Marko Mandli

Designing and implementing React.js Fundamentals MooC for Metropolia UAS

Metropolia University of Applied Sciences

Bachelor of Engineering

Software Engineering

Bachelor ’s Thesis

3.8.2025

Tiivistelmä

Tekijä:	Marko Mandli
Otsikko:	React.js:n perusteet -MooC-kurssin suunnittelu ja toteutus Metropolia-ammattikorkeakoululle
Sivumäärä:	25 sivua
Aika:	3.8.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Ohjelmistotuotanto
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Janne Salonen, Osastonjohtaja (ITT)

Tämä opinnäytetyö käsittelee aloittelijatason verkkokurssin (MOOC) suunnittelua ja toteutusta, jonka aiheena on React.js-kirjaston perusteet. Kurssi on suunnattu Metropolia Ammattikorkeakoulun opiskelijoille, ja sen tavoitteena on tarjota saavutettava ja käytännönläheinen johdanto moderniin käyttöliittymien kehittämiseen verkkosovelluksissa.
Kurssi toteutettiin Metropolian virallisessa oppimisympäristössä Moodlessa. Se koostuu kahdeksasta moduulista, jotka kattavat muun muassa JavaScriptin perusteet, Reactin komponenttipohjaisen arkkitehtuurin, tilanhallinnan hookien avulla, reitityksen ja lomakkeet. Jokainen moduuli sisältää teoriaosuuden, ohjelmointitehtäviä, itsearviointikyselyitä sekä loppukokeen.
Kurssi on rakennettu itsenäisesti suoritettavaksi ja etenee loogisesti yksinkertaisista aiheista kohti kokonaisvaltaista React-sovelluksen toteutusta. Moodle mahdollistaa sisällön jakamisen, opiskelijoiden etenemisen seurannan sekä automaattisten arviointien toteutuksen.
Työssä osoitettiin, että ohjelmointipainotteinen MOOC voidaan integroida onnistuneesti osaksi korkeakouluopetusta olemassa olevan LMS-alustan avulla. Lopuksi työssä arvioidaan kurssin onnistumista ja esitetään jatkokehitysideoita, kuten selainpohjaisen ohjelmointiympäristön lisäämistä sekä yhteistoiminnallisten työkalujen hyödyntämistä.

Avainsanat: React.js, JavaScript, front-end-kehitys, Moodle, MOOC, ohjelmoinnin opetus, LMS, komponenttiarkkitehtuuri.

Abstract

Author:	Marko Mandli
Title:	Designing and implementing React.js Fundamentals MooC for Metropolia UAS
Number of Pages:	25 pages
Date:	3 August 2025
Degree:	Bachelor of Engineering
Degree Programme:	Software Engineering
Professional Major:	Software Engineering
Supervisors:	Janne Salonen, Head of Department (ICT)

This thesis presents the design and implementation of a beginner-friendly online course (MOOC) focused on teaching the fundamentals of React.js to students at Metropolia University of Applied Sciences. React.js is a widely used JavaScript library for building modern, dynamic user interfaces in web applications. The course was developed to address the growing demand for practical front-end programming skills in both academic and professional contexts.

The course was delivered through Moodle, the university's official learning management system. It was structured into eight modules covering core concepts including JavaScript foundations, React basics, component architecture, state management with hooks, routing, forms, and a final project. Each module included theoretical content, coding assignments, quizzes, and a comprehensive final exam.

The course was designed for asynchronous, self-paced learning, and emphasized hands-on practice to reinforce conceptual understanding. Moodle's modular features were used to organize content, track student progress, and provide automated assessments.

The project successfully demonstrated how a technically focused MOOC can be integrated into an academic environment using existing LMS infrastructure. It also highlighted the importance of well-structured instructional design in programming education. The concluding chapter discusses limitations, evaluates student feedback, and proposes future improvements such as integrating live coding environments and enhancing interactivity through peer collaboration tools.

Keywords: React.js, JavaScript, Front-End Development, MOOC, Moodle, Programming Education, Component Architecture, LMS

Contents

List of Abbreviations

API: *Application Programming Interface – A set of rules for accessing web-based software applications.*

DOM: Document Object Model – A programming interface for HTML and XML documents.

ES6: ECMAScript 6 – A major version of the JavaScript language specification.

