LUND
UNIVERSITY

UTVÄRDERING AV RAMVERK FÖR WEBBASERADE ANVÄNDARGRÄNSSNITT MED FOKUS PÅ TILLÄMPNINGAR INOM FINITA ELEMENTBERÄKNINGAR

ADAM PERSSON

Bachelor's Dissertation
at Structural Mechanics

DEPARTMENT OF CONSTRUCTION SCIENCES
DIVISION OF STRUCTURAL MECHANICS

ISRN LUTVDG/TVSM--26/6004--SE (1-32) | ISSN 0281-6679
BACHELOR'S DISSERTATION

UTVÄRDERING AV RAMVERK FÖR WEBBASERADE ANVÄNDARGRÄNSSNITT MED FOKUS PÅ TILLÄMPNINGAR INOM FINITA ELEMENTBERÄKNINGAR

ADAM PERSSON

Supervisor: **HENRIK DANIELSSON**, Assistant Professor, Div. of Structural Mechanics, LTH.
Assistant Supervisor: Dr **JONAS LINDEMANN**, Div. of Structural Mechanics, LTH | Lunarc.
Examiner: Professor **KENT PERSSON**, Div. of Structural Mechanics, LTH.

Printed by V-husets tryckeri LTH, Lund, Sweden, April 2026 *(Pl)*.

For information, address:
Div. of Structural Mechanics, LTH, Lund University, Box 118, SE-221 00 Lund, Sweden.
Homepage: http://www.byggmek.lth.se

Sammanfattning

I detta arbete undersöks möjligheten att införa ett webbaserat alternativ till den nuvarande Qt-baserade gränssnittsdelen i kursen VSMN20, Programutveckling för tekniska tillämpningar, vid Lunds Tekniska Högskola. Bakgrunden är en tydlig rörelse på programvarumarknaden mot webbaserade och molnbaserade lösningar, och frågan om hur man skulle kunna få in den förändringen i en kurs där studenter arbetar med mjukvaruutveckling med CALFEM för Python.

Syftet med arbetet är att identifiera det mest lämpliga ramverket för att skapa ett webbaserat gränssnitt till projektet i kursen, utan att göra det svårare för studenterna att uppnå kursens krav. För att göra detta valdes tre Python-baserade ramverk ut: Dash, NiceGUI, och Streamlit. Kursens slutprojekt återskapades i samtliga tre ramverk för att kunna jämföra dessa. Jämförelsen baserades på kriterier kopplade till pedagogik och komplexitet, robusthet och felsökbarhet, tillståndshantering, gränssnittets möjligheter, utvecklingsbarhet, driftsättning till molntjänster, och marknadsrelevans.

Resultatet visar att de studerade ramverken har olika styrkor och svagheter. Dash framstår som det mest strukturerade och etablerade alternativet, med ett arbetssätt som inte bara ligger närmare moderna komponentbaserade ramverk som React, utan även hur det nuvarande Qt-baserade ramverket är uppbyggt. Det erbjuder även starka möjligheter för vidareutveckling, tydlig uppdelning mellan olika delar av applikationen, och ett rikt ekosystem av komponentbibliotek. Samtidigt kräver Dash mer kod, mer explicit struktur, och en högre programmeringsförståelse. Detta gör att tröskeln blir högre i en kurs där programmering inte är det primära förkunskapskravet.

Streamlit framstår som det ramverk som snabbast och enklast låter utvecklaren komma till ett fungerande och presentabelt resultat. Dess arbetssätt är mer förenklat, mer förutbestämt, och därmed mindre likt de strukturer som återfinns i etablerade ramverk. Just denna förenkling är också en styrka i denna jämförelsen då det för utvecklaren blir en mindre tröskel att ta sig över för att uppnå målen.

NiceGUI placerar sig i flera avseenden mellan de andra två, men bedöms i detta arbete vara det svagaste alternativet på grund av ett mindre ekosystem och ett arbetsflöde som upplevs som hackigt och mindre smidigt.

Arbetet visar också att driftsättning till molntjänster är möjligt för samtliga tre ramverk, men att det kräver vissa kompletterande filer för att importera beroenden och systempaket. Streamlit Community Cloud togs fram som den med den enklaste lösningen, men är dock begränsad till Streamlit.

Om fokus ligger på låg tröskel för att det webbaserade alternativet ska nå samma svårighetsgrad som den nuvarande Qt ramverket, och Qt designer metoden, framstår Streamlit som det mest lämpliga alternativet. Men vid andra prioriteringar kan man komma fram till andra slutsatser. Om kursen i framtiden kan ge mer stöd i gränssnittsutveckling och höja den programmeringsmässiga nivån, skulle Dash kunna bli ett mer attraktivt val, särskilt med tanke på att dess styrkor ligger på långsiktiga pedagogiska

fördelar samt likhet med arbetsmarknadens arbetssätt.

Slutsatsen i detta arbete blir därför att Streamlit är det mest lämpliga alternativet för kursen i dess nuvarande form. Samtidigt pekar resultaten på att Dash också har tydliga kvaliteter som skulle kunna göra det relevant som ett alternativ. Ett naturligt nästa steg vore därför att låta ett webbaserat alternativ prövas i kursen i mindre skala och samla in återkoppling för att vidareutveckla upplägget.

Abstract

This study evaluates three Python-based frameworks for developing web applications for finite element analysis: Dash, Streamlit, and NiceGUI. The evaluation focused on interactivity, usability, and suitability for inclusion in a course at LTH about software development for technical applications. To compare the frameworks, the final course project was recreated in each of them while keeping the underlying computational model constant. The comparison was based on pedagogigal complexity, debugging, state-management, UI capabilities, maintainability, deployment, and relevance to current industry practices. The results indicate that Streamlit is the most suitable option for the course in its current form, mainly due to the lowered programming threshold and a faster path to a functional result. Dash showed stronger long-term benefits in pedagogical and market-related qualities, but required much more effort in development. NiceGUI was the weakest option in this context due to a less smooth development workflow and a less mature ecosystem. The study concludes that Streamlit is currently the most appropriate choice, while Dash could become more attractive if additional support was provided in the course.

Förord

Detta examensarbete genomfördes under våren 2026 vid Avdelningen för Byggnadsmekanik, Lunds Universitet.

Jag vill rikta ett varmt tack till min examinator Kent Persson samt mina handledare Jonas Lindemann och Henrik Danielsson för deras hjälp med att ge arbetet en riktning och deras stöd genom hela processen. Arbetet har gett mig möjligheten att undersöka en del av mjukvaruutveckling som jag inte har provat på tidigare, och har fördjupat mina kunskaper på ett sätt som jag känner kommer vara värdefullt i framtiden. Jag tackar därför er för ert engagemang och tillgänglighet när jag sökt er vägledning.

Jag vill även tacka de personer som ställde upp på intervju och bidrog med viktiga perspektiv till arbetet. Muhammed Najeem för hans syn på Dash och molntjänster, samt Pierre Olsson för hans insikter i hur Strusoft arbetar med övergången till webbaserade lösningar och hur detta förändringsarbete genomförs i praktiken.

Lund, Mars 2026.

Adam Persson

Innehåll

Sammanfattning **i**

Abstract **iii**

Förord **v**

1 Inledning **1**
1.1 Bakgrund . . . 1
1.2 Syfte . . . 1
1.3 Metod . . . 2
1.4 Avgränsning . . . 2
1.5 Användning av AI . . . 3

2 Teoretisk bakgrund **5**
2.1 Kurs . . . 5
2.2 CALFEM . . . 6
2.3 Marknadens rörelse . . . 6
2.4 Jämförelsekriterier . . . 7
2.4.1 Pedagogik och komplexitet . . . 7
2.4.2 Robusthet och felsökbarhet . . . 7
2.4.3 Interaktiv modell och tillståndshantering . . . 8
2.4.4 Gränssnittets möjligheter och visualisering . . . 8
2.4.5 Utvecklingsbarhet och kodkvalitet . . . 8
2.4.6 Driftsättning och marknadsrelevans . . . 8

3 Översikt av ramverk **9**
3.1 Dash . . . 9
3.1.1 Exempel Dash . . . 10
3.2 Streamlit . . . 11
3.2.1 Exempel Streamlit . . . 12
3.3 NiceGUI . . . 12
3.3.1 Exempel NiceGUI . . . 13
3.4 Molntjänster . . . 13

4 Jämförelse av ramverk **15**
4.1 Pedagogik och komplexitet . . . 16
4.2 Robusthet och felsökbarhet . . . 17
4.3 Interaktiv modell och tillståndshantering . . . 18
4.4 Gränssnittets möjligheter och visualisering . . . 18

4.5 Utvecklingsbarhet och kodkvalitet 19
4.6 Driftsättning och marknadsrelevans 19

5 Slutsatser **21**

Kapitel 1

Inledning

1.1 Bakgrund

Programvarumarknaden har under de senaste åren rört sig mot mer molnbaserade och webbaserade modeller än lokalt installerade program. Detta både för att undvika att behöva installera program på varje dator, och för att enklare kunna utesluta de som ej har behörighet att använda programmet. Det görs även i en uttryckt vilja att modernisera sin tekniska miljö [1]. Red Hat gjorde en undersökning där de kom fram till att 95% av de som svarade anser att applikationsmodernisering är viktigt för organisationens framgång [2]. I sin tur projicerade Gartner 2024 att 90% av organisationer, alltså företag och andra verksamheter, kommer att implementera en hybridmolnstrategi med sina program före 2027 [3]. Gartner kopplar de detta till AI-industrin. I denna snabbt ändrade marknad är det lätt för LTH:s kurser att falla efter. Med den bakgrunden kan det anses relevant att undersöka möjligheten att inkludera webbaserad mjukvaruutveckling i kurser för att förbereda studenterna på den nuvarande verkligheten när de lämnar universitetet.

