Course Introduction
Design of Dynamic Web System

Main Teacher: Johan Kristiansson

TA: Casper Lundberg

Course Aim

The student should have the skills and knowledge to:

- Build a dynamic web system based on own and publicly available program modules.
- Create an application-programming interface for communication between web client and web server.
- With broad expertise in the field of web technology, understand the context at system level, and apply knowledge in mathematics and science for specific issues. This is shown through presenting concepts for modern techniques for client-server communication, secure handling of user information and user interface design.
- Have a good understanding of ethical problems related to handling of sensitive data related to individuals.
- Model, simulate, predict and evaluate web systems, even with limited information. This is shown through laboratory work and own developed prototype, including preparatory work and analysis of results.

Contents and Realization

- The course contains an introduction to various web related technologies for creating dynamic web systems for human-computer and computer-computer interaction
- The course is conducted by students ***working in pairs*** in order to solve a number of lab assignments
- The students are given freedom to choose what technologies they want to use in order to solve each assignment, and the chosen technologies are discussed during examination
- Required knowledge needed is gathered via searches in literature and on the Internet.
- A number of lectures will be provided in order to introduce relevant technologies and current problems
- Lab assignment work is documented through an online versioning system that is used continuously during the course

Why This Course Matters?

The tech industry is facing an unprecedented shortage of engineers who can build and deploy production-ready web systems. While many developers can code, few possess the complete skill set this course provides:

What Industry Desperately Needs

- Full-stack engineers who understand both frontend and backend
- DevOps-capable developers who can deploy and monitor their own code
- Surging demand for Cloud-Native experts
- Production-minded engineers who write tests, implement CI/CD, and think about security

https://www.refontelearning.com/blog/top-paying-devops-skills-in-2025-observability-kubernetes-more

https://www.cncf.io/wp-content/uploads/2025/04/cncf_annual_survey24_031225a.pdf

https://www.uplers.com/blog/the-future-of-cloud-computing-why-kubernetes-developers-are-in-high-demand

Assessment & Examination

- Course Assessment: 7.5 ECTS
  - Laboratory work with practical exercises and prototype development
  - Oral examination demonstrating understanding and reasoning ability

AI-Assisted Learning

Key Principle: AI is a tool, not a substitute for understanding. You must be able to defend every line of code and every design decision during oral examination.

- GenAI Usage Guidelines
  - Learning assistant: Use ChatGPT, Claude, Cursor for mentoring and guidance
  - Code generation: Accelerate development with AI-generated code
  - Problem solving: Explore solutions and debug with AI assistance
  - Documentation: Generate comments and technical documentation
- Your Responsibilities
  - Understand everything: Explain all design choices and architecture decisions
  - Test thoroughly: All AI-generated code must have comprehensive tests (>60% coverage)
  - Demonstrate competence: Be able to reason about test strategies and prove correctness
  - Own the code: You are accountable for all submitted work, regardless of origin

How to fail the course?

If you cannot explain it simply, you don't understand it well enough

The oral exam tests understanding, not memorization. Students who rely entirely on AI without developing comprehension will struggle to defend their work

Examples:

- Submitting another student's work
- Unable to draw system architecture on whiteboard
- Cannot describe alternative solutions they considered
- Fails to explain why they chose specific libraries or frameworks
- Fails to understand the flow of data through their application
- Cannot explain what their submitted code does line-by-line
- Cannot explain what their tests are actually testing
- Tests are superficial (only happy path, no error cases)
- Unable to identify edge cases in tests

What is a Dynamic Web System?

