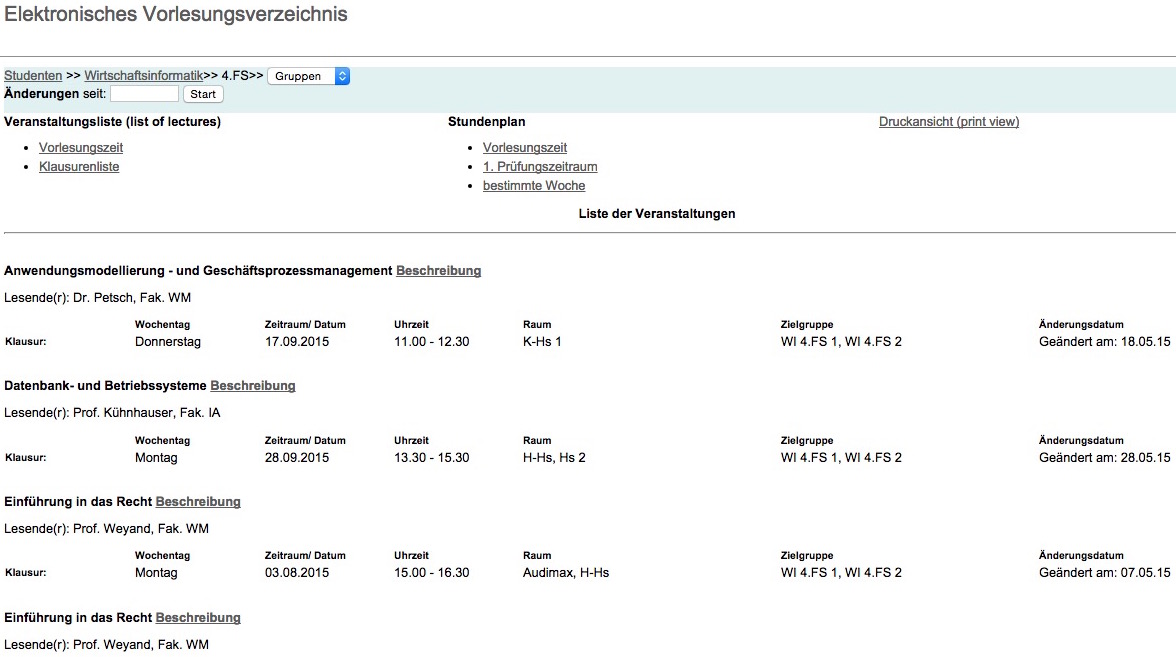
Dieses Dokument und dessen Inhalt knüpfen an Sachverhalte des ersten Review-Dokuments an.

## Technischer Hintergrund

### Initialisierung



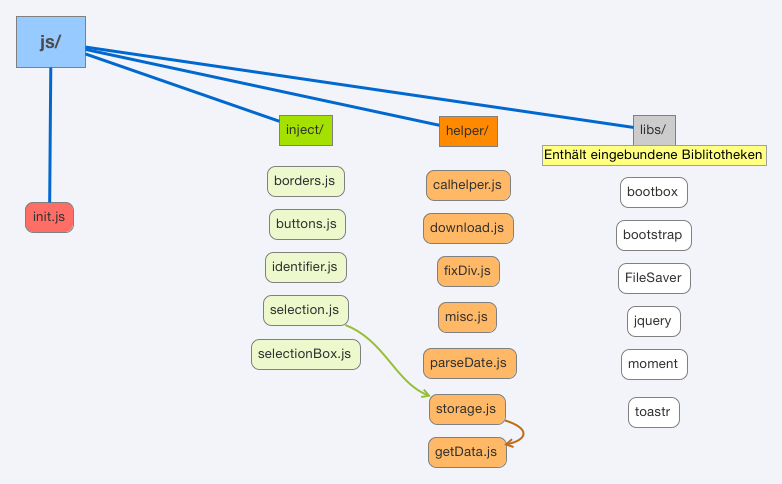
Der Einstieg in die Programmlogik befindet sich in der Datei namens init.js. Diese initialisiert das Programm, indem es benötigte Variablen entweder aus dem Speicher liest, oder diese leer erzeugt, wie zum Beispiel ein Array für den "Warenkorb". Im Anschluss erzeugt es die Seitenleiste für den Warenkorb, der auf allen Seiten des VLV vorhanden sein wird. Es folgen dann einige Überprüfungen, die je nachdem auf welcher Unterseite man sich befindet, bestimmte Elemente erzeugen und Funktionen aufrufen. Im Folgenden bezieht sich der gesamte Ablauf auf die Textansicht eines bestimmten Studienganges.