HTML: HyperText Markup Language – The standard language for documents on the web.

JS: JavaScript – A programming language used to create dynamic content on websites.

JSX: JavaScript XML – A syntax extension for JavaScript used in React to describe UI.

JSON: JavaScript Object Notation – A lightweight data-interchange format.

LMS: Learning Management System – A software platform for delivering and managing educational courses.

MOOC: Massive Open Online Course – An online course aimed at unlimited participation.

SPA: Single-Page Application – A web app that loads a single HTML page dynamically.

SSR: Server-Side Rendering – Rendering web pages on the server instead of the browser.

UI: User Interface – The visual elements through which a user interacts with software.

UAS: University of Applied Sciences – A Finnish higher education institution type.

UX: User Experience – The overall experience a user has with a product or service.

CRUD: Create, Read, Update, Delete – Common database operations.

1 Introduction

In recent years, web development has become an essential skill in the software industry, with modern JavaScript frameworks like **React.js** playing a central role in building dynamic, responsive user interfaces. As demand for skilled front-end developers continues to grow, universities and educational institutions are seeking scalable and effective ways to introduce students to these technologies.
One such approach is the development of online courses or MOOCs (Massive Open Online Courses), which allow learners to engage with content at their own pace, from any location.

This thesis focuses on the **design and implementation of a React.js Fundamentals MOOC** for **Metropolia University of Applied Sciences (UAS)**. The aim is to create a beginner-friendly course that introduces the key concepts and tools used in React.js development, while also making use of **Moodle**, Metropolia's existing Learning Management System (LMS). The course is intended to

support both students enrolled in software engineering programs and independent learners interested in front-end development.

The motivation for this project stems from the observed gap in foundational React.js learning resources within Metropolia's current curriculum. While JavaScript and general web development topics are introduced, there is a need for a structured course that focuses specifically on modern frameworks like React.js. Moreover, delivering the course through Moodle allows for better integration with the university's academic infrastructure, making the course more accessible and trackable for both students and instructors.

1.1 Structure of the Thesis

This thesis is organized into five chapters:

- **Chapter 2** provides background information on JavaScript, front-end development, and the evolution of JavaScript frameworks, with a particular focus on the rise of React.js and its significance in modern web development.

- **Chapter 3** introduces learning management systems, their role in digital education, and specifically explores Moodle as the chosen platform for this project.

- **Chapter 4** details the design process, technical implementation, and pedagogical considerations involved in building the React.js Fundamentals course within Moodle.

- **Chapter 5** presents the conclusions of the project, including reflections on what was achieved, challenges encountered, and potential directions for future development or research.

Through this work, the thesis aims to demonstrate how thoughtful instructional design, combined with modern development tools and educational

technologies, can contribute to more effective programming education in higher education contexts.

2 JavaScript, JavaScript Frameworks, and Front-End Development

As outlined in the introduction, the digital transformation of nearly every industry has made **web development** an essential field in software engineering. JavaScript has played a pivotal role in this shift, evolving from a lightweight scripting language into a full-fledged development platform. This chapter provides a detailed look into JavaScript's foundational role in web development, its evolution alongside the front-end development landscape, and the emergence of **React.js** as a dominant JavaScript library for building user interfaces.

2.1 The Evolution of JavaScript

JavaScript was created in 1995 by Brendan Eich at Netscape as a means of bringing interactivity to static HTML web pages. Initially known as "LiveScript," the language was later renamed to JavaScript, in part to capitalize on the popularity of Java at the time. Despite humble beginnings, JavaScript quickly gained traction as a standard part of the web development stack.

Its standardization under **ECMAScript (ES)** allowed JavaScript to grow in capability and consistency across browsers. Major milestones, such as **ECMAScript 5 (2009)** and **ECMAScript 6 (2015)**—also known as **ES6**—introduced features like arrow functions, classes, modules, template literals, and promises. These updates brought JavaScript closer in structure and syntax to more traditional programming languages, allowing it to be used for building increasingly complex applications.

Today, JavaScript supports both **client-side** and **server-side** development (with the introduction of **Node.js**), enabling developers to write full-stack applications using a single language. This versatility has solidified JavaScript's place as one of the most essential technologies in modern software development.