A Dynamic Web System is a web application that generates content in real-time based on user interactions, database queries, and server-side processing, rather than serving static, pre-built pages

- Facebook/Instagram: Personalized feeds, real-time notifications
- Gmail/Outlook: Email management, instant updates
- Slack/Discord: Live messaging, presence indicators
- Amazon/eBay: Product recommendations, shopping carts
- Uber/Lyft: Real-time driver tracking, dynamic pricing
- Airbnb/Booking.com: Availability updates, personalized search
- Netflix/Spotify: Content recommendations, watch history
- Google Docs/Office 365: Collaborative editing, auto-save
- GitHub/GitLab: Code repositories, pull requests
- Online Banking: Account balances, transaction history

- Old definition: Anything that is generated and rendered in the browser
- New definition: All data comes from a database, and heaviliy personalised

What is NOT a Dynamic Web System?

- Documentation & Content Sites
  - Documentation sites: Read the Docs (pre-generated HTML)
  - Technical blogs: Dev.to exports, Medium publications (static versions)
  - API references: Swagger/OpenAPI static documentation
- Client-Only Web App
  - Calculator apps: All logic in JavaScript, no backend needed
  - Browser games: Chess, Sudoku running entirely client-side
  - Formatters: JSON/XML beautifiers, code formatters
- Key Characteristic
  - Run entirely in the browser & work offline once loaded
  - Don't require user accounts
  - Process data locally
  - Can be hosted on CDNs only
  - Update through redeployment, not database changes

Dynamic Web System - Core Characteristics

- Real-Time Content Generation
  - Pages assembled on-the-fly via server (SSR) OR browser (CSR)
  - Modern Single-Page Apps update content without full page reloads
  - API-driven architecture separates data from presentation
- Personalized & Contextual
  - Personalized experiences based on user profiles
  - Content varies based on user, time, context, and data state
  - Client-side state management for instant UI updates
- Interactive & Responsive
  - User actions trigger immediate responses
  - Asynchronous operations without blocking UI
  - Offline-first: local caching, sync when connected

Dynamic Web System - Core Characteristics

- Data-Driven Architecture
  - Persistent data storage (SQL/NoSQL databases)
  - Data relationships and complex queries
  - Real-time synchronization between client and server
- Often Cloud-Native & Scalable
  - Auto-scaling based on traffic demands
  - Microservices and containerized deployments
  - Serverless functions for event-driven computing
  - CDN distribution for global performance
- Mostly Continuously Evolving
  - Frequent updates through CI/CD pipelines
  - Feature flags for gradual rollouts
  - A/B testing and experimentation built-in
  - Infrastructure as Code for reproducible environments
  - Observable with real-time monitoring and analytics

Learning Objectives

- Build Full-Stack Production Systems with focus on backends
  - Design and implement complete web applications from frontend, database to cloud-native microservices
  - Design APIs and scalable web systems supporting millions of users
  - Design resilient systems with graceful error handling
  - Implement secure authentication, prevent vulnerabilities, and ensure GDPR compliance
- Apply Modern Industry-Standard Practices
  - Write comprehensive unit, integration, and end-to-end tests
  - Set up professional CI/CD pipelines and automated deployments
  - Configure A/B testing and feature flags for controlled rollouts
- Analytical & Design Skills
  - Apply scientific principles for performance prediction and system analysis
  - Model and simulate system behavior before implementation
  - Reason about architecture databases and caching strategies based on requirements
  - Balance trade-offs between performance, cost, and complexity

Overview

Week 1 – Build something to deploy

Lecture 1: Database and Backend Design Patterns

Hands-on Workshop 1: Practical Development and Testing

Week 2 – Getting Familiar with Kubernetes

Lecture 2: Modern Cloud-Native Platforms

Hands-on Workshop 2: Kubernetes & Cloud native development

Week 3 – Connect clients and services

Lecture 3: API Design & Communication

Week 4 – Build a production system

Lecture 4: CI/CD/GitOps, Observability & Monitoring

Hands-on Workshop 4: ArgoCD, A/B deployment on Kubernetes

Week 5 – Security

Lecture 5: Security

Week 6 – Advanced topics

Lecture 6: Real-time Web Systems

Lecture 7: AI Integration in Dynamic Web Systems

Week 8 - Final Project Presentations/Demo Day

December 15, on-demand

Week 1 – Build something to deploy

1. Database and Backend Design Patterns

- Backend
  - Monolithic vs Microservices
  - Service-Oriented Architecture (SOA)
  - Load balancing strategies
  - Circuit breakers and resilience patterns
  - Services meshes
- Data Management
  - SQL vs NoSQL
  - ACID vs BASE consistency models
  - Database scaling patterns (sharding, replication)
  - Event sourcing and CQRS
  - Client-side caching
  - Content-Delivery Networks (CDN)
  - Redis and in-memory databases
  - Cache invalidation patterns

