De super-recepten app

Er zijn al een boel recepten-sites op internet, dus er is vast nog plaats voor één site meer, zeker als die handige foefjes heeft!

Basisvoorwaarden

1) Er moet een pagina zijn waarop je een lijst met tenminste de eerste recepten ziet, gesorteerd op aantal sterren (de recepten met de beste recensie komen het eerst).

2) Je moet de receptencollectie kunnen filteren op
- categorie (voorgerecht, hoofdgerecht, bijgerecht, toetje)
- ingrediënten (1 of meer)
- ongewenste/verboden ingrediënten (zoals aardbeien, peper, varkensvlees, mayonaise)
- dieetwensen (vlees, vis, gevogelte, vegetarisch, veganistisch, eventueel glutenvrij e.d.)
- naam (dit kan meerdere recepten opleveren als je zoekt op iets als 'salade')
… eventueel andere relevante zaken naar keuze, zoals aantal calorieën

3) Als je op een recept klikt krijg je de details van het recept te zien. Op die pagina kun je ook een rating geven.

4) Bij het overzicht is er ook een knop waarmee je naar een pagina wordt geleid waar je een nieuw recept mag aanmaken. Let op: de naam van dat recept moet uniek zijn!

Extra uitdagingen

A) voorzie het recept van een foto!

B) maak het mogelijk commentaren onderaan het recept toe te voegen

C) sta toe een gebruikersprofiel te maken met favorites, zodat je bij opgave van je gebruikersnaam en wachtwoord de zoekcriteria op jouw criteria worden gezet, en de door jou gebookmarkte recepten als bovenste worden getoond
-sta het dus ook toe recepten te bookmarken!
-mogelijk ook functionaliteit om recepten op een "ignore list" te zetten, die je in een apart window kan bekijken
-en eventueel ook 'cook again' (uitgeprobeerd en succesvol) en 'want to try'-windows.

D) Bepaalde termen (zoals blancheren) worden in recepten automatisch gehyperlinkt naar een video en/of tekst die het uitlegt; de applicatie kan een woordenboek hebben die woorden zoals blancheren automatisch daarnaar linkt.

E) Als je een recept selecteert (grotere uitdaging: of meerdere) dan kan je een boodschappenlijst downloaden met de ingrediënten

F) Sta ook 'niet-exclusieve' categorieën toe, zoals "kindvriendelijk"

G) Zorg dat recepten ook kunnen worden gesorteerd op nieuwheid (dus de nieuwste recepten eerst)

H) Maak mogelijk een persoonlijke "recept(en)" van de dag pagina aan, zodat iemand die inlogt het geplande recept/de geplande recepten op zijn/haar mobiel ziet en kan gaan koken.

De supermarkt-app

Let wel: normaal maak je voor elke hoofdcategorie gebruiker een aparte app; dus klanten en managers zouden totaal andere apps hebben, al zou de database (en mogelijk de backend) gedeeld kunnen worden. In dit voorbeeld zou je het managers-gedeelte kunnen zien als een makkelijkere manier om de database te updaten, maar je kan het natuurlijk ook uitbreiden.

Basisvoorwaarden

1) Op het openingsscherm krijg je een lijst te zien van aanbiedingen.

2) Er is ook een scherm waar je de produkten kunt zien die je het vaakste koopt (in deze app zou je daarvoor op het scherm zelf gebruikersnaam en wachtwoord moeten opgeven), voor-ingevuld met de hoeveelheid waarin je ze het meeste koopt (1x, 2x of 3x ofzo)

3) Produkten hebben
-een prijs
-een beschrijving
-1 of meer categorieën/tags (groente, fruit, huishoudelijk, “zero-calories”, “rijk aan calcium”)

4) Je moet produkten (ook meerdere malen) in een winkelmandje kunnen doen; er moet ook duidelijk zijn hoeveel er nu in je mandje zitten – en je moet ze er ook gelijk uit kunnen halen als je teveel hebt aangeklikt.

5) Je moet je winkelmandje kunnen bekijken, er produkten aan kunnen toevoegen (of eruit halen), en kunnen afrekenen (het winkelmandje wordt geleegd)

6) Je moet ervoor zorgen dat mensen produkten kunnen vinden (via naam/categorie en/of andere mogelijkheden)

---

7) Voor de management-app: de manager moet produkten met beschrijvingen en prijzen kunnen toevoegen en verwijderen

8) de manager moet prijzen kunnen wijzigen

Extra uitdagingen

A) Zorg dat items voorzien kunnen worden van een foto!

B) laat de manager aanbiedingen zetten (een bepaalde prijs voor een bepaalde periode, zeg altijd tot het einde van de week)

C) zorg dat de manager een overzicht krijgt van hoeveel produkten er van elk soort afgelopen week/maand verkocht zijn

D) zorg dat de volgorde van de produkten in “mijn lijst” zo is dat de laatst gekochte produkten bovenaan staan

E) Zorg dat de gebruiker zijn/haar lijst ook kan sorteren op categorie (groente, fruit, huishoudelijk) in plaats van op laatst gekocht, en met een druk op de knop naar de juiste categorie kan gaan (binnen de categorieen kan je overwegen de volgorde te doen op laatst gekocht of juist alfabetisch).

