Planeten Browsergame

Tim Felix Tanner (1151110)

Bastian Schneider (1151420)

Einleitung

Die Folgende Projektbeschreibung schildert den Aufbau und Ablauf des Projektes im Modul Mobile Applikationen der Fachhochschule Bielefeld, welches wir im Laufe des Semesters durchgeführt haben.

Projektbeschreibung

Die Entwicklung eines Spiels, das auf Browserspielen basiert, die früher populär waren. Das Thema des Spiels ist, aufgrund der Tatsache das der Fantasie kaum Grenzen gesetzt sind, Science-Fiction. Das Spiel findet in einem fiktiven Universum statt und der Spieler übernimmt die Rolle des Herrschers eines Planeten. Der Spieler ist in der Lage Gebäude, Raumschiffe zu bauen und Forschung zu betreiben. Er kann sich mit gezielten Überfällen auf andere Spieler einen Spielfortschritt erarbeiten, um die größte Flotte aufbauen.

Projektziel

Die entwickelte Applikation soll anschaulich und intuitiv sein. Es wird bewusst auf Animationen und Ähnliches verzichtet, um den Stil und Charme der vorher genannten Browserspiele beizubehalten. Die Benutzeroberfläche soll simpel gehalten werden und dennoch alle nötigen Funktionen klar erkennbar darstellen, um eine einfache Bedienung zu gewährleisten. Das Design soll einheitlich sein, um der Applikation ein natürliches Aussehen zu verleihen.

Das Spielkonzept soll einfach gehalten werden und trotzdem dafür sorgen dem Spieler Spaß und die Lust zum weiterzuspielen bringen. Berechnungen und Ähnliches sollen daher auch im Hintergrund geschehen, um den Fluss des Spiels so wenig wie möglich zu unterbrechen.

Projektbegründung

Aufgrund der Tatsache, dass kaum Browserspiele, so wie sie sind, als Mobile Applikationen veröffentlicht wurden, haben wir uns dazu entschieden dieses Projekt zu bearbeiten.

Projektschnittstellen

Die Applikation, die vom Benutzer bedient wird, ist nur ein Teil eines Systems. Die Eingaben des Benutzers werden an einen Server gesendet und dort verarbeitet. Der Server wurde mithilfe von Node.js programmiert. Des Weiteren gibt es eine MySQL Datenbank zur Speicherung von verschiedensten Werten. Es wurde außerdem MySQL Workbench zur Überwachung und zum Einpflegen der Daten verwendet. Die Kommunikation zwischen Server und Datenbank ist auch mit Node.js geregelt und im Server implementiert. Genauere Informationen befinden sich im Abschnitt Implementierung. Der Benutzer interagiert also nur mit dem Client, während alles andere an anderer Stelle passiert. Mehr dazu im Abschnitt, in dem die Architektur beschrieben wird.

Projektabgrenzung

Da die Arbeitszeit am Projekt beschränkt ist, soll die Applikation mit wenigen Spielern im Sinn implementiert werden.

Stand der Technik

Spieleentwicklung

In der heutigen Spieleentwicklung wird viel Arbeit in das Design und in Animationen investiert. Viele der aktuellen Spiele werden mit einer Game Engine entwickelt. Weil die Arbeit an einer eigenen Engine zu viel Zeit in Anspruch nimmt, kann man eine Engine benutzen die der Öffentlichkeit zur Verfügung gestellt wird wie die Unreal Engine 3 oder Unity. Andere Game Engines wie die Frostbite Engine von EA Games werden nur von EA Games selbst benutzt oder an andere Entwicklerstudios verliehen.

Weil die Entwicklung mit einer Game Engine sehr aufwendig ist und das Designen und Animieren sehr viel Zeit in Anspruch nimmt und unseren Projektrahmen übersteigt, haben wir uns bewusst für ein Spiel ohne die genannten Komponenten entschieden.

Spiele überzeugen heutzutage oftmals durch sehr schönes Design oder umfangreiche Spielmöglichkeiten. Durch immer leistungsstärkere Hardware ist es mittlerweile mögliche sogar in Mobilen Applikationen auf umfangreiche Grafik zu setzen.

Eine andere Möglichkeit ein Spiel interessant zu machen ist durch ein gutes Spielkonzept. Hier haben wir die Idee aufgegriffen und unser Spieledesign an das, der früheren Browsergames angelehnt. Dadurch dass es keine Runde gibt, die endet, wird der Spieler ermutigt aktiver am Spielgeschehen teilzunehmen und durch viel Teilnahme wird man mit einer größeren Flotte belohnt.

HTTPS

Bei der Übertragung von kritischen Nutzerdaten, wie zum Beispiel Kontoinformationen oder Adressen ist HTTP nicht mehr ausreichend. Angreifer können diese Nachrichten relativ leicht auslesen, da ihr Inhalt nicht verschlüsselt ist.

Deswegen gibt es eine Erweiterung des Protokolls in Form von HTTPS. Im Gegensatz zu HTTP werden hierbei die gesendeten Daten verschlüsselt. Außerdem gibt es eine Authentifizierung anhand eines Zertifikates, die dem Client sicherstellt das der Server, der ist, für den er sich ausgibt.

Die Verschlüsselung basiert auf einem Prinzip aus mehreren Schlüsseln. Es gibt einen öffentlichen Schlüssel, der für jeden Client Sichtbar ist, der sich mit dem Server verbindet. Hat der Client mithilfe des Zertifikats den Server Authentifiziert (und der Server den Client) dann erhält er den Öffentlichen Schlüssel (public key). Der Server hat den zweiten Schlüssel, der privat ist (private key). Nun erzeugt der Client mithilfe des ersten Schlüssels einen verschlüsselten dritten Schlüssel (session key). Dieser Schlüssel wird nun mit dem Server geteilt. Mithilfe des privaten Schlüssels kann der Server den dritten Schlüssel entschlüsseln.

Somit wurde die zuvor Asymmetrische Verschlüsselung, also Verschlüsselung mit zwei verschiedenen zusammengehörigen Schlüsseln, zu Symmetrischer Verschlüsselung, also Verschlüsselung mit einem gemeinsamen Schlüssel und Nachrichten können sicher ausgetauscht werden.

Im Fall unseres Projektes ist diese Art der Sicherheit nicht notwendig, da die einzige kritische Information, die übertragen wird, die E-Mail-Adresse ist (siehe auch Sicherheit). Des Weiteren haben wir unseren Server selbst programmiert und lassen ihn lokal auf einem privaten Laptop laufen, was zur Folge hat das wir keine offizielle Authentifizierung haben. Daher macht es keinen Sinn die erweiterte Variante von HTTP zu verwenden.

Anforderungsdokumentation

Folgend werden die Anforderungen an die entwickelte Applikation zu Beginn des Projektes mit denen die am Ende tatsächlich erfüllt wurden gegenübergestellt. Außerdem wird die anfängliche Konzeption erläutert und wieder mit der endgültigen entgegengestellt

Übersicht

An dieser Stelle werden die gestellten Anforderungen dargestellt und durch einen Haken wird gezeigt, welche tatsächlich erfüllt wurden.

Must have:

- [x] Basen können ausgebaut werden (Minen, Truppen)
- [x] Angriffe auf Spieler
- [x] Ressourcenabbau
- [x] Bilder
- [x] Server-Client Architektur
- [x] Datenbank
- [x] Mindestens drei verschiedene Einheiten mit mindestens zwei verschiedenen Werten (Angriff, Leben)
- [x] Funktionierendes Kampfsystem
- [x] Erstellung eines Kontos und Anmeldung

Should have:

- [ ] Nicht-Spieler-Charaktere
- [x] Forschung
- [x] Verbesserungen (Truppen/Gebäude)
- [x] Verteidigungsanlagen
- [x] Ansprechenderes Design
- [x] Einfaches Kampfsystem

Could have:

- [ ] zweidimensionale Karte
- [ ] neue Server (Verschiedene Spielmodi)
- [ ] Sound bzw. Musik
- [x] Mehr Einheitenauswahl
- [x] Komplexes, ausbalanciertes Kampfsystem

Won't have:

- [ ] Animationen
- [ ] Werbung
- [ ] Bildschirmdrehung

Verlauf

Zu Beginn des Projektes waren wir sehr optimistisch und haben geglaubt, dass wir auch die could haves teilweise noch erreichen könnten. Im Laufe der Arbeit am Projekt hat sich herausgestellt das diese Einschätzung sehr optimistisch war und wir haben uns darauf geeinigt uns mit den should haves zufrieden zu geben. Am Ende ist es dann doch noch dazu gekommen, dass wir unsere erste Zielsetzung erreicht haben.

