HOGESCHOOL ROTTERDAM / CMI

Web application development

INFWAD01-D
INFWAD21-D

Number of study points: 12 ECTs
Course owners: S. Anwar, R. Vonk
Course teachers: S. Anwar, M.B. Çetin, R. Vonk

Module description

Module name:	Web Application Development
Module code:	INFWAD01-D, INFWAD21-D
Study points and workload:	This module gives 12 ECTs, in correspondence with 336 hours: INFWAD01-D (fulltime): • 14 x 2 hours frontal lecture • 14 x 6 hours assisted practicum • 224 hours of personal study and working on project INFWAD21-D (parttime): • 14 x 2 hours frontal lecture • 14 x 2 hours assisted practicum • 280 hours of personal study and working on project
Examination:	Exam and project assignment
Course structure:	Lectures, practical sessions, and project-based learning (teamwork)
Prerequisite knowledge:	Knowledge of year 1 (Basecamp, OODP, Relational Databases basics)
Learning goals:	At the end of the course, the student should be able to: CLO1. Apply basic knowledge of fundamental web technologies (HTML, CSS, JavaScript, forms and DOM manipulation) while building applications for the web as a full-stack developer. CLO2. Design and develop front-end components and user interfaces using the React library together with TypeScript, applying the functional programming principles. CLO3. Design and develop RESTful Web APIs using .NET Core CLO4. Utilize .NET Core features such as dependency injection, configuration, routing, and data persistence. CLO5. Integrate front-end components with back-end APIs through asynchronous HTTP requests, using authentication and authorization mechanisms in .NET Core Web APIs to secure API endpoints.
Course owner(s):	S. Anwar, R. Vonk
Date	September 2, 2025

General description

1.1 Introduction

This course will take you on a first exploration of web application development using front-end technologies such as TypeScript and React, and back-end technologies such as .NET Core and Entity Framework for building an API. You will learn to design, develop, and deploy full-stack web applications. In frontal lectures, we will introduce you to a stack of current web technologies and the programming paradigms that should accompany them. In the practicum lessons that follow the theory of every week, we will guide you through applying what you have learned. You will also work in teams on a realistic web project.

1.2 Relationship with other teaching units

In the first year you've learned to program using Python and C#. You are now familiar with the concept of OODP, and have knowledge of working with relational databases. This year we will build upon that experience and take your first steps into the exciting world of web application development.

1.3 Learning aids

Your main learning aid is your laptop, and freely available sources from the internet. The material will be provided during the course.

2. Rough program

This is intended as our guideline of topics to cover. During the course, 'Class subject' and 'Content' might not always correspond exactly to the mentioned lesson weeks.

Week	Class subject	Content	Project moments
1 (1.1)	Introduction	How the web works	
2 (1.2)	Front-end fundamentals	HTML & CSS	
3 (1.3)	Front-end fundamentals	JavaScript & DOM manipulation Client-server interaction Modern front-end stacks	
4 (1.4)	React & Typescript	Introduction to TypeScript	
5 (1.5)	React & Typescript	Functional Programming	
6 (1.6)	React & Typescript	Introduction to React	
7 (1.7)	React & Typescript	Asynchronous code State management	

8 (1.8)	React & Typescript	Routing Data fetching	Assessment 1: Formative + mandatory (for feedback)
9 (1.9)	Backend C# / EF	Databases & Entity Framework	
10 (1.10)	Backend C# / EF	Minimal APIs	
11 (2.1)	Backend C# / EF	MVC architecture Dependency injection	
12 (2.2)	-	Self-study week (continue work on project)	
13 (2.3)	Backend C# / EF	Logging & security	Assessment 2: Formative + mandatory (for feedback)
14 (2.4)	Integration	Integrating front-end and back-end	
15 (2.5)	Integration	Final steps	
16 (2.6)	Project	Final project assessment	Deadline for project submission Assessment 3: Summative (graded)
17 (2.7)	Project	Final project assessment	Assessment 3: Summative (graded) (continued)
18 (2.8)	Exam	Theory exam (multiple choice) Insight moment (theory exam)	
19 (2.9)	Resit project	Re-assessment for project	Retake assessment
20 (2.10)	Resit exam	Resit theory exam (multiple choice) Insight moment (resit theory exam)	

3. Assessment

The assessment consists of two parts: a theory exam and a project assessment. Your final grade will be the average of the two parts.