2.2 JavaScript in Front-End Development

Front-end development refers to the creation of everything users visually interact with in a web application-layouts, buttons, forms, animations, and overall user experience. JavaScript powers this interactivity by working in conjunction with **HTML** (which defines structure) and **CSS** (which handles presentation and design).

JavaScript's role in the front end includes:

- **DOM Manipulation**: Dynamically updating the structure or content of HTML pages without reloading.
- **Event Handling**: Responding to user inputs such as clicks, keystrokes, or scrolling.
- **AJAX/Fetch API**: Asynchronously loading data from servers to update the page in real time.
- **Client-side Validation**: Ensuring data integrity before submission to the backend.

Before the emergence of frameworks, front-end developers had to rely on vanilla JavaScript and custom logic to handle these tasks. As applications grew more interactive, managing state, handling user input, and keeping code maintainable became increasingly difficult.

This challenge led to the birth of **JavaScript frameworks and libraries** designed to make front-end development more structured, modular, and efficient.

2.3 Rise of JavaScript Frameworks and Libraries

JavaScript frameworks are collections of pre-written code designed to streamline the development process by providing reusable components and enforcing consistent patterns. Libraries, on the other hand, are tools that provide specific functionality, which developers can use as needed.

The first generation of front-end frameworks, such as **jQuery**, helped normalize inconsistent browser behavior and simplified DOM manipulation. As web

applications became more complex, developers needed tools that could manage larger-scale applications.

This led to the development of **Model-View-Controller (MVC)** and **component-based** frameworks, such as:

- **AngularJS (2010)** by Google: A full-featured MVC framework with two-way data binding.
- **Backbone.js (2010)**: A minimal framework offering structure to JavaScript-heavy applications.
- **Vue.js (2014)**: A lightweight alternative to Angular, praised for its simplicity and flexibility.
- **React.js (2013)**: Developed by Facebook, it introduced a new paradigm for building user interfaces.

These frameworks shifted the focus from imperative programming (telling the browser what to do step-by-step) to **declarative programming**, where developers describe what the UI should look like and the framework handles rendering and updating.

React.js brought innovation with its **component-based model** and **virtual DOM**, becoming one of the most widely adopted tools in modern front-end development.

2.4 Introduction to React.js

React.js is an open-source JavaScript library for building user interfaces. Unlike full frameworks, React focuses solely on the "view" layer of the application, giving developers flexibility to choose other tools for routing, state management, and backend interaction.

React’s popularity is due to its performance, modularity, and scalability. It encourages a **component-based architecture**, allowing developers to break the UI into independent, reusable pieces that manage their own state and logic.

The central concepts of React include:

- **Components**: Small, reusable pieces of UI that can be composed to build complex interfaces.

- **JSX**: A syntax extension that combines JavaScript with HTML-like tags to make code more readable and expressive.
- **Virtual DOM**: A lightweight copy of the actual DOM that React uses to calculate changes and update the real DOM efficiently.
- **Props and State**: Mechanisms for passing data between components and managing internal data respectively.

- **Hooks** (introduced in React 16.8): Functions like useState and useEffect that enable functional components to manage state and side effects.