Fokus på detta arbete har varit att finna webbaserade alternativ för kursen VSMN20, Programutveckling för tekniska tillämpningar. Kursen använder idag Qt, samt Qt designer för att ta fram en skrivbordsapplikation som kan beräkna och visualisera enkla finita element baserade problem med hjälp av CALFEM för Python [4].

1.2 Syfte

Arbetet syftar till att föreslå det mest lämpliga webbaserade ramverket för gränssnittsutveckling som kan användas i kursen VSMN20: Programutveckling för tekniska tillämpningar. För att komma fram till vilket som är mest lämpligt kommer flera olika gränssnittsbibliotek för webbapplikationer att jämföras. Ett grundläggande krav är att ramverken måste kunna hantera de uppgifter som finns i kursen på ett effektivt, intuitivt, samt estetiskt tilltalande sätt.

Arbetet har följande delmål:

- Implementera kursens slutprojekt i ett antal webbaserade ramverk.
- Analysera skillnader mellan de utvalda ramverken.
- Ta fram en modell för jämförelsen som är anpassad specifikt för kursen VSMN20.

- Jämföra de utvalda ramverken mellan varandra och med nuvarande Qt ramverk, samt med etablerade ramverk.

- Undersöka vilka möjligheter det finns för driftsättning av den färdiga webbapplikationen i olika molntjänster, samt andra sätt för den alternativa metoden i kursen att kännas attraktiv för studenter.

- Välja ut det ramverk som utgör det lämpligaste alternativet i kursen för studenter som vill använda en alternativ metod utan att göra det svårare att uppnå kursens krav.

- Undersöka vad som behöver ändras eller uppdateras i kursen eller i CALFEM för att förenkla övergången.

- Jämföra slutsatser med hur företag inom beräkningsprogramvaror utför sin förflyttning till webbaserade applikationer.

1.3 Metod

Arbetet består av ett antal praktiska moment samt en jämförande analys av dem. För att hitta möjliga ramverk utförs en webbstudie om vilka ramverk som skulle passa för uppgiften.

Efter att ramverken är utvalda, återskapas slutprojektet i kursen VSMN20 i alla tre ramverk för att få en tydlig jämförelse av hur väl de passar till kursen. I samband med detta tas även kriterier fram för att kunna göra en tydlig och objektiv jämförelse. Kriterierna väljs utifrån kurs och studentbehov.

För att förstå bakgrunden och arbetets plats i det större sammanhanget behövs en webbstudie kring marknadens rörelse mot webbaserad mjukvara. Ett led i detta är också en intervju med en representant för Strusoft för att få en lokal och mer personlig förankring till koncepten.

1.4 Avgränsning

En avgränsning i arbetet är att endast tre olika ramverk jämförs. Det finns ytterligare ramverk, men tre valdes ut med avseende på hur de representerar de vanligaste typerna för att inte göra arbetet för omfattande

Programutvecklingen i detta arbete utgår ifrån samma kod som utgjorde projektuppgiften i kursen VSMN20. Den representerar bra vilken typ av kod som studenterna tar fram under projektuppgiften. Samma uppgift hade kunna implementerats på ett mer optimalt sätt, men det är inte syftet med kursen.

På grund av att arbetet utgår från en befintlig kod som utvecklats av författaren är utgångspunkten kanske inte precis samma som för de studenter som går kursen. Detta kan orsaka en vinkling i mina jämförelser, samt att man kan missa saker som kan upplevas svårt för andra. Författaren har också en bakgrund i Teknisk Fysik, och ett antal programmeringskurser, vilket inte är samma bakgrund som majoriteten av de studenter som tar kursen. Detta kan till viss del påverka bedömningen av lämpligheten relaterat till målgruppen.

I kursen finns också tre olika alternativ. Spänningsberäkning, grundvattenströmning, eller värmesflödesproblem. I detta arbete utgår vi från spänningsexemplet som det implementerade exemplet.

1.5 Användning av AI

Under arbetets gång har generativ AI använts som ett verktyg i olika delar av processen. AI har använts i uppbyggnaden av koden för att lista vilka funktioner som behöver skrivas om, samt vilka widgets som man kan behöva användas i de olika ramverken. AI användes också i utvecklingsfasen för att kontrollera felmeddelanden. Det har också använts i första steget för framtagning av jämförelsekriterier och som ett bollplank. I skrivandet användes AI för källhantering, dubbelkontroll av språket, samt som hjälpmedel med LaTeX.

Kapitel 2

Teoretisk bakgrund

2.1 Kurs

Kursen VSMN20, Programutveckling för tekniska tillämpningar, vilken är fokus för detta arbete syftar till att ge studenterna förmågan att utveckla komplexa beräkningsprogram för tekniska tillämpningar, med grafiska gränssnitt.

Förkunskapskraven är några olika alternativa kurser i Finita elementmetoden eller Teknisk modellering: Bärverksanalys. Den har däremot inga förkunskapskrav för ren programmering mer än den grundläggande kursen i beräkningsprogrammering, vilket begränsar nivån av komplexitet. Det finns krav på att studenten ska kunna förstå problemet som utvecklas under kursens gång, men är också många studenters introduktion till större programutvecklingsprojekt. [5].

Under kursens gång går man från det mest grundläggande python-programet ”Hello, World!” till slutprojektet som innehåller nätgenerering, beräkning, och gränssnitt, bara några exempel. Ett grundvattenflödesproblem, ett plant spänningsproblem, eller ett värmeflödesproblem. Projektet byggs upp i steg vecka för vecka genom att mer och mer funktionalitet implementeras i varje arbetsblad. Geometrin definieras utifrån det valda exemplet, ett elementnät skapas för beräkning, förskjutningar beräknas, och i fallet med plan spänning beräknas sedan också spänningen i varje punkt. I ett av arbetsbladen utvecklar man sedan det grafiska gränssnittet med hjälp av Qt och Qt designer. Qt designer är då ett separat program som låter användaren skapa olika widgets och placera dem på ett fönster och sedan exportera en .ui fil som används i koden för att skapa gränssnittet i Python utan att skriva en rad kod. Namnen som ges i Qt designer för indatafält och komponenter kan sedan användas som om de skapats programatiskt. På så sätt kan man enkelt skapa så kallade *callbacks*, det vill säga funktioner som anropas när en viss händelse inträffar, för att hantera interaktion och spara värden.

I det sista momentet i projektet implementeras en funktion där man utför en parameterstudie. En av parametrarna varieras stegvis mellan två värden och en beräkning sker för varje steg. Varje beräkning sparas sedan som en .vtk fil som kan importeras i Paraview för att visualisera vad som händer med modellen när parametern ändras [6].

Examination i kursen sker genom en muntlig projektredovisning, samt kamratgranskning. Under presentationen förväntas man köra det program man skapat och förklara de olika delarna av koden, samt vad som händer när något ändras.

2.2 CALFEM

CALFEM står för Computer Aided Learning of the Finite Element Method och är ett bibliotek med funktioner för att formulera och lösa problem med Finita elementmetoden. Det har utvecklats vid avdelningen för byggnadsmekanik samt hållfasthetslära vid Lunds Universitet. Biblioteket har rötter tillbaks till 70-talet [7] men fortsätter utvecklas även idag. Under åren har det också översatts från olika programmeringsspråk ursprungligen från Fortran sedan från MATLAB till Python. Under många år har grundversionen varit en MATLAB-toolbox [8], men ett beslut fattades att översätta det till Python för att det ska vara mer tillgängligt. Python har nämligen, till skillnad från Matlab, öppen källkod. Python-versionen innehåller även stöd för meshing med Gmsh samt visualisering med Matplotlib [4]. På LTH har man också bytt introduktionsspråket i beräkningsprogrammering från MATLAB till Python, vilket innebär att fokus på Python version har ökat.

