Documentatie S2

Wat komt er in de documentatie van semester 2?

* **Requirements**
* **User-Cases**
* **Contextdiagram**
* **Conceptueel model**
* **User-Stories**
* **Kwaliteitseisen**
* **UI-schetsen**
* **Testplan**
* **Architectuur**
* **Domeinmodellen**
* **Databaseontwerp**
* **Geautomatiseerd testen**
* **Acceptatietesten**
* **De SOLID principes**

**Requirements**

Requirements hebben vaak 3 elementen:

Requirement: Het gedrag wat het systeem moet vertonen

Beperkingen (op een requirement): Verdere verscherping van het observeerbare gedrag van het systeem (denk aan beperkingen aan invoer, zaken die wel of niet mogen)

Kwaliteitseisen (op een requirement): Algemene eisen aan het systeem, meestal geen verscherping van het gedrag, maar de manier waarop het gedrag zich manifesteert (denk aan het tonen van duidelijke foutmeldingen indien iets fout gaat).

**User-Cases**

Een Use-Case beschrijft “wie” met het betreffende systeem “wat” kan doen. Use-Cases bestaan uit een scenariobeschrijving. De relatie tussen de Actoren en hun Use-Cases kunnen in een Use-Casediagram worden weergegeven. Let er op dat zo'n diagram waardeloos is zonder de beschrijvingen.

**Contextdiagram**

Een software applicatie draait altijd in een bepaalde context: Een aantal externe systemen en actoren waarmee de applicatie interacteert. Als je deze context hebt, dan is een dergelijk diagram in te zetten als input voor requirements of user stories, en geeft bijvoorbeeld een overzicht van de actoren van het systeem. In het ontwerp kan een contextdiagram ook gebruikt worden als basis voor de software architectuur.

**Conceptueel model**

Naast een schets van de context is ook zinvol om vroeg in het project conceptueel model te maken om structuur aan te brengen in het domein van de applicatie. Het dient als praatplaatje met de klant en bevat geen technische details, maar gebruikt 'de taal van de klant'. Daarom neem je hierin nog geen technische beslissingen als "welk type heeft elk attribuut", "hoe sla ik dit dan op in de database" en "welk gedrag van het systeem komt in welke entiteit".

Het conceptueel model bevat dan dus ook alleen entiteiten, relaties en de belangrijkste attributen voor de applicaite.

**User-Stories**

Volgens Fowler moet een goede user-story aan de volgende zaken voldoen:

Independent: the stories can be delivered in any order

Negotiable: the details of what's in the story are co-created by the programmers and customer during development.

Valuable: the functionality is seen as valuable by the customers or users of the software.

Estimable: the programmers can come up with a reasonable estimate for building the story

Small: stories should be built in a small amount of time, usually a matter of person-days. Certainly you should be able to build several stories within one iteration.

Testable: you should be able to write tests to verify the software for this story works correctly.

Meestal hebben ze het formaat:

"Als <rol> wil ik <feature> om <waarde> te bereiken."

**Kwaliteitseisen**

Om te kunnen bepalen of een user-story klaar is, wordt over het algemeen een Defintion of Done opgesteld: een checklist met eisen waaraan voldaan moet zijn. Daarnaast is er waarschijnlijk een set acceptatiecriteria waaraan voldaan moet worden.

Onderdeel van je D.O.D. kan bijvoorbeeld zijn:

De Story moet gedekt zijn met unit-tests die slagen

Alle code is door een tweede ontwikkelaar gereviewed

Het klassendiagram is uitgebreid met details over de nieuwe Story

De gebruikershandleiding is uitgebreid met details over de nieuwe Story

De D.O.D. geldt voor alle features, maar elke User Story kan eigen kwaliteitseisen of acceptatiecriteria hebben.

**UI-schetsen**

Een UI-schets kan helpen bij het verzamelen van requirements. Door na te denken over hoe de site eruit ziet, dwing je jezelf ook wat er allemaal aan functionaliteit aanwezig moet zijn om dat voor elkaar te krijgen.