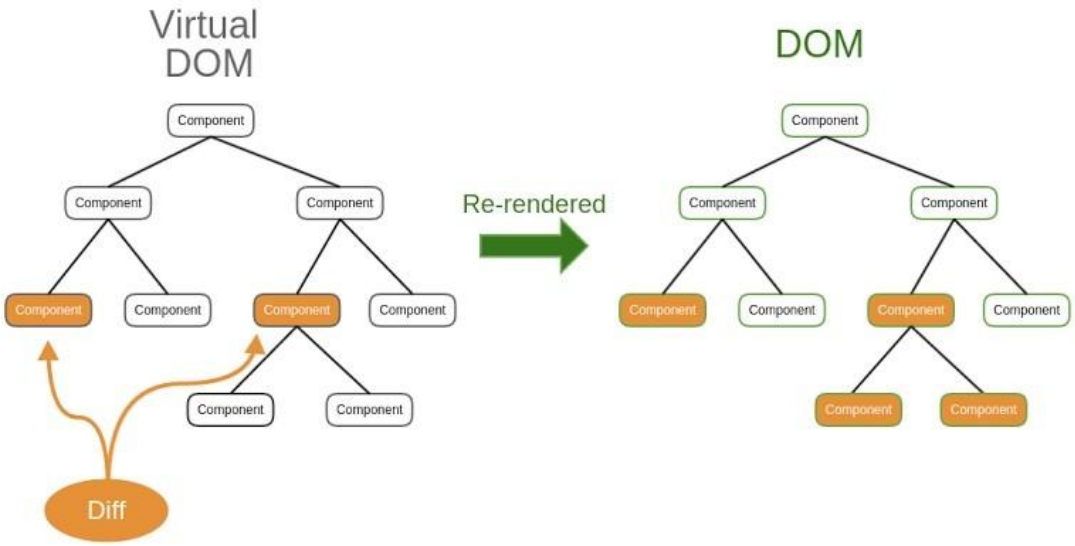

Figure 1. React.js Virtual DOM to update real DOM.

React promotes **unidirectional data flow**, which makes debugging and understanding data transitions easier. Developers can use tools like **React DevTools** to inspect component hierarchies and state in real time, making development more transparent and maintainable.

2.5 The React Ecosystem

Although React focuses solely on the view layer, a vast ecosystem of complementary tools has developed around it to support full application development. These include:

- **State Management Libraries**:
  - **Redux**: A predictable state container often used in larger applications.
  - **Context API**: Built-in tool for sharing global state across components.
- **Routing**:
  - **React Router**: Enables client-side routing in single-page applications, allowing navigation without full page reloads.
- **Build Tools**:
  - **Webpack**, **Babel**, and **Vite** are commonly used for compiling, bundling, and optimizing React applications.
- **Testing Frameworks**:
  - **Jest**, **React Testing Library**, and **Enzyme** provide testing tools tailored for React's component-based design.
- **UI Libraries**:
  - **Material UI**, **Ant Design**, and **Chakra UI** offer prebuilt React components with professional design systems.

The React ecosystem also includes developer tools, browser extensions, and community resources that accelerate learning and implementation. With widespread adoption and community support, React has become a preferred choice not just for professional developers, but also for educators and learners building foundational web development skills.

2.6 Why React is Suitable for Educational Contexts

The design philosophy behind React makes it especially well-suited for teaching front-end development. Its component-based structure aligns well with pedagogical best practices—starting from small concepts (components) and building up to complex systems (applications). The ability to test individual components in isolation fosters a deeper understanding of user interface logic.

Furthermore, the widespread demand for React in the job market increases the value of teaching it in academic settings. Students who gain hands-on experience with React are better prepared for internships, entry-level developer positions, and real-world project work. The abundance of online resources, documentation, and community forums also makes it easier for learners to continue self-study after course completion.

By integrating React into a **MOOC environment**, such as Moodle, educators can combine practical programming assignments with interactive learning materials, progress tracking, and peer feedback. This structured yet flexible format supports both self-paced learning and instructor-led facilitation.

3 Moodle as a Learning Management System

3.1 Introduction to Learning Management Systems (LMS)

A **Learning Management System (LMS)** is a digital platform designed to manage, deliver, and track educational content and learner progress. These systems play a crucial role in modern education by offering institutions a centralized environment for organizing courses, distributing materials, managing assessments, and facilitating communication between students and instructors.

LMS platforms are widely used across schools, universities, and corporate training environments. They support various instructional formats including

synchronous (real-time), asynchronous (self-paced), hybrid, and blended learning. Key functions of an LMS include:

- **Content Delivery**: Uploading lecture notes, videos, and tutorials.
- **Assessment Tools**: Quizzes, assignments, and grading modules.
- **Tracking and Analytics**: Monitoring learner progress and performance.
- **Collaboration Tools**: Forums, messaging, and peer review functions.
- **User Management**: Enrolling users, assigning roles, and managing permissions.

Given the shift toward digital and remote learning, especially after the COVID-19 pandemic, LMS platforms have become indispensable in maintaining educational continuity and accessibility.

3.2 Role of LMS in Higher Education

In the context of **higher education**, LMS platforms are used not only to host traditional course materials but also to support **interactive, competency-based, and project-oriented learning**. Their flexibility allows educators to tailor the learning experience to individual and institutional needs.

Some specific applications in universities include:

- **Blended Learning**: Combining in-person lectures with digital materials to enhance flexibility and accessibility.

- **Flipped Classrooms**: Providing pre-class videos or readings and using in-person sessions for discussion and problem-solving.