F) Geef de klant de mogelijkheid een dag en dagdeel af te spreken voor de bezorging...(eventueel met e-mail-bevestiging)

De OCA-app

Programmeerexamens zijn vaak niet de leukste of makkelijkste dingen om je op voor te bereiden. Maar kan je je programmeervaardigheden inzetten om juist daarmee te helpen?

Basisvoorwaarden

1. Er moet uiteraard een scherm zijn waarop je een quiz kan aanvragen: quizzes kunnen algemeen/globaal zijn, per onderwerp/trefwoord, per hoofdstuk, of individueel (de gebruiker moet inloggen met gebruikersnaam en wachtwoord, en krijgt eerder gefaalde vragen in zijn quiz). Je kunt ook aangeven hoeveel vragen je in de quiz wilt hebben (default = 10)

2. Er moet een scherm zijn waarop je een quizvraag kunt aanmaken. Bij elke vraag:
    -de vraag zelf
    -zorg dat het makkelijk is 1 of meerdere blokken code (in een ander lettertype) aan de vraag toe te voegen (hint: mogelijk geven layout-talen als MarkDown, AsciiDoc of LaTeX ideeën daarvoor).
    -opties: ja/nee vraag, multiple choice (1 goed), multiple selection (meerdere goed, aantal moet worden opgegeven)
    -bij multiple choice en multiple selection uiteraard de mogelijke antwoorden
    -aangeven wat het goede antwoord / de goede antwoorden zijn
    -liefst uitleg waarom het antwoord goed is
    -de paragraaf in het Gupta-boek waarin het wordt besproken (bijvoorbeeld 3.8.2)
    -verplicht: 1 of meer tags die het onderwerp aangeven, zoals “parameters”

3. Er moet een administratie-scherm zijn waar je tags kan toevoegen zodat er geen wildgroei komt aan tags (bijvoorbeeld “parameter”, “parameters”, “Parameter”, “PARAMETERS”). Zorg liefst voor bepaalde checks, zoals dat de nieuwe tag niet een versie van een bestaande tag is met een andere casing (“Class” vs “class”)

4. Tags moet je kunnen verwijderen zolang ze niet aan vragen zijn gekoppeld; als je een tag wilt verwijderen die aan een vraag is gekoppeld moet je de tag hernoemen.

5. En je moet de quiz zelf kunnen doen! Na afloop moet je je score kunnen zien en vragen kunnen aanklikken om je antwoord en de uitleg te kunnen bekijken.

Extra uitdagingen

A) Handig: maak import- en export-functionaliteit zodat je je vragenlijst kan exporteren en de vragen van anderen kan inlezen. Vragen die precies hetzelfde zijn (de tekst van de vraag is hetzelfde) worden uiteraard niet overgeschreven.

B) Dat je vragen als ‘favorites’ kan zetten om later nog eens over te praten met een docent of je mede-deelnemers. Ook als je denkt dat de vraag onduidelijk is of het antwoord fout (je moet ze daarna uiteraard ook kunnen ‘ont-favoriten’)

C) Je kunt ook een soort ‘spaced repetition’ inbouwen, dat als iemand iets niet weet, dat de vraag na 4 dagen weer gesteld wordt, als het dan niet lukt na nogmaals 4 dagen, anders na 4x4=16 dagen, enzovoorts (https://en.wikipedia.org/wiki/Spaced_repetition)

D) Zorg (dat als de gebruiker zijn/haar gebruikersnaam en wachtwoord geeft) hij/zij kan zien van welk hoofdstuk en/of onderwerp het percentage gefaalde vragen de afgelopen tijdsperiode (dag/week/maand) het hoogste en laagste was.

E) Eventueel: dat de gebruiker alle vragen van een bepaald onderwerp of paragraaf moet kunnen zien/quizzen.

F) Een grafiekje waarin de gebruiker het toenemen van scores over de tijd kan zien toenemen!

Algemene opmerkingen

1) Ik heb uiteraard ook persona's en scenario's gemaakt voor de projecten, maar niet ontzettend uitgebreid. Dus als je zelf op bepaalde punten een ander of mogelijk beter idee hebt, overleg het met mij (of mede-Product Owner Sem). Dat is ook een algemene regel bij programmeren: zie je fouten, onduidelijkheden of verbeterpunten in een opdracht? Overleg dat met je PO. Een vergissing in de software zetten of (te) eigenwijs zijn en een ongewenste verbetering implementeren zijn beide niet wat een product owner of een klant wil zien!