Auf die restlichen could haves und alle won't haves wurde bewusst verzichtet, da diese kaum neue Funktionalität erzeugt hätten. Sie hätten dazu beigetragen die Applikation vermarktbar zu machen, was für uns den Rahmen dieses Projektes sprengen würde.

Erste Konzepte

Im Folgenden werden die Verschiedenen Konzepte, die zu Beginn des Projektes erstellt wurden anhand von Mockups dargestellt und erläutert. Es wurden grundlegend zwei unterschiedliche Ideen verfolgt.

Landscape Layout

Nachfolgend sind Mockups zu sehen, die auf einem Konzept beruhen, das mit dem Landscape Layout erdacht wurde.

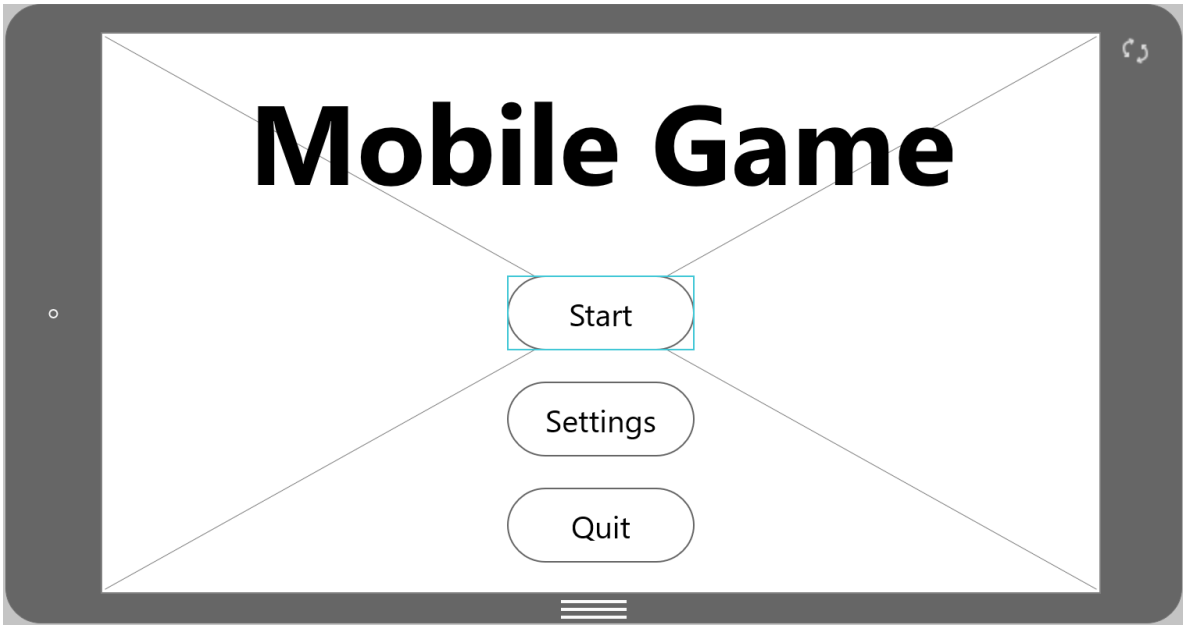

Bild 1: Mockup der Startseite

In diesem Bild sieht man einen Platzhalter für den Namen der Applikation sowie drei Knöpfe. Die Knöpfe sind selbsterklärend. Der erste startet das eigentliche Spiel, der zweite öffnet das Optionsmenü und der letzte beendet die Applikation. Im Hintergrund befindet sich ein Platzhalter für ein Hintergrundbild.

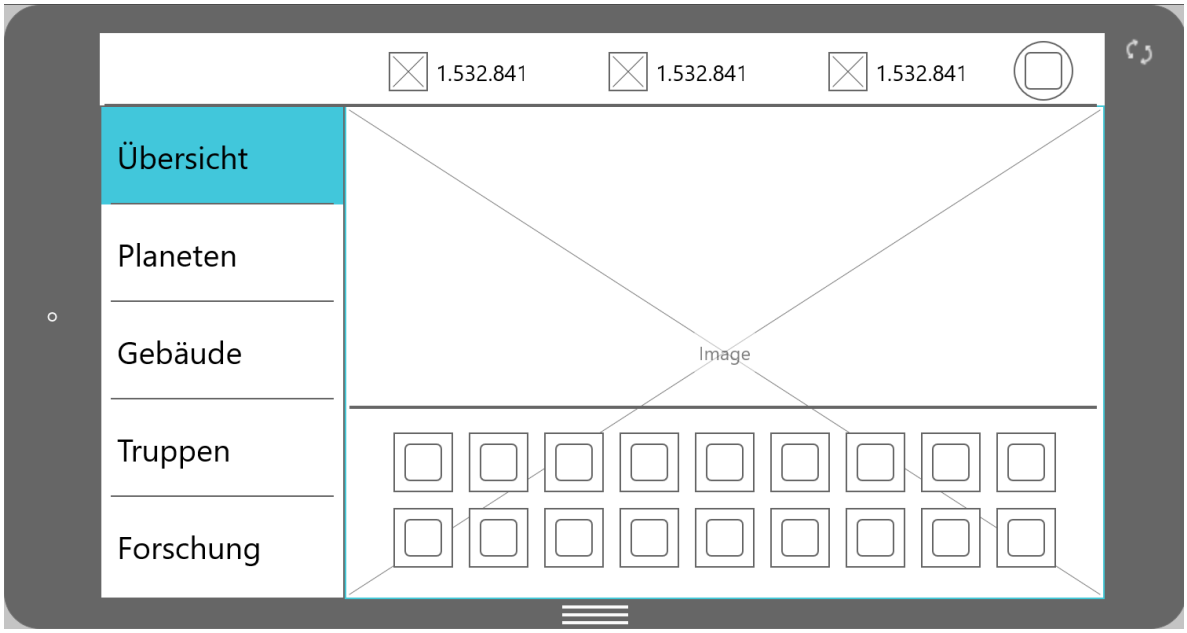

Bild 2: Mockup der Übersichtsseite

Hier ist die Übersichtsseite des Spiels dargestellt. Oben rechts sieht man die drei Ressourcen mit zugehörigen Bildern, sowie einen Knopf, der in das Optionsmenü führt. Am linken Rand befindet sich die Navigationsleiste zum Wechseln der Seiten. Der Hauptteil des Bildschirms ist mit einem Bild und verschiedenen Knöpfen befüllt, die zur Anzeige verschiedenster Informationen führen.