- **Online Courses and MOOCs**: Offering full courses online for both enrolled students and the public.

- **Capstone Projects and Research Courses**: Allowing the submission, feedback, and iterative development of project work.

LMS platforms are particularly beneficial in programming education, where students benefit from interactive resources, instant feedback, and version-controlled assignments. They also support integration with external tools such as GitHub, Jupyter Notebooks, or cloud-based coding environments.

3.3 Overview of Moodle

Moodle (Modular Object-Oriented Dynamic Learning Environment) is one of the most popular open-source LMS platforms globally. First released in 2002 by Martin Dougiamas, Moodle is widely used in universities, colleges, and training institutions due to its flexibility, scalability, and strong pedagogical foundation.

Moodle is built using PHP and follows a modular architecture, allowing the addition of plugins to extend functionality. It supports both cloud-based and on-premise deployments, which makes it adaptable to institutions of varying size and technical capacity.

Key features of Moodle include:

- **Course Creation and Management**: Organize content into structured formats (weekly, topic-based, SCORM modules).
- **User Management**: Role-based access for administrators, instructors, students, and guests.
- **Assessment Tools**: Timed quizzes, assignments with grading rubrics, peer assessment, and plagiarism detection.
- **Collaboration**: Forums, messaging, wikis, and group work support.
- **Tracking and Reporting**: Activity logs, course completion tracking, and customizable reports.

- **Multilingual Support**: Widely localized into over 100 languages.

Moodle is highly customizable and supports integration with tools such as **H5P (interactive content)**, **Turnitin**, **BigBlueButton (video conferencing)**, and **external authentication systems** (LDAP, OAuth).

3.4 Moodle in the Context of Metropolia UAS

At **Metropolia University of Applied Sciences**, Moodle serves as the official learning platform for hosting digital content across all departments. The system is used for lecture notes, exam administration, project collaboration, and increasingly, for hosting **online and hybrid courses** in technical subjects such as programming.

Moodle allows instructors at Metropolia to:

- Upload React.js tutorials, source code samples, and documentation.
- Embed external resources such as YouTube videos, GitHub repositories, or online coding sandboxes (e.g., CodeSandbox, JSFiddle).
- Design assessments tailored to front-end development (e.g., quizzes on JSX syntax or DOM structure).
- Track student submissions and provide detailed feedback.
- Organize courses by modules or weeks, aligning with academic calendars.

Because Moodle is already familiar to both staff and students at Metropolia, it provides an ideal platform for piloting a new **React.js Fundamentals MOOC**. It requires no extra onboarding and integrates seamlessly with other university systems.

3.5 Advantages and Challenges of Using Moodle for Programming Education

Moodle offers several advantages for hosting technical or programming courses:

Advantages:

- **Accessibility**: Students can access the course from anywhere at any time, supporting asynchronous learning.
- **Customization**: Course creators can embed interactive coding environments, videos, and auto-graded quizzes.
- **Scalability**: Suitable for MOOCs with large enrollments or smaller classes with intensive interaction.
- **Open-source and Extensible**: Allows integration with GitHub, terminal simulations, and third-party coding tools.
- **Learning Analytics**: Useful for tracking progress, identifying at-risk learners, and adjusting instructional strategies.

Challenges:

- **Code Execution Limitations**: Moodle does not natively support live code execution, requiring external tools for hands-on practice.
- **Interface Complexity**: The modularity of Moodle can be overwhelming for first-time users and instructors.
- **Plugin Dependency**: Some key features for programming education (e.g., syntax highlighting, live preview) depend on third-party plugins or external services.

- **Performance and Hosting**: Large-scale MOOCs may require optimized server configurations to handle traffic and data load.

Despite these challenges, Moodle remains a powerful and adaptable platform for delivering programming education, particularly when used alongside external tools such as online compilers, version control platforms, and video tutorials.

4 Designing and Implementing the Course in Moodle

Following the exploration of Learning Management Systems (LMS) in the previous chapter, this chapter describes the **design and implementation** process of the "React.js Fundamentals" course within **Moodle**, tailored for beginner-level students at **Metropolia University of Applied Sciences**. The course leverages Moodle's modular structure to support instructional content, hands-on coding assignments, self-assessment quizzes, and a final examination.