Anledningen att man utvecklade CALFEM var inte av någon kommersiell anledning, utan för att skapa ett verktyg som kan användas för att lära ut och förenkla implementeringen av FEM beräkningar så att studenter tydligt ser koppling mellan teori, modell, numerisk metod samt faktisk kod [7]. Källkoden och dokumentationen finns att ladda ner. Detta för att erbjuda ett transparent och öppet verktyg för studenten att ta hjälp av. Många av de verktyg studenterna på LTH kommer behöva under sina studier kommer med instruktioner och förklaringar. På detta sätt tillåter det alla att se och granska dess byggstenar, till skillnad från andra FEM-bibliotek. Detta hjälper studenter förstå FEM särskilt i implementationsfasen. CALFEM fyller ett syfte att fylla gapet mellan FEM-teorin och färdiga program ute i industrin.

2.3 Marknadens rörelse

Det finns flera anledningar till varför marknaden rör sig mot webbaserade applikationer snarare än skrivbordsbaserade. En av de stora är modernisering. IBM beskriver modernisering som att man tar existerande äldre applikationer och moderniserar deras infrastruktur, interna arkitektur, och dess verktyg. De menar också att genom modernisering, och genom distribution i molnet, fås en mycket snabbare funktionsutveckling och prestanda. De är också tydliga med att det inte alltid är positivt, då det ofta är en investering som behöver göras, men att med rätt strategi kommer man få en bra avkastning på investeringen (ROI). [9]. Tittar man på Red Hats rapport i samma ämne så verkar det tydligt att företagen de undersökt har kommit fram till att det är värt att göra. 95% av de som svarade på deras undersökning anser att modernisering av applikationerna är viktigt för organisationens framgång. De som svarade anser även att säkerhet, tillförlitlighet, och skalbarhet är de viktigaste anledningarna att modernisera på detta sätt. RedHat:s rapport kom även fram till att AI spelar en stor roll i detta, då 78% av de som svarade använder eller planerar att använda AI i deras moderniseringsarbete [2]. Samma trend kan ses i Gartners prognos. Den säger att 90% av företag och andra verksamheter kommer ha en molnhybridstrategi i sin struktur till 2027. Även de pekar på AIs påverkan på molntjänster och påpekar att den viktigaste utmaningen att bemöta är datasynkronisering[3].

För att sätta denna studie i en mer lokal kontext intervjuades Pierre Olsson, teamledare för mjukvaruutveckling på Strusoft. Strusoft är ett mjukvaruföretag med kontor i Malmö som fokuserar på mjukvaror inom byggnadsmekanik. I intervjun förklarar han

att även de ser moderniseringen som en viktig anledning till att de övergår till webbaserat i nästan alla av deras program, men även enhetlighet för programmen, samt för att göra användningen enklare för användaren. Beräkningar sker på en molntjänst driven av Strusoft istället för på en lokal dator och på detta sätt behöver användaren inte längre installera något program. Detta går fullt i linje med ett mål han uttrycker om att användaren snabbt och enkelt ska kunna plocka upp deras verktyg och förstå det utan att behöva en lång inlärningsperiod. Förenkling för användaren beskrivs också som deras största prioritering när de gör om sina program. Man vill kunna förenkla alla delar, även kanske introducera en AI som kan hjälpa med frågor, men utan att ta bort någon tidigare funktionalitet[1].

I deras implementering bygger de gränssnitt baserat på React, Node och TypeScript. Backend blir istället i C++ eller C#. Han beskriver ett system med microtjänster istället för ett monolitiskt. De har enligt honom ett ekosystem av kanske 10-15 API:er för olika delar som betong, trä eller stål.

Ett annat mål med moderniseringen är licenshantering. Genom molntjänsterna har de byggt en egen portal som identifierar användaren, utan att spåra plats eller IP, för att kontrollera den personliga licensen. Detta förenklar också för kunden, då all licenshantering nu faller på Strusoft själva [1].

2.4 Jämförelsekriterier

Kriterierna som togs fram utifrån kursen, studenter, samt kursansvarig delas upp i 6 kategorier. Under varje kategori finns några ämnen som bedöms. I detta skede är inget kriterium viktigare än något annat, utan hela jämförelsen görs för att få en utförlig studie.

2.4.1 Pedagogik och komplexitet

- Tydlighet i arbetsflöde
- Tydlighet i instruktion
- Kodmängd
- Kodkomplexitet
- Förenlighet med CALFEM:s pedagogiska mål

2.4.2 Robusthet och felsökbarhet

- Felhantering och felsökning
- Kraschrisk vid felhantering
- Felisolering

2.4.3 Interaktiv modell och tillståndshantering

- Eventmodell
- Tillståndshantering
- Styrning av uppdateringar

2.4.4 Gränssnittets möjligheter och visualisering

- Layoutkontroll
- Komponentutbud
- Plotintegrering

2.4.5 Utvecklingsbarhet och kodkvalitet

- Arkitektur och möjlighet att hålla isär olika element
- Möjlighet till kreativitet och frihet

2.4.6 Driftsättning och marknadsrelevans

- Hur enkel lösningen är att driftsätta, både lokalt och i moln
- Beroenden
- Jämförbarhet med marknadens verktyg
- Kompatibilitet med vanliga verktyg

Kapitel 3

Översikt av ramverk

Tre ramverk har valts ut i detta arbete för en jämförelse. Dessa är Dash, NiceGUI, och Streamlit. I följande avsnitt beskrivs var och en av dem för sig med ett kort exempel var som visar på skillnaderna. Ett urval av molntjänster gås också igenom.

3.1 Dash

Dash är ett open source-ramverk för Python, utvecklat av Plotly, för att bygga interaktiva webbapplikationer med grafiska gränssnitt [10]. Ramverket bygger på Flask på serversidan. Detta innebär att applikationen körs som en webbserver i Python och hanterar HTTP-förfrågningar och kommunikation mellan webbläsaren och applikationslogiken. Gränssnittet i webbläsaren använder Dashs egen frontend som är baserat på React [11]. React är ett JavaScript-bibliotek för att bygga användargränssnitt i webbläsaren genom återanvändbara komponenter [12]. Dash-komponenter är i grunden React-komponenter som exponeras så att de kan användas direkt med Python istället för att utvecklaren ska behöva skriva i JavaScript [13]. För visualisering använder Dash Plotly.js, ett JavaScript-bibliotek för interaktiva grafer i webbläsaren. Detta gör att grafer kan visualiseras och uppdateras dynamiskt [11]. På detta sätt kan Dash erbjuda ett rikt, interaktivt gränssnitt utan att utvecklaren behöver kunna skriva i HTML, CSS, eller JavaScript för att skapa några komponenter, (även om det är en möjlighet som ges ifall man vill ta fram egna komponenter). Kopplat till Dash finns flera utvecklade bibliotek med komponenter att importera som community-bibliotek, exempelvis Dash Mantine Components, för att enkelt få ett bredare komponentutbud som ser enhetligt ut [14].

Programmering i Dash är komponentbaserat och bygger på ett deklarativt upplägg där gränssnittet konstrueras genom att komponenter definieras i Python som en layout [15]. Interaktivitet inom programmet implementeras med hjälp av callbacks, där utvecklaren deklarerar samband mellan uppdaterande egenskaper, t.ex. klick, ändrade värden, eller andra egenskaper som då behöver uppdateras som ett resultat [16]. *Inputs* anger vilka indatavärden som utlöser en callback, *states* anger vilka värden som ska läsas vid körning, och slutligen anger *outputs* vilka komponentegenskaper som uppdateras. När en användare interagerar med gränssnittet skickas förändrade komponentvärden från webbläsaren till serversidan, vilket triggar de relevanta callbacks och resulterar i uppdateringar i webbläsaren. På detta sätt synkroniseras state mellan klient och server, och gränssnittet kan uppdateras dynamiskt. För att behålla data mellan interaktioner kan state lagras explicit i komponenter avsett för detta, t.ex.

`dcc.store`, som används för att lagra modellparametrar eller resultat som sedan används av andra callbacks [17]. Dash tillåter också att flera händelser hanteras i samma callback [18].