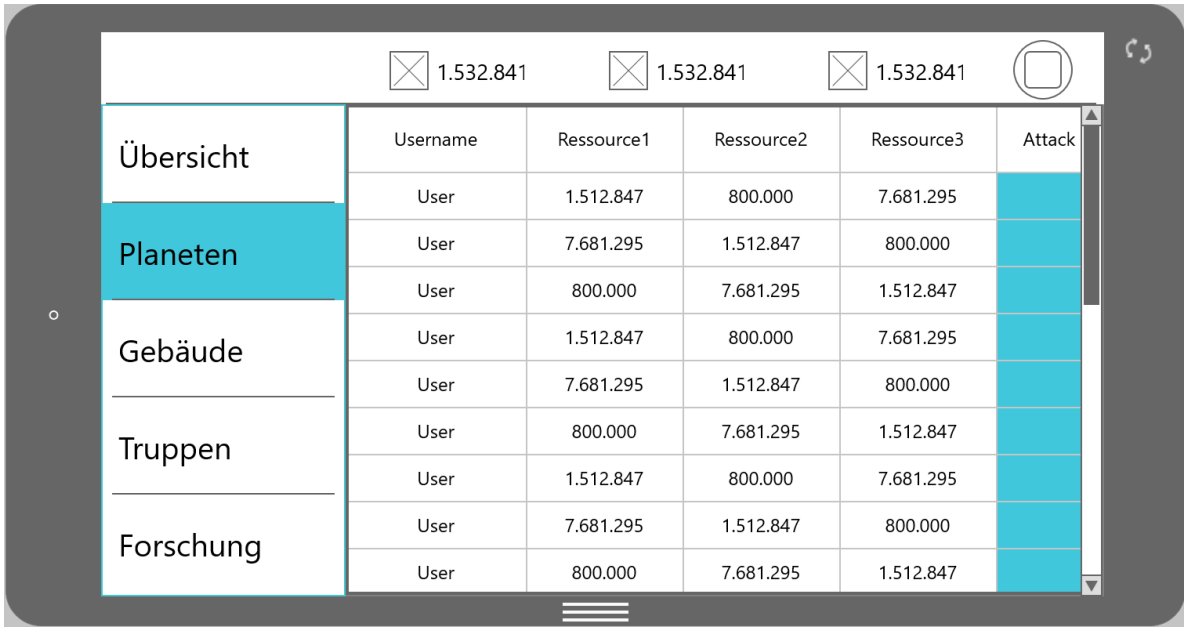

Bild 3: Mockup der Planetenseite

In dieser Darstellung wird die Planetenseite gezeigt. Der linke und obere Rand ist identisch zum vorherigen Bild. Rechts sieht man nun eine Übersicht aller Planeten mit Informationen über den Besitzer und seine Ressourcen. Des Weiteren gibt es für jeden Planeten einen Knopf, der einen Angriff startet.

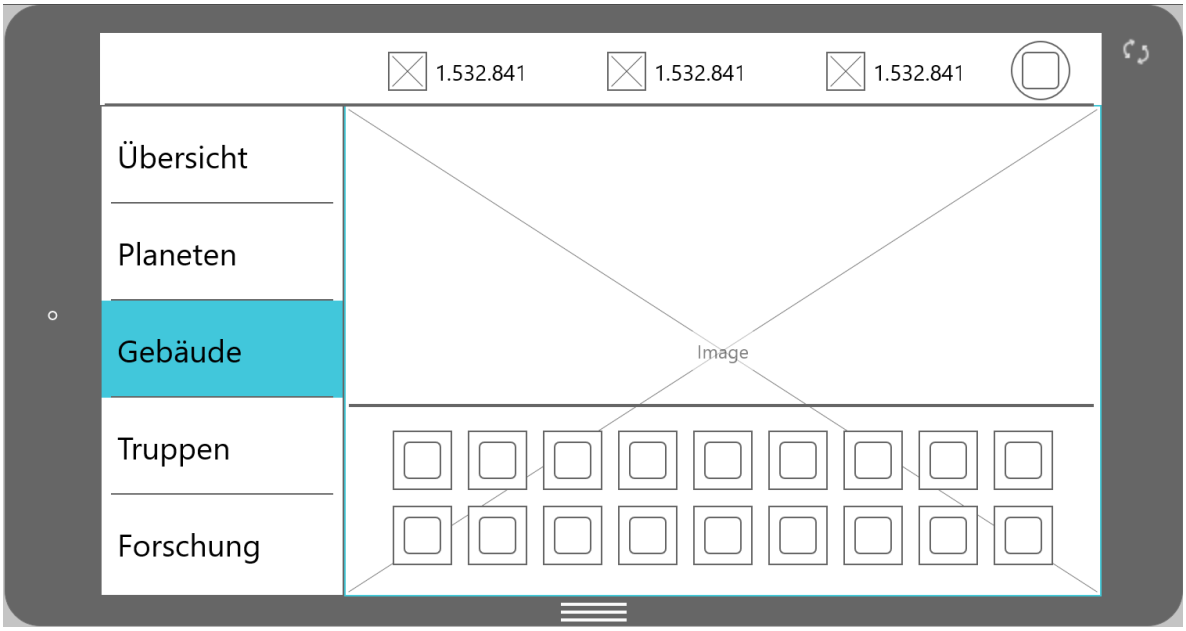

Bild 4: Mockup der Gebäudeseite

Dieses Bild stellt die Gebäudeseite dar. Sie ähnelt *Bild 2* vom Aufbau, jedoch ist die Funktion der Knöpfe anders. Hier können sie dazu benutzt werden die verschiedenen Gebäude aufzuwerten.

Die weiteren Einträge des Navigationsmenüs (Truppen, Forschung) wurden an dieser Stelle nicht weiter Beschrieben, da sie im Aufbau und der Funktionalität *Bild 4* entsprechen.

Portrait Layout

Hier werden Mockups gezeigt, die mit dem Portrait Layout im Sinn entwickelt worden.

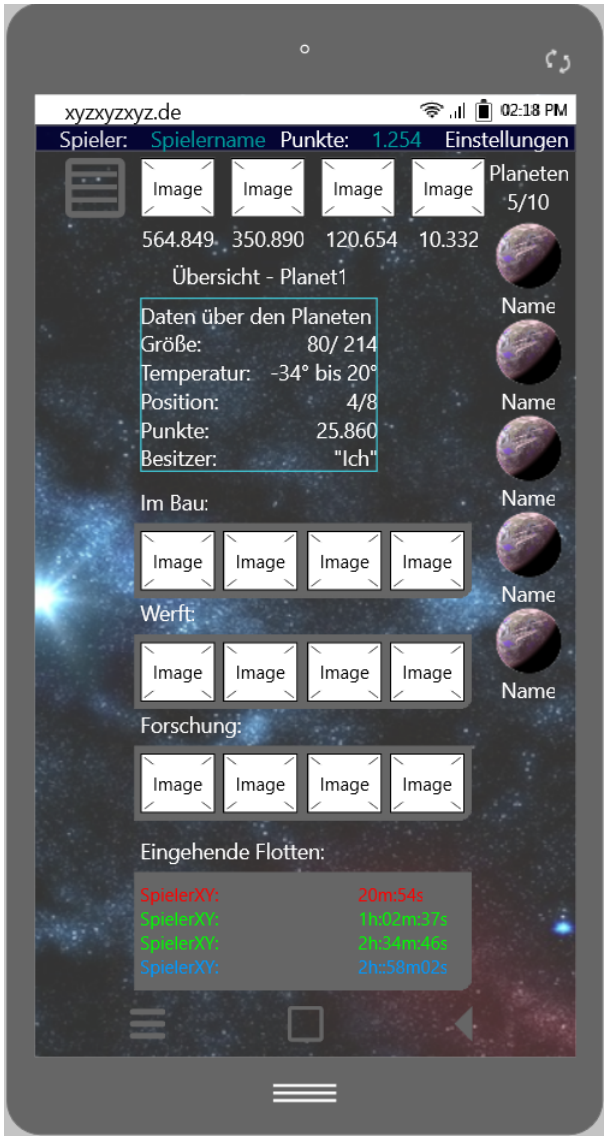

Bild 5: Mockup der Übersichtsseite

In diesem Bild sieht man die Übersichtsseite des Spiels. Am oberen Rand befindet sich eine Leiste, die den Spielernamen, Punktestand und einen Knopf für Einstellungen beinhaltet. Darunter befindet sich links oben in der Ecke ein Knopf, der das Dropdown Menü öffnet. Rechts davon befinden sich Bilder, die die verschiedenen Ressourcen repräsentieren. Diesen Bildern sind die Zahlenwerte der jeweiligen Ressource zugeordnet. Am rechten Rand befindet sich eine Übersicht über die Planeten eines Spielers, die Namen und Bilder zeigt. Mittig befindet sich die Übersicht des ausgewählten Planeten mit allen Relevanten Informationen. Darunter befinden sich mehrere Boxen, die Gebäude, Raumschiffe und Forschungen, die gerade im Bau beziehungsweise in der Erforschung sind mit deren Restzeit darstellt. Am unteren Rand befindet sich ein Informationsfeld über die gestarteten Angriffe auf den Planeten des Spielers und wann die dazugehörigen Flotten eintreffen.