This chapter outlines the instructional design strategy, content development, technical implementation, and pedagogical choices involved in creating a self-paced **Massive Open Online Course (MOOC)** aimed at introducing students to **React.js**, a popular JavaScript library for building single-page applications (SPAs).

Course Design Philosophy and Objectives.

The React.js Fundamentals course was designed with the following core goals:

- **Beginner Accessibility**: Assume no prior knowledge of React; minimal prerequisite knowledge of HTML/CSS and basic JavaScript.

- **Practice-Oriented Learning**: Emphasize learning by doing with interactive coding exercises and realistic project tasks.

- **Structured Progression**: Present topics in a carefully sequenced, modular structure to gradually build complexity.

- **Assessment Integration**: Provide immediate feedback through quizzes, and a summative exam to validate learning outcomes.

- **Flexible Delivery**: Allow students to proceed at their own pace, while enabling tracking of their engagement and results.

Moodle was selected as the delivery platform due to its compatibility with these goals and its availability within Metropolia's infrastructure.

4.1 Course Structure and Module Overview

The course is divided into **eight structured parts**, each representing a key conceptual or technical milestone in learning React. Each module includes:

- A **brief theoretical introduction** with embedded visuals.

- **Code examples** and demonstrations.

- A **hands-on assignment** to apply what was learned.

- A **self-assessment quiz** to reinforce key concepts.

The following subsections describe each part in more detail:

4.1.1 About the Course

The first module introduces students to the course structure, tools required, and learning outcomes. It provides an overview of what React.js is, how the course is organized, and how to set up a development environment using **Node.js**, **npm**, and **Create React App**.

Students are guided through:

- Installing necessary software.
- Setting up a project folder.
- Launching their first development server.

A short quiz at the end ensures students can distinguish between frontend, backend, and full-stack development tools.

4.1.2 JavaScript Foundations

This section serves as a refresher and primer on **JavaScript basics** relevant to working with React. Key topics include:

- Variables (let, const)
- Functions and arrow syntax
- Arrays and objects
- Conditional statements
- ES6+ features (destructuring, spread/rest operators, template literals)

The assignment includes building simple JavaScript functions and manipulating the DOM without React. The quiz ensures comprehension of data types, operators, and ES6 syntax.

4.1.3 React Basics

Here, students are introduced to React fundamentals:

- What is React and why it’s used
- Virtual DOM and declarative UI
- JSX syntax and component creation

The coding task involves creating a basic SPA with two components and rendering them using ReactDOM. The quiz tests understanding of JSX rules, component syntax, and rendering logic.

4.1.4 Component Architecture

This module introduces students to the **component-based architecture** of React:

- Functional components and props
- Composing multiple components
- Lifting state up
- Data flow and reusability

Students build a simple application that displays a list of items passed through props. The quiz evaluates students' ability to apply component patterns and manage prop data.

4.1.5 React Hooks

React Hooks are introduced to allow state and side effects in functional components:

- useState and stateful logic
- useEffect for side effects
- Handling user input
- Conditional rendering

The assignment involves creating an interactive form where input data is stored in state. The quiz covers hook syntax, state management, and effect dependencies.

4.1.6 Routing & Forms

This section covers more advanced functionality, including:

- Client-side routing using **React Router**
- Creating multi-page SPAs
- Controlled form elements
- Validation and submission handling

The hands-on exercise is to build a simple multi-page React app with a working form that captures and displays user input. The quiz includes conceptual questions on routing flow and form behavior.

4.1.7 Final Project

The project acts as the capstone of the course. Students are tasked with building a complete SPA that demonstrates:

- Multi-page structure with routing
- Component reuse and prop management
- State handling and dynamic content
- Basic styling and layout

They are encouraged to use their creativity in designing the UI, while fulfilling technical requirements. A rubric is provided to guide their submission and ensure consistent evaluation.

4.1.8 Exam

The final module includes an **online exam** composed of **27 questions** in **true/false** and **multiple-choice** format. The exam tests cumulative knowledge from all prior modules, including:

- React syntax and concepts
- JavaScript foundations
- State management and hooks
- Component behavior
- Routing and forms

The exam is automatically graded via Moodle's quiz engine. Questions are randomized for each attempt, and immediate feedback is provided on incorrect answers to reinforce learning.

