|  |
| --- |
| Summa College |
| Mobile Application Development 2 |
|  |

|  |
| --- |
| Buenen, Stef  30-12-2022 |



Inhoudsopgave

[Inleiding 2](#_Toc132105264)

[Hoofdstuk 1: Crud 3](#_Toc132105265)

[Inleiding 3](#_Toc132105266)

[Wireframe 3](#_Toc132105267)

[Datamodel 5](#_Toc132105268)

[API 5](#_Toc132105269)

[Opdracht: Maak de app 6](#_Toc132105270)

[Conclussie 19](#_Toc132105271)

[Hoofdstuk 2: Personeel 20](#_Toc132105272)

[Inleiding 20](#_Toc132105273)

[De PersoneelsApp 20](#_Toc132105274)

[De AuthenticationServices 21](#_Toc132105275)

[De PersoneelsPage (deels) 22](#_Toc132105276)

[De LoginPage (deels) 22](#_Toc132105277)

[Hoofdstuk 3: Eindopdracht 24](#_Toc132105278)

[Inleiding 24](#_Toc132105279)

[Opdracht 24](#_Toc132105280)

[Beoordeling 25](#_Toc132105281)

[Conclusie 26](#_Toc132105282)

# Inleiding

Flutter is een relatief jong framework, maar wordt toch al zeer veel gebruikt, onder andere door Alibaba, Google, Philips en BMW. Doordat het jong is, is de technologie echt ‘state of the art’. De code van flutter is beschikbaar via GIT: <https://github.com/flutter/flutter>.

Hoewel je met flutter ook websites en desktop applicaties kunt maken, richten we ons in MAD op de mobiele applicaties.

In de module MAD 1 heb je jezelf de basisvaardigheden van Flutter eigen gemaakt. Je kunt al een zeer behoorlijke app maken. In module MAD 2 gaan we hiermee verder. Aan het eind van de module ben je in staat om een volwaardige CRUD te maken. Hierbij zorg je ervoor dat je jouw programma netjes structureert in folders voor de pages, de api-calls, models en assets 7en en (directories).

Hoofdstuk 1 behandeld alle stof om een CRUD te maken. Merk op: je maakt hierbij gebruik van een API. De API zelf maak je niet binnen de modules MAD; hiervoor zijn de modules API bedoeld.

In hoofdstuk 2 ga je zelf een Widget maken, dat je kunt gebruiken in jouw flutter applicaties. Het doel van hiervan is dat je efficienter leert te programmeren (DRY) en dat je programma’s er uiteindelijk overzichtelijker uit gaan zien.

Hoofdstuk 3 bevat dan al de eindopdracht. Je leest hier aan welke eisen jouw eindopdracht moet voldoen. De eindopdracht maak je alleen of in een groepje van maximaal 2 personen. Tijdens de oplevering van deze eindopdracht zorg je ervoor dat je alle code (ook de code die je niet zelf gemaakt hebt), kunt toelichten.

# Hoofdstuk 1: Crud

## Inleiding

In dit hoofdstuk wordt aan de hand van de champions app, behandeld hoe je binnen de module MAD 2 een CRUD maakt, gebruik makend van API’s. We gaan hier uit van een eenvoudige API zonder autorisaties. We veronderstellen dat je zelf in staat bent om autorisaties toe te passen.

We hanteren de volgende uitgangspunten:

* Elke class in een apart bestand (statefull classes w)
* Classes worden gestructureerd in directories:
  + Pages (de pages van de app).
  + Models (de classes met de data van jouw app)
  + Services (de classes die gebruik maken van de api’s)

Achtereenvolgens tref je hierna het volgende aan:

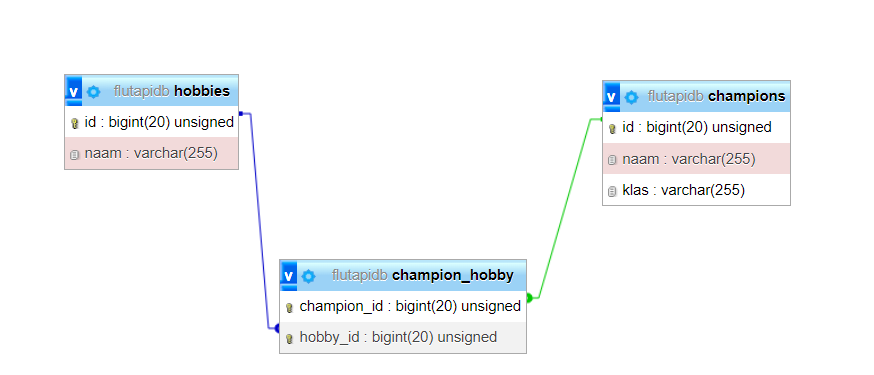
* Wireframe van de app : layout en interactie met de gebruiker
* Datamodel : de data die de app gebruikt
* Api’s die de app nodig heeft
* Opdrachten en Theorie : hier vind je uitleg over de aanpak en opdrachten om de app

te completeren.

## Wireframe

| Home | Champions | Hobbies |
| --- | --- | --- |
|  |  |  |
| Home Screen  Bevat een afbeelding met daaroverheen de tekst Champions App tegen een transparant witte achtergrond. | Index Champions  Toont van alle champions:   * Naam van de champion * Aantal hobbies   Interactie via Iconen:   * zie Champions Hobbies (koppelen van hobbies). * Zie Champions - Edit * Verwijdert Champion * zie: Champion - Create | Index Hobbies  Toont van alle hobbies:   * Naam hobby   Interactie via Iconen:   * zie Hobbies - Details * Zie Hobbies – Edit * Verwijdert Hobby * zie: Hobbies-Create |
|  |  |  |
|  | Toevoegen van een nieuwe champion. Zowel de naam als de klas zijn verplichte tekstvelden.  Interactie:   * Bewaren:   + Controle invoer.   + Toevoegen van een champion.   + Naar Index met champions. * Annuleren   + Naar Index met champions. | Toevoegen van een nieuwe hobby. De naam is een verplichte tekstveld.  Interactie:   * Bewaren:   + Controle onvoer.   + Toevoegen van hobby.   + Naar Index met hobbies. * Annuleren   + Naar Index met hobbies. |
|  |  |  |
|  | Toont naam van de champion met daarachter, tussen haakjes, de klas waarin hij is ingedeeld.  Onder de champion staat een lijst met alle hobby’s. De hobby’s die de champion beoefent zijn ‘aan’gevinkt.  Interactie:   * Via de checkBox kan de gebruiker aangeven of de champion de hobby beoefent (vinkje aan) of juist niet beoefent (vinkje uit). | Toont de naam van de hobby met daaronder een lijst van alle champions die de hobby beoefenen.  NB: Er kunnen hier dus geen champions gekoppeld worden. |
|  |  |  |
|  | Wijzigen van een champion. Zowel de naam als de klas zijn verplichte tekstvelden.  Interactie:   * Bewaren:   + Controle invoer.   + Wijzigen van champion (naam en klas).   + Naar Index met champions. * Annuleren   + Naar Index met champions. | Wijzigen van een hobby. De naam is een verplichte tekstveld.  Interactie:   * Bewaren:   + Controle invoer.   + Wijzigen van de naam van de hobby.   + Naar Index met hobbies. * Annuleren   + Naar Index met hobbies |