Bild 6: Mockup des Dropdown Menüs

Hier sieht man das ausgeklappte Dropdown Menü, das die Navigation steuert, inklusive repräsentativer Bilder und Namen der Seite, auf der man landet.

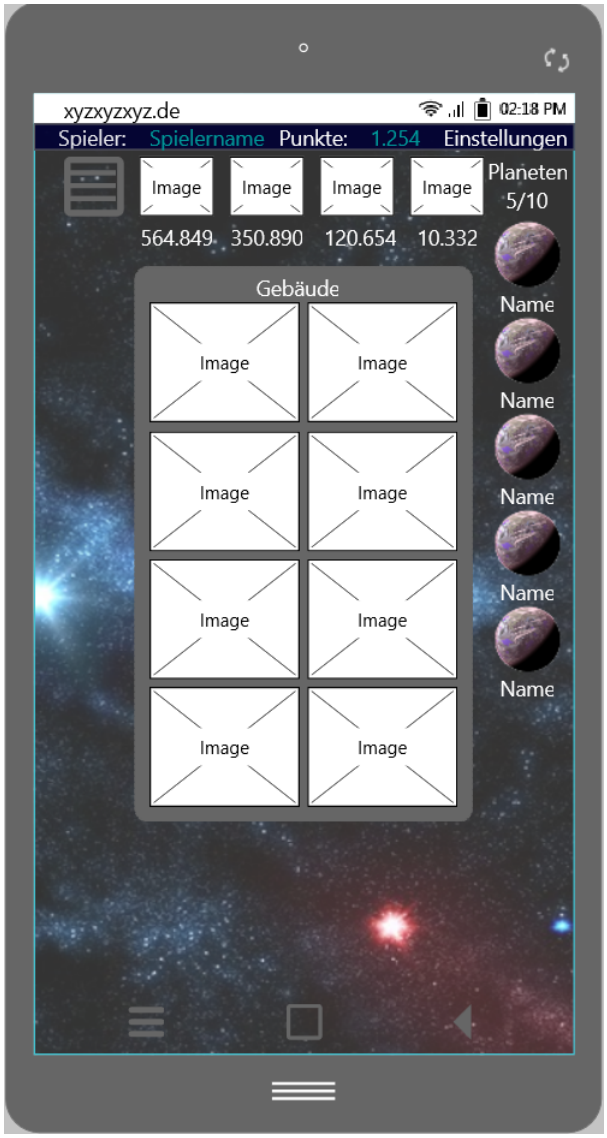

Bild 7: Mockup der Ressourcenseite

Diese Darstellung zeigt eine Übersicht der Ressourcenproduzierenden Gebäude mitsamt Bild und Level.

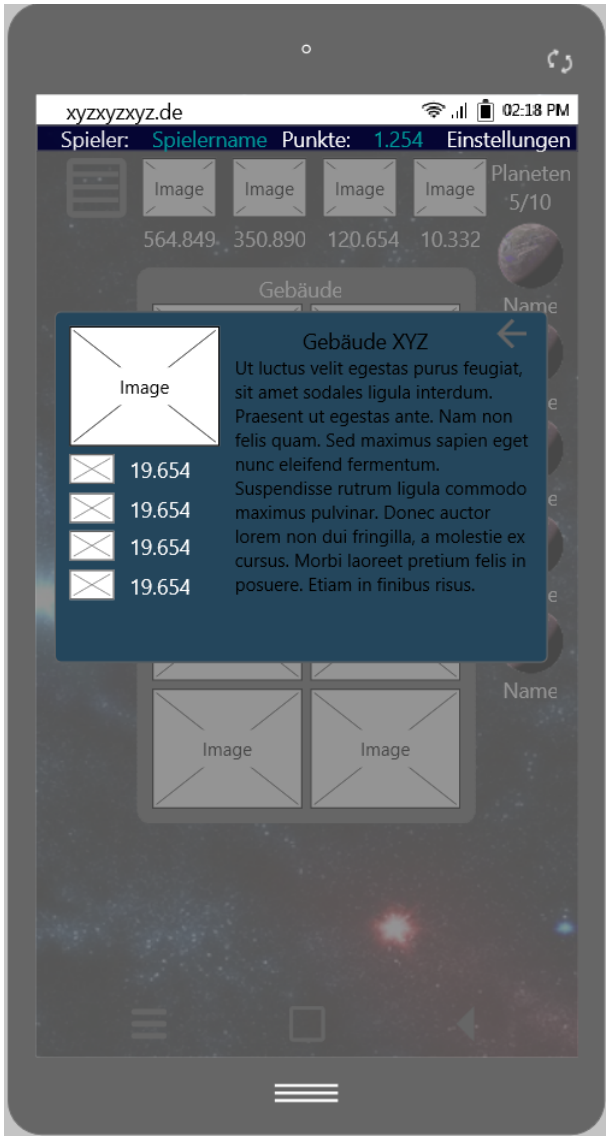

Bild 8: Mockup der Detailansicht

Wenn man im vorherigen Bild (*Bild 7*) auf einen Eintrag tippt, öffnet sich die hier dargestellte Detailansicht. Diese zeigt den Namen, ein Bild, eine Beschreibung, sowie die Kosten für die Verbesserung des Gebäudes an.

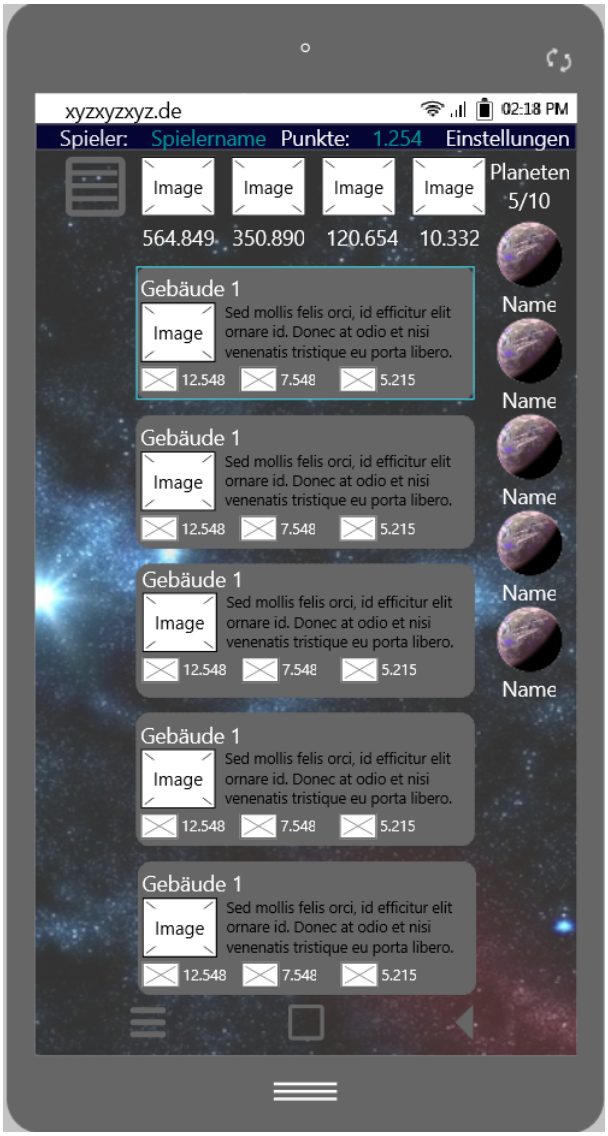

Bild 9: Mockup der Gebäudeseite

Dieses Bild zeigt die Gebäudeseite, die Informationen zu eben diesen enthält. Die einzelnen Elemente enthalten den Namen, ein Bild, eine Beschreibung, sowie die Kosten für die Verbesserung des Gebäudes.