3.1.1 Exempel Dash

För att exemplifiera hur Dash fungerar visas här ett exempel med enkel knapp som räknar antalet gånger den klickas på. Detta är kanske inte det mest optimala sättet att utföra uppgiften, utan mer ett sätt utvalt för att visa på de egenskaper som beskrivits. Tanken är också att efterlikna koden i huvudprojektet. I nedanstående kod kan man se de callbacks, states, och store som beskrivits tidigare. Under koden finns även en illustration av hur gränssnittet ser ut.

```
from dash import Dash, html, dcc, Input, Output, State, callback
import dash_mantine_components as dmc

app = Dash(__name__)

app.layout = dmc.MantineProvider(
    [
        dmc.Stack(
            [
                dmc.Title("Exempel i Dash", order=4),
                dmc.Grid(
                    [
                        dmc.GridCol(
                            dmc.Button("Klicka här",
                            id="count-button", fullWidth=True),
                            span=3,
                        ),
                        dmc.GridCol(
                            "",
                            span=6,
                        ),
                        dmc.GridCol(
                            dmc.Text("Antal klick: 0",
                            id="count-text"),
                            span=3,
                        ),
                    ]
                ),
                dcc.Store(id="count-store", data=0),

            ],
            gap="md",
        )
    ]
)

@callback(
    Output("count-store", "data"),
```

```
        Output("count-text", "children"),
        Input("count-button", "n_clicks"),
        State("count-store", "data"),
        prevent_initial_call=True,
    )

    def update_count(n_clicks, current_value):
        new_value = current_value +1
        return new_value, f"Antal klick: {new_value}"

    if __name__ == "__main__":
        app.run(debug=True, port=8051)
```

Figur 3.1: Gränssnittet som exempelkoden med Dash skapade.

3.2 Streamlit

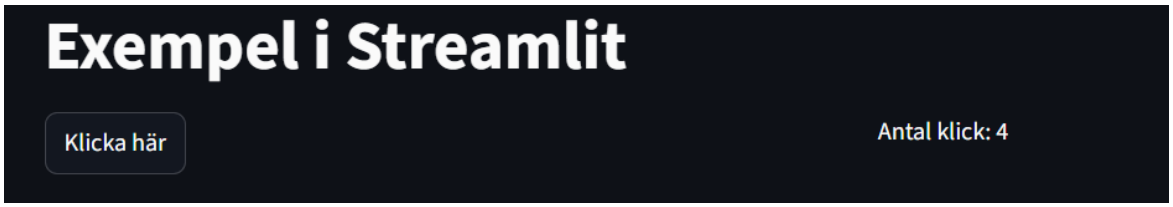
Streamlit är ett open source-ramverk för Python, utvecklat för att enkelt bygga webbapplikationer med grafiska gränssnitt där utvecklaren beskriver gränssnittet direkt i Python [19]. Applikationens serversida bygger på Tornado, ett Python-ramverk för webbtjänster och nätverkskommunikation [20]. Kommunikationen mellan servern och webbläsaren sköts av WebSockets, vilket är en teknik för en persistent tvåvägskanal som gör att servern kan skicka uppdateringar under körning [21]. Gränssnittet använder Streamlits egen frontend, som är implementerad med React och TypeScript. React är ett JavaScript-bibliotek för att bygga användargränssnitt [12], och TypeScript är en typsäker variant av JavaScript som används för större webbapplikationer [22]. Detta innebär att Streamlits inbyggda gränssnitts-komponenter renderas i webbläsaren, medan utvecklaren bygger applikationen genom Python-kod utan att normalt behöva skriva HTML, CSS, eller JavaScript [19]. Deras Custom komponenter kan däremot vara byggda med hjälp av valfri webbteknik, även om de erbjuder mallar till React och Typescript [21].

Programmering i Streamlit bygger på ett skriptbaserat och deklarativt arbetssätt. Gränssnittet skapas genom att koden körs uppifrån och ned och anrop beskriver vad som ska visas i webbläsaren [23]. När användaren interagerar med gränssnittet, genom att t.ex. ändra värden eller trycker på en knapp, körs då hela scriptet om [24]. Detta gör att gränssnittet genereras på nytt utifrån den aktuella data varje gång något ändras [23]. För att behålla data mellan omkörningar används `st.session_state` [25], där applikationen lagrar både modellparametrar, aktuella figurer, status, och text [26]. Komponenternas tillstånd kopplas till session state genom unika key-värden, och applikationslogik triggas antingen av callbacks eller genom rena funktionsanrop [24]. Flera av dessa triggers fungerar som engångsflaggor, t.ex. `st.button` [27] som ger ett booleskt värde (boolean) som beskriver vad som ska köras under nästa loop av scriptet [28].

3.2.1 Exempel Streamlit

Här under visas samma exempel som i Dash men skrivet i Streamlit för att visa upp dess skillnader. Exemplet är en knapp med en räknare som visar hur många gånger den klickats. Här ser vi tydligt hur den måste först kontrollera ifall `st.session_state.count` finns, och om inte så skapas det. Detta är för att varje gång knappen trycks körs scriptet om från toppen igen, och antalet kontrolleras då varje gång.

```
import streamlit as st

st.title("Exempel i Streamlit")

if "count" not in st.session_state:
    st.session_state.count = 0

col1, col2, col3 = st.columns([1,2,1])

with col1:
    if st.button("Klicka här"):
        st.session_state.count += 1

with col3: st.write(f"Antal klick: {st.session_state.count}")
```



Figur 3.2: Gränssnittet som exempelkoden med Streamlit skapade.

3.3 NiceGUI

NiceGUI är ett open source bibliotek för Python, utvecklat av Zauberzeug, för att bygga grafiska gränssnitt som körs i en webbläsare. I projektets dokumentation presenterar utvecklarna det som ett alternativ till Streamlit som de ansåg gör "för mycket magi" när det kommer till tillståndshantering [29]. Ramverket är byggt i grunden över FastAPI, ett Python-ramverk som driver serversidan och sköter kommunikationen mellan webbläsaren och koden [30]. Gränssnittet i webbläsaren är istället baserat på Vue och har Quasar som komponentbibliotek. Vue är ett javascript-ramverk för att bygga användargränssnitt i webbläsaren [31]. Quasar i sin tur är ett komponentbibliotek som använder Vue i grunden. Quasar har ett stort utbud av färdiga gränssnittskomponenter och layoutlösningar [32]. Detta innebär att NiceGUI kan erbjuda många app-liknande komponenter utan att utvecklaren behöver implementera dem själv med HTML och CSS.

Programmering i NiceGUI är komponentbaserad och händelsedriven. Gränssnittet byggs upp genom deklarerade komponenter direkt i Python [33]. Komponenter kopplas direkt till händelser, t.ex. klick eller ändrade värden, som anropar funktioner när användaren interagerar med gränssnittet [33]. Ramverket hanterar synkronisering mellan

server och klient, och gör så att komponenter uppdateras dynamiskt när applikationens state förändras [34]. State kan uttryckas genom komponenters aktuella värden, och vid behov, via inbyggd lagring om man vill behålla värden mellan interaktioner [35].

3.3.1 Exempel NiceGUI

Här under visas samma exempel för NiceGUI som för de andra två ramverken. Det är en knapp och en räknare som visar hur många gånger den klickats på. Här syns tydligt den direkta kopplingen mellan knappen och funktionen. Vi undviker därmed de callbacks som sågs i Dash.

```
from nicegui import ui

count = 0

def increase_count():
    global count
    count += 1
    count_label.set_text(f"Antal klick: {count}")

ui.label("Exempel i NiceGUI").classes("text-h6")

with ui.row().classes("w-full items-center gap-4"):
    ui.button("Klicka här", on_click=increase_count).classes("w-32")
    ui.label("").classes("grow")
    count_label = ui.label("Antal klick: 0").classes("w-64")

ui.run()
```

Figur 3.3: Gränssnittet som exempelkoden med NiceGUI skapade.

3.4 Molntjänster

I arbetet användes och utvärderades flera molntjänster för att se hur enkelt det är att driftsätta applikationerna till molnet. Molntjänster som passar våra ramverk är : Streamlit Community Cloud, Plotly Cloud, PyCafé, Render, och Hugging Face Spaces.

Streamlit Community Cloud är begränsat till endast Streamlit-program [36]. Där kopplar man sitt Github-konto och ett publikt repositorium för att kunna ladda koden till molnet [37]. En `requirements.txt`-fil krävs för att beskriva vilka pythonpaket som programmet är beroende av [38]. Utöver detta behövs även en `packages.txt`-fil som laddar in eventuella externa Linux-paket. En av anledningarna till att vi behöver externa Linux-paket är på grund av att CALFEM för Python kräver programmet, gmsh, som används för meshgenerering. `Gmsh` behöver utöver pythonpaket också underliggande systembibliotek i Linux-miljön. Dessa behöver specifikt anges för att programmet

ska fungera i molntjänsten. Streamlit Community Cloud är helt gratis för personligt bruk, och man får själv bestämma en egen URL [39].

Med Plotly Cloud, som Dash förespråkar, uppstod samma problem med gmsh. I deras gratisversion gick det inte att importera Linux-paketen [40]. På grund av detta är Plotly Cloud inte möjligt att använda i kursen, förrän det beroendet av Linux-paket förändras.