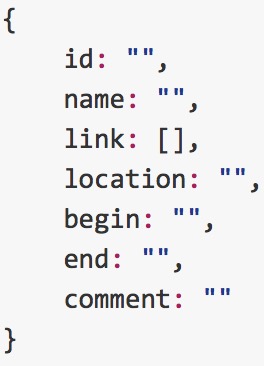
Text Version des VLV

Da die Überprüfung, ob es sich um die Textversion handelt, positiv ausfällt, wird nun die Logik aufgerufen, welche alle Veranstaltungen auf der Seite sucht und diese in einem Array temporär speichert. An diesem Punkt wird außerdem überprüft, ob alle Elemente eine ID enthalten, welche zum weiteren Programmablauf benötigt wird. Ist diese nicht vorhanden, so wird eine zufällige erzeugt, dem Element zugeordnet und die ID persistent abgespeichert, damit in Zukunft einem Objekt, welches schon eine neue ID erhielt, wieder dieselbe zugewiesen werden kann. Anhand des Arrays der Veranstaltungen werden auf dieser Seite nun Buttons eingefügt, die es erlauben, eine Veranstaltung zu dem Warenkorb hinzuzufügen. Des weiteren werden Ränder eingefügt, um Veranstaltungen optisch von anderen abzugrenzen. Des weiteren wird in diesem Punkt eine Logik aufgerufen, die den Nutzer darüber informiert, ob in den letzten 2 Wochen (Zeitraum wird in zukünftigen Versionen konfigurierbar sein) Änderungen der Daten stattgefunden haben. Dies erfolgt über eine optische Hervorhebung der Aktualisierungsdaten und über eine Benachrichtigung in der rechten oberen Ecke der Seite. Am Ende der Initialisierung wird aus dem Speicher ein eventuell vorhandener Warenkorb aus einer vorherigen Sitzung ausgelesen und diese Auswahl erneut hervorgenommen. Nun ist die Initialisierung abgeschlossen und der weitere Programmablauf wird durch die Interaktionen des Nutzers bestimmt.

### Auswahl einer Veranstaltung



Wird eine Veranstaltung zum Warenkorb hinzugefügt, so wird eine Funktion namens saveToCart() aufgerufen. Diese Funktion erzeugt ein leeres JSON Objekt nach folgendem Aufbau:



Dieses noch leere Objekt wird nun nach und nach mit Informationen gefüllt: - id: Die eindeutige ID des Container-Objekts (ein div Objekt) dieser Veranstaltung - name: Der Name der Veranstaltung - link: Der DOM-Pfad zu dem Container-Objekt dieser Veranstaltung (um später wieder auf das ursprüngliche Objekt schließen zu können) - location: Der Ort, an dem die Veranstaltung stattfinden wird - begin: Der Startzeitpunkt, an dem die Veranstaltung beginnt - end: Der Endzeitpunkt, an dem die Veranstaltung endet - comment: Ein Kommentar zu der Veranstaltung, in dem wir die/den Lesende(n) einfügen

Die werte name, location, begin, end und comment sind durch den Nutzer später durch einen modalen Dialog änderbar.

