capere vexillum

**Project Dossier**

Inhoudstabel

[Opdracht omschrijving 3](#_Toc501622147)

[Product Backlog 4](#_Toc501622148)

[Sprint 0 6](#_Toc501622149)

[Technologie 6](#_Toc501622150)

[Mobiele applicatie 6](#_Toc501622151)

[Webapplicatie 7](#_Toc501622152)

[Game server 7](#_Toc501622153)

[Database 7](#_Toc501622154)

[Sprint 1 7](#_Toc501622155)

[Sprint backlog 7](#_Toc501622156)

[Vragen tonen 8](#_Toc501622157)

[Antwoorden nakijken 8](#_Toc501622158)

[Beacon zoeken 8](#_Toc501622159)

[Beacon vangen 9](#_Toc501622160)

[Beacon cooldown instellen 9](#_Toc501622161)

[Tijd tonen 9](#_Toc501622162)

[Tijd berekenen 9](#_Toc501622163)

[Sprint 2 10](#_Toc501622164)

[Sprint backlog 10](#_Toc501622165)

[Sprint 3 11](#_Toc501622166)

[Sprint backlog 11](#_Toc501622167)

[Sprint 4 14](#_Toc501622168)

[Sprint backlog 14](#_Toc501622169)

[Sprint 5 16](#_Toc501622170)

[Sprint backlog 16](#_Toc501622171)

# Opdracht omschrijving

Het spel wordt gespeeld met 2 teams (bv. rood vs groen) die liefst even groot zijn. De bedoeling van het spel is om in een bepaalde tijd (bv. 10 minuten) een zo hoog mogelijke score te halen door vlaggen (iBeacons) te veroveren.

Deze vlaggen (iBeacons) staan verspreid over een bepaald domein (bv. de campus van de UCLL).

Om een vlag te veroveren moet je eerst een vlag vinden. Als je in de buurt van een vlag bent zal je 3 multiple choice vragen moeten beantwoorden via de mobiele app. Indien de quiz volledig juist opgelost wordt, heeft jouw team deze vlag veroverd.   
Als er een fout in de quiz zat zal de vlag enkel tijdelijk (bv. 30 sec) vergrendeld worden. Deze vergrendeling dient om te verhinderen dat het andere team meteen de vlag terug verovert en zodat je niet oneindig kan proberen om het juiste antwoord te raden.

Na een bepaald tijdsinterval (bv. iedere 5 seconden) krijgt ieder team afhankelijk van het aantal vlaggen dat ze veroverd hebben een aantal punten bij  
hun score. Het spel eindigt wanneer de tijd op is. Het team met de meeste punten wint.

### Mobiele applicatie

* Aanmaken van lobby’s
  + Instellingen van naam, wachtwoord en tijdslimiet van het spel
  + Starten van spel als iedereen in de lobby is en een team gekozen heeft
* Aansluiten bij bestaande lobby a.d.h.v naam en wachtwoord ingesteld door maker
  + Lijst van spelers in lobby
  + Keuze van team
* Detectie van vlaggen(iBeacons) eens het spel gestart is
  + Multiple choice quiz starten
* Weergave huidige score van beide teams
* Weergave van resterende tijd in het spel
* Weergave van winnaar na afloop van het spel
  + Scores winnaar en verliezer weergeven
  + Mogelijkheid om score per speler / speler met hoogste score per team te laten zien

### Webapplicatie

* Beheer van alle quizvragen die in het spel gebruikt worden
* Beveiligd met aanmeldsysteem

### Game server

* Lobby systeem
  + Meerdere spellen tegelijk
* Synchronisatie van de spellen (timer)
  + Sockets
* Voorziening van vragen aan de clients

# Product Backlog

Bij een eerste analyse van de opdrachtomschrijving tracht je de grote functionele eenheden af te splitsen. In scrum technologie noemen we dit “epic stories”.