PyCafé har en möjlighet att skriva och driftsätta koden rakt ifrån fönstret oavsett vilket av ramverken som används. Det är dessutom helt gratis och har en enkel process för att driftsätta ett program. Problemen dyker upp i två fall. PyCafé har en pyodide-baserad miljö som kan installera rena Python-wheels från Pypi, men paket som inte har tillgänglig wheel kan inte alltid installeras [41]. Utöver dessa problem var en del paket föråldrade som t ex PyVTK som senast uppdaterades 2016. PyCafé har dessutom samma problem med gmsh, och inget sätt att externt importera Linux-paketen. Därför anses inte denna molntjänsten heller vara möjlig att använda i kursen.

Render är en tjänst som möjliggör driftsättning av ett stort utbud av olika Python-baserade webbapplikationer. För att använda den importeras koden från ett publikt eller privat GitHub-repositorium [42]. För denna applikation krävs som sagt extra systembibliotek, men Render detta löses genom att använda Docker. Man behöver utöver `requirements.txt` även en `Dockerfile` som beskriver för Render hur och vad den behöver installera till sin miljö [43]. Man behöver alltså manuellt installera de olika paketen som behövs. Render erbjuder helt gratis driftsättning men med automatisk nedstängning efter inaktivitet samt en uppstart som tar någon minut. Man har också max 750 timmar aktivitet per månad, men det lär inte uppnås under kursen oavsett [44].

Hugging Face Spaces är ytterligare en molntjänst som möjliggör driftsättning av webbapplikationer [45]. Man kan få en mall till både Streamlit [46] och Dash [47], samt flera andra ramverk, men inte NiceGUI. Genom den mallen kan man få en uppfattning om hur det fungerar, och det är strukturerat som ett GitHub-repositorium. Man kan skriva allt rakt på hemsidan om man vill. Samma filer som behövs i Render för att importera Pythonpaket samt de externa linuxpaketen behövs även här, skillnaden är att Hugging Face Spaces kräver en förutbestämd port för driftsättning [48]. Gratisversionen av Hugging Face Spaces har en gratis driftsättning med begränsad kapacitet. Om man betalar får man t.ex. en Dev mode som förenklar om man utvecklar appen på hemsidan, genom liveuppdatering bland annat [49].

Kapitel 4

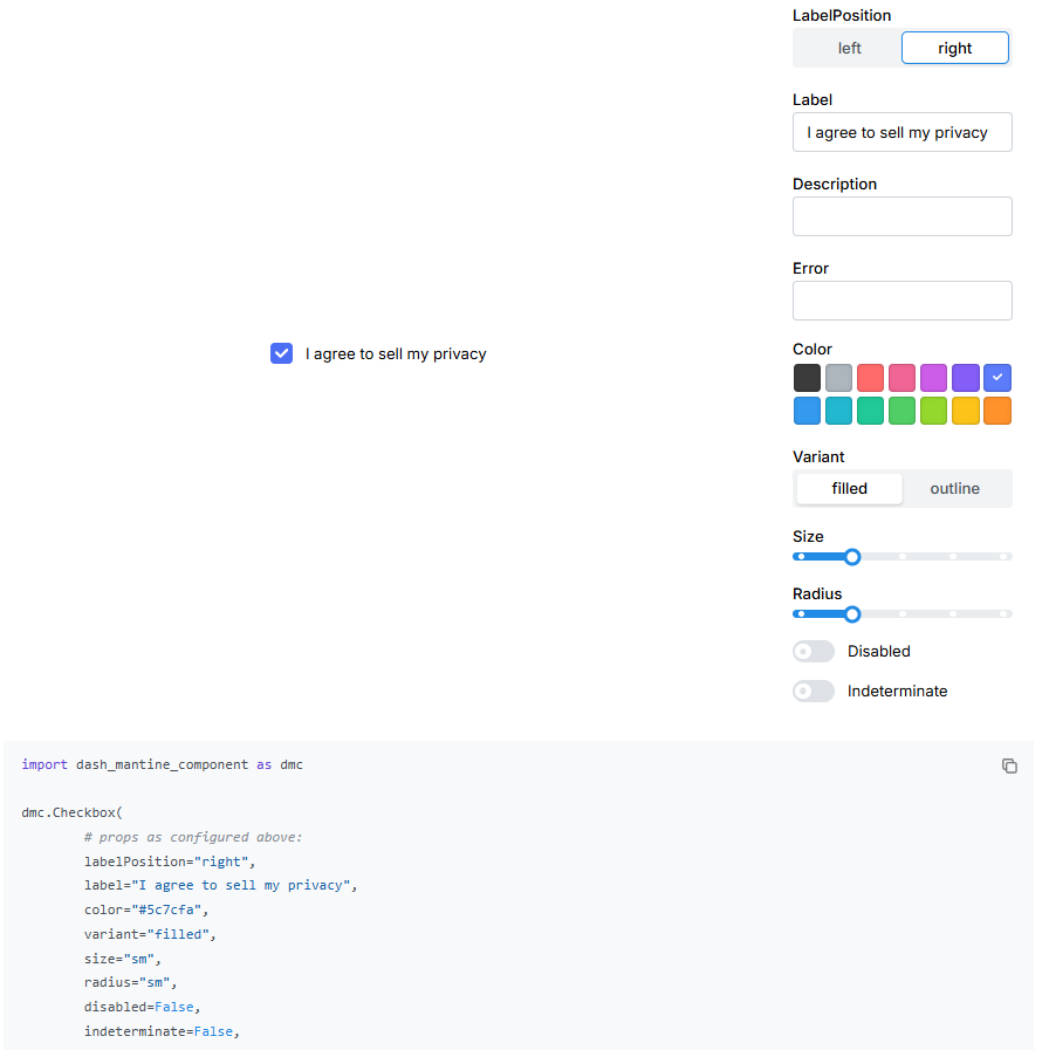
Jämförelse av ramverk

För att på bästa sätt representera de tre ramverken, har vi bytt visualiseringpaket från Matplotlib till Plotly. Anledningen är att kunna göra interaktiva visualiseringar i applikationen. Det sågs av skribenten som en viktig del av presentationen, särskilt med webbapplikationer. Denna förändring påverkar inte jämförelsen mellan de tre ramverken. Själva beräkningslogiken, och den gemensamma modellstrukturen bibehålls i en separat modul som kallas `PlaneStress.py`. Gränssnitt och anrop är det som ändras mellan ramverken och allt det sker i en egen applikationsfil `app.py`.

Jämförelsen mellan de tre ramverken görs enligt de kriterier som beskrevs tidigare.