## Datamodel



## API

### champions

|  |  |  |  |
| --- | --- | --- | --- |
| methode | url | json parameters | json resultaat |
| get | api/champions |  | Alle champions |
| get | api/champions/{id} |  | Champion met opgegeven id |
| post | api/champions | naam, klas | gecreëerde champion |
| patch | api/ champions/{id} | naam en/of klas | champion met opgegeven id |
| delete | api/werknemers/{id} |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| methode | url | json parameters | json resultaat |
| post | api/champions/{id1}hobbies/{id2} |  | - , hobby id2 wordt door champion id1 beoefend |
| delete | api/champions/{id1}hobbies/{id2} |  | -, hobby id2 wordt niet meer door champion id1 beoefend |

### hobbbies

|  |  |  |  |
| --- | --- | --- | --- |
| methode | url | json parameters | json resultaat |
| get | api/hobbies |  | Alle hobby’s |
| get | api/ hobbies /{id} |  | Hobby met opgegeven id |
| post | api/ hobbies | naam | gecreëerde hobby |
| patch | api/ hobbies/{id} | naam | Hobby met opgegeven id |
| delete | api/ hobbies/{id} |  |  |

## Opdracht: Maak de app

### Inleiding

Je gaat nu de app maken. Hierbij hanteer je de volgende uitgangspunten:

* De data classen zet je allemaal in een directory met de naam models
* Alle api calls maak je in een aparte class in de directory services
* De pages structureer je in de directory met de naam pages.

Een deel van de code krijg je. Soms omdat de kennis al lang bekend is (dus daar hoef je jezelf nu niet opnieuw in te verdiepen); soms als voorbeeld, zodat je daarna zelf aan de slag kunt. Wordt de code als voorbeeld getoond, kijk dan ook altijd naar de verwijzingen in de officiele flutter documentatie. Deze documentatie heb je namelijk altijd bij de hand; deze reader/workshop waarschijnlijk niet.

Als je alle opdrachten af hebt, heb je een werkende app.

### Stappen

#### Opdracht: Maak aan de hand van de wireframes, het datamodel en de api’s de structuur

1. Maak een flutter project met de naam **championsapp**
2. Richt de structuur van de app als volgt in:

|  |  |
| --- | --- |
|  | Champions.jpg kun je downloaden van ItsLearning  Directory (folder) met de models die worden gebruikt  Bevat de class Champion  Bevat de class Hobby  Directory met de pages die worden gebruikt  Directory met de pages voor de tab Champions   * Create/edit/details (hobby’s) / index   Directory met de pages voor de tab Hobby’s   * Create/details/edit/index   Home page voor de tab Home  Directory met services. Alle api-calls worden hier gecodeerd. |

#### Opdracht: Vul de code voor de model classes in

1. Maak de models champion en hobby (je mag onderstaande code overnemen)

|  |
| --- |
| **import 'hobby.dart'**;  **class** Champion {  **final** int **id**;  **final** String **naam**;  **final** String **klas**;  **final** List<Hobby>? **hobbies**;   Champion({  required **this**.**id**,  required **this**.**naam**,  required **this**.**klas**,  **this**.**hobbies**,  });   **factory** Champion.fromJson(Map<String, **dynamic**> json) {  **return** Champion(  id: json[**'id'**],  naam: json[**'naam'**],  klas: json[**'klas'**],  );  }  } |
| **De lijst met hobby’s is hier een nullable.** |

|  |
| --- |
| **import 'champion.dart'**;  **class** Hobby {  **final** int **id**;  **final** String **naam**;  **final** List<Champion>? **champions**;   Hobby({  required **this**.**id**,  required **this**.**naam**,  **this**.**champions**,  });   **factory** Hobby.fromJson(Map<String, **dynamic**> json) {  **return** Hobby(  id: json[**'id'**],  naam: json[**'naam'**],  );  }   @override  bool **operator**==(Object o) {  **if** (o **is** Hobby) {  **return** o.**id** == **this**.**id**;  }  **else** {  **return false**;  }  } }  In de code hierna gebruik je de list-method *contains* om vast te stellen of een hobby al in de lijst met hobby’s van de champion zit:  **.champion.hobbies!.contains(hobby)**,  Wij willen dat *contains* 2 Hobby-objecten hetzelfde vindt, als de id’s van de Hobby-objecten aan elkaar gelijk zijn. Om die reden overriden we hier de ***operator ==***. |
| *Merk op dat de List fields nog geen waarde krijgen via de factory. De factory is nog niet af. Wil je van de factory gebruik maken, zul je dat eerst moeten doen.* |

#### Opdracht: Maak de code om de Champions via de API’s te beheren

NB: je maakt hier gebruik van het http packages. Zorg dat je het **http** package aan je flutter-project hebt toegevoegd (raadpleeg eventueel pub.dev).

1. Maak champions\_service.dart

|  |
| --- |
| **import 'dart:convert'**; **import 'package:championsapp/models/champion.dart'**; **import 'package:championsapp/models/hobby.dart'**; **import 'package:http/http.dart' as** http; |
| Bovenstaande, staan de imports die je gebruik. Deze zouden geen verrassingen mogen opleveren. |
| **class** ChampionService {  } |
| Bovenstaande staat de class ChampionService. In deze class gaan we de methods hierna toevoegen. Vergelijk de methods met de methods die je in een database class zou maken:   * getAll WPF: GetAll * post WPF: Create * put WPF: Update * delete WPF: Delete   Daarnaast zijn er nog 2 extra methods om hobbies aan een champion te koppelen, of juist te ontkoppelen:   * addHobbyToChampion * deleteHobbyFromChampion   Merk op dat elke method asynchroon wordt uitgevoerd. Het resultaat (type) van de methods is dan ook steeds een Future. |
| Future<List<Champion>> getAll() **async** {  List<Champion> champions = [];  **final** response =  **await** http.get(Uri.*parse*(**'https://flutapi.summaict.nl/api/champions'**));  **if** (response.**statusCode** != 200) {  **throw** Exception(  **'Fout bij het ophalen van alle champions (**${response.**statusCode**}**).'**);  }   **final** List<**dynamic**> data = jsonDecode(response.**body**);  **for** (int i = 0; i < data.**length**; i++) {  **final** champion = Champion(  id: data[i][**'id'**],  naam: data[i][**'naam'**],  klas: data[i][**'klas'**],  hobbies: []);   **final** List<**dynamic**> hobbies = data[i][**'hobbies'**];  **for** (int j = 0; j < hobbies.**length**; j++) {  **final** hobby = Hobby(  id: hobbies[j][**'id'**], naam: hobbies[j][**'naam'**], champions: []);  champion.**hobbies**!.add(hobby);  }  champions.add(champion);  }    **return** champions;  } |
| Opmerking: [**https://flutapi.summaict.nl/api/champions**](https://flutapi.summaict.nl/api/champions) geeft als resultaat een JSON array terug (begint met een [ en eindigt met een ]. Het overeenkomstig type in Flutter is dan een List met objecten: **List<dynamic>**.  De elementen van de lijst beginnen elk met een { en eindigen met een }. Het overeenkomstig type in Flutter is een **Map<String, dynamic>**.  Hier worden de champions gemaakt met de standaard constructor. Wil je beter voldoen aan de standaards van Flutter, dan maak je gebruik van een factory. In Mad1 heb je hiermee gewerkt. |