|  |  |  |  |
| --- | --- | --- | --- |
| Epic | Story | Story points | priority |
| Mobile App | Als gebruiker wil ik een vlag kunnen veroveren door een quiz juist te beantwoorden zodat mijn team meer punten kan krijgen | 13 | 1 |
| Mobile App | Als gebruiker wil ik wanneer ik in de buurt van een vlag kom de keuze krijgen of ik aan de quiz begin zodat ik de vlag kan veroveren of niet | 8 | 2 |
| Mobile App | Als gebruiker wil ik de tijd die over blijft kunnen zien zodat ik weet wanneer het spel gedaan is | 2 | 3 |
| Mobile App | Als gebruiker wil ik het aantal vlaggen (gewichtsfactor) van mijn team zien zodat ik kan zien of we aan het winnen zijn | 2 | 4 |
| Mobile App | Als gebruiker kan ik de score van beide teams zien zodat ik weet wie aan het winnen is | 1 | 5 |
| Server | Als gebruiker wil ik dat de app in verbinding staat met de spelserver zodat het spel samen gespeeld kan worden | 13 | 6 |
| Mobile App  Server | Als gebruiker wil ik een lobby kunnen aanmaken zodat ik een team kan kiezen, een spel kan starten en een tijdslimiet kan instellen | 5 | 7 |
| Mobile App  Server | Als gebruiker kan ik mij aansluiten aan een lobby via een unieke code (naam en wachtwoord) zodat ik deel uit maak van een lobby | 5 | 8 |
| Server | Als gebruiker wil ik het spel kunnen starten als alle spelers een team hebben gekozen zodat het spel gespeeld kan worden | 5 | 9 |
| Mobile App | Als gebruiker wil ik aan het einde van het spel een scherm zien zodat ik kan zien of ik gewonnen of verloren heb | 1 | 10 |
| Mobile App | Als gebruiker wil ik dat de quiz om een vlag te veroveren uit een willekeurige selectie vragen wordt samengesteld zodat er meer variatie in het spel zit | 2 | 11 |
| Web App | Als beheerder wil ik vragen kunnen toevoegen aan de database zodat deze deel uit maken van het spel | 1 | 12 |
| Web App | Als beheerder wil ik een vraag kunnen bewerken zodat bestaande vragen aangepast kunnen worden | 2 | 13 |
| Web App | Als beheerder wil ik een vraag kunnen verwijderen zodat deze niet langer deel uit maakt van het spel | 1 | 14 |
| Web App | Als beheerder wil ik categorieën kunnen toevoegen zodat er meerdere categorieën zijn | 2 | 15 |
| Web App | Als beheerder wil ik een categorie bewerken zodat je op meerdere categorieën kan filteren | 3 | 16 |
| Web App | Als beheerder wil ik vragen kunnen filteren op categorie zodat ik een beperkt aantal vragen kan zien | 2 | 17 |
| Web App | Als beheerder wil ik mij kunnen aanmelden op de web app zodat ik toegang heb tot de beheertools | 1 | 18 |
| Web App | Als beheerder wil ik mij kunnen afmelden van de web app zodat ik niet langer toegang heb tot de beheertools | 0 | 19 |
| Mobile App | Als gebruiker kan ik kiezen om het spel te herstarten of het spel te stoppen zodat er nog een spel gespeeld kan worden of niet | 5 | 20 |
| Web App | Als beheerder wil ik een categorie kunnen verwijderen zodat deze niet meer in het systeem zit | 5 | 21 |

# Sprint 0

Volgens de scrum methode moet elke sprint een stukje werkende code opleveren met een toegevoegde waarde voor het gehele project.

Elke Sprint bevat hiervoor een combinatie van analyse, ontwerp, infrastructuur, ontwikkeling, testen en integratie.

In deze optiek verschilt sprint 0 van alle andere sprints. Sprint 0 is de voorbereiding van het gehele project en levert geen werkende code op. Sprint 0 mag echter ook nooit de intentie hebben om tegemoet te komen aan het waterval systeem.