**Testplan**

Op basis van de requirements kun je vaak al een aantal scenario's verzinnen waaraan je applicatie moet voldoen. Zowel de 'happy flow', waarbij je de stappen doorloopt zoals het hoort, maar ook de foutsituaties. Voor allebei deze scenario's kun je een test verzinnen om te verifiëren dat je software werkt zoals de bedoeling is. Je kunt deze tests vastleggen in een testplan en aantonen dat je je requirements afdekt met een testmatrix.

**Architectuur**

Een architectuur bestaat meestal uit een aantal *componenten* - of *lagen* - waaronder bijvoorbeeld:

Presentatie - ook wel View, UI: De connectiviteit tussen gebruiker en de applicatie

Applicatie - ook wel Logic, Business: Het gedrag van de applicatie zelf

Persistentie - ook wel DAL: De connectiviteit tussen de applicatie en de gekozen data opslag methode

Domein - ook wel Model: De datacontainers die gebruikt worden door de applicatie

ApiWrapper: De connectiviteit tussen de applicatie en externe API's

Hardware Abstraction Layer - ook wel HAL: De connectiviteit tussen de applicatie en externe hardware

In de architectuur beslis je welke *componenten* je nodig hebt en hoe deze met elkaar verbonden zijn. Als input hiervoor gebruik je de analyse en dan met name de context. In je ontwerpdocument geef je een toelichting waarom je voor deze architectuur gekozen hebt.

**Domeinmodellen**

Een klassendiagram toont het overzicht van de klassen in een systeem, evenals de onderlinge relaties tussen deze klassen. Voor iedere klasse wordt de naam evenals de attributen en methoden weergegeven. Met deze diagrammen wordt de statische structuur van je software gemodelleerd.

**Databaseontwerp**

Uiteindelijk ga je ook dingen in een database opslaan (dit is één van de leeruitkomsten). Probeer na te denken, op basis van je domeinmodel, welke tabellen je nodig gaat hebben en hoe de relaties tussen deze tabellen zijn. Je DBMS (DataBase Mangement Systeem) dwingt deze relaties af als je op de juiste velden sleutels zet (primaire en vreemde sleutels). Ook zijn er technische beperkingen aan de datatypes die je kunt opslaan in je DBMS. Deze komen ook terug in je Databaseontwerp.

**Geautomatiseerd testen**

Een vorm van tests die je kunt maken zijn unit tests. Een unit is in deze context bijvoorbeeld een component met een bepaalde interface. Een class is hier een voorbeeld van. Alle publiek aanspreekbare zaken in een class zou je kunnen (en moeten) testen.

**Acceptatietesten**

Als je een (deel van je) applicatie hebt gebouwd, zul je, naast testen dat alles technisch werkt ook moeten controleren of het voldoet aan de gebruikerseisen en niet-functionele eisen die je hiervoor hebt opgesteld.

Afhankelijk van hoe je de requirements hebt opgesteld doe je dit als volgt:

1. Use-Cases
   * De tegenhanger van use-cases zijn test-cases. Hierbij doorloop je een bepaald scenario binnen je use-case met concrete data. De testcase beschrijft zowel de exacte invoer en de exacte verwachte resultaten. Voor alle paden door je use-case heb je een aparte test-case nodig, zowel voor de 'happy flow' waar alles goed gaat als alle uitzonderingen die je hebt beschreven.
   * De resultaten van het uitvoeren van alle test-cases, een test-run, leg je vast in een test-rapportage. Daarmee kun je bij je opdrachtgever aantonen dat je software voldoet aan alle eisen.
2. User-Stories
   * Bij een user story zijn als het goed is acceptatiecriteria gedefinieerd.
   * In de Definition of Done die je gebruikt staat waarschijnlijk dat een feature aan al die criteria moet voldoen om 'Af' te mogen zijn. Dat moet je dus aftesten. Dat kan met handmatige tests zoals hierboven beschreven.