|  |
| --- |
| Future<Champion> post(Champion champion) **async** {  **final** response =  **await** http.post(Uri.*parse*(**'https://flutapi.summaict.nl/api/champions'**),  headers: <String, String>{  **'Content-Type'**: **'application/json'**,  },  body: jsonEncode({  **'naam'**: champion.**naam**,  **'klas'**: champion.**klas**,  }));   **if** (response.**statusCode** != 201) {  **throw** Exception(**'Het is niet gelukt om de champion toe te voegen'**);  }   **final** result = jsonDecode(response.**body**);  **return** Champion(  id: result[**'id'**], naam: result[**'naam'**], klas: result[**'klas'**]);  } |
| Zie ook: [**https://docs.flutter.dev/cookbook/networking/send-data**](https://docs.flutter.dev/cookbook/networking/send-data) |
| Future<Champion> put(int id, Champion champion) **async** {  **final** response =  **await** http.put(Uri.*parse*(**'https://flutapi.summaict.nl/api/champions/**$id**'**),  headers: <String, String>{  **'Content-Type'**: **'application/json'**,  },  body: jsonEncode({  **'id'**: champion.**id**,  **'naam'**: champion.**naam**,  **'klas'**: champion.**klas**,  }));   **if** (response.**statusCode** != 200) {  **throw** Exception(**'Het is niet gelukt om de champion toe te voegen'**);  }   **final** result = jsonDecode(response.**body**);  **return** Champion(  id: result[**'id'**], naam: result[**'naam'**], klas: result[**'klas'**]);  } |
| Zie ook: [**https://docs.flutter.dev/cookbook/networking/update-data**](https://docs.flutter.dev/cookbook/networking/update-data) |
| Future<bool> delete(int championId) **async** {  **final** response = **await** http.delete(  Uri.*parse*(**'https://flutapi.summaict.nl/api/champions/**$championId**'**));   **return** response.**statusCode** == 200;  } |
| Zie ook: [**https://docs.flutter.dev/cookbook/networking/delete-data**](https://docs.flutter.dev/cookbook/networking/delete-data) |
| Future<bool> addHobbyToChampion(int championId, int hobbyId) **async** {  **final** response =  **await** http.post(Uri.*parse*(  **'https://flutapi.summaict.nl/api/champions/**${championId}**/hobbies/**${hobbyId}**'**));   **return** response.**statusCode** == 200;  } |
| Hier gebruik je de post om een hobby aan een champion te koppelen. Dat je een post moet gebruiken, zie je in de api die je moet gebruiken: |
| Future<bool> deleteHobbyFromChampion(int championId, int hobbyId) **async** {  **final** response =  **await** http.delete(Uri.*parse*(  **'https://flutapi.summaict.nl/api/champions/**${championId}**/hobbies/**${hobbyId}**'**));   **return** response.**statusCode** == 200;  } |

#### Opdracht: Maak de code om de Hobby’s via de API’s te beheren

Deze code kun je aan de hand van de voorafgaande code zelf maken.

1. Maak hobby\_service.dart

Met bovenstaand voorbeeld, moet je nu in staat zijn om de **HobbyService** class te maken. Je hoeft alleen methods te maken voor :

* Future<List<Hobby>> getAll() (GetAll)
* Future<Hobby> post(Hobby hobby) (Create)
* Future<Hobby> put(int id, Hobby hobby) (Update)
* Future<bool> delete(int hobbyId) (Delete)

1. Maak in de class HobbyService een method *Future<List<Hobby>> getAllWithChampions()*

De method getAll is zo gemaakt dat deze enke alle Hobbies ineen List<Hobby> zet. Omdat je op het detailsscherm van een hobby ook alle champions moet tonen die de hobby uitoefenen, heb je nog een extra method nodig, namelijk een method die alle hobbies ophaalt met daarbij alle champions.



Wel moet je nog een extra method maken, die ervoor zorgt dat ook alle champions bij een hobby worden bijgelezen.

**Je hebt nu alle gegevens klaarstaan om de app verder vorm te geven:**

#### Opdracht: Maak de code voor de pages (Navigatie in main)

1. Pas main.dart aan. Zet hier de tab-navigatie in voor de app, zodat je juiste schermen worden aangestuurd.

Omdat dit reeds aanwezige kennis is, mag je onderstaande code gebruiken:

|  |
| --- |
| **import 'package:championsapp/pages/champions/champions\_index.dart'**; **import 'package:championsapp/pages/hobbies/hobbies\_index.dart'**; **import 'package:championsapp/pages/home\_page.dart'**; **import 'package:flutter/material.dart'**;  **void** main() {  runApp(**const** ChampionsApp()); }  **class** ChampionsApp **extends** StatelessWidget {  **const** ChampionsApp({Key? key}) : **super**(key: key);   @override  Widget build(BuildContext context) {  **return** MaterialApp(  home: DefaultTabController(  length: 3,  child: Scaffold(  bottomNavigationBar: Container(  color: Colors.*blue*,  child: **const** TabBar(  tabs: [  Tab(icon: Icon(Icons.*home*)),  Tab(icon: Icon(Icons.*accessibility\_new\_rounded*)),  Tab(icon: Icon(Icons.*hourglass\_top*)),  ],  ),  ),  body: **const** TabBarView(  children: [  HomePage(),  ChampionsIndex(),  HobbiesIndex(),  ],  ),  ),  ),  );  } } |
| NB: Om deze code te testen. Zorg je ervoor dat de pages HomePage, ChampionsIndex en HobbiesIndex een Widget ‘returnen’ (bijvoorbeeld een simpele scaffold). |