In sprint 0 gaan we de globale aspecten van het project vast leggen. De programmeer omgeving en de te gebruiken technologieën worden vastgelegd evenals een basisstructuur voor de database en een globale opzet voor de userinterface. Belangrijk is dat je bij de design elementen zoals de basisstructuur voor de database en de globale opzet van de userinterface niet te ver in detail gaat (waterval systeem). Het moet telkens een leidraad zijn waarop je verder werkt in de volgende sprints.

Wij noemen dit sprint 0, anderen noemen dit de voorbereiding.

## Technologie

Het spel bestaat uit een mobiele applicatie en een webapplicatie. De mobiele app wordt gebruikt door de spelers van het spel en de webapp wordt gebruikt door de beheerder(s) van het spel.

### State manager

#### Probleem

Doorheen de app zijn er verschillende variabelen en constanten die door verschillende klassen gebruikt worden en steeds hetzelfde zijn. Daarnaast moeten deze ook worden opgeslagen wanneer de app naar de achtergrond gaat. Om te zorgen dat iedere klasse steeds dezelfde waardes gebruikt hebben we iets nodig.

#### Oplossing

Een oplossing voor dit probleem is een state manager.

##### Principe

Een state manager is een klasse die de status van de app bijhoudt. Het oorspronkelijke idee was om met een sleutel (een enum key) een bepaalde waarde te kunnen opvragen en/of wegschrijven.

##### Oorspronkelijk design

**public interface** IStateManager<TKey> {  
  
 **void** addStateChangedListener(OnStateChangedListener listener);  
 **void** removeStateChangedListener(OnStateChangedListener listener);  
  
 Object get(TKey key);  
 **void** set(TKey key, Object value);  
  
 **void** save();  
 **void** load();  
}

Met de methodes get en set kunnen door middel van een TKey waardes geschreven en opgevraagd worden. In de implementatie van de oorspronkelijke versie waren dit eenvoudige enum waardes van het StateManagerKey type.

De methodes save en load kunnen gebruikt worden om de opgeslagen waardes op te slaan op het toestel en om deze er achteraf ook terug vanaf te halen. Dit is nuttig om bepaalde statussen (states) te onthouden ook wanneer de app naar de achtergrond verhuist.

Tot slot zijn er de methodes add- en removeStateChangedListener. Deze worden gebruikt om te luisteren naar wanneer een state van de app verandert. Als in klasse A een state wordt geschreven en klasse B staat als listener geregistreerd dan wordt klasse B op de hoogte gebracht door de methode StateChanged van de OnStateChangedListener (die meegegeven wordt als parameter).

De implementatie van deze interface bleek moeilijker als eerst verwacht. De methodes get, set, add- en removeStateChangedListener werkten snel maar met het opslaan en laden van state waren er problemen. Het opslaan gebeurde door de map met al de objecten te serialiseren en dan als een string op te slaan. Helaas gebeurde bij het deserialiseren een fout waardoor bepaalde objecten naar een verkeerde klasse werden omgevormd. Dat gaf natuurlijk verder in de app fouten.

##### Oplossing voor de save en load methodes

Aangezien er types werden omgewisseld leek het de meest logische oplossing om een tweede map bij te houden waar de types per key in werden opgeslagen. Zo kon er geen verwarring bestaan over de klasse van een object. Helaas was dit ten koste van een deel van de gebruiksvriendelijkheid.

**private static final** Map<StateManagerKey, IState> ***KEY\_TYPE\_MAP*** = **new** HashMap<StateManagerKey, IState>() {{

put(StateManagerKey.***TEAMS***, **new** State(**new** TypeToken<List<Team>>() {

}.getType(), **true**, **false**));

put(StateManagerKey.***FLAGS***, **new** State(Flags.**class**, **true**, **true**));

put(StateManagerKey.***LOBBY\_SETTINGS***, **new** State(LobbySettings.**class**, **false**, **true**));

put(StateManagerKey.***USER\_ID***, **new** State(String.**class**, **true**, **true**));

put(StateManagerKey.***SCORE***, **new** State(**long**.**class**, **true**, **false**));

put(StateManagerKey.***LOBBY\_ID***, **new** State(String.**class**, **true**, **true**));

put(StateManagerKey.***GAME\_TIME***, **new** State(Float.**class**, **true**, **false**));

put(StateManagerKey.***GAME\_STARTED***, **new** State(**boolean**.**class**, **true**, **true**));

put(StateManagerKey.***MY\_FLAGS***, **new** State(Flags.**class**, **true**, **true**));