### Datenextraktion

Der Aufbau einer Veranstaltung im VLV sieht aus wie folgt:



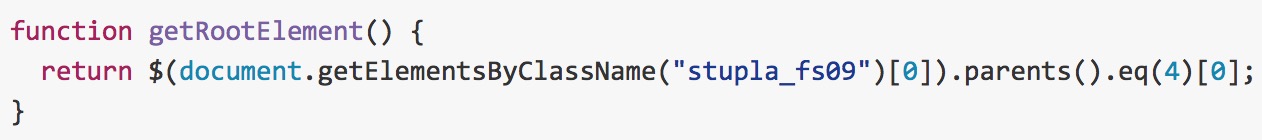
Haben wir den Elternknoten (<div id="nr...></div>), so können wir die Kindknoten leicht auslesen. Dies erfolgt beispielsweise durch einen Aufruf wie:



Das object wäre hierbei unser oberster Elternknoten. Von diesem aus wird das 3. Kindelement aufgerufen, davon dann der Wert innerText (da man sonst auch die HTML-Tags bekommt, welche wir nicht wollen). Abschließend wird durch slice(12) noch das vorhergehende Lesende(r): abgeschnitten, damit wir als Ergebnis nur einen String mit dem Inhalt Prof. Nissen, Fak. WM erhalten. Analog dazu erfolgt das Auslesen der restlichen Informationen, wobei gegebenenfalls Informationen, wie die Uhrzeit und Datum vorher noch geparsed werden müssen.

### Parsen der Veranstaltungsinformationen aus dem Vorlesungsverzeichnis

Das Vorlesungsverzeichnis hat bei seinen einzelnen Veranstaltungen eine gleichbleibende Struktur (siehe oberen HTML-Ausschnitt des Vorlesungsverzeichnis der Veranstaltung Einführung in ERP-Systeme). Der Beginn des Auslesens wird mit dem mitteilen eines Einstiegspunkt in der Datei GetData.js in der Methode getRootElement() festgelegt.



In dieser Methode wird der übergreifende DIV-Container in dem sich die einzelnen Veranstaltungsdetails befinden ausgewählt. Praktisch der erste mit dem wir arbeiten. Es wird das Skript nach Elementen der Klasse stupla\_fs09 durchsucht und deren übergreifender Elternknoten ausgewählt. Im nächsten Schritt werden die einzelnen Details Name der Veranstaltung, Lesender, Wochentag, Uhrzeit, Raum, Zielgruppen und Änderungsdatum durch einzelne Funktionen zurückgegeben. Hierbei wird der übergreifende Elternknoten auf die einzelnen Arrays der Kindknoten heruntergebrochen. Im unteren Beispiel wird der Wochentag durch die Methode getDayOfWeek(object) zurückgegeben. Als Input bekommt die Methode den Elternknoten und durchläuft dann eine Kette aus Kindknoten, bis es zum richtigen Knoten gelangt, der den Wochentag beinhaltet. Den Inhalt dessen liefert das Attribut .innerText.



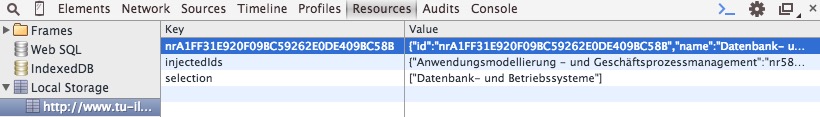
In der Funktion getEventData(subject) (siehe oben) wird dann den einzelnen Keys (name, comment, location, begin, end) des JSON Objektes data ein durch die Funktionen zurückgegebener Wert zugewiesen. Dies sind die Informationen einer Veranstaltung, die der externen Kalenderapplikation in Form einer ics-Datei mitgeteilt werden. Die Informationen Zielgruppe und Änderungsdatum fallen hier heraus.

#### Zeit und Datum parsen

Aufgrund des in der ics-Datei erlaubten Formats muss die Zeit und der Wochentag, sowie die Wiederholungen der Kalenderwochen in einen auf die Uhrzeit genauen Start- und Endtermin konvertiert werden. Dies erfolgt in der Datei parseDate.js . !Wird noch vervollständigt!