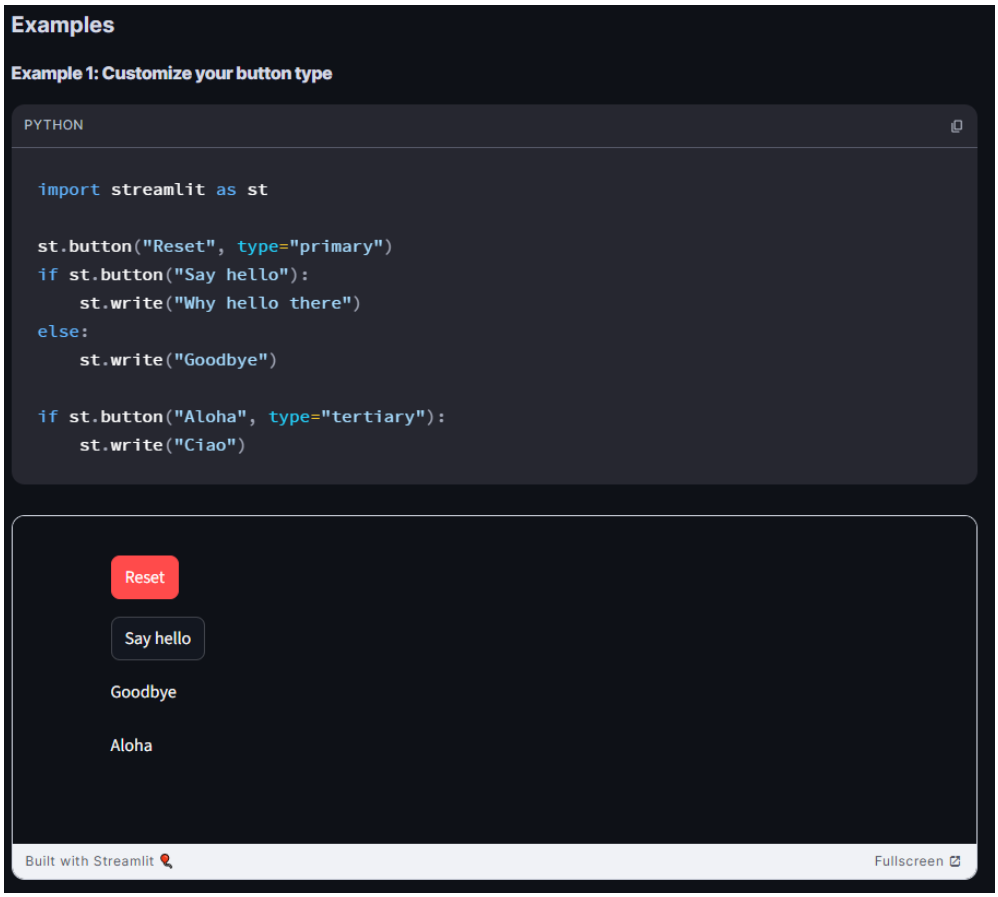
4.1 Pedagogik och komplexitet

En objektiv skillnad är mängden kod som måste skrivas. Mellan Dash och NiceGUI applikationerna skiljer över 150 rader kod, där Dash är den som har mest med sina 557 rader, och NiceGUI minst på 396 rader. Streamlit ligger däremellan på 434 rader. Även om koden inte är perfekt skriven, eller kompakt, blir skillnaden väldigt tydlig. För absoluta storleken på källkodsfilerna har Streamlit har 12799 tecken, NiceGUI 12464 tecken, och Dash har hela 19149 tecken för projektkoden [50]. En stor del av skillnaden kommer ifrån att Dash har ett annat sätt att hantera callbacks. Om man räknar raderna där callbacks deklareras i Dash uppnår det nästan 70 rader beroende på hur man räknar (och något tusen tecken). Men det är en tydlig skillnad i hur man skriver sitt gränssnitt. Dash kräver en mer explicit och detaljerad beskrivning av sin layout. Det går att få hjälp, t.ex. genom att Dash Mantine Components har interaktiva och justerbara widgets man kan kopiera koden från när man fått dem att se ut som man vill. Se figur 4.1 nedan för referens.



Figur 4.1: Exempel på interaktiva instruktioner i Dash Mantine Components.

De har utöver detta flera exempel samt utförlig dokumentation kring varje widget. Streamlit och NiceGUI har istället bara dokumentationen samt exempel. Se exempel från Streamlit i Figur 4.2 nedan:



Figur 4.2: Exempel från Streamlits dokumentation.

Interaktionen förenklar förståelsen för vad varje del av koden gör. Komplexiteten av koden är svårare att bedöma. Det är enkelt att dela upp det i Dash genom att ha en *main controller*, alltså en central styrande del, och centralisera alla sina callbacks till en plats, men det är inget krav. I Streamlit och NiceGUI skrivs däremot logiken oftare direkt när komponenten definieras eller händelsen hanteras. Då blir uppdateringen lokal och direkt synlig i den delen du skriver.

CALFEM för Python skapades, som skrivet tidigare i rapporten, som ett verktyg i utbildningen och för att förenkla lärandet speciellt med programmeringen med finita elementmetoden. Eftersom kursen är tätt kopplad till CALFEM är det viktigt att just förståelsen prioriteras.

4.2 Robusthet och felsökbarhet

Uppstart- och importfel riskerar att stoppa applikationen oavsett vilket ramverk som används, men skillnaden är tydligare vid interaktion med applikationen under körning. I Dash uppstår felen oftast i callbacks, och kan ge tydliga kopplingar till felet i webbläsaren efter ett fel, (iallafall om debug mode"är på). Streamlit kör om hela koden konstant vid varje ändring. Detta gör att ett litet lokalt fel lätt kan påverka hela programmets körning. Det är väldigt tydligt att något är fel, och deras felmeddelanden i webbläsaren dokumenterar klart var det avbröts, men det är mer abrupt än Dash. NiceGUI har inte samma direkta uppdatering som Dash och Streamlit har, så problem upptäcks inte förrän man sparat och kört om hela programmet från början

igen. Det gör att det kan ta längre tid att upptäcka problem. I detta arbetet upplevdes problemen ofta vara lokala i funktioner vilket gav en relativt enkel felsökbarhet.

4.3 Interaktiv modell och tillståndshantering

Eventmodellen mellan de tre olika ramverken är ganska olika från grunden.

Dash hanterar callbackskedjor i form av inputs, outputs och tillstånd, eller states, genom att inputs orsakar en förändring som gör att states läses in i funktionen så att man får en output. En knapptryckning innebär alltså att värden läses in, en funktion utförs, och outputs skickas vidare. Detta framgår av listan av callbacks.

I Streamlit, som tidigare nämnts, körs koden hela tiden, dvs om något ändras körs hela scriptet om från toppen. Är det ett event som händer, till exempel en knapp som aktiveras, ändras ett värde, t.ex. en boolean, för att göra en förändring i nästa omkörning. Man har `st.session_state` för att spara värden mellan omkörningar. Det blir en direkt uppdatering om du inte tvingar fram ett annat beteende. Man kan koda om i Streamlit manuellt för att tvinga den bete sig lite annorlunda och styra uppdateringsflödet genom att koppla det mer direkt till funktioner snarare än booleans och omkörningar.

NiceGUI har en mer direkt koppling mellan event och funktion. En funktion kopplas till exempel rakt till knapptryckningar, vilket gör att uppdateringslogiken ligger i själva komponenten. Man kan också tvinga NiceGUI att aktivt uppdatera en funktion varje gång värde ändras, till exempel för att visa det aktuella värdet i ett skjutreglage.

4.4 Gränssnittets möjligheter och visualisering

Dash har ett väldigt fritt layoutsystem genom sina komponenter, där man kan skriva exakta storlekar och struktur av varje komponent. När man använder deras komponentbibliotek däremot, som Dash Mantine eller Dash Bootstrap, förenklar de layouten och använder ett 12-kolumnsystem. Där varje komponent beskriver hur många av de 12 tillgängliga kolumnerna den tar upp. Man bestämmer alltså bredden av varje del genom att beskriva hur många 12:e-delar av sidbredden den ska använda, som vi ser i avsnitt 3.1.1. Detta gör att applikationen mycket enklare skalas om, om fönstrets storlek ändras.

Streamlit har ett liknande system i grunden där de använder sig av relationen mellan alla komponenters bredd. De sätter inte upp 12 kolumner specifikt, utan har istället en friare relation mellan komponenterna, som ses i avsnitt 3.2.1. Detta ger samma fördelar i skalning som för Dashs komponentbibliotek, men kan göra att man behöver räkna en extra gång för att få saker uppradade perfekt om den ena raden har fler komponenter som delar på platsen än den andra.

I NiceGUI har du möjligheten att välja själv vilket system du vill ha. I sin dokumentation framstår NiceGUI som mer stil- och klassbaserat. Det gör att exakta mått ser enklare ut och ligger nära till hands, detta görs tydligt i avsnitt 3.3.1. Det kräver någon extra rad kod för att implementera en relationsbaserad layout, men det går utan större problem.

När det kommer till komponentutbud är Dash den med det största ekosystemet med enhetliga komponenter. Detta beror främst på att de har etablerade paket med fulla komponentbibliotek i samma stil som passar ihop med enkla instruktioner som

Dash Mantine och Dash Bootstrap. Streamlit har Streamlit Extra som det närmaste liknande alternativet, men den är inte lika komplett. Annars har Streamlit ett stort bibliotek av community-skapade komponenter i sitt Components Gallery. Problemet är att få till enhetlighet, då de kan se lite olika ut om du tar komponent för knappar från en skapare och textrutor från en annan. NiceGUI har ett stort eget bibliotek av komponenter, vilket också är dess styrka, men har mindre variation i tillgängliga komponenter jämfört med de andra två om det är det man letar efter.

När det gäller plotintegrering fungerar de alla bättre med Plotly än med Matplotlib. Detta då Matplotlib endast erbjuder statiska figurer som inte går att interagera med på samma sätt som graferna från Plotly. På denna punkt är skillnaden mellan ramverken relativt liten.

4.5 Utvecklingsbarhet och kodkvalitet

Dash är bättre lämpat när man vill hålla isär olika funktionella element för bättre tydlighet. Layout, komponenter, och underliggande modell/logik kan betraktas helt separat från varandra med callbacks som kopplar ihop dessa. Callback-systemet gör det även tydligt vilken del av applikationen som uppdaterar vad, samtidigt kan det uppfattas som att strukturen känns tung om den inte organiseras noggrant.

Med Streamlit är det enklare att nå ett fullt fungerande resultat, utan att behöva lägga stor vikt på strukturen. Applikationen ser ut och känns som ett sammanhängande flöde, men samtidigt kan det vara svårt att strukturera och hålla isär olika elementen när projekten blir större. Det går att strukturera det, men det är svårt att få samma tydliga kopplingar som i Dash.

NiceGUI ligger i ett mellanläge. Funktioner kopplas ofta direkt till händelse vilket gör att strukturen kan vara tydlig när man går igenom den, även om man måste hoppa runt bland raderna. Det är också mycket beroende på hur utvecklaren lägger upp strukturen då ramverket ger utvecklaren stora friheter.