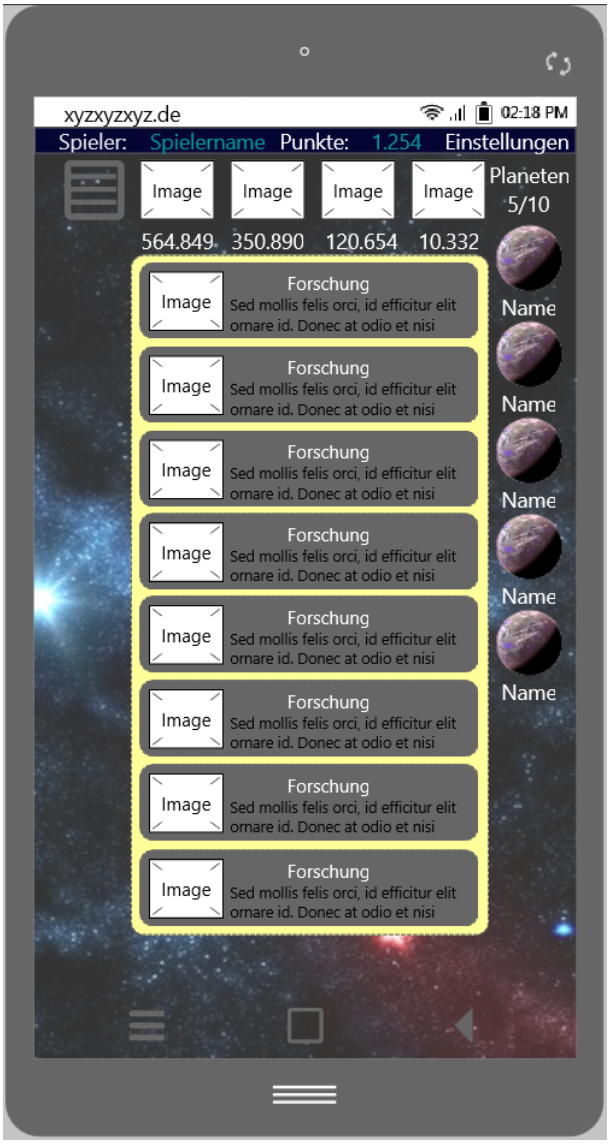

Bild 10: Mockup der Forschungsseite

Hier ist die Forschungsseite dargestellt. Man sieht ein Bild, den Namen sowie eine Beschreibung.

Die Detailansicht für die Forschung sowie die weiteren Einträge des Navigationsmenüs (Einheiten, Flotten und Galaxie) werden an dieser Stelle nicht gezeigt, da sie sich kaum von den vorhandenen Darstellungen und Erläuterungen unterscheiden würden.

Finales Konzept

Hauptsächlich wurden in den tatsächlichen Entwurf der Benutzeroberfläche Ideen aus den Mockups im Portrait Layout übernommen, da sich diese Darstellungsweise besser für Auflistungen eignet als das Landscape Layout.

Der Startbildschirm wie er in *Bild 1* zu sehen ist wurde verworfen, da es innerhalb der Applikation kein Einstellungsmenü gibt. Damit verliert der Startbildschirm seine Funktion. Nun ist der Startbildschirm eine Übersichtsseite, die der in *Bild 5* ähnlichsieht. Das dient dem Zweck das der Spieler direkt, wenn er das Spiel startet Zugriff auf nahezu alle relevanten Informationen hat. Auch das Dropdown Menü (*Bild 6*) wurde fast genauso übernommen wie hier dargestellt.

Die Planetenleiste am rechten Rand der Bilder mit Portrait Layout ist ganz weggefallen, da jeder Spieler nur ein Planet hat und nicht mehrere, wie zu Anfang festgelegt. Die Seiten zur Anzeige der Gebäude (*Bild 9*) und Forschung (*Bild 10*) sind fast eins zu eins übernommen worden. Der einzige Unterschied hier ist, dass sie jetzt die gesamte Breite einnehmen.

Die Wahl der Farbe und des Hintergrunds ist eher moderat ausgefallen. Das basiert auf der Tatsache das ein Hintergrund wie in den *Bildern 5 bis 10* zu sehen für schlechte Lesbarkeit sorgt. Daher ist der Hintergrund nun weiß und alle Elemente sind entweder Schwarz oder Blau.

Architekturbeschreibung

Aufbau

Wir haben nach dem Entwicklungsmodell Model View Controller gearbeitet, um die Logik des Programms von der Benutzeroberfläche zu trennen. Wobei wir für die Datenspeicherung also der View eine Datenbank verwenden. Der Controller wurde durch einen Server umgesetzt und die View durch den Client. Der Client ist nur dazu da die Daten anzufordern und sie anzuzeigen, die Logik ist im Controller also bei uns im Server realisiert. Nachfolgend werden diese Komponenten beschrieben und unsere Entscheidungen erklärt. Durch Einhalten des Model View Controller Entwicklungsmodell, ist das Projekt erweiterbar, so dass im Nachhinein neue Features realisiert werden können.

Client

Da wir eine mobile Applikation schreiben wollten haben wir uns für die Entwicklungsumgebung Android Studio entschieden, um den Client darzustellen. Android Studio hat eine umfangreiche Auswahl von Bibliotheken und Möglichkeiten zur Gestaltung der Benutzer-Oberfläche. Der Client ist dazu gedacht die Daten darzustellen und sendet HTTP-Requests an den Server, um Ressourcen anzufordern.

Server

Der Server verwaltet die Daten aus der Datenbank und beantwortet die Anfragen des Clients. Er berechnet die Kämpfe und die manipuliert die Daten nach Aktion des Spielers. Er hat die Aufgabe die gesamte Spiellogik zu halten und ist deshalb auch das komplexeste der drei Teilprogramme. Wir haben uns hier für die Umsetzung in Node.js entschieden. Als Programmiersprache wird Javascript verwendet. So fiel es uns besonders leicht mit JSON-Objekten umzugehen. Javascript unterstützt Methoden um aus den Datenbank Queries erhaltene Datensets, direkt in JSON umzuwandeln und diese an die HTTP-Responses anzuhängen.

Datenbank

Wir haben uns für die Nutzung einer Datenbank entschieden, um die Daten von Spielern zu speichern und um den Spielfortschritt festzuhalten. Als Datenbankmanagementsystem (DBMS) haben wir uns für MySQL entschieden. Der Grund dafür ist die einfache Handhabung und die nativen SQL-Queries. Wir haben im Umgang mit diesem DBMS schon positive Erfahrungen gemacht und konnten unser Vorwissen nutzen, um relativ schnell eine funktionsfähige Datenbank aufzubauen. Mit der MySQL Workbench konnten wir ein Entity-Relationship-Modell erstellen. Aus diesem Modell wurde dann ein Schema erstellt, das auf einem Datenbank-Server läuft.

Beispiel

Es folgt eine Erläuterung der Architektur anhand eines konkreten Beispiels aus dem Projekt. Der Aspekt, der gewählt wurde, ist die Übersicht der Gebäude.