#### Opdracht: Maak de code voor de home page

1. Maak Home

Omdat dit reeds aanwezige kennis is, mag je onderstaande code gebruiken:

|  |
| --- |
| **import 'package:flutter/material.dart'**;  **class** HomePage **extends** StatelessWidget {  **const** HomePage({Key? key}) : **super**(key: key);   @override  Widget build(BuildContext context) {  **return** SafeArea(  child: Scaffold(  body: Stack(  children: [  Image.asset(  **'lib/assets/images/champions.jpg'**,  width: MediaQuery.*of*(context).**size**.**width**,  height: MediaQuery.*of*(context).**size**.**height**,  fit: BoxFit.**cover**,  ),  Center(  child: Container(  padding: EdgeInsets.all(50),  color: Colors.*white70*,  child: Text(  **'Champions App'**,  style: TextStyle(color: Colors.*deepPurpleAccent*, fontSize: 40),  ),  ),  ),  ],  ),  ),  );  } } |

*Zorg via de* ***pubspec.yaml*** *dat je de afbeelding champions.jpg kunt gebruiken in de Image.*

#### Opdracht: Maak de code voor de Champions pages

Deze code wordt als voorbeeld gegeven. Je dient de toelichting op de code te begrijpen en in staat te zijn deze zelfstandig toe te passen in de code die je hierna zelf gaat maken.

1. Maak de code voor Champions – Index

Je maakt hier de code voor de lijst met Champions. De naam voor de Widget heb je al bepaald (anders zou je main.dart het niet doen) en is in dit geval: **ChampionsIndex**. Zorg ervoor dat de widget de statefull versie extends, want je zult de lijst, willen refreshen, nadat je champions toegevoegd, gewijzigd en/of verwijderd hebt.

Neem de volgende code over:

|  |
| --- |
| **import 'package:championsapp/models/champion.dart'**; **import 'package:championsapp/pages/champions/champions\_create.dart'**; **import 'package:championsapp/pages/champions/champions\_edit.dart'**; **import 'package:championsapp/pages/champions/champions\_hobbies.dart'**; **import 'package:championsapp/services/champion\_service.dart'**; **import 'package:flutter/material.dart'**;  **class** ChampionsIndex **extends** StatefulWidget {  **const** ChampionsIndex({Key? key}) : **super**(key: key);   @override  State<ChampionsIndex> createState() => \_ChampionsIndexState(); }  **class** \_ChampionsIndexState **extends** State<ChampionsIndex> {  @override  Widget build(BuildContext context) {  **return** Scaffold(  appBar: AppBar(  title: **const** Text(**'Champions Index'**),  ),  body: <<CODE VOLGT HIERNA>>  floatingActionButton: <<CODE VOLGT HIER>>  } } |
| Bovenstaande code is een basis statefulWidget. In de verplichte build method is enkel een Scaffold met een AppBar opgenomen. We gaan de code hierna completeren:   * In de **body** gaan de de api resultaten zetten die we ophalen met de method getAlllvan uit de ChampionService * In de **floatingActionButton** moeten we nog iets zetten, zodat we naar de CreatePage voor een champion gaan (om een nieuwe champion toe te voegen). |
| floatingActionButton: FloatingActionButton(  child: **const** Icon(Icons.*add*),  onPressed: () => \_createChampion(context), ), |
| Allereerst hierboven de **floatingActionButton** inhoud. Om de code overzichtelijk te houden, maken we gebruik van een aantal hulp methods. In dit geval \_createChampions(context). Deze method wordt hierna toegelicht. |

|  |
| --- |
| body: FutureBuilder<List<Champion>>(  future: ChampionService().getAll(),  builder: (context, snapshot) {  **if** (snapshot.**hasError**) {  **return** Text(snapshot.**error**.toString());  }  **if** (!snapshot.**hasData**) {  **return const** CircularProgressIndicator();  }  **return** \_championsIndex(snapshot.**data**!, context);  }, ), |
| Hierboven vind je de standaard code die we gebruiken om de api resultaten te tonen. De method die alle champions ophaalt is **ChampionService().getAll()**. Het type van deze method is een **Future<List<Champion>>**.  Om de gegevens van het type Future te tonen, gebruik je een FutureBuilder. In elke FutureBuilder moet je de volgende 2 eigenschappen vullen:   * future: hier zet je de future in, in dit geval het resultaat van de method **ChampionService().getAll()**. * Builder:hierin ‘return’ je een Widget die getoond moet worden. Hierbij hebben we 2 smaken:   + Als er een fout optrad, dan wordt hier de foutcode getoond   + Als er geen fout optrad, maar er ook nog geen data was (! = not), wordt een ronddraaiende cirkel getoond   + Als er geen fout optrad en er wel data was, dan wordt de ontvangen data (de inhoud van List<Champion>) getoond. Hierbij maken we voor de overzichtelijkheid gebruik van een **\_championsIndex**. Merk op dat we als argument een uitroepteken achter data zetten; omdat data een nullable is en omdat we zeker weten dat er data in zit (hasdata = true). |
| Hierna volgt de code voor de verschillende methods. Ook hier zie je weer volop gebruik van methods om de code overzichtelijk te houden. |
| **\_championsIndex** |
| Widget \_championsIndex(List<Champion> list, BuildContext context) {  **return** ListView.builder(  itemCount: list.**length**,  itemBuilder: (context, index) {  **return** ListTile(  leading: \_championHobbies(list[index], context),  title: Row(  children: [  Expanded(child: Text(list[index].**naam**)),  \_editChampion(list[index], context),  \_deleteChampion(list[index].**id**, context)  ],  ),  subtitle: Text(**'Aantal hobbies:** ${list[index].**hobbies**?.**length** ?? 0}**'**),  );  },  );  } |
| De code voor de methods \_editChampion en \_deleteChampion volgt hierna.  Zoek zelf op wat de volgende code doet:  list[index].**hobbies**?.**length** ?? 0 |