}};

Dit heeft voor een tijd goed gegaan maar omdat het moeilijk was om nieuwe states toe te voegen is de map verwijderd en zijn de keys aangepast. Hier hebben we geleerd da een enum een interface kan implementeren. Hierdoor kan de key dus ook het type van de waarde achter de key bevatten. Wanneer we dan iets opslaan of laden, kan van de key opgevraagd worden welk type het is.

**public interface** IStateManagerKey {  
 Class getValueClass();  
}

Omdat deze oplossing nog altijd soms onstabiel was, hebben we uiteindelijk gezorgd dat alleen bepaalde types kunnen worden opgeslagen. Eén van deze types is een ISerializable object. Het bevat een methode serialize en deserialize om het object te converteren naar een string.

**public interface** ISerializable {  
 String Serialize();  
 **void** Deserialize(String serializedObject);  
}

**public interface** IStateManager<TKey> **extends** IObservable {

**boolean** save();

**boolean** load();

**void** clear();

*/\* ------------------------- GETTERS ------------------------- \*/*

ISerializable getSerializable(TKey key);

Integer getInt(TKey key);

Long getLong(TKey key);

Float getFloat(TKey key);

String getString(TKey key);

Boolean getBoolean(TKey key);

*/\* ------------------------- SETTERS ------------------------- \*/*

**void** SetSharedPreferences(SharedPreferences sharedPreferences);

<T **extends** ISerializable> ISerializable setSerializable(TKey key, T value);

Integer setInt(TKey key, Integer value);

Long setLong(TKey key, Long value);

Float setFloat(TKey key, Float value);

String setString(TKey key, String value);

Boolean setBoolean(TKey key, Boolean value);

}

Hierdoor is bij ieder type zeker wat in de state manager wordt opgeslagen. En kunnen we dat er dus ook zeker terug uithalen.

##### Server

De server wordt gebruikt om de variabelen die voor alle verschillende spelers gemeenschappelijk zijn bij te houden. In zekere zin zijn de waardes die deze twee onderling uitwisselen allemaal states. Dat wil zeggen dat de state manager die eigenlijk ook zou moeten kunnen afhandelen. Daarnaast is het ook interessant als de rest van de app niet weet welke statussen op de server worden bijgehouden en welke lokaal. Dat maakt het ontwikkelen makkelijker. De verantwoordelijkheden worden meer gescheiden.

Daarom kwam het idee om de socket variabelen ook in de statemanager te bewaren. Hiervoor moesten er natuurlijk wel een aantal aanpassingen gebeuren aan de state manager en de keys.

**public interface** IStateManagerKey {  
 Class getValueClass();  
  
 ESocketOnKey getSocketOnKey();  
 ESocketEmitKey getSocketEmitAskKey();  
 ESocketEmitKey getSocketEmitPutKey();  
  
 Object getDefaultValue();  
 **boolean** needsToBeStored();  
}

In de IStateManagerKey bestaan er nu de methodes om op te vragen of de key naar de server moet worden doorgestuurd of de waarde op de server wordt bijgehouden. Als een object een key opvroeg met een “AskKey” dan werd de waarde aan de server afgevraagd en het object dat vroeg op de hoogte gebracht. Zo ook voor de “PutKey”. Als deze bestond werd de waarde naar de server gestuurd.

Om de verandering op de server te detecteren, luistert de state manager naar de server naar veranderingen. Als er iets verandert, brengt hij zijn luisteraars op de hoogte.