Alla ramverken erbjuder stora möjligheter till utveckling om man är villig att gå ett extra steg för det. Med tanke på att ramverken skall användas i en kurs kan det vara bra med ett ramverk som erbjuder möjligheter utan allt för många rader kod. Dash med sitt stora komponentbibliotek erbjuder detta. De har också en omfattande dokumentation med enkla interaktiva instruktioner som gör att man både förstår vad koden gör och kan justera allt från form och marginal till färg. Streamlit har ett stort bibliotek av komponenter skapade av användare i mycket varierande stil. Det skapar möjligheten för utvecklaren att hitta en skapare vars stil passar just dig.

Sammanfattningsvis framstår Streamlit som det mer färdiga arbetssättet av de tre, medan Dash och NiceGUI ger mer möjlighet att göra om applikationen till precis det du vill ha. Men det kommer ofta med en del extra arbete för att nå samma enkelhet i utvecklingsflödet.

4.6 Driftsättning och marknadsrelevans

Den senaste versionen av Streamlit och NiceGUI har ett beroende av åtminstone Python 3.10 [51] [52], medan Dash är mer bakåtkompatibelt med stöd för version 3.8 [53]. Det innebär att det inte krävs att alla paket är av senaste version. Python 3.10 släpptes i slutet av 2021, och version 3.8 släpptes i slutet av 2019.

Angående driftsättning undersöktes flera alternativ. I praktiken var det bara tre aktuella molntjänster, och en av dem fungerar endast för Streamlit. Streamlit Community Cloud var enklast att arbeta med, vilket gör den speciellt intressant. De andra två molntjänsterna var Render och Hugging Face Spaces. Mellan dessa två skiljer sig inte arbetsättet särskilt mycket. Mallar finns att utgå ifrån för Dash och Streamlit. Detta kan vara bra att ta med sig i en kurssituation för en bättre förståelse och användbarhet. Alla tre ramverken går att sätta på en molntjänst. Det enda problemet som dyker upp är att lista upp beroende för Linux-paket som krävs för speciellt CALFEM.

Med marknadsrelevans menas hur likt arbetsprocessen och strukturen är det som studenter kan stöta på ute i arbetslivet. Det som jämförs med är då att implementera ett gränssnitt i React. Då används ett annat programmeringsspråk, JavaScript, men strukturen och arbetssätt är ändå jämförbara. React har ett komponentbaserat och deklarativt arbetssätt [54]. Av de tre ramverken i denna studie är det framförallt Dash som ligger närmast detta arbetssätt, genom komponentbaserade uppbyggnad, callbacks, och inputs, state och outputs. NiceGUI är också ganska likt med sina direkta händelsefunktioner, men Streamlit skiljer sig tydligt i arbetssätt. Streamlit, med sin loop-baserade skriptmodell har en struktur som skiljer sig mest bland ramverken och skiljer också mest mot vanliga förekommande modeller i verkligheten. Det hänger samman med att ramverket i huvudsak är utformat för att förenkla utvecklingen av Pythonbaserade webbapplikationer.

Kapitel 5

Slutsatser

Streamlit är det ramverk som utifrån jämförelserna bedöms vara mest lämpat att använda i kursen. En utvecklare kan mycket enkelt och snabbt nå ett fungerande och presentabelt gränssnitt. Dess loop-baserade skriptmodell kan begränsa det för större projekt. Avsaknaden av callbacks eller en lika explicit deklarativ struktur gör också att den är svår att direkt koppla till React, som är mer vad marknaden bygger sina applikationer med, eller till Qt som den nuvarande kursen använder. Men i ljuset av att kursen är en fortsättningskurs i mekanik med begränsade förkunskapskrav inom programutveckling är det svårare att rekommendera de andra två, eftersom de ställer högre krav på programmeringskunskaper.

Samtidigt framstår Dash i flera avseenden som ett starkare mer långsiktigt inlärningsverktyg, vilket är intressant då syftet med detta arbete är relaterat till kursutveckling och undervisning. Dash med sitt stora komponentbibliotek, dokumentation, sina många guider, men också just för att det är mer likt de verktyg som studenterna mer sannolikt kommer använda ute i arbetslivet framöver. Strukturen med callbacks är även mer likt Qt som används i den nuvarande kursen. Det kräver dock en hel del mer tid och ansträngning för att utveckla en lösning som håller samma kvalitet i kursen som Streamlit kan erbjuda. Särskilt då den nuvarande metoden med Qt och Qt Designer innebär att studenterna kan använda verktyg för att rita upp gränssnittet utan att skriva kod.

NiceGUI framstår i jämförelse med de andra två svagare. Ramverket har inte samma typ av community kring vidareutvecklade komponenter och verktyg. Ramverkets arbetssätt gjorde det också svårare att skapa motsvarande gränssnitt i detta arbete. Programmet avbröts ofta när filer sparades och att sidan då behövde laddas om gör att arbetsflödet blir mer oförutsägbart, särskilt man försöker bygga ett gränssnitt. Det fanns alltså ett mindre utbud av verktyg, samtidigt som de tillgängliga verktygen i högre grad medförde problem.

Dash skulle kunna vara det mest lämpliga alternativet om studenterna fick mer anpassade exempel och funktioner att använda. Det är mindre låst och förenklat än Streamlit, och därmed friare samtidigt som dess dokumentation och guider är bättre. Om tidsskillnaden i utvecklingsfasen mellan Streamlit och Dash kunde minskas skulle Dash därför kunna vara att föredra. Detta förutsätter dock en sådan grad av förenkling att vissa av de egenskaper som gör Dash pedagogiskt och tekniskt intressant riskerar att gå förlorade. Om ramverket struktureras upp för mycket, kan man till slut fråga sig om man inte lika gärna borde använda den enklare, men mer begränsade, modellen som Streamlit redan erbjuder.

Om man tar in molntjänsterna i bedömningen så erbjuder Streamlit Community Cloud ett bättre alternativ än de andra i undersökningen, då det var enklare att koppla till GitHub och inte hade krav på att lära sig vad en Dockerfile var. Streamlit Community Cloud kan enbart använda Streamlit-baserade appar, så det ger ytterligare en fördel för detta ramverk. Denna tjänst upplevdes också ge tydligare felmeddelanden för att bättre kunna analysera fel i applikationen. Oavsett vilken man väljer kan kursansvarig presentera färdiga filer för beroenden och paket som behöver laddas ner. Studenten kan erhålla en färdig Dockerfile om man väljer Render eller Hugging Face Spaces. Detta är ett viktigt steg i att förenkla processen med driftsättning.

En viktig del att ha i åtanke är att bedömningen oundvikligen innehåller kvalitativa inslag. Även om vissa delar kan stödjas med mer konkreta mått, som antal rader eller tecken, bygger flera av kriterierna på avvägningar kring estetik, enkelhet, och arbetsflöde. Dessa framstår som svåra att bedöma helt objektivt utan större undersökningar eller testomgångar med studenter i kursen.

Andra slutsatser som kan dras från arbetet är att om kursen ska erbjuda ett spår för webbaserade applikationer kan det vara bra att implementera en uppritningsmodul i CALFEM för Plotly motsvarande vis_mpl för Matplotlib.

Arbetet visar också att företags prioriteringar när de flyttar sina skrivbordsapplikationer till webben och till molnet ofta handlar om modernisering och användarens upplevelse, medan i detta arbete prioriteras utvecklarens och studentens upplevelse. Både med avseende på hur bra det fungerar som läroverktyg samt om det blir svårare eller mer tidskrävande än den nuvarande metoden. Även om det ena påverkar det andra, är därför olika ramverk och verktyg värdefulla.

Sammantaget pekar arbetet därför mot att Streamlit är det mest lämpliga alternativet för kursen i dess nuvarande form, främst på grund av den lägre tröskeln för att uppnå ett fungerande resultat. Samtidigt visar arbetet också att Dash har tydliga styrkor, särskilt ur ett långsiktigt perspektiv med pedagogik och marknadsrelevans. Om kursen kan ge mer stöd i själva gränssnitt-utvecklingen och höja ambitionsnivån, skulle Dash kunna bli det mer attraktiva alternativet i framtiden.

Ett naturligt nästa steg vore därför att låta ett sådant webb-baserat alternativ testas i kursen i mindre skala, helt frivilligt, och samla in studenternas återkoppling, för att vidare kunna justera upplägget och identifiera den mest lämpliga modellen för framtiden.

Referenser

[1] Pierre Olsson. *Intervju om webbaserade ramverk, och moderniseringen som sker på Strusoft.* Intervju utförd av Adam Persson, 2026-02-04. 2026.

[2] Red Hat. *The State of Application Modernization*. Resource type: E-book. Juni 2024. URL: https://www.redhat.com/en/resources/app-modernization-report (hämtad 2026-03-17).