**De SOLID principes**

Wanneer deze vijf principes samen worden toegepast zal de code eenvoudig te onderhouden en uit te breiden zijn. Specifieke implementatie-leerdoelen zijn gebaseerd op deze principes.

De letters van SOLID staan voor:

|  |  |
| --- | --- |
| **S** | Single responsibility |
| **O** | Open / Closed |
| **L** | Liskov substitution |
| **I** | Interface segregation |
| **D** | Dependency inversion |

* **Requirements**

FR-01 Gebruikers kunnen inloggen / een nieuwe inlog maken.

B-01.1 Wanneer een gebruiker inlogt met een verkeerde combinatie van gebruikersnaam en wachtwoord, dan moet daar een melding van komen.

B-01.2 Wanneer een gebruiker op de klop “Ik heb nog geen account” drukt, komt er een scherm waar de gebruikersnaam en het wachtwoord ingesteld kan worden, waarna deze wordt toegevoegd aan de database.

FR-02 Gebruikers kunnen posts maken.

B-02.1 Via hun eigen pagina kan de gebruiker in een invulveld typen.

B-02.2 De gebruiker kan de post verzenden naar de homepagina.

FR-03 Er wordt een aparte pagina aangemaakt voor het leaderboard.

B-03.1 Gebruikers kunnen de leaderboard-pagina bekijken.

B-03.2 Alleen admins kunnen het leaderboard veranderen.

B-03.3 Het leaderboard ververst regelmatig en wordt aangepast naar de meer recente data.

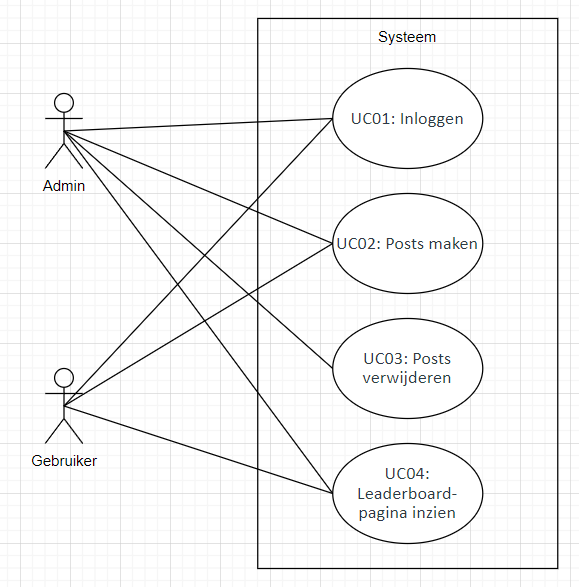
* **User-Cases**

|  |  |
| --- | --- |
| Naam | UC01: Inloggen |
| Samenvatting | Na het ingeven van de vereiste gegevens wordt dit als gebruiker aan het systeem toegevoegd. |
| Actors | Admin, Gebruiker |
| Aannamen | Geen |
| Scenario | 1. De actor bezoekt de website. 2. De website toont een inlog-scherm. 3. De actor vult de vereiste informatie in de velden in. 4. Het systeem controleert de ingevoerde gegevens en logt de actor in. |
| Uitzonderingen | 1. Niet de juiste gegevens zijn ingevuld. Toon een foutmelding. 2. Niet alle vereiste velden zijn ingevuld. Toon een foutmelding. 3. De actor is nog geen gebruiker en drukt op de knop “Ik ben nog geen gebruiker”. De actor wordt toegevoegd als gebruiker. |
| Resultaat | De gebruiker wordt ingelogd. |