|  |
| --- |
| Future<**void**> \_createChampion(BuildContext context) **async** {  **await** Navigator.*of*(context).push(MaterialPageRoute(  builder: (context) => **const** ChampionsCreate(),  ));  setState(() {});  } |
| Merk op: Door middel van await, wordt gewacht totdat er wordt teruggekeerd van ChampionsCreate.  Vervolgens wordt met setState() het scherm ververst (de lijst wordt opgenieuw samengesteld).  NB: Zorg dat je een default Scaffold hebt voor **ChampionsCreate** |
| Widget \_editChampion(Champion champion, BuildContext context) {  **return** GestureDetector(  onTap: () **async** {  **await** Navigator.*of*(context).push(MaterialPageRoute(  builder: (context) => ChampionsEdit(champion: champion),  ));  setState(() {});  },  child: **const** Icon(Icons.*edit*),  );  } |
| NB: Zorg dat je een default Scaffold hebt voor **ChampionsEdit** |
| Widget \_deleteChampion(int id, BuildContext context) {  **return** GestureDetector(  onTap: () **async** {  bool gelukt = **await** ChampionService().delete(id);  **if** (gelukt) {  *// refresh scherm* setState(() {});  } **else** {  **if** (context.**mounted**) {  showDialog(  context: context,  builder: (context) {  **return** AlertDialog(  title: **const** Text(**'Champions - Delete'**),  content: **const** Text(**'Verwijderen is mislukt'**),  actions: [  TextButton(  onPressed: () => Navigator.*of*(context).pop(),  child: **const** Text(**'Ok'**))  ],  );  },  );  }  }  },  child: **const** Icon(Icons.*delete\_outline*),  );  } |
| Als het verwijderen mistlukt (de champion had bijvoorbeeld nog hobbies), dan wordt hiervan een melding gegeven. |

|  |
| --- |
| \_championHobbies(Champion champion, BuildContext context) {  **return** GestureDetector(  onTap: () **async** {  **await** Navigator.*of*(context).push(MaterialPageRoute(  builder: (context) => ChampionsHobbiesPage(champion: champion),  ));  setState(() {});  },  child: **const** Icon(Icons.*account\_box\_outlined*),  );  } } |
| Toon de ChampionsHobbiesPage en refresh bij terugkeer (aantal hobbies kan gewijzigd zijn).  NB: Zorg dat je een default Scaffold hebt voor **ChampionsHobbiesPage** |

1. Maak de code voor Champions – Create

Deze code vind je hieronder. Kijk er nog eens goed naar. Als het goed is, is het allemaal gesneden koek.

|  |
| --- |
| **import 'package:championsapp/models/champion.dart'**; **import 'package:championsapp/services/champion\_service.dart'**; **import 'package:flutter/material.dart'**;  **class** ChampionsCreate **extends** StatefulWidget {  **const** ChampionsCreate({Key? key}) : **super**(key: key);   @override  State<ChampionsCreate> createState() => \_ChampionsCreateState(); }  **class** \_ChampionsCreateState **extends** State<ChampionsCreate> {  **final \_formKey** = GlobalKey<FormState>();  **final \_naamController** = TextEditingController();  **final \_klasController** = TextEditingController();   @override  **void** dispose() {  **\_naamController**.dispose();  **\_klasController**.dispose();  **super**.dispose();  }   @override  Widget build(BuildContext context) {  **return** Scaffold(  appBar: AppBar(title: Text(**'Champions - Create'**)),  body: Form(  key: **\_formKey**,  child: Column(  children: [  *// Naam* TextFormField(  controller: **\_naamController**,  textInputAction: TextInputAction.**next**,  decoration: InputDecoration(  border: OutlineInputBorder(),  labelText: **'Naam'**,  ),  validator: (value) {  **if** (value == **null** || value.**isEmpty**) {  **return 'Vul naam in'**;  }  **return null**;  },  ),   *// Klas* TextFormField(  controller: **\_klasController**,  textInputAction: TextInputAction.**next**,  decoration: InputDecoration(  border: OutlineInputBorder(),  labelText: **'Klas'**,  ),  validator: (value) {  **if** (value == **null** || value.**isEmpty**) {  **return 'Vul klas in'**;  }  **return null**;  },  ),   *// Save / Cancel* Row(  children: [  ElevatedButton(  onPressed: () **async** {  **if** (**\_formKey**.**currentState**!.validate() == **false**) {  ScaffoldMessenger.*of*(context).showSnackBar(  **const** SnackBar(content: Text(**'Verbeter de fouten'**)),  );  };  **var** champion = Champion(id: 0,  naam: **\_naamController**.**text**,  klas: **\_klasController**.**text**);  champion = **await** ChampionService().post(champion);  Navigator.*pop*(context);  },  child: Text(**'Bewaren'**),),  ElevatedButton(  onPressed: () {  Navigator.*of*(context).pop();  },  child: Text(**'Annuleren'**),),  ],  )  ],   )   )  );  } } |
| Om de gegevens van een nieuwe champion met de API toe te voegen, gebruik je de method **await ChampionService().post(champion).** Await is belangrijk, want anders wordt er direct een pop uitgevoerd, en dat levert een ongewenst resultaat. |

1. Maak de code voor Champions – Edit

Deze code vind je hieronder. Kijk er nog eens goed naar. Als het goed is, is het allemaal gesneden koek.

|  |
| --- |
| **import 'package:championsapp/models/champion.dart'**; **import 'package:championsapp/services/champion\_service.dart'**; **import 'package:flutter/material.dart'**;  **class** ChampionsEdit **extends** StatefulWidget {  **const** ChampionsEdit({Key? key, required **this**.**champion**}) : **super**(key: key);  **final** Champion **champion**;   @override  State<ChampionsEdit> createState() => \_ChampionsEditState(); }  **class** \_ChampionsEditState **extends** State<ChampionsEdit> {  **final \_formKey** = GlobalKey<FormState>();  **final \_naamController** = TextEditingController();  **final \_klasController** = TextEditingController();   @override  **void** initState() {  **\_naamController**.**text** = **widget**.**champion**.**naam**;  **\_klasController**.**text** = **widget**.**champion**.**klas**;  **super**.initState();  }   @override  **void** dispose() {  **\_naamController**.dispose();  **\_klasController**.dispose();  **super**.dispose();  }   @override  Widget build(BuildContext context) {  **return** Scaffold(  appBar: AppBar(title: Text(**'Champions - Edit'**)),  body: Form(  key: **\_formKey**,  child: Column(  children: [  *// Naam* TextFormField(  controller: **\_naamController**,  textInputAction: TextInputAction.**next**,  decoration: InputDecoration(  border: OutlineInputBorder(),  labelText: **'Naam'**,  ),  validator: (value) {  **if** (value == **null** || value.**isEmpty**) {  **return 'Vul naam in'**;  }  **return null**;  },  ),   *// Klass* TextFormField(  controller: **\_klasController**,  textInputAction: TextInputAction.**next**,  decoration: InputDecoration(  border: OutlineInputBorder(),  labelText: **'Klas'**,  ),  validator: (value) {  **if** (value == **null** || value.**isEmpty**) {  **return 'Vul klas in'**;  }  **return null**;  },  ),   *// Save / Cancel* Row(  children: [  ElevatedButton(  onPressed: () **async** {  **if** (**\_formKey**.**currentState**!.validate() == **false**) {  ScaffoldMessenger.*of*(context).showSnackBar(  **const** SnackBar(content: Text(**'Verbeter de fouten'**)),  );  }  ;  **var** champion = Champion(  id: **widget**.**champion**.**id**,  naam: **\_naamController**.**text**,  klas: **\_klasController**.**text**);  champion = **await** ChampionService()  .put(**widget**.**champion**.**id**, champion);  Navigator.*pop*(context);  },  child: Text(**'Bewaren'**),  ),  ElevatedButton(  onPressed: () {  Navigator.*of*(context).pop();  },  child: Text(**'Annuleren'**),  ),  ],  )  ],  )));  } } |
| Om de gegevens van een champion te wijzigen, maak je gebruik van de method **await ChampionService().put(widget.champion.id, champion);**  Await is belangrijk, wamt anders wordt er direct een pop uitgevoerd, en dat levert een ongewenst resultaat. |