Die Klasse `BuildingsActivity` ist im Zusammenspiel mit der Klasse `RecyclerViewAdapterBuildings` für das Anzeigen der Gebäude auf dem Endgerät des Benutzers (wird ab jetzt **Client** genannt) zuständig. Damit das Anzeigen von Daten passieren kann, muss eine HTTP-Anfrage an den **Server** gestellt werden. Dafür ist die Klasse `HttpGetRequest` zuständig. Um die Anfrage zu stellen und die Antwort des **Servers** verarbeiten zu können, muss eine Instanz erzeugt und die Methode `setUpdateListener` implementiert werden. Folgender Codeausschnitt veranschaulicht diesen Ablauf:

```
HttpGetRequest buildingsdata = new HttpGetRequest();
buildingsdata.setUpdateListener(new HttpGetRequest.OnUpdateListener() {
    @Override
    public void onUpdate(String result) {
        Gson gson = new GsonBuilder().create();
        Buildingsdata[] buildingsdata = gson.fromJson(result,
Buildingsdata[].class);
        for (int i = 0;i < 4; i++){
            mHeadlines.add(buildingsdata[i * 11 + 1].name);
            mmaterial.add(buildingsdata[i * 11 + mLevels.get(i)].material);
            melectronics.add(buildingsdata[i * 11 +
mLevels.get(i)].electronics);
            mfuel.add(buildingsdata[i * 11 + mLevels.get(i)].fuel);
            mDescriptions.add("Beschreibung: " + i);

            initImageBitmaps();
        }
    }
});
buildingsdata.execute("http://192.168.0.80:8000/?type=buildings");
```

Die for-Schleife iteriert über ein Array, in dem die Informationen über die Verschiedenen Gebäudetypen enthalten sind und fügt diese einer Recyclerview hinzu. Durch das Aufrufen der Methode `execute()` wird die Anfrage ausgeführt. Der Parameter `type=buildings` wird im **Server** verarbeitet und dadurch wird erkannt welche Daten angefordert worden sind. Die HTTP-Anfrage wird anschließend im **Server** (`server.js`) beantwortet. Der Codeausschnitt der verantwortlichen Funktion folgt:

```
function SetupServer(connection) {
    server = http.createServer(function(request, response) {
      const queryObject = url.parse(request.url,true).query;

      if (request.method == "GET") {
        queries.database(connection, queryObject)
        .then(e => {
            response.writeHead(200, {'Content-Type': 'application/json'})
            response.end(JSON.stringify(e));
        });
      }
    });
}
```

Der Aufruf von `queries.database()` führt dazu das `function database(connection, queries){...}` (`query.js`) aufgerufen wird. Hier die Darstellung des Codeausschnitts:

```
function database(connection, queries){
    const queryobject = queries;
    var string = "";
    switch(queryobject.type){
        case "buildings":
            string = "SELECT name, level, material, electronics, fuel FROM
buildings JOIN buildingsdata ON buildings.id = buildingsdata.Buildings_id"
            break;
    }
    return connection.query(string);
}
```

Somit wird das Ergebnis der Query an den **Server** weitergegeben und dieser sendet die Daten im HTTP-Body an den **Client**, wo dieser die Daten ausliest und darstellt.

Implementierung

Im folgenden Abschnitt wird der Ablauf des Projektes beschrieben. Der gesamte Verlauf wurde in die Phasen Projektplanung und Projektdurchführung eingeteilt, um die Prozesse besser nachvollziehen zu können.

Projektplanung

Zeitplanung

Es war ein Zeitraum von 13 Wochen vorgegeben. Wir haben uns für das Spiralmodell entschieden. Die Begründung kann im Abschnitt Entwicklungsprozess eingesehen werden. Eine grobe Zeiteinteilung haben wir wie folgt vorgenommen. Die Angaben sind in Prozent, weil wir in einem iterativen Modell immer wieder zwischen den Phasen wechseln.

Projektphasen	**geplante Zeit in %**
Planung	10
Entwurf	20
Implementierung	30
Zusammenführung	5
Tests	30
Dokumentation	5

Gantt-Diagramm

Das nachfolgende Gantt-Diagramm zeigt die Zeiteinteilung beginnend mit der Phase Planung, die direkt nach dem Projektstart beginnt. Hier wurden die ersten Absprachen bezüglich des Projektaufbaus gemacht. Die Entwickler einigen sich auf die Details und den Umfang des Projekts. Es werden die ersten Mockups erstellt, um die eigenen Vorstellungen zu vergleichen und zu kommentieren. Die grundlegendsten Themen werden geklärt, damit jeder weiß worauf man sich im Projekt konzentrieren muss.

Anschließend beginnt die Phase Entwurf in der die ersten Modelle entstehen, die für die Implementierung später wichtig sind. Die Modelle werden besprochen und iterativ ausgebaut. Wichtig ist hier das die Modelle nicht kurz vor Projektende fertig sind, sonst übertragen sich die Änderungen auf die anderen Teile des Projekts. Erst wenn die Entwürfe fertig sind kann mit dem Erstellen des Codes angefangen werden.

Die ersten Arbeiten zielten auf das Designen der Benutzeroberfläche in Android Studio. Die Oberfläche wurde zu Beginn des Projekts fertig gestellt und nur noch kleinere Komponenten wurden im Nachhinein ergänzt oder bearbeitet. Für das Erstellen der Benutzeroberfläche wurde ein Zeitraum von 3 Wochen eingerechnet, parallel dazu wurden Tests und Recherchen für die Entwicklung einer Client-Server Verbindung und die Benutzung einer Datenbank durchgeführt.

Nach ausreichender Recherche und Planung wurde dann die Datenbank aufgesetzt. Die Datenbank wurde in mehreren Iterationen verfeinert, bis das fertige Modell nach 2 Wochen stand. Eine Änderung in der Datenbank bedeutete immer viel Aufwand, weshalb hier auch ein durchdachter Aufbau seine Wichtigkeit zeigte.

Der Server wurde anschließend realisiert, als Bindeglied zwischen Client und Datenbank. Es musste zu diesem Zeitpunkt klar sein wir die Datenbank aussieht, damit man die entsprechenden Queries schreiben konnte.

Nachdem das Projekt jetzt vorläufig in Betrieb genommen werden konnte, haben sich die Entwickler auf die Erweiterung des Projekts und die Implementierung neuer Features konzentriert.

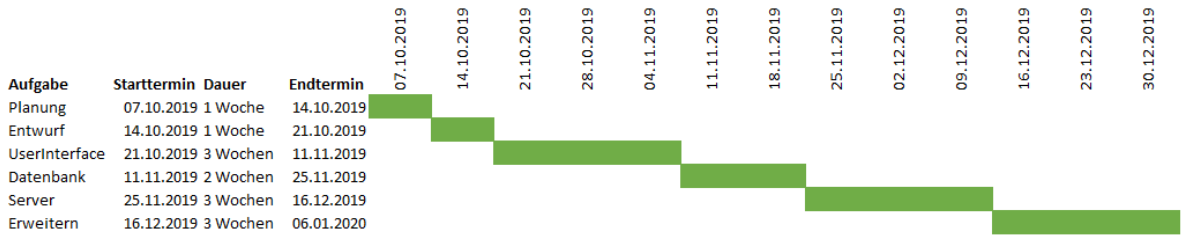

Bild 11: Gantt-Diagramm

Terminplanung

Nachfolgend ist der Meilensteinplan. Es wurden alle Meilensteine eingehalten.

Nummer	Bezeichnung	Soll Termin	Ist Termin
1	Projektstart	14.10.2019	14.10.2019
2	Projektplanung	21.10.2019	21.10.2019
3	Server Client Kommunikation hergestellt	28.10.2019	28.10.2019
4	Oberfläche entworfen	11.11.2019	11.11.2019
5	Datenbank eingebunden	25.11.2019	25.11.2019
6	erste Funktionalität benutzbar	16.12.2019	16.12.2019
7	Projekt funktionsfähig	06.01.2020	06.01.2020
8	Projektübergabe	13.01.2020	13.01.2020

Personalplanung

An dem Projekt haben zwei Entwickler gearbeitet. Die wichtigsten Entscheidungen wurden zusammengetroffen. Die Arbeiten wurden aufgeteilt und es standen für Nachfragen oder Hilfestellungen zwei Dozenten zur Verfügung.

Sachmittelplanung

Sämtliche benutzte Software ist Open-Source-Software, also kostenfrei verfügbar. Da es kein gewerbliches Projekt ist und auch nie als eines geplant war, wurden keine Investitionen vorgenommen. Von Mockplus wurde eine einwöchige Testversion genutzt. Diese Programme und Bibliotheken wurden benutzt:

- MySQL Workbench
- Node.js
- Gson
- Android Studio
- Visual Studio Code
- Mockplus
- git

Es wurden zwei Computer für die Programmierung genutzt und ein Laptop für die Präsentation.

Entwicklungsprozess

Wir haben uns für das Spiralmodell als Entwicklungsmodell geeinigt, weil das Projekt zu komplex ist, um es in einem Zug zu implementieren. Es muss immer wieder getestet werden bevor neue Funktionalitäten implementiert werden können. Ansonsten hätten lange Fehlersuchen den Entwicklungsprozess zu stark verlangsamt. Nachdem das Grundgerüst des Programms stand, wurden im gesamten Entwicklungsprozess, iterativ, neue Features implementiert. Damit das Zusammenspiel mit dem schon vorhandenen Code getestet werden konnte.