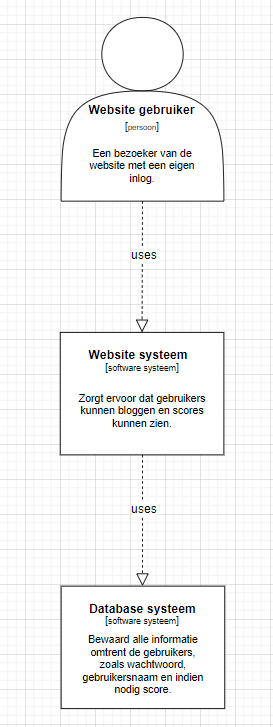
|  |  |
| --- | --- |
| Naam | UC02: Posts maken |
| Samenvatting | Na het ingeven van de vereiste gegevens wordt dit als post aan het systeem toegevoegd. |
| Actors | Admin, Gebruiker |
| Aannamen | Geen |
| Scenario | 1. De actor bezoekt de posts-pagina. 2. De actor drukt op de knop “Maak post”. 3. De actor vult de vereiste informatie in de velden in. 4. De actor drukt op de knop “Post”. 5. Het systeem controleert de ingevoerde gegevens en voegt de post toe. |
| Uitzonderingen | 1. Niet alle vereiste velden zijn ingevuld. Toon een foutmelding. 2. De actor probeert de pagina te verlaten zonder op de “Post” knop gedrukt te hebben. Geef “Weet u zeker dat u de pagina wil verlaten zonder te posten?” melding. |
| Resultaat | Er is een nieuwe post toegevoegd aan het systeem. |

|  |  |
| --- | --- |
| Naam | UC03: Posts verwijderen |
| Samenvatting | Na het ingeven van de vereiste gegevens wordt dit als post van het systeem verwijderd. |
| Actors | Admin |
| Aannamen | Geen |
| Scenario | 1. De inhoud van een post valt buiten het kader van acceptabel taalgebruik. 2. De actor selecteert de post. 3. De actor verwijderd de post. |
| Uitzonderingen | 1. De actor verwijderd de verkeerde post. De actor heeft de mogelijkheid de laatst-verwijderde post terug te halen. |
| Resultaat | De post wordt uit het systeem verwijderd. |

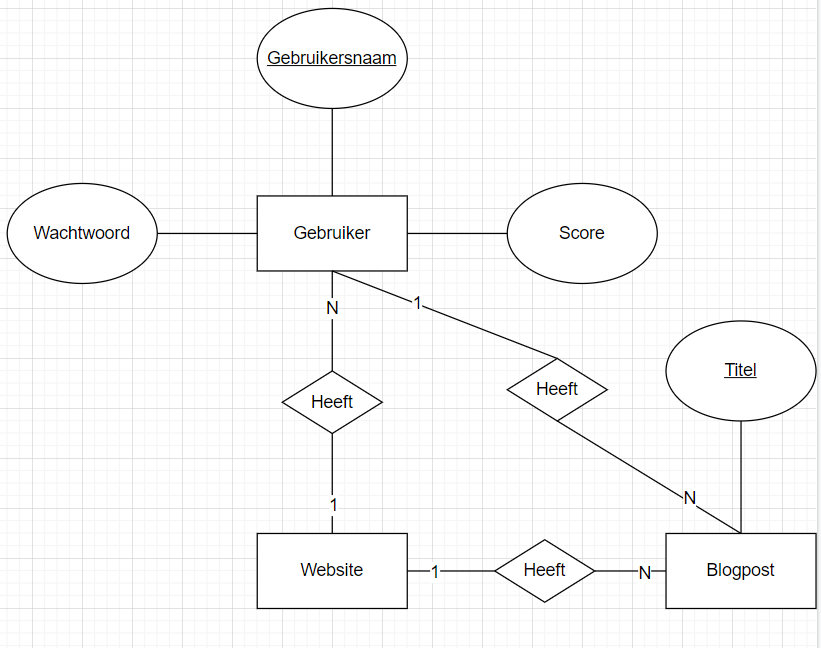
|  |  |
| --- | --- |
| Naam | UC04: Leaderboard-pagina inzien |
| Samenvatting | De leaderboard-pagina wordt gegenereerd met gegevens uit het systeem. |
| Actors | Admin, Gebruiker |
| Aannamen | Geen |
| Scenario | 1. De actor bezoekt de leaderboard-pagina. 2. De leaderboard-pagina laat de top 3 spelers zien. |
| Uitzonderingen | 1. De actor drukt op de knop “Volledige ranking”. De leaderboard-pagina laat alle spelers in het systeem zien. |
| Resultaat | Er is een duidelijk overzicht van de top 3 spelers. |



* **Contextdiagram**



* **Conceptueel model**



* **User-Stories**

[Zie requirements en user-cases.]