1. Maak de code voor Champions - Hobbies

Deze code vind je hieronder. Kijk er nog eens goed naar. Het meeste zal gesneden koek zijn.

|  |
| --- |
| **import 'package:championsapp/models/champion.dart'**; **import 'package:championsapp/models/hobby.dart'**; **import 'package:championsapp/services/champion\_service.dart'**; **import 'package:championsapp/services/hobby\_service.dart'**; **import 'package:flutter/material.dart'**;  **class** ChampionsHobbiesPage **extends** StatefulWidget {  ChampionsHobbiesPage({Key? key, required **this**.**champion**}) : **super**(key: key);   **final** Champion **champion**;   @override  State<ChampionsHobbiesPage> createState() => \_ChampionsHobbiesPageState(); }  **class** \_ChampionsHobbiesPageState **extends** State<ChampionsHobbiesPage> {  @override  Widget build(BuildContext context) {  **return** Scaffold(  appBar: AppBar(title: Text(**'Champion Hobbies'**)),  body: Column(  children: [  *// Champion Data* Text(**'**${**widget**.**champion**.**naam**} **(**${**widget**.**champion**.**klas**}**)'**),   *// Hobbies* Expanded(  child: FutureBuilder<List<Hobby>>(  future: HobbyService().getAll(),  builder: (context, snapshot) {  **if** (snapshot.**hasError**) {  **return** Text(snapshot.**error**.toString());  }  **if** (snapshot.**hasData** == **false**) {  **return** CircularProgressIndicator();  }  **return** ListView.builder(  itemCount: snapshot.**data**!.**length**,  itemBuilder: (context, index) {  **return** Row(  mainAxisAlignment: MainAxisAlignment.**spaceBetween**,  children: [  Text(snapshot.**data**![index].**naam**),  **Checkbox**(  value: **widget.champion.hobbies!  .contains(snapshot.data![index])**,  onChanged: (value) {  **if** (value == **true**) {  *// 'Toevoegen van hobby;* ChampionService().addHobbyToChampion(  **widget**.**champion**.**id**, snapshot.**data**![index].**id**);  **widget**.**champion**.**hobbies**!  .add(snapshot.**data**![index]);  } **else** {  ChampionService().deleteHobbyFromChampion(  **widget**.**champion**.**id**, snapshot.**data**![index].**id**);  Hobby hobby = **widget**.**champion**.**hobbies**!.firstWhere(  (element) =>  element.**id** == snapshot.**data**![index].**id**);  **if** (hobby != **null**) {  **widget**.**champion**.**hobbies**!.remove(hobby);  }  }  setState(() {});  },  )  ],  );  },  );  },  ),  )  ],  ),  );  } } |
| De waarde van de checkbox kan hier true (aangevinkt) of false (niet aanegvinkt) zijn. Deze waarde wordt hier bepaald door  **widget.champion.hobbies!.contains(snapshot.data![index]**   * In **widget.champion.hobbies** staan alle hobbies die de champion beoefent. * In snapshot.data! staan alle hobbies. * In snapshot.data![index] staat de hobby van de betreffende regel.   Widget.champion.hobbies!.contains(snapshot.data![index]) is true, als de hobby van de betreffende regel voorkomt in de hobbies die de champion beoefent.  Kijk nog eens naar de class Hobby. Hier zie je dat is is aangegeven dat 2 hobbies aan elkaar gelijk zijn als de id’s van die hobbies aan elkaar gelijk zijn.  @override bool **operator**==(Object o) {  **if** (o **is** Hobby) {  **return** o.**id** == **this**.**id**;  }  **else** {  **return false**;  } }  Dankzij deze code werkt contains nu volgens verwachting. Zou je deze code niet opnemen, dan zullen 2 hobbies alleen maar aan elkaar gelijk zijn (en gevonden worden met een contains) als ze hetzelfde computeradres hebben. |

#### Opdracht: Maak de code voor de Hobby’s pages

1. Maak de code voor Hobbies – Index

Je hebt nu voldoende voorbeeldcode om deze page met functionaliteit zelf te maken.

1. Maak de code voor Hobbies – Create

Je hebt nu voldoende voorbeeldcode om deze page met functionaliteit zelf te maken.

1. Maak de code voor Hobbies – Edit

Je hebt nu voldoende voorbeeldcode om deze page met functionaliteit zelf te maken.

1. Maak de code voor Hobbies – Details

Je hebt nu voldoende voorbeeldcode om deze page met functionaliteit zelf te maken.

Let op: Hier zit geen functionaliteit in om champions te koppelen. Je toont hier dus ook niet alle champions. Je toont alleen alle champions die aan een hobby gekoppeld zijn.

## Conclussie

Je hebt in dit hoofdstuk een volwaardige CRUD applicatie gemaakt. Code is gestructureerd in directories en (dart)classen zodat de code goed leesbaar blijft.

# Hoofdstuk 2: Personeel

## Inleiding

Binnen de module API2 heb je geleerd hoe je met een (rotating) token de toegang tot een api kunt beveiligen. Een dergelijk token krijgt de gebruiker van je app, nadat hij zich heeft geauthenticeerd (inloggen).

In dit hoofdstuk behandelen we een manier waarop je met deze tokens kunt omgaan.

De volgende onderwerpen komen aan de orde:

* Het tonen van een Page, afhankelijk van een conditie (PersoneelsApp)

In dit geval: als de gebruiker is ingelogd, wordt de PersoneelsPage getoond. Is de gebruiker niet ingelogd, wordt de inlog page getoond.

* Registreren, inloggen en uitloggen via een static method (AuthenticationService)

De token wordt hierbij als static variabele bewaard. Hierdoor gaat de waarde van de token niet verloren, als het object om in te loggen niet meer nodig is. Ook de methods zijn static, waardoor je niet steeds een object hoeft te maken, als je ervan gebruik wilt maken.