Week 1 - Getting Familiar with Kubernetes

2. Modern Cloud-Native Platforms

- Docker & Containerization
  - Container fundamentals
  - Docker best practices
  - Multi-stage builds
- Kubernetes architecture
  - Pods, Services, Deployments
  - ConfigMaps and Secrets
  - Persistent storage in Kubernetes
  - Horizontal Pod Autoscaling
  - Ingress controllers and load balancing
  - Helm charts for application packaging
  - Kubernetes operators
- Serverless-Computing, FaaS

Week 3 - Connect clients and services

3. API Design & Communication

- REST principles and best practices
- Long-polling
- API versioning strategies
- Authentication patterns (JWT, OAuth 2.0)
- Rate limiting and throttling
- GraphQL vs REST
- WebSockets for real-time communication
- Server-Sent Events (SSE)
- Pub/Sub
- Message brokers (RabbitMQ, MQTT)
- gRPC

Week 4 - Build a production system

4. CI/CD/GitOps, Observability & Monitoring

- DevOps
  - GitHub Actions / GitOps ArgoCD/Flux
  - Automated testing strategies
  - Container image building and scanning
  - Deployment strategies (A/B, Blue-Green, Canary, Rolling)
- Optimization, Observability & Monitoring
  - The three pillars: Logs, Metrics, Traces
  - Distributed tracing with OpenTelemetry
  - Application performance monitoring and testing

Week 5 - Full-stack integration

5. Frontend Design Patterns

- Architectural Patterns (e.g. Model-View-Controller)
- Single-Page vs Multi-page design
- Progressive Web Apps
- State Synchronization
- Data Fetching Patterns
- Caching Patterns
- API Communication (REST, GraphQL etc.)

Week 5 - Full-stack integration

6. Security

- Authentication & Authorization
  - OAuth 2.0, JWT, OIDC
  - Multi-factor authentication
  - API key management
  - Zero-trust architecture
- Container & Kubernetes Security
  - Container security scanning
  - Kubernetes RBAC
  - Network policies
  - Pod Security Standards
  - Secrets management
- GDPR & Data Protection
  - Privacy by design
  - Data minimization and retention
  - User consent management
  - Right to be forgotten implementation

Week 6 – Advanced topics

7. Real-time Web Systems

- Immediate UI feedback
- WebRTC
- Collaborative Editing
  - Operational Transformation
  - Conflict-Free Replicated Data Types (CRDT)
- Presence & Awareness

Week 6 – Advanced topics

8. AI Integration in Dynamic Web Systems

- Personalization/Recommender System
- Collaborative Filtering
- AI/LLM chatbot integration
- Agentic AI and Model Context Protocol (MCP)
- Autonomous Task Planning and Execution

Lab Assignment: Build Your own Dynamic Web System

Students choose their own project idea, but must fulfill core technical requirements that align with learning objectives

Project Proposal (Week 1)

Students submit a 1-page proposal including:

- Problem they want to solve
- Core features
- Requirements
- Roadmap
- Target users
- Technology stack justification
- System architecture diagram
- How do demonstrate grading requirements

Example Project Ideas:

- Social platform for student organizations
- Collaborative study tool with real-time features
- Personal finance tracker with AI insights
- Recipe sharing platform with recommendations
- Fitness tracking with social challenges
- Local news aggregator with personalization
- Movie night planning with voting system
- Lecture note sharing with collaborative annotations