@Override  
**public void** update(Observable observable, Object args) {  
 **if** (!(args **instanceof** SocketValueChangedArgs)) {  
 **return**;  
 }  
  
 ISocketKey socketKey = ((SocketValueChangedArgs) args).getKey();  
 Object socketArgs = ((SocketValueChangedArgs) args).getArgs();  
  
 **for** (IStateManagerKey stateManagerKey : EStateManagerKey.*values*()) {  
 **if** (stateManagerKey.getSocketOnKey() == socketKey) {  
 **super**.updateState(  
 stateManagerKey,  
 getMapForClass(stateManagerKey.getValueClass()),  
 ArgsConverter.*ConvertSocketArgsToStateManagerState*(  
 stateManagerKey,  
 socketArgs  
 )  
 );  
 }  
 }  
}

Helaas heeft deze methode niet lang stand gehouden. Wanneer een key van de server werd opgevraagd, wordt de statemanager op de hoogte gebracht die dan op zijn beurt de luisteraars op de hoogte bracht. Omdat deze op de hoogte werden gebracht, vragen zij opnieuw de waarde af aan de state manager die dan op zijn beurt weer naar de server gaat. Dit zorgde voor een oneindige loop. Een moeilijke fout om te vinden maar we zijn dan maar op een andere strategie overgegaan.

###### Oplossing oneindige loop

Om de problemen van de loop op te lossen hebben we dan de GameController gemaakt. Dit is een object dat al de logica van het spel bevat.

**public interface** IGameController  
 **extends** IStateManager<IStateManagerKey> {  
  
 **void** createLobby(LobbySettings lobbySettings);  
  
 **void** leaveLobby();  
 **void** joinLobby(LobbySettings lobbySettings);  
 **void** joinTeam(String team);  
  
 **void** leaveTeam();  
  
 **void** startGame();  
 **void** askPlayers();  
 **void** askFlags();  
 **void** startGameListeners();  
 **void** setContext(AActivityWithStateManager context);  
  
 AActivityWithStateManager getContext();  
}

Wanneer één van deze methodes wordt aangeroepen, wordt de nodige logica uitgevoerd om het gewenste resultaat te bekomen. Voor sommige van deze methodes moet met de server gecommuniceerd worden, voor andere niet. Dat beslist de game controller.

### Socket

Als verbindingsmiddel tussen de server en de app gebruiken we socket.io. Om deze eenvoudig te gebruiken hebben we ook hiervoor een service geschreven. De SocketService.

**public interface** ISocketService<TKey> **extends** IObservable {  
 **void** send(TKey key, Object value);  
  
 **void** connect();  
  
 String getServerAddress();  
  
 **int** getServerPort();  
  
 **boolean** isConnected();  
}

Deze werkt op een gelijkaardige manier aan de StateManager. Er wordt met keys een waarde verzonden. Als key gebruiken we opnieuw een enum die de ISocketKey implementeerd. Deze interface bevat opnieuw de verwachte klasse achter de key, de mode (on of emit) en de identifier waarmee de server kan gecontacteerd worden.

**public interface** ISocketKey {  
 String getStringIdentifier();  
 EMode getMode();  
 Class getValueClass();  
  
 **enum** EMode {  
 ***ON***, ***EMIT*** }  
}

Wanneer de service een update krijgt van de server brengt die de luisteraars (listeners) op de hoogte.

**private void** registerListeners() {

**for** (**final** ESocketOnKey key : ESocketOnKey.*values*()) {

**if**(key.getStringIdentifier().equals(**null**))

**return**;

**socket**.on(key.getStringIdentifier(), **new** Emitter.Listener() {

@Override

**public void** call(Object... args) {

Class c = key.getValueClass();

**if** (c==**null**){

SocketValueChangedArgs socketValueChangedArgs = **new** SocketValueChangedArgs(key, args);

notifyObservers(socketValueChangedArgs);

}

**else if** (args.getClass().isAssignableFrom(c)) {

SocketValueChangedArgs socketValueChangedArgs = **new** SocketValueChangedArgs(key, args);

notifyObservers(socketValueChangedArgs);

} **else if** (!ArrayHelpers.*IsNullOrEmpty*(args) && args[0].getClass().isAssignableFrom(c)) {

SocketValueChangedArgs socketValueChangedArgs = **new** SocketValueChangedArgs(key, args[0]);

notifyObservers(socketValueChangedArgs);

}

}

});

}

}

Wanneer de methode send wordt aangeroepen worde de waarde met de identifier achter de key verzonden naar de server.