* Gebruik maken van een call back functie (PersoneelsPage, LoginPage)

Indien de gebruiker via de login page inlogt, zal dit doorgegeven worden via een call back method van de PersoneelsApp. Logt de gebruiker zichzelf uit op de PersoneelsPage, wordt ook dit via een call back bethod van de PersoneelsApp doorgegeven.

## De PersoneelsApp

|  |
| --- |
| **import 'package:flutter/material.dart'**; **import 'package:personeels\_app/pages/login\_page.dart'**; **import 'package:personeels\_app/pages/personeels\_page.dart'**;  **void** main() {  runApp(**const** PersoneelsApp()); }  **class** PersoneelsApp **extends** StatefulWidget {  **const** PersoneelsApp({Key? key}) : **super**(key: key);   @override  State<PersoneelsApp> createState() => \_PersoneelsAppState(); }  **class** \_PersoneelsAppState **extends** State<PersoneelsApp> {  bool **\_signedIn** = **false**;   **void** setSignedIn(bool signedIn) {  setState(() {  **\_signedIn** = signedIn;  });  }   @override  Widget build(BuildContext context) {  print(**\_signedIn**);  **return** MaterialApp(  home: **\_signedIn** ? PersoneelsPage(setSignedIn: setSignedIn)  : LoginPage(setSignedIn: setSignedIn),  );  } } |
| In de boolean **\_signedIn** wordt bijgehouden of de gebruiker is ingelogd:   * True: in dat geval wordt de PersoneelsPage getoond * False: in dat geval wordt de LoginPage getoond   Met de method setSignedIn wordt de boolean **\_signedIn** van een waarde voorzien. Deze method wordt als argument aan zowel de PersoneelsPage als aan de LoginPage meegegeven, en zodat de method binnen PersoneelsPage en LogingPage gebruikt kan worden (call back method). |

## De AuthenticationServices

|  |
| --- |
| **import 'dart:convert'**; **import 'package:http/http.dart' as** http;  **class** AuthenticationServices {  **const** String *\_baseApi* = **'http://127.0.0.1:8000/api'**;  **static** String *\_bearerToken* = **''**;   *// api/register/* **static** Future<bool> *register* (String email, String password, String name) **async** {  **final** response = **await** http.post(  Uri.*parse*(**'**$*\_baseApi***/register'**),  headers: <String, String>{  **'Content-Type'**: **'application/json; charset=UTF-8'**,  },  body: jsonEncode(<String, String>{  **'name'**: name,  **'email'**: email,  **'password'**: password,  **'password\_confirmation'** : password  }),  );   **return** response.**statusCode** == 200;  }   *// api/login/* **static** Future<bool> *login* (String email, String password) **async** {  **final** response = **await** http.post(  Uri.*parse*(**'**$*\_baseApi***/login'**),  headers: <String, String>{  **'Content-Type'**: **'application/json; charset=UTF-8'**,  },  body: jsonEncode(<String, String>{  **'email'**: email,  **'password'**: password  }),  );   **if** (response.**statusCode** == 200) {  **final** result = jsonDecode(response.**body**);  print(response.**body**);  *\_bearerToken* = result[**'access\_token'**];  }   **return** response.**statusCode** == 200;  }   *// api/logout/* **static** Future<bool> *logout* () **async** {  **final** response = **await** http.post(  Uri.*parse*(**'**$*\_baseApi***/logout'**),  headers: <String, String>{  **'Content-Type'**: **'application/json; charset=UTF-8'**,  **'Authorization'** : **'Bearer** $*\_bearerToken***'** },  );   **return** response.**statusCode** == 200;  } } |
| Binnen de class AuthenticationServices zijn 2 static variabelen gebruikt:   * Een constante (moet in Dart static zijn) die de base URL van de API bevat.   NB: de base URL werkt alleen als je de app als web of windows applicatie runt. In de emulator moet je een andere URL gebruiken. Ga na op welke manier je dit dynamisch kunt implementeren.   * Een variabele die de bearer token bevat.   De methods zijn allen static. Hierdoor kun je AuthenticationServices. *login* gebruiken i.p.v. AuthenticationServices().. |

## De PersoneelsPage (deels)

|  |
| --- |
| **import 'package:flutter/material.dart'**;  **class** PersoneelsPage **extends** StatefulWidget {  **const** PersoneelsPage({Key? key, required **this**.**setSignedIn**}) : **super**(key: key);   **final void** Function(bool signedIn) **setSignedIn**;   @override  State<PersoneelsPage> createState() => \_PersoneelsPageState(); }  **class** \_PersoneelsPageState **extends** State<PersoneelsPage> {  @override  Widget build(BuildContext context) {  **return** Scaffold(  appBar: AppBar(title: Text(**'Personeels Index'**), actions: [  IconButton(  onPressed: () {  **widget**.**setSignedIn**(**false**);  },  icon: Icon(Icons.*logout*),  ),  ]),  body: ListView(  children: **const** [  *// Combobox met functies* Text(**'Zet hier een combobox met alle functies'**),  *// Lijst met employees* SizedBox(height: 10,),  Text(**'Zet hier een lijst met alle employees met deze functie'**),  ],  ),  );  } } |
| Hier zie je op welke manier je de call back als parameter in de statefull widget LoginPage declareert. De rest van de code zou gesneden koek moeten zijn.  NB: Breidt e.e.a. eventueel zelf uit zodat er een volwaardige CRUD wordt gemaakt. Zorg ervoor dat je in de services die de API’s call uitvoeren, steeds de bearer-token meegeeft en bijwerkt. Je mag zelf een oplossing bedenken hoe je dat wilt doen. |

## De LoginPage (deels)

|  |
| --- |
| **import 'package:flutter/material.dart'**; **import 'package:personeels\_app/services/authentication\_services.dart'**;  **class** LoginPage **extends** StatefulWidget {  **const** LoginPage({Key? key, required **this**.**setSignedIn**}) : **super**(key: key);   **final void** Function(bool signedIn) **setSignedIn**;   @override  State<LoginPage> createState() => \_LoginPageState(); }  **class** \_LoginPageState **extends** State<LoginPage> {  **final \_formKey** = GlobalKey<FormState>();  **final \_emailTextController** = TextEditingController();  **final \_passwordTextController** = TextEditingController();   @override  **void** dispose() {  **\_emailTextController**.dispose();  **\_passwordTextController**.dispose();  **super**.dispose();  }   @override  Widget build(BuildContext context) {  **return** Scaffold(  body: Form(  key: **\_formKey**,  child: ListView(  children: [  *// email* TextFormField(  controller: **\_emailTextController**,  textInputAction: TextInputAction.**next**,  decoration: **const** InputDecoration(  border: OutlineInputBorder(), hintText: **'email adres'**),  validator: (value) {  **if** (value == **null** || value.**isEmpty**) {  **return 'Vul emailadres in'**;  }  **return null**;  },  ),  *// password* TextFormField(  controller: **\_passwordTextController**,  textInputAction: TextInputAction.**next**,  obscureText: **true**,  decoration: **const** InputDecoration(  border: OutlineInputBorder(), hintText: **'password'**),  validator: (value) {  **if** (value == **null** || value.**isEmpty**) {  **return 'Vul password in'**;  }  **return null**;  },  ),  *// submit button* ElevatedButton(  onPressed: () **async** {  **if** (**\_formKey**.**currentState**!.validate()) {  **try** {  **final** result = **await** AuthenticationServices.*login*(  **\_emailTextController**.**text**,  **\_passwordTextController**.**text**,  );  **widget**.**setSignedIn**(result);  } **catch** (e) {  **widget**.**setSignedIn**(**false**);  }  }  },  child: **const** Text(**'Inloggen'**),  ),  ],  ),  ),  );  } } |
| Hier zie je op welke manier je de call back als parameter in de statefull widget LoginPage declareert. De rest van de code zou gesneden koek moeten zijn.  NB: Breidt e.e.a. eventueel zelf uit, zodat de gebruiker zich ook kan registreren. |