### Speicherung der Daten im Local Storage des Browsers

Damit wir diese Informationen nach einem Schließen des Browserfensters nicht verlieren, speichern wir diese dann im local Storage ab. Jeder moderne Browser unterstützt diese Art der Speicherung. Die Objekte bleiben so lange erhalten, bis man sie explizit löscht. Implementiert ist dieser Storage durch einen simplen Key-Value-Storage, der es erlaubt zu einem beliebigen Key einen beliebigen Value zu speichern, wobei der Value aber nur in Form eines Strings abgespeichert werden kann.



Um pro Key mehr Informationen abspeichern zu können, bieten sich die Funktionen JSON.stringify() und JSON.parse() an. Erstere wandelt ein JSON Objekt in einen String um und zweitere in umgekehrte Richtung.

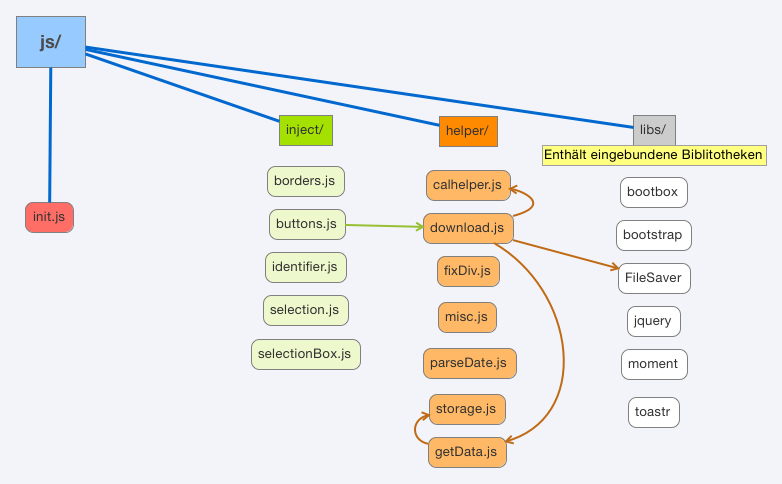
##### Beispiel



Dieser kleine Ablauf erzeugt ein JSON Objekt, welches eine ID und einen Namen besitzt. Dieses wird dann in den Local Storage geschrieben, indem es in einen String umgewandelt wird und unter dem Key Test123 abgespeichert wird. Durch JSON.parse() wird der gelesene String wieder in ein JSON Objekt umgewandelt und kann dann in der Konsole ausgegeben und als Objekt betrachtet werden.

Dies kann erweitert werden, sodass auch komplexere Objekte, wie z.B. ein Array von JSON Objekten abgespeichert werden können.

### Download der Kalenderdatei



Da eine gültige Kalenderdatei nach dem vCalendar Format lediglich eine Textdatei ist, die bestimmten Regeln folgt, ist es sehr einfach, eine solche in Javascript zu bauen. Zu diesem Zweck erzeugen wir einen Array, der nach und nach mit Informationen ergänzt wird. Ein Feld des Arrays entspricht einer Zeile der fertigen Datei.



Nachdem die letzte Zeile (END:VCALENDAR) hinzugefügt wurde, wird der Array mittels .join("\n") zusammengefügt. Zwischen allen Feldern wird allerdings zusätzlich noch ein Zeilenumbruch eingefügt. Dieser fertige String wird dann in einen Blob umgewandelt. Dieser wird anschließend mit angegebenem Dateinamen als .ics Datei heruntergelden.

### Waffle.io

Waffle erstellt eine vollständige Projektmanagementlösung von vorhandenen GitHub Themen. Das Waffle Bord zeigt die GitHub Issues und Pull Requests in Echtzeit. Waffle achtet auf die Aktionen im Arbeitsablauf um zu wissen, wann die Arbeit beendet ist und aktualisiert den Status automatisch. Jeder kann einem öffentlichen Waffle Bord zu sehen.  
Hinzufügen eines Waffle Abzeichen auf Readme lassen die Mitwirkenden wissen, was bearbeitet werden muss und was schon fertig ist. Waffle ist mit dem GitHub-Repository integriert, so dass die Issues an dem Board standardmäßig organisiert werden. Das spart Zeit und Arbeit. Waffle hilft die laufenden Arbeiten zu visualisieren und zu priorisieren.