3.1 Theory exam (50% of your final grade)

The theory exam (in ANS) consists of multiple-choice questions. The retake will follow the same form as the first chance exam. The grading scheme is included in this document.

3.2 Project assessment (50% of your final grade)

The project you will be asked to deliver will be a team effort, and you are required to be a good team player. You will work together with your team to design and implement a web application, fulfilling the requirements to be found in the project description document. Some time to work on the project will be allocated during the practicum lessons, so we can guide you along the way. The bigger part of time needed for your project will have to be spent unaccompanied (see study points and workload in the module description above).

Formative feedback about the progress will be given during the course. Take this feedback seriously, because the suggested improvements will be required to get a good grade on your final assessment.

Instructions for the presentation and submission of the project will be communicated during the course.

Even though the project will be a group effort, evaluation and **grading will be individual** and based on your contribution to the complete work, and on the shown understanding of the concepts applied in the code base. During assessments, you will be asked to present and explain your own contributions.

Everyone must make a **significant contribution**. If your contribution is not significant (in relation to your teammates, or, for example, the teacher decides that you have generated too much of your work with AI), you can be excluded from the course and receive a No Go (a ND in Osiris). The teacher will monitor your contributions during the course and can ask you for a clarification at any point.

There will be 3 moments assessment within the project part (and one retake):

- Assessment 1: Formative, mandatory (for feedback)
- Assessment 2: Formative, mandatory (for feedback)
- Assessment 3: Summative (graded)
- Retake assessment (graded)

The work will be evaluated using the project rubric in this document.

3.3 Grading Criteria

- To successfully pass the course, a student must achieve a **sufficient grade (≥ 5.5)** in **both** the exam and the project components.
- The **final grade** will be the **sum of the exam grade and the project grade**, but **only if both components are sufficient (≥ 5.5)**.
- If either the exam or the project grade is insufficient (< 5.5), that grade will **not contribute** to the final grade.
- Students are allowed to improve **only the insufficient component(s)** during the second attempt.
- **Partial grades are valid only within the current academic semester.** If the course is not successfully completed within the same semester, all assessment activities (exam and project) must be repeated in the following academic year/semester.

3.4 Attendance

Attendance is mandatory for both online and offline lessons. Attendance is monitored during each lesson. To meet the attendance requirement, students must be present for at least 80% of the lessons. Being late (arriving after the scheduled start time), counts as an absence. The course coordinator, manager or dean will determine whether your absence qualifies as an ‘exceptional circumstance’ (such as a prolonged illness or the funeral of an immediate family member). Exceptions to the attendance requirement can only be made in consultation with the instructor(s). Work, a scheduled doctor's or dentist's appointment, a few days of flu, or problems with public transport, etc., do not qualify as exceptional circumstances. Public transport strikes are an exception.

If you do not meet the attendance requirement, you will receive a No Go/ND. You will not be able to retake the course. This means you will have to retake the course next academic year.

3.5 Plagiarism

Plagiarism is defined as copying texts, images, or code without correctly citing the source. This also includes texts, images, or code generated by artificial intelligence, such as ChatGPT. You are allowed to use other people's work in your own work, if you make it clear that it is not your own work and provide a correct citation. Plagiarism is a form of fraud.

3.6 Warning

If, during the course, you do not meet one of the above required criteria or are acting in a way that is unacceptable to the teacher or to your fellow students, you will get an official warning. You will be given the chance to make changes (unless you don’t meet the attendance requirement, which is non-negotiable). If you fail to do so in time, you will receive a No Go/ND.