# Hoofdstuk 3: Eindopdracht

## Inleiding

In dit hoofdstuk vind je informatie over de eindopdracht. Je vraag aan je docent of je aan je eindopdracht mag beginnen. De eindopdracht moet je namelijk geheel zelfstandig kunnen maken.

Je mag zelf een onderwerp voor je eindopdracht bedenken. Kun je niks bedenken, geen probleem, dan krijg je een opdracht. Je hebt dan geen invloed op het onderwerp. Overigens: wij vinden het leuker, als ze zelf iets kiest.

Wil je gebruik maken van een API die je zelf hebt ontwikkeld? Laat deze dan eerst beoordelen door jouw docent; de API moet namelijk voldoende uitdaging voor de eindopdracht bevatten.

Hierna vind je eerst de eisen die aan de eindopdracht worden gesteld. Daarna zie je een beoordelingsoverzicht. Deze beoordeling moet jezelf invullen, met bewijsmateriaal.

## Opdracht

Je gaat een app maken die aan de volgende eisen voldoet:

**MAD1**

* De app bestaat uit tenminste 4 pages, waaronder een about screen, die elke voor een eindgebruiker te bereiken zijn. Maximaal 1 scherm bevat statische data.
* Tijdens de navigatie worden er gegevens tussen de pages uitgewisseld, door middel van parameters. Maak gebruik van Tab en Stack navigation.
* De gegevens die de app toont, zijn afkomstig van het internet. Deze gegevens haal je volgens de richtlijnen op door middel van een API en zet je in een Dart class.
* De app kent een nette en verzorgde lay-out. De look-and-feel van android wordt gevolgd. Bij MAD is lay-out belangrijk: doet er toe. Zorg ervoor dat dit goed en gedegen in elkaar zit:
  + Logische navigatie voor de gebruiker,
  + Goede ondersteuning van de gebruiker,
  + Voorspelbaar gedrag (kijk wat common practice is),

**MAD2**

* Je maakt gebruik van API’s voor opvragen, toevoegen, wijzigen en verwijderen
* Maak gebruik van tenminste 1 herbruikbare component
* Uniforme look-and-feel

NB:

* Het datamodel en de API’s mogen eigen ontwikkelingen zijn.
* Maak ook eens gebruik van een nieuwe widget. Zie bijvoorbeeld: [Material Components widgets | Flutter](https://docs.flutter.dev/development/ui/widgets/material)
* Voor het connecten van jouw emulator met een local host: zie [Connecting Flutter application to Localhost | by Seddiq Sorush | Medium](https://medium.com/@podcoder/connecting-flutter-application-to-localhost-a1022df63130). Het is natuurlijk nog beter wanneer je jouw API online hebt gezet.

## Beoordeling

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Onderwerp | NB | B | G | Toelichting |
| * Tab navigatie |  | B |  |  |
| * Stacknavigatie |  | B |  |  |
| * About page met licenties (showAboutDialog) |  |  | G |  |
| * Layout |  |  | G |  |
| * Netheid code |  |  | G |  |
| * Netwerk gebruik |  | B |  |  |
| * Complexiteit API |  | B |  |  |
| * >= 4 pages, maximaal 1 statische data en 1 about scherm |  |  | G |  |
| * Gegevensuitwisseling bij stacknavigatie |  | B |  |  |
|  |  |  |  |  |
| * Gebruiker kan gegegevens toevoegen, wijzigen, verwijderen en opvragen (CRUD) |  | B |  | Alle CRUD functionaliteiten moeten gemaakt zijn |
| * Tenminste C/U/D moeten via een beveiligde API (bearer token) |  | B |  |  |
| * Rotating bearer token |  |  | G |  |
| * APP draait in deploy mode |  |  | G |  |

* Indien aangeduide onderdelen in kolom B allemaal behaald zijn, dan:
  + Indien aangeduide onderdelen in kolom G allemaal behaald zijn,
    - dan Goed
  + anders Voldoende
* Anders Onvoldoende

# Conclusie

Je hebt in module 2 een volwaardige CRUD applicatie gemaakt, gebruik makend van API’s. Hierbij heb je gebruik gemaakt van de kennis die je al in de module MAD 1 had opgedaan. Voor de CRUD functionaliteiten heb je ook gebruik gemaakt van de stabdaard flutter documentatie op het internet.

Heb je MAD 2 afgerond, dan ben je in staat om een eenvoudige maar volwaardige app te maken.

**Vervolg**

Wil je jezelf nog verder verdiepen in de mogelijkheden van flutter? Hiervoor is ruim voldoende informatie te vinden op het internet, zoals:

* Testen is noodzakelijk om zeker te zijn dat jouw APP goed werkt.
  + Een goed begin vind je op <https://docs.flutter.dev/cookbook/testing>. Hier wordt onder andere ingegaan op Integratie testen, Unit testen, en Widget testen.
  + Kijk daarnaast ook eens in de flutter documentatie onder ‘Testing & debugging’.
* Wil je je verder verdiepen in het ontwerpen van nette lay-outs. Kijk dan nog eens naar:
  + <https://docs.flutter.dev/development/ui/layout>
  + <https://docs.flutter.dev/development/ui/layout/adaptive-responsive>
  + [🖥📱RESPONSIVE DESIGN • Flutter Tutorial - YouTube](https://www.youtube.com/watch?v=MrPJBAOzKTQ) (8 minuten)
  + [🖥📱Responsive Dashboard UI • FLUTTER Tutorial ♡ - YouTube](https://www.youtube.com/watch?v=9bo1V9STW2c) (29 minuten)
* Flutter is open source. De code van Flutter staat op Git
  + <https://github.com/flutter/flutter>