* **Kwaliteitseisen**

[Zie requirements en user-cases.]

Definition of done:

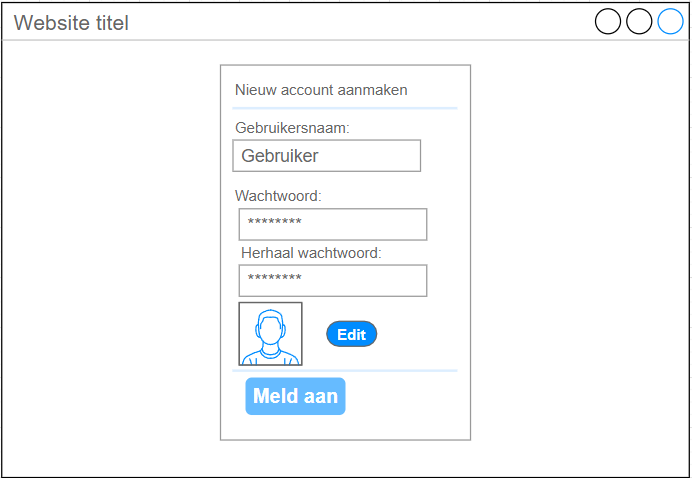
De code moet gedekt zijn met unit-tests die slagen.

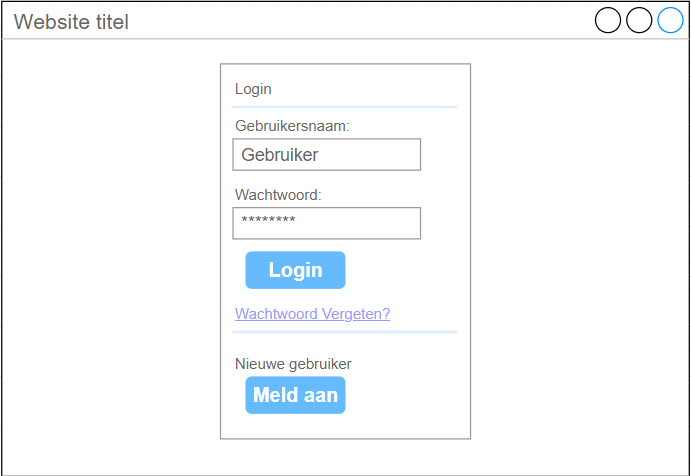
Alle code is door een tweede ontwikkelaar gereviewed.

Het klassendiagram is uitgebreid met details die overeenkomen met de code.

De gebruikershandleiding is compleet.