3.7 Theory exam grading scheme

Correct Answers	Grade	Remarks
<8	0	Fail
8-10	1	Fail
11-13	2	Fail
14-16	3	Fail
17-19	4	Fail
20	5,5	Pass
21	6	Pass
22	6,5	Pass
23	7	Pass
24	7,5	Pass
25	8	Pass
26	8,5	Pass
27	9	Pass
28	9,5	Pass
29	10	Pass
30	10	Pass

3.8 Final grade calculation examples

PG = Project Grade

EG = Exam Grade

Grade = ={(PG>=5.5? PG:) + (EG>=5.5? EG: 0)} / 2

If any of PG or EG is ND, then final grade will be ND.

First Chance				**Retake / Second Chance**			
Project Grade	**Exam Grade**	**Grade**	**Remarks**	**Project Grade**	**Exam Grade**	**Grade**	**Remarks**
8	9	(8+9)/2 = 8,5	Success				
10	5	(10+ 0) / 2 = 5	Fail (ReTake Exam)		6	(10+6)/2 = 8	Success
3	9	(0+9)/2=4,5	Fail (Repair Project)	7		(7+9)/2=8	Success
3	4	(0+0)=0	Fail (Retake Exam, Repair Project)	6	6	(6+6)/2=6	Success
9	5	(9+0)= 4,5	Fail (ReTake Exam)		4	(9+0)/2= 4,5	Fail Redo entire course Next year
5	5	(0+0)/2=0	Fail (Retake Exam, Repair Project)	ND	10	(ND+10)/2 = ND	Fail Redo entire course next year
ND	10	(ND+10)/2 = ND	Fail (Repair Project)	8		(8+10)/2 = 9	Success
3	4	(0+0)=0	Fail (Retake Exam, Repair Project)	ND	10	(ND+10)/2 = ND	Fail Redo entire course next year

3.9 Rubric for project evaluation

Criteria	Excellent (9-10)	Good (7-8.5)	Fair (5-6.5)	Poor (3-4.5)	Incomplete (0-2.5)
CLO1: Core front-end technologies					
Basic knowledge of fundamental web technologies	Demonstrates comprehensive understanding of HTML, CSS, forms. Builds fully functional, responsive, and well-structured web applications. HTML and CSS are clean, efficient, and maintainable.	Demonstrates solid knowledge of HTML, CSS, forms with only minor gaps. Builds functional applications that are mostly responsive and user-friendly. HTML and CSS are generally well-structured but may lack optimization.	Demonstrates basic working knowledge of HTML, CSS, forms. Applications are functional but may lack polish, responsiveness, or efficiency. HTML and CSS work but do not always follow best practices.	Demonstrates limited knowledge of HTML, CSS, forms. Applications may have significant usability or functionality issues. HTML and CSS are unorganized, repetitive, or contain frequent errors.	Demonstrates minimal to no understanding of HTML, CSS, forms. Unable to build functional web applications. HTML and CSS contain major flaws and fail to meet basic requirements.
CLO2: TypeScript and React					
User interfaces using React library in TypeScript, applying component-based architecture	The solution is a well-designed application that effectively applies complex functional programming aspects. It follows pure functional programming principles meticulously, demonstrating mastery of advanced concepts such as pure functions, immutability, higher-order functions, function composition, polymorphism, recursion, currying, and closures. The codebase is clean, concise, and elegantly designed. The React implementation shows mastery of the concepts of modularity, separation of concerns and well-maintainable code. It is well structured and follows best practices.	The solution predominantly adheres to functional programming principles, with most aspects correctly applied. It demonstrates a solid understanding of functional programming concepts, including pure functions, immutability, higher-order functions, and function composition. While there may be minor flaws or inconsistencies, the codebase is generally well-structured. The React implementation shows understanding of the concepts of modularity, separation of concerns and well-maintainable code. It is reasonably well structured and follows some best practices.	The solution follows functional programming principles to some extent but lacks consistency or clarity in its implementation. While functional aspects are present, they may be poorly integrated or applied inconsistently throughout the codebase. The application design may be somewhat convoluted or overly complex, detracting from the overall quality of the solution. The React implementation shows some basic understanding of the concepts of modularity, separation of concerns and well-maintainable code. It is somewhat structured and doesn't completely follow some best practices.	The solution deviates significantly from functional programming principles, with a majority of the codebase following imperative programming paradigms. Functional aspects are either absent or poorly implemented, leading to a codebase that is difficult to understand, maintain, or extend. The application design may be cluttered, disorganized, or overly reliant on mutable state and side effects. The React implementation lacks basic understanding of the concepts of modularity, separation of concerns and well-maintainable code. It is not well structured and doesn't follow best practices.	The solution is incomplete or severely flawed, with little evidence of functional programming principles being applied. It may rely heavily on imperative programming techniques, resulting in an ineffective or non-functional application. The codebase lacks coherence, structure, or adherence to any discernible programming paradigm. The React implementation is severely flawed or incomplete.