@Override

**public void** send(ISocketKey key, Object value) {

**if** (key.getMode() != ISocketKey.EMode.***EMIT***) {

**throw new** UnsupportedOperationException(**"The key "** + key.getStringIdentifier() + **" does not support sending."**);

}

Class c = key.getValueClass();

String identifier = key.getStringIdentifier();

Log.*d*(***TAG***, **"sending to server: "** + key + **" : "** + value);

**if** (c== **null**) {

**socket**.emit(identifier);

} **else if** (value.getClass().isAssignableFrom(c)) {

**socket**.emit(identifier, value);

} **else** {

String error = **"Value not of right type. (Type is "** + value.getClass() + **". Needed type is "** + c + **"."**;

**throw new** IllegalArgumentException(error);

}

}

## Database

# Sprint 1

Start datum : 05-10-2017

Aflever datum: 19-10-2017

## Sprint backlog

|  |  |  |  |
| --- | --- | --- | --- |
| User Story | Globale Requirements/ Stappenplan | Taken | Requirements/Toelichting |
| Als gebruiker wil ik een vlag kunnen veroveren door een quiz juist te beantwoorden zodat mijn team meer punten kan krijgen | * Start App * App toont spelpagina en start het spel * App zoekt naar beacons en toont de dichtstbijzijnde beacon en niet verder is dan + 1m * Als beacon is gevonden, toon een nieuwe pagina met vragen + antwoorden * Als fout antwoord, stop quiz * Als juist antwoord, toon volgende vraag * Als alle vragen beantwoord zijn, stop quiz en wijs beacon toe aan het juiste team * Bij einde quiz, toon een melding * Stel cooldown in * Toon spelpagina met de cooldown in | * Vragen tonen * Antwoorden nakijken * Beacon zoeken * Beacon vangen * Beacon cooldown instellen | * Vragen uit een locale database halen * Speciefiek aantal vragen tonen * Als vraag wordt beantwoord kijk of deze juist is * Alle vragen beantwoord, stop de quiz en geeft melding * Juist antwoord, toon volgende vraag * Fout antwoord, stopt quiz en geeft melding * Beacon tonen bij + 1m * Enkel dichtstbijzijnde beacon tonen * Beacon aan team van de speler toewijzen * Tijdens cooldown kunnen er geen vragen getoond worden bij deze beacon * Cooldown is 30 seconden * Cooldown moet in een fragment, dit wordt getoond als de beacon in cooldown is |
| Als gebruiker wil ik de tijd die over blijft kunnen zien zodat ik weet wanneer het spel gedaan is | * Start app * Spel start * App laat de overgebleven tijd zien * Timer loopt af * Spel stopt | * Tijd tonen * Tijd berekenen | * Toon de resterende tijd * Object tijd * Fragment * Timer * Eindtijd * Event tonen |

## Vragen tonen

* Toelichting
  + Vragen uit een locale database halen
  + Speciefiek aantal vragen tonen
  + Als vraag wordt beantwoord kijk of deze juist is
  + Alle vragen beantwoord, stop de quiz en geeft melding
* Code

## Antwoorden nakijken

* Toelichting
  + Juist antwoord, toon volgende vraag
  + Fout antwoord, stopt quiz en geeft melding
* Code

## Beacon zoeken

* Toelichting
  + Beacon tonen bij + 1m
  + Enkel dichtstbijzijnde beacon tonen
* Code