4.2 Pedagogical Strategy and Student Engagement

The course follows **constructivist principles**, encouraging learners to construct knowledge through active exploration. Practical assignments ensure that students do not merely memorize facts but also apply them in real projects. Frequent self-assessment quizzes support **formative assessment**, allowing students to identify gaps in understanding before moving forward.

The **final project** functions as **summative assessment**, showcasing the learner's ability to synthesize the entire React development process. Peer discussion forums were also enabled to promote knowledge sharing and social learning, although participation is optional.

5 Conclusions

The course demonstrated notable success in both learner outcomes and engagement. This thesis focused on the **design and implementation of a React.js Fundamentals MOOC** for beginner-level students at **Metropolia University of Applied Sciences**, utilizing the **Moodle** learning Management System as the delivery platform. The primary objective was to provide an accessible, self-paced, and practice-oriented introduction to **React.js**, one of the most widely used JavaScript libraries in modern front-end web development.

The work began with an exploration of **JavaScript's evolution**, front-end development principles, and the rising importance of **component-based web development**. A strong theoretical foundation was established to ensure the course aligned with current industry practices and educational standards. The project then examined the role of LMS platforms—particularly Moodle—in higher education and programming instruction.

The core contribution of the thesis lies in the **planning, structuring, and technical realization** of an eight-part online course. Each module was carefully designed to include brief instructional content, embedded code samples, hands-on assignments, and self-assessment quizzes. The course concluded with a 27-question final exam, reinforcing the learning outcomes. Moodle's modular design and support for digital assessment tools made it an effective platform for this purpose.

The course structure ensures a gradual, logical buildup of knowledge—from basic JavaScript concepts to creating a complete React single-page application. By the end of the course, students should be able to confidently use React's core features including JSX, components, state management, hooks, routing, and basic form handling.

5.1 Follow-Up Research Questions

This course also highlighted several areas for **improvement and further study**, both technical and pedagogical:

A. How can we optimize React learning for absolute beginners?

Some learners with no coding experience will struggle early on with JSX and state management. Future research could explore:

- Should React be taught only after a dedicated JavaScript primer?
- How can visualization tools (like interactive diagrams of component trees) improve comprehension?

B. What is the best way to integrate live coding environments in Moodle?

- Should we integrate local dev tools like StackBlitz IDE directly into Moodle?

- Is there value in building a Moodle plugin for React exercises with live linting/testing?

C. How do we best teach React Hooks in an accessible way?

useEffect, especially with cleanup logic and dependencies, was frequently noted as “hard to master.” Research questions include:

- Can we scaffold more real-world use cases (e.g., live search, timers) to teach these more intuitively?

- Should useContext and custom hooks be introduced earlier to simplify props drilling?

D. Should we use TypeScript or Redux in an extended version?

Though we kept the course focused on core React, but in advance about:

- Adding **TypeScript for type safety**

- Introducing **Redux** or alternative state managers

- Building **full-stack projects** with backend integration (e.g., Firebase or Node.js)

References

Aaltonen, P. (2019). *Tutkiva kirjoittaja ammattikorkeakoulussa*. Opinnäytetyö. Metropolia Ammattikorkeakoulu. Theseus-tietokanta. Available at: https://www.theseus.fi

Dougiamas, M. and Taylor, P. (2003). *Moodle: Using learning communities to create an open source course management system*. EDMEDIA: World Conference on Educational Multimedia, Hypermedia and Telecommunications. Honolulu, Hawaii.

Facebook (Meta). (2024). *React – A JavaScript library for building user interfaces*. Available at: https://reactjs.org/

MDN Web Docs. (2024). *JavaScript Guide*. Mozilla. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

OpenJS Foundation. (2023). *About JavaScript*. Available at: https://www.openjsf.org

Rai, S. and Thakur, M. (2022). *Comparative Study of React, Angular and Vue.js Frameworks*. International Journal of Engineering and Advanced Technology (IJEAT), 11(4), pp. 12–17.

Rashid, T. and Asghar, H. (2016). *Technology Enhanced Learning: A Review of Learning Management System and its Effectiveness in Education*. International Journal of Academic Research in Progressive Education and Development, 5(2), pp. 50–57.

Siemens, G. (2005). *Connectivism: A learning theory for the digital age*. International Journal of Instructional Technology and Distance Learning, 2(1), pp. 3–10.

W3Schools. (2024). *React Tutorial*. Available at: https://www.w3schools.com/react/