2) Dit is de eerste keer dat we deze opdrachten doen, dus we kunnen niet garanderen dat je in twee weken klaar bent. Het belangrijkste is dat de code die er is:

a) aan de Java/JavaScript(of TypeScript, wat ook van toepassing is)-stijlgidslijnen voldoet

b) begrijpelijk/duidelijk leesbaar en onderhoudbaar is

c) dat het happy path werkt en 'unhappy paths' voldoende worden afgehandeld (geen crashes!)

d) alles wat je hebt, werkt! Dus geen knoppen die niets doen of zoekbalken die niets doen; als de gebruiker het ziet, moet het werken! Vandaar dat het ook goed kan zijn een aparte master-branch te hebben, zodat er wel halve features als tussenproduct in de development-branch kunnen zijn, maar dat de masterbranch altijd klaar is voor een demo aan klanten!

Vergeet niet dat deze twee weken niet gericht zijn op het maken van een complete app waarmee ITvitae grof geld kan verdienen (dat lukt toch bijna nooit met een app waar je niet door een opdrachtgever van tevoren voor betaald wordt), maar om de *processen* van samenwerking en professioneel programmeren te leren; focusen op het 'resultaat' is in deze fase van het leerproces contraproductief, dat is pas voor als Scrum en professioneel programmeren (redelijk) automatisch en makkelijk gaan.

Checklist

[] Zet het scrumboard op (Trello, GitHub Projects, of iets anders naar keuze van je team).
[] Begin met het maken van een backlog met mogelijk een paar van de volgende items. Elk item heeft een duidelijke beschrijving/Definition of Done nodig, pokeren/estimating is ook nodig zodat het team informatie kan uitwisselen.
[] Laat iemand een repository opzetten en toegang geven aan de rest van het team.
[] Maak persona en scenarios, zet die in een directory van je repo.
[] Standaard verwacht ik React/Java met Spring Boot/Postgres. Als je dingen wilt aanpassen (Java en IETS React-achtigs zijn verplicht, Postgres en de precieze React-achtige implementatie niet) of extra libraries gebruikt (TailWind of iets dergelijks) verwacht ik dat je in de repository Architectural Decision Records daarvoor aanmaakt en minstens 3 alternatieven vindt en vergelijkt.
[] Het maken van papieren wireframes of andere schetsen om met je team het uiterlijk van de GUI kort te sluiten. Na de papierfase overweeg schetsen vast te leggen in je repo, bijvoorbeeld met tekenprogramma's zoals https://sketch.io/sketchpad/ of iets als https://online.visual-paradigm.com/ of Figma. Ik adviseer wel het simpel te houden: het is heel goed mogelijk een dag aan Figma te besteden en dat je uiteindelijk het toch anders eruit laat zien of moet laten zien, en dat 90% van de Figma-tijd eigenlijk verspild is.
[] Maak eventueel schetsen van grove structuur applicatie (3-5 klassen per A4, ook begeleidende tekst). Voeg dat ook toe in een directory van je repo.
[] Definieer de (eerste versie van de) "ubiquitous language": hoe ga je alles noemen? Zet dat document ook ergens in je repository.
[] Leg eventueel overige code-conventies en tools vast. Gebruik je bijvoorbeeld ESLint en Prettier, of andere plugins? Settings voor IntelliJ's actions on save? Voeg die documentatie ook toe in een directory van je repo.

Overige adviezen

1. verwerk het nummer van de taak in de naam van de branch

2. Laat de reviewer ook het pull request uittesten.

3. Gebruik de procedure: pull development, merge development in feature branch, los eventuele merge conflicten op, merge feature branch naar development, push dan de geupdate development.

4.Probeer taken vertikaal op te delen, niet horizontaal. Liefst doet iemand bij een taak zowel front-end als back-end/API als de database. Al mag je iemand anders wel om hulp vragen als je ergens mee vastloopt.

5. Probeer taken klein genoeg te houden (1 uur tot 1-2 dagen), anders krijg je teveel merge conflicts of wordt de reviewer overweldigd.

6. Kies een scrum-master voor de week (of gebruik random.org), spreek ook tijden af voor de daily scrum. Donderdag de 20e de eerste review, retrospective en de planning voor week 2; donderdag de 27e de tweede review, retrospective, en demo aan de andere groepen

7. Als je een dag niet kan komen, laat dat aan je teamgenoten weten (gebruik een Teams of Discord of WhatsApp-kanaal, of wat je ook daarvoor kiest). Neem nooit aan dat de studentbegeleiders (of ik) tijd hebben om je afwezigheid door te geven – als het gebeurt, is dat een bonus, maar er is zeker geen garantie dat het gebeurt!

8. Vergeet niet de documentatie te updaten als je een belangrijke keuze herziet of een nieuwe keuze maakt. Dus bijvoorbeeld als je iets invoegt/verandert in een wireframe, of data toevoegt aan een scenario, of een nieuwe grotere/discutabelere technische beslissing moet maken.

9. Hou je eigen voortgang met een taak in de gaten; na 15 of 30 minuten vastzitten met een probleem (of wat je in het team ook afspreekt als redelijke tijd) vraag iemand anders om hulp/advies. Dat moet je later op je werk ook doen...