[3] Gartner. *Gartner Forecasts Worldwide Public Cloud End-User Spending to Total $723 Billion in 2025*. Nov. 2024. URL: https://www.gartner.com/en/newsroom/press-releases/2024-11-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-total-723-billion-dollars-in-2025 (hämtad 2026-03-17).

[4] Division of Structural Mechanics, Lund University. *CALFEM for Python 3.6.12 documentation*. 2026. URL: https://calfem-for-python.readthedocs.io/en/latest/ (hämtad 2026-03-17).

[5] Lunds Tekniska Högskola. *Kursplan för VSMN20*. Kursplan för läsåret 2025/26. 2025. URL: https://kurser.lth.se/lot/course-syllabus/25_26/VSMN20 (hämtad 2026-03-17).

[6] Jonas Lindemann. *Software Development for Technical Applications - VSMN20*. Kurswebbplats med arbetsblad och projektinformation. 2026. URL: https://vsmn20.readthedocs.io/sv/latest/ (hämtad 2026-03-17).

[7] Division of Structural Mechanics, Lund University. *CALFEM*. Accessed on the website of the Division of Structural Mechanics, LTH. 2017. URL: https://www.byggmek.lth.se/english/resources/software/calfem/ (hämtad 2026-03-17).

[8] Division of Structural Mechanics, Lund University. *calfem-3.6-manual-full.pdf*. MATLAB manual hosted on GitHub. 2026. URL: https://github.com/CALFEM/calfem-matlab/blob/master/calfem-3.6-manual-full.pdf (hämtad 2026-03-17).

[9] IBM. *What is application modernization?* 2026. URL: https://www.ibm.com/think/topics/application-modernization (hämtad 2026-03-17).

[10] Plotly. *Dash Documentation & User Guide*. 2026. URL: https://dash.plotly.com/ (hämtad 2026-03-18).

[11] Plotly. *plotly/dash*. GitHub repository for Dash. 2026. URL: https://github.com/plotly/dash (hämtad 2026-03-18).

[12] Meta Platforms, Inc. *React*. 2026. URL: https://react.dev/ (hämtad 2026-03-18).

[13] Plotly. *Writing Your Own Components*. Dash for Python documentation. 2026. URL: https://dash.plotly.com/build-your-own-components (hämtad 2026-03-18).

[14] Dash Mantine Components. *Dash Mantine Components Documentation*. 2026. URL: https://www.dash-mantine-components.com/ (hämtad 2026-03-18).

[15] Plotly. *Part 1. Layout | Dash for Python Documentation*. 2026. URL: https://dash.plotly.com/layout (hämtad 2026-03-18).

[16] Plotly. *Basic Callbacks | Dash for Python Documentation*. 2026. URL: https://dash.plotly.com/basic-callbacks (hämtad 2026-03-18).

[17] Plotly. *Part 4. Sharing Data Between Callbacks | Dash for Python Documentation*. 2026. URL: https://dash.plotly.com/sharing-data-between-callbacks (hämtad 2026-03-18).

[18] Plotly. *Determining Which Callback Input Changed | Dash for Python Documentation*. 2026. URL: https://dash.plotly.com/determining-which-callback-input-changed (hämtad 2026-03-18).

[19] Streamlit. *Streamlit*. 2026. URL: https://streamlit.io/ (hämtad 2026-03-18).

[20] Tornado. *Tornado Web Server*. 2026. URL: https://www.tornadoweb.org/en/stable/ (hämtad 2026-03-18).

[21] Streamlit. *Understanding Streamlit's client-server architecture*. 2026. URL: https://docs.streamlit.io/develop/concepts/architecture/architecture (hämtad 2026-03-18).

[22] TypeScript. *TypeScript: JavaScript With Syntax For Types*. 2026. URL: https://www.typescriptlang.org/ (hämtad 2026-03-18).

[23] Streamlit. *Basic concepts of Streamlit*. 2026. URL: https://docs.streamlit.io/get-started/fundamentals/main-concepts (hämtad 2026-03-18).

[24] Streamlit. *Widget behavior | Streamlit Docs*. 2026. URL: https://docs.streamlit.io/develop/concepts/architecture/widget-behavior (hämtad 2026-03-18).

[25] Streamlit. *Session State | Streamlit Docs*. 2026. URL: https://docs.streamlit.io/develop/api-reference/caching-and-state/st.session_state (hämtad 2026-03-18).

[26] Streamlit. *Add statefulness to apps | Streamlit Docs*. 2026. URL: https://docs.streamlit.io/develop/concepts/architecture/session-state (hämtad 2026-03-18).

[27] Streamlit. *st.button | Streamlit Docs*. 2026. URL: https://docs.streamlit.io/develop/api-reference/widgets/st.button (hämtad 2026-03-18).

[28] Streamlit. *Button behavior and examples | Streamlit Docs*. 2026. URL: https://docs.streamlit.io/develop/concepts/design/buttons (hämtad 2026-03-18).

[29] NiceGUI. *Why NiceGUI?* 2026. URL: https://nicegui.io/#why (hämtad 2026-03-18).

[30] FastAPI. *FastAPI Documentation*. 2026. URL: https://fastapi.tiangolo.com/ (hämtad 2026-03-18).

[31] Vue.js. *Introduction | Vue.js*. 2026. URL: https://vuejs.org/ (hämtad 2026-03-18).

[32] Quasar Framework. *Introduction to Quasar*. 2026. URL: https://quasar.dev/introduction-to-quasar/ (hämtad 2026-03-18).

[33] NiceGUI. *Action & Events | NiceGUI Documentation*. 2026. URL: https://nicegui.io/documentation/section_action_events (hämtad 2026-03-18).

[34] NiceGUI. *Binding Properties | NiceGUI Documentation*. 2026. URL: https://nicegui.io/documentation/section_binding_properties (hämtad 2026-03-18).

[35] NiceGUI. *Storage | NiceGUI Documentation*. 2026. URL: https://nicegui.io/documentation/storage (hämtad 2026-03-18).

[36] Streamlit. *Streamlit Community Cloud*. 2026. URL: https://docs.streamlit.io/deploy/streamlit-community-cloud (hämtad 2026-03-18).

[37] Streamlit. *Connect your GitHub account*. 2026. URL: https://docs.streamlit.io/deploy/streamlit-community-cloud/get-started/connect-your-github-account (hämtad 2026-03-18).

[38] Streamlit. *Manage your app*. Includes information about requirements.txt and packages.txt. 2026. URL: https://docs.streamlit.io/deploy/streamlit-community-cloud/manage-your-app (hämtad 2026-03-18).

[39] Streamlit. *Deploy your app on Community Cloud*. 2026. URL: https://docs.streamlit.io/deploy/streamlit-community-cloud/deploy-your-app/deploy (hämtad 2026-03-18).

[40] Plotly. *Plotly Pricing*. 2026. URL: https://plotly.com/pricing/ (hämtad 2026-03-18).

[41] PyCafe. *Building Python packages*. 2025. URL: https://py.cafe/docs/howto/build (hämtad 2026-03-18).

[42] Render. *Render*. General platform information. 2026. URL: https://render.com/ (hämtad 2026-03-18).

[43] Render. *Deploying with Docker*. 2026. URL: https://render.com/docs/docker (hämtad 2026-03-18).

[44] Render. *Deploy for Free*. 2026. URL: https://render.com/docs/free (hämtad 2026-03-18).

[45] Hugging Face. *Spaces*. 2026. URL: https://huggingface.co/docs/hub/spaces (hämtad 2026-03-18).

[46] Hugging Face. *Streamlit Spaces*. 2026. URL: https://huggingface.co/docs/hub/spaces-sdks-streamlit (hämtad 2026-03-18).

[47] Hugging Face. *Dash on Spaces*. 2026. URL: https://huggingface.co/docs/hub/spaces-sdks-docker-dash (hämtad 2026-03-18).

[48] Hugging Face. *Docker Spaces*. 2026. URL: https://huggingface.co/docs/hub/spaces-sdks-docker (hämtad 2026-03-18).

[49] Hugging Face. *Hugging Face Pricing*. 2026. URL: https://huggingface.co/pricing (hämtad 2026-03-18).

[50] Adam Persson. *Python-Web-Frameworks*. GitHub repository. 2026. URL: https://github.com/Rashio97/Python-Web-Frameworks/tree/main (hämtad 2026-03-24).

[51] Python Package Index. *nicegui*. 2026. URL: https://pypi.org/project/nicegui/#data (hämtad 2026-03-18).

[52] Python Package Index. *streamlit*. 2026. URL: https://pypi.org/project/streamlit/#data (hämtad 2026-03-18).

[53] Python Package Index. *dash*. 2026. URL: https://pypi.org/project/dash/#data (hämtad 2026-03-18).

[54] React. *Thinking in React*. 2026. URL: https://react.dev/learn/thinking-in-react (hämtad 2026-03-18).