#### Real-time

Änderungen an dem Bord werden angezeigt ohne dass die Seite neu geladen werden muss.

#### Milestones

Organisieren der Issues in Milestones dient zur einfacheren Planung und Verfolgung.

#### Multiple Repos on a single Board

Verbinden von beliebig vielen GitHub Repositories auf einem Board ist möglich.

#### Anpassbare Workflow

Das Board kann geändert werden, um es an den Arbeitsablauf an zu passen.

### Bewertung unseres Vorgehensmodell: Scrum

Scrum ist ein sehr agiles Vorgehensmodell. es setzt nicht auf die klassischen Phasen der Softwareentwicklung. Die Ziele sind aber dieselben: Produkterfolg im Sinne der Interessen der Stakeholder, Prozesserfolg als termintreue Sprints. Durch den Einsatz des Product Owner ist die Zufriedenheit des Auftraggebers gesichert. Weiter kann der der Product Owner User Storys im Backlog streichen oder bestätigen um so Individuelle und kurzfristige Wünsche zu realisieren. Das Ergebnis ist das Release Backlog. Der Product Owner ist somit Bestandteil des Produkterfolges. Für uns ist der Product Owner unser Betreuer. Scrum stellt somit die Zufriedenheit des Betreuers sicher, was nur im Interesse des Teams seien kann.

Ferner liegt ein Vorteil von Scrum im Prozesserfolg. Durch die Organisation in Sprints mit von uns angepassten Terminen (Milestones) haben wir zu jedem Zeitpunkt eine lauffähige Software die während der Dauer der Entwicklung weiter ergänzt und verbessert werden kann. Die Aufgabe des Scrum Masters ist es hier das Release Backlog in Sprints aufzuteilen. Die Sprintmeetings helfen uns ein gemeinsamen Konsens über etwaige Probleme zu finden und fördern so die Teamarbeit. Ein „Feature freeze“ soll es in Scrum nicht geben.

Scrum ist für uns also ein großer Erfolg. Zunächst weil wir nur geringe praktische Erfahrung in der Softwareentwicklung haben und so auf Probleme die in der Entwicklungsphase nicht erkannt worden sind zu reagieren, aber auch weil wir als junges Team die Möglichkeit haben in Zusammenarbeit mit dem Product Owner die Software während der Entwicklung selber mit mitzugestalten. Abschließend lässt sich also sagen, dass Scrum für ein gut gewähltes Vorgehensmodell ist, da es ein sehr agilen und dynamischen Charakter hat.

## Überprüfung der Nutzerfreundlichkeit mittels Testpersonen

Um unsere Software auf Intuitivtät und Nutzerfreundlichkeit zu testen, haben wir verschiedene Personen gebeten unser Plug-In zu bedienen. Dabei fiel uns auf, dass die Bedienung auf aktuellem Stand der Software noch nicht Intuitiv genug ist. Lediglich einer einzigen Testperson gelang es, einen Termin in den Warenkorb einzufügen und anschließend auch herunterzuladen. Infolge dessen haben wir uns dazu entschlossen ein Interaktives Tutorial zu entwickeln, welches beim ersten Ausführen des Plug-Ins ausgeführt wird.

## Glossar

* **local Storage**: Ein lokaler Speicher, der mit HTML5 eingeführt wurde. Er ist in allen modernen Browsern verfügbar und ermöglicht eine persistente Speicherung von Informationen über das Schließen einer Seite hinaus.
* **JSON**: Die JavaScript Object Notation ist ein kompaktes Datenformat in einer einfach lesbaren Textform zum Zweck des Datenaustauschs zwischen Anwendungen.
* **BLOB**: Binary Large Objects (BLOBs) sind große binäre Objekte. Dies dient uns dazu aus einem String eine Datei zu erstellen.