CLO3: Endpoint					
Endpoint Implementation, request/ response handling	All endpoints are correctly implemented, inline documented, and adhere to RESTful principles. They effectively handle various HTTP methods and provide appropriate responses including status codes.	Most endpoints are correctly implemented and adhere to RESTful principles, with a few minor endpoints incomplete, or some request / responses might not be handled properly.	Some endpoints are correctly implemented but lack documentation or adherence to RESTful principles. The core functionality of the application is still there with peripheral cases missing or incomplete.	Few endpoints are correctly implemented, poorly documented, or do not adhere to RESTful principles. They handle HTTP methods and resource representations inadequately, causing issues.	Most endpoints are missing, incorrectly implemented, undocumented, or fail to adhere to RESTful principles, rendering the API unusable.
CLO4: .NET Core features					
Dependency Injection, configuration, routing, and data persistence.	Dependency injection is correctly implemented throughout the application, promoting modularity, and maintainability. The data is made persistent, and crud operations are working properly.	Dependency injection is mostly correct and functional but may have minor issues or inconsistencies. Data is persistent with most of the crud operations working properly	Dependency injection is present but has issues in some modules, The crud operation on data has minor issues.	Dependency injection is poor, or data is not made consistent	Dependency injection is severely flawed, and/or data is not being made persistent. It may not function as intended or disrupt application functionality.
CLO5: Integrate frontend and backend					
Communication between front-end React components and back-end	Demonstrates exceptional proficiency in designing and implementing effective communication between the front and backend. It uses asynchronous HTTP requests (fetch) and adheres closely to RESTful API conventions, ensuring efficient data exchange, error handling, and synchronization. The communication process is robust, reliable, and follows best practices for client-server communication. Demonstrates proficiency in implementing middleware and filters to secure API endpoints and enhance security measures.	Demonstrates solid proficiency in designing and implementing effective communication between front and back-end. It utilizes asynchronous HTTP requests (fetch) to exchange data and follows RESTful API conventions for consistency and clarity. While there may be minor issues or inconsistencies, the communication process is generally well-implemented and adheres to best practices. Middleware implementation is mostly correct and functional but may have minor issues or inconsistencies. It generally enhances API security	Demonstrates basic proficiency in designing and implementing communication between front-end and back-end. It uses asynchronous HTTP requests (fetch) and attempts to adhere to RESTful API conventions, but there may be noticeable issues with efficiency, error handling, or synchronization. The communication process may lack clarity or consistency in implementation. Middleware implementation has noticeable issues, such as inefficiencies, inconsistencies, or incomplete functionality. It may not fully enhance API security.	Solution lacks proficiency in designing and implementing communication between front and backend. It may deviate significantly from RESTful API conventions or fail to handle async requests effectively. The communication process may be inefficient, error-prone, or poorly synchronized, leading to suboptimal data exchange and integration between client and server. Middleware implementation is poor, causing security vulnerabilities or insufficient protection of API endpoints. It may lack key functionality	Solution is incomplete or severely flawed, with little evidence of proficiency in designing and implementing communication between front and backend. It may lack essential components or fail to establish functional communication channels required for effective data exchange. The integration between client and server may be incomplete, non-functional, or poorly implemented. Middleware implementation is severely flawed or incomplete, leading to frequent security breaches or inadequate protection of API endpoints.