Projektdurchführung

Nachdem die vorbereitenden Maßnahmen abgeschlossen waren, kommen wir zur Projektdurchführung. In diesem Kapitel beschreiben wir den Ablauf des Projekts, gehen auf einige Arbeitsschritte und die Qualitätssicherung ein.

Entscheidungsfindung

Als erstes galt es ein geeignetes Datenbank Management System zu evaluieren. Die am weitesten verbreiteten Datenbank Management Systeme sind Oracle, MySQL und Microsoft SQL-Server, für andere sinkt mit der Popularität auch deren Unterstützung durch Dokumentation. Wir beschränken die Entscheidungsfindung auf diese drei, weil wir auch schon Vorerfahrung haben und so ein schnelles Anlaufen des Projektes gewährleisten können. Da Microsoft SQL-Server kostenpflichtig ist und uns keine Ressourcen zu Verfügung stehen, blieben noch Oracle und MySQL übrig. Durch die umfangreiche Unterstützung von MySQL in verschiedenen Sprachen und auf diversen Plattformen haben wir uns letztendlich dafür entschieden.

Für die Auswahl des Servers haben wir uns für eine lokale Variante entschieden, weil uns die Ressourcen fehlen einen Server anzuschaffen. Da wir unsere privaten Computer dafür nutzen, wollten wir bei der Gefahr eine Sicherheitslücke zu erzeugen keinen Zugriff auf unsere privaten Daten ermöglichen. Deshalb haben wir den Server selbst in Node.js erzeugt und das Verhalten implementiert.

Beschreibende Arbeitsschritte

Im Folgenden beschreiben wir einige elementare Arbeitsschritte, die notwendig waren, um dieses Projekt durchzuführen.

Aufsetzen der Datenbank

Um die Datenbank aufzusetzen, wurde ein Entity-Relationship-Model erstellt. Da es Probleme bei der Struktur der Datenbank gab, wurde die Hilfe der Dozenten in Anspruch genommen. Die Daten wurden getrennt aufgeteilt in eine Benutzerbezogene Gruppe und in Daten die ausschließlich für das Spiel gedacht sind und nicht in irgendeiner Art vom Client verändert werden können. Anschließend wird das Entity-Relationship-Model in ein SQL-Script umgewandelt, damit es in die Datenbank eingespielt werden kann. Dort erzeugt es ein Schema, in dem die Daten eingesehen und verändert werden können.

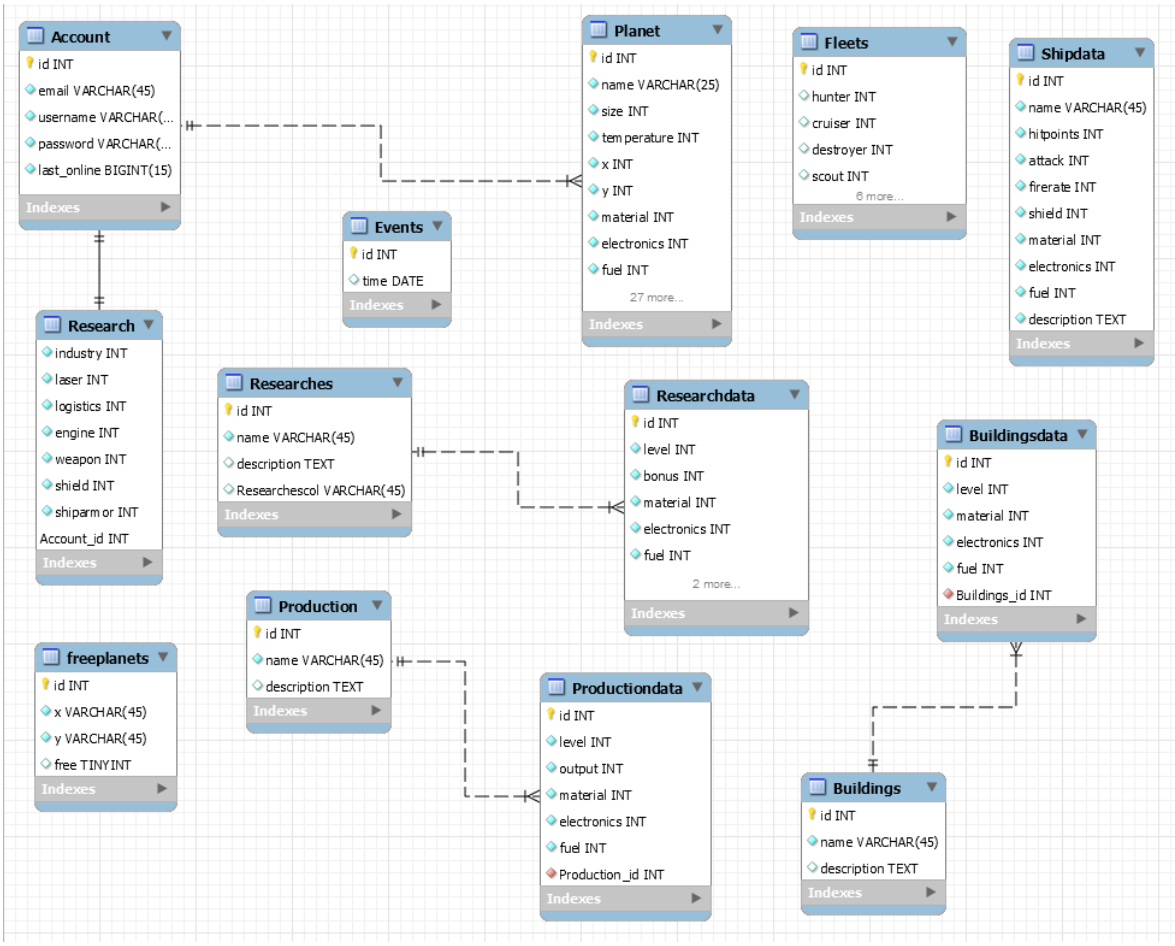

Bild 12: Entity-Relationship-Model

Implementierung des Servers

Der Server wurde in seiner ersten Version in Java geschrieben, um die Kommunikation zu testen. Durch Recherchen sind wir dann auf eine bessere Methode gestoßen und haben den Server in Node.js implementiert. Da in Javascript Funktionen asynchron ausgeführt werden, haben wir mit Promise-Objekten gearbeitet, die das Programm zwingen auf den Rückgabewert einer Funktion zu warten. Das war unabdingbar, um die Ergebnisse der SQL-Queries an die HTTP-Responses anzuhängen. Ohne dieses Verfahren wurden die HTTP-Responses ohne Inhalt im body zurück an den Client gesendet.

Umsetzung des Clients

Zunächst haben wir uns auf die Gestaltung der Oberfläche konzentriert. Da die Oberfläche größtenteils unabhängig von dem Backend entwickelt werden kann, bietet es sich an damit anzufangen. Danach haben wir in den ersten Anläufen verschiedene Kommunikationsarten und Bibliotheken für die Verwendung im Backend getestet. Die Entscheidung fiel dann auf HttpURLConnection, durch die leichte Erlernbarkeit und durch die ausführliche Dokumentation. Zum Schluss haben wir die HTTP-Anfragen als sogenannte Async Tasks implementiert. Um innerhalb dieser Klasse Zugriff auf Elemente der Benutzeroberfläche zu haben, wurde in der Klasse ein Interface angelegt, das immer, wenn die Klasse aufgerufen wird, gefüllt wird.

Qualitätssicherung

Kontinuierliche Tests

Während der Entwicklungsphase wurde die Software in Intervallen getestet. Es wurden Tests nach jedem neuen Feature gemacht und die Datenbank wurde auf Anomalien überprüft. Siehe hierzu auch das Kapitel Tests.