* **UI-schetsen**

****



Afbeelding met tekst

Automatisch gegenereerde beschrijving

Afbeelding met tekst

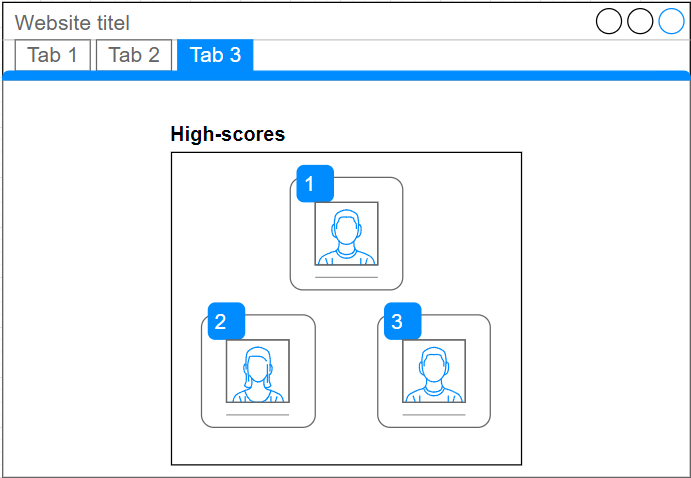
Automatisch gegenereerde beschrijving

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Afbeelding met tekst

Automatisch gegenereerde beschrijving



* **Testplan**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | User-case | Invoer | Verwachte uitvoer |
| TC01 | UC01 | Omschrijving: “Inloggen”  User ID: 2 | Er wordt ingelogd door de gebruiker. |
| TC02 | UC01 | Omschrijving: “Inloggen en aanmelden”  User ID: 2 | Er wordt een nieuw account toegevoegd door de gebruiker. |
| TC03 | UC02 | Omschrijving: “Posts maken”  User ID: 2 | Er wordt een nieuwe post toegevoegd door de gebruiker. |
| TC04 | UC03 | Omschrijving: “Posts verwijderen”  User ID: 1 | Er wordt een bestaande post verwijderd door de admin. |
| TC05 | UC04 | Omschrijving: “Leaderboard-pagina inzien”  User ID: 2 | De leaderboard-pagina wordt geladen en de gegevens opgehaald voor de gebruiker. |

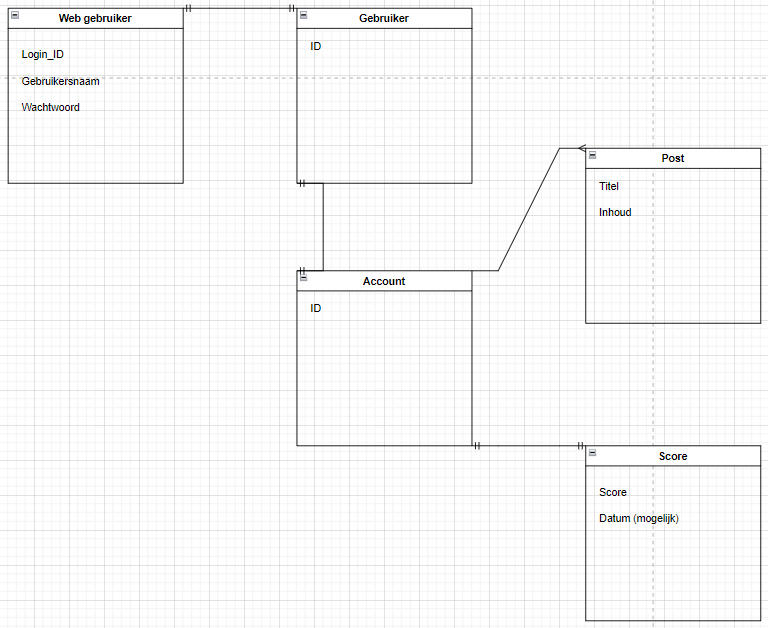
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | FR-01 | B-01.1 | B-01.2 | FR-02 | B-02.1 | B-02.2 | FR-03 | B-03.1 | B-03.2 | B-03.3 |
| TC01 | X |  |  |  |  |  |  |  |  |  |
| TC02 | X |  | X |  |  |  |  |  |  |  |
| TC03 |  |  |  | X | X | X |  |  |  |  |
| TC04 |  |  |  |  |  |  |  |  |  |  |
| TC05 |  |  |  |  |  |  |  | X |  | X |

* **Architectuur**

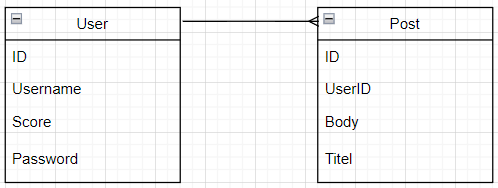
Afbeelding met tekst, binnen

Automatisch gegenereerde beschrijving

* **Domeinmodellen**



* **Databaseontwerp**



* **Geautomatiseerd testen**

…

* **Acceptatietesten**

…

* **De SOLID principes**

…

Trello bord:

Afbeelding met tekst, binnen, schermafbeelding

Automatisch gegenereerde beschrijving