Lab Assignment: Build Your own Dynamic Web System

All projects MUST include these elements:

- Full-Stack Implementation *(ILO: Build dynamic web systems)*
  - Frontend (React/Vue/Angular - student choice) **NOTE: no focus on frontends!**
  - Backend API (Node.js/Python/Go - student choice)
  - Database (PostgreSQL/MongoDB - justified choice)
  - All deployed on Kubernetes
- API Design *(ILO: Create application-programming interface)*
  - RESTful API
  - Proper authentication (JWT/OAuth)

Lab Assignment: Build Your own Dynamic Web System

All projects MUST include these elements:

- System Design & Modeling *(ILO: Model, simulate, predict and evaluate)*
  - Architecture diagram (C4 model or similar)
  - Database schema design with relationships (TODO: page load time implications)
  - Performance analysis (load testing results)
  - Scalability plan (how to handle 100x users)
- Security & Ethics *(ILO: Ethical handling of sensitive data)*
  - Secure authentication implementation
  - Input validation and sanitization
  - HTTPS

Lab Assignment: Build Your own Dynamic Web System

All projects MUST include these elements:

- Production Readiness
  - CI/CD pipeline with GitHub Actions/GitLab CI
  - Automated tests (minimum 60% coverage on backend code, not UI)
    - Also, at least two endpoint test failures (e.g. user failed to login, or profile update can only be done by owner of the profile)
  - Monitoring with Prometheus/Grafana
  - ArgoCD for CI/CD/GitOps deployment TODO: Think about this:

Measure knowledge and not feature set,

Lab Assignment: Build Your own Dynamic Web System

Choose Advanced Features to improve grade

Option A: Real-Time Feature

- WebSocket implementation
- Live notifications
- Collaborative editing
- Real-time chat

Option B: AI Integration

- LLM chatbot integration
- Recommendation system
- Semantic search
- Content generation

Option C: Performance Optimization

- Caching strategy (Redis)
- CDN integration
- Database optimization

Option D: Advanced DevOps

- Grafana Monitoring
- Blue-green deployment
- Feature flags system
- Chaos engineering tests

Grading example

NOTE: This is not a design course, grade is determined by the architecture quality, security, scalability, and production-readiness of your dynamic web system, not the visual polish of your interface.

3 Grade Project

- Basic requirements met
- Simple monolithic architecture
- 60% test coverage
- Standard features only
- Adequate documentation
- Can explain basics in oral exam

4 Grade Project

- Well-structured architecture with clear separation of concerns
- More comprehensive testing (e.g. 70% test coverage including integration tests: More than 2 edge case testing.
- One advanced feature (AI, real-time, or advanced DevOps)
- Good documentation (README, API docs, architecture diagram)
- Solid oral defense - can explain and justify design decisions
- CI/CD pipeline with dev/prod environments

5 Grade Project

- Innovative technical idea solving real problem
- Clean architecture with microservices
- Two advanced features (e.g. AI + real-time)
- Excellent documentation (README, API docs, architecture diagrams, deployment guide)
- Strong oral defense with deep technical understanding
- Production-ready with monitoring, logging, and error handling
- Security best practices implemented beyond basic requirements, specifically RBAC
- Resilience

Support

- Students works work in pairs (2, max 3)
- Weekly Lab Sessions
- Open office hours for technical help (on demand)
- Canvas discussion forum (tell students can be anomomy, TODO: remember to configure Canvas correctly.)
- 4 Hands-on Workshop (Peer programming sessions)
- Kubernetes cluster access
  - 1 VM, 12 cores, 48GB memory, 300 GB NVMe
  - K3s and Rancher UI
- Tutorials, e.g. how to install Kubernetes

What’s next?

- Send in a Proposal
- Send Backlog link to Casper
- Recommended Gannt chart timeline


LULEÅ
UNIVERSITY
OF TECHNOLOGY