## Beacon vangen

* Toelichting
  + Beacon aan team van de speler toewijzen
* Code

## Beacon cooldown instellen

* Toelichting
  + Tijdens cooldown kunnen er geen vragen getoond worden bij deze beacon
  + Cooldown is 30 seconden
  + Cooldown moet in een fragment, dit wordt getoond als de beacon in cooldown is
* Code

## Tijd tonen

* Toelichting
  + Toon de resterende tijd
  + Object tijd
  + Fragment
* Code

## Tijd berekenen

* Toelichting
  + Timer
  + Eindtijd
  + Event tonen
* Code

# Sprint 2

Start datum : 19-10-2017

Aflever datum: 02-11-2017

## Sprint backlog

|  |  |  |  |
| --- | --- | --- | --- |
| User Story | Globale Requirements/ Stappenplan | Taken | Requirements/Toelichting |
| Als gebruiker wil ik wanneer ik in de buurt van een vlag kom de keuze krijgen of ik aan de quiz begin zodat ik de vlag kan veroveren of niet | * Start app * Spel start * Beacon wordt gevonden * Vraag komt of je met de quiz wilt starten * Zo ja, begin quiz * Zo nee, cooldown begint | * Statemanager * Keuze quiz starten * Achtergrond: sla op * Sluiten: verwijder alles |  |
| Als gebruiker wil ik het aantal vlaggen (gewichtsfactor) van mijn team zien zodat ik kan zien of we aan het winnen zijn | * Start app * Spel start * Bovenaan zie je het aantal vlaggen | * Vlaggen worden weergeven |  |
| Als gebruiker kan ik de score van beide teams zien zodat ik weet wie aan het winnen is | * Start app * Spel start * Bovenaan zie je de score in een progressbar | * Score van beide teams in progressbar |  |
| Als gebruiker wil ik dat de app in verbinding staat met de spelserver zodat het spel samen gespeeld kan worden | * Start app * Spel start * Spel start op de socket server * Spel start op de app * Spel wordt gesynchroniseerd | * Teams * Flags * Tijd * Connectie * Variabele op server |  |

# Sprint 3

Start datum : 02-11-2017

Aflever datum: 16-11-2017

## Sprint backlog

|  |  |  |  |
| --- | --- | --- | --- |
| User Story | Globale Requirements/ Stappenplan | Taken | Requirements/Toelichting |
| Als gebruiker wil ik een lobby kunnen aanmaken zodat ik een team kan kiezen, een spel kan starten en een tijdslimiet kan instellen | * Start app * Welkom scherm opent * Als join lobby   Geef lobbynaam, spelersnaam en lobbywachtwoord in en druk op join   * Als create lobby   Geef lobbynaam, spelersnaam, lobbywachtwoord en tijd in en druk op create | * Scherm create / join lobby * Scherm lobby aanmaken met instellingen * Lobby joinen * Speler toevoegen aan team "geen team" * Lobby online aanmaken | * Tijd (float) * Naam (string uniek) * Wachtwoord (string) * Tijd (float) * Naam (string uniek) * Wachtwoord (string) * Teams initialiseren op server |
| Als gebruiker wil ik het spel kunnen starten als alle spelers een team hebben gekozen zodat het spel gespeeld kan worden | * Start app * Creeer of join een lobby * Stel al de gegevens in * Druk op creeer of join * Join team * Als alle gebruikers een team hebben, druk op start * Spel start | * Scherm lobby | * Startknop voor lobby owner (instellingen vanuit socket => gameactivity) of "leave lobby"-knop voor speler die lobby gejoint heeft * Overige gebruikers willekeurig indelen in teams (gelijke teams) |
| Als gebruiker wil ik aan het einde van het spel een scherm zien zodat ik kan zien of ik gewonnen of verloren heb | * Start app * Creeer of join een lobby * Stel al de gegevens in * Druk op creeer of join * Join team * Als alle gebruikers een team hebben, druk op start * Spel start * Spel wordt gespeeld * Timer loopt af * Activity start met boodschap gewonnen of verloren | * Scherm einde spel | * Punten vergelijken * Boodschap tonen gewonnen of verloren * Wanneer timer afloopt |
| Als gebruiker wil ik dat de quiz om een vlag te veroveren uit een willekeurige selectie vragen wordt samen gesteld zodat er meer variatie in het spel zit |  | * Opzetten database * Functie in REST om random vragen te verkrijgen * DBHandler in mobiele app aanpassen | * Structuur: * ­Categorie * Vraag * Antwoorden * url => (server/api/v1/get   /RandomVragen  /:categorie/:aantal) => categorie="algemeen"   * statische DB door GET request vervangen |
| Als beheerder wil ik vragen kunnen toevoegen aan de database zodat deze deel uit maken van het spel |  | * Scherm dat alle vragen weer geeft * Vraag moet in de database terecht komen met | * Filterbaar * ID * Vraag * Antwoord * Categorie * Categorie * Vraag * Max 4 antwoorden |
| Als beheerder wil ik een vraag kunnen bewerken zodat bestaande vragen aangepast kunnen worden |  | * Gebruikt hetzelfde scherm als (12) * De aangepaste waarde moet opgeslagen worden |  |
| Als beheerder wil ik een vraag kunnen verwijderen zodat deze niet langer deel uit maakt van het spel |  | * Gebruikt hetzelfde scherm als (12) * De geselecteerde vraag wordt verwijderd uit de database |  |