Sicherheit

Die Sicherheit der Nutzerdaten ist hauptsächlich durch die Architektur und das genutzte Protokoll (HTTP) gewährleistet. Die genannten Daten sind in diesem Fall E-Mail-Adressen und Passwörter. Diese Daten werden in einer Datenbank gespeichert, auf die nur der Server Zugriff hat. Der Client, also die vom Benutzer bediente Applikation hat keinerlei Möglichkeit auf die Datenbank zuzugreifen (siehe Architektur). Durch die Verwendung von HTTP ist ein gewisses Maß an Sicherheit gewährleistet. Zum Beispiel wird für die Übertragung von kritischen Daten die Post-Methode benutzt, damit die Daten im Body übermittelt werden, anstatt direkt in der URL wie bei der Get-Methode.

Ein mögliches Sicherheitsrisiko ist in der lokalen Speicherung von Accountdaten zu sehen. Wenn sich der Nutzer vor dem Schließen der Applikation nicht ausloggt, wird er automatisch beim Start wieder eingeloggt. Sollte nun jemand anderer Zugriff auf das Endgerät des Nutzers haben, könnte er auf den Account des Nutzers zugreifen. Dieses Risiko ist jedoch vernachlässigbar, da keinerlei persönliche Informationen, wie zum Beispiel Kontodaten oder Adressen benutzt werden. Außerdem kann es sein das der Nutzer die Kombination von E-Mail und Passwort auch bei anderen Diensten verwendet und somit hätte der Angreifer auch Zugriff auf diese. Zum Auslesen dieser Informationen sind jedoch Programmierkenntnisse nötig, daher wird diese Risikowahrscheinlichkeit auf niedrig eingeschätzt.

Spielkonzepte

Planeten, Ressourcen, Gebäude

Jeder Spieler kann auf seinem Planeten drei verschiedene Ressourcen abbauen, die für verschiedenste Gebäude und Verbesserungen benötigt werden. Der Abbau findet passiv statt. Das heißt, dass auch wenn man nicht aktiv im Spiel - also "Offline" - ist, werden Ressourcen erzeugt. Diese Ressourcen sind Baumaterialien, Computerchips und Treibstoff. Sie können verwendet werden, um Gebäude zu errichten, Raumschiffe und Verteidigungsanlagen zu bauen und Forschung zu betreiben. Des Weiteren können Gebäude verbessert werden, was verschiedene Boni mit sich bringt. Zum einen gibt es Gebäude, die die Produktion bestimmter Ressourcen verbessern, zum anderen gibt es Gebäude, die die maximale Kapazität der Ressourcen erhöht. Außerdem gibt es Gebäude die die Produktivität verschiedener Aspekte, wie zum Beispiel die Geschwindigkeit der Forschung oder das Bauen von Raumschiffen verbessert.

Durch die Verbesserung von Verteidigungsanlagen kann der Spieler seine Chancen vergrößern Angriffe von anderen Spielern abzuwehren.

Angriffe, Raumschiffe

Angriffe sind ein weiterer Weg, um Ressourcen zu erhalten. Der Spieler kann seine Raumschiffe zu einem anderen Planeten schicken und versuchen einem anderen Spieler Ressourcen zu stehlen. An dieser Stelle kommen die Verteidigungsanlagen, die man auf einem Planeten errichten und verbessern kann nun ins Spiel. Durch diese erhöht man seine Chance den Kampf zu gewinnen, da diese zusätzlich zu den eigenen Raumschiffen am Kampf teilnehmen.

Raumschiffe haben vier verschiedene Werte. Trefferpunkte, Schilde, Angriff und Feuerrate. Anhand dieser Werte wird ein Kampf simuliert. Die Verschiedenen Schiffe haben Stärken und Schwächen, wie zum Beispiel ein besonders hoher Schaden aber dafür eine geringere Feuerrate oder ähnliches.

Die Simulation des Kampfes basiert auf folgender Formel:

$$Angriff_{Gesamt} = Angriff_{Schiff} * Feuerrate_{Schiff} * Anzahl_{Schiff}$$
$$Schild_{Gesamt} = Schild_{Schiff} * Anzahl_{Schiff}$$
$$Trefferpunkte_{Gesamt} = Trefferpunkte_{Schiff} * Anzahl_{Schiff}$$

Diese Werte basieren auf den verschiedenen Typen von Schiffen und werden pro Typ berechnet und dann addiert. Nun wird iterativ die folgende Formel berechnet bis die Trefferpunkte einer Seite (Angreifer oder Verteidiger) auf null sinken:

$$Trefferpunkte_{Übrig} = Trefferpunkte_{Gesamt} + Schild_{Gesamt} - Angriff_{Gesamt}$$

Tests und Usability

Tests

Im Verlauf der Entwicklung wurden entsprechend des Vorgehensmodells immer wieder Testdaten in die Datenbank eingepflegt und neu implementierte Funktionen mit diesen getestet. Oft sind uns dabei Fehler aufgefallen, die wir vorher übersehen haben.

Usability

Die Benutzeroberfläche wurde so entworfen, dass sich alle Funktionen entweder als Einträge im immer sichtbaren Dropdown Menü wiederfinden oder als Buttons, an dem von der ausgeführten Aktion betroffenem Element sind. Des Weiteren wurde für alle Seiten mit ähnlichem, Listenartigen Aufbau das gleiche Muster verwendet. Einzelne, zusammengehörende Elemente sind von einem Rahmen umgeben, um sie voneinander abzugrenzen. Das Dropdown Menü beinhaltet fast ausschließlich Punkte zum Wechseln der Seite. Das sind Aktionen, die man nicht permanent benötigt und daher werden sie durch die Platzierung im Dropdown Menü versteckt. Dadurch kann Platz für die wichtigeren Aktionen geschaffen werden, die somit mehr ins Auge fallen.

Zusammenfassung

Soll-/Ist-Vergleich

Bei einer rückblickenden Betrachtung des Projektes, kann festgehalten werden, dass fast alle zuvor festgelegten Anforderungen erfüllt wurden. Außerdem wurden Aspekte erfüllt, die über die Planung hinaus gehen. Die erwarteten Ressourcen wurden eingehalten und es wurden keine weiteren Ressourcen benötigt. Das Projektziel konnte im Rahmen der gegebenen Möglichkeiten erfüllt werden. Der vorher erstellte Zeitplan und die Meilensteine konnten eingehalten werden.

Lessons learned

Im Laufe des Projekts haben wir wertvolle Erfahrungen bezüglich der Entwicklung von Applikationen sammeln können. Des Weiteren haben wir festgestellt das die Entwicklung eines Spieles eine größere Herausforderung ist, als wir Anfangs dachten. Außerdem haben wir viel neues über die Client Server Architektur, explizit die Kommunikation über HTTP sowie die Nützlichkeit von Node.js lernen können. Node.js regelt einen Großteil des handshakes von HTTP, sodass man sich nur noch um die Verarbeitung der Daten kümmern muss. Auch im Umgang mit einer Datenbank ist es von großem Nutzen, da man Queries als Strings sehr einfach ausführen kann und man ein Objekt zurückbekommt, dass man direkt versenden kann. Auch haben wir von Neuem gemerkt, dass das Dateiformat JSON die Kommunikation und Datenverarbeitung durch die Objektifizierung deutlich erleichtert.

Ausblick

Obwohl das Projektergebnis die definierten Anforderungen erfüllt hat, können in Zukunft noch neue Anforderungen und Erweiterungsvorschläge realisiert werden. Das Projekt bietet das Potential noch neue Ideen einfließen zu lassen und kann auch noch optimiert werden. Es war nicht geplant das Spiel zu veröffentlichen und es würde auch noch einige Arbeitsstunden beanspruchen.

Quellenverzeichnis

HTTPS:

https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol_Secure 10.01.2020 19:35

https://de.wikipedia.org/wiki/Transport_Layer_Security#Vor-_und_Nachteile 10.01.2020 19:35

https://tiptopsecurity.com/how-does-https-work-rsa-encryption-explained/ 10.01.2020 20:00

Spieleentwicklung:

https://de.wikipedia.org/wiki/Spiel-Engine 10.01.2020 20:30