# Sprint 4

Start datum : 16-11-2017

Aflever datum: 30-11-2017

## Sprint backlog

|  |  |  |  |
| --- | --- | --- | --- |
| User Story | Globale Requirements/ Stappenplan | Taken | Requirements/Toelichting |
| Als beheerder wil ik categorieën kunnen toevoegen zodat er meerdere categorieën zijn |  | * Scherm dat alle categorieën weergeeft * Knop: Bewerk * Knop: Verwijder * Knop (algemeen): toevoegen | * ID * Naam * toont venster * naam + knop OK en Annuleren * categorie moet in db bijkomen na OK knop |
| Als beheerder wil ik een categorie bewerken zodat je op meerdere categorieën kan filteren |  | * Knop bewerken van categorie (zie 15) toont venster * Categorie moet opgeslagen worden nadat er op de knop opslaan gedrukt is | * naam * knop opslaan * knop annuleren |
| Als beheerder wil ik vragen kunnen filteren op categorie zodat ik een beperkt aantal vragen kan zien |  |  |  |
| Als beheerder wil ik mij kunnen aanmelden op de webapp zodat ik toegang heb tot de beheertools |  | * table gebruikers in DB * pagina inloggen (= startpagina) |  |
| Als beheerder wil ik mij kunnen afmelden van de webapp zodat ik niet langer toegang heb tot de beheertools |  | * Knop afmelden op elke pagina (ga terug naar aanmeldpagina) |  |
| Als gebruiker kan ik kiezen om het spel te herstarten of het spel te stoppen zodat er nog een spel gespeeld kan worden of niet |  | * Bereken winnaar * Knop leavy lobby | * Terug naar startscherm van app |
| Als beheerder wil ik een categorie kunnen verwijderen zodat deze niet meer in het systeem zit |  | * categorie verwijderen als er op de verwijder knop gedrukt wordt bij (15) * gebruiker moet bevestigen (dialoog) * categorie moet verwijderd worden uit de database |  |

# Sprint 5

Start datum : 30-11-2017

Aflever datum: 14-12-2017

## Sprint backlog

|  |  |  |  |
| --- | --- | --- | --- |
| User Story | Globale Requirements/ Stappenplan | Taken | Requirements/Toelichting |
| Als gebruiker kan ik mij aansluiten aan een lobby via een unieke code (naam en wachtwoord) zodat ik deel uit maak van een lobby |  | * Scherm lobby joinen * Scherm lobby | * Naam + wachtwoord invullen * Joinen (socket) * 3 Teams (orange, green, no\_team) * Verlaat lobby (socket) * Team kiezen (orange / green) |