Bavarian Graduate School of Computational Engineering

Technical University of Munich

BGCE Honours project report

Interactive preCICE Online Tutorial

Authors: Hasan Ashraf,
Pei-Hsuan Huang,
Felix Lachenmaier,
Kirill Martynov,
Dmytro Sashko,
Jan Sültemeyer,
Advisors: Benjamin Uekermann TUM, (Topic Advisor)
Friedrich Menhorn TUM, (Team Advisor)

Preface

The Bavarian Graduate School of Computational Engineering's (BGCE) honours project is a 10-month project where students conduct research on cutting-edge topics in the field of Computational Engineering, in cooperation with a partner in industry or academia. The BGCE program is funded by the Elite Network of Bavaria and includes students selected from - but not exclusively - the International Master's program in Computational Science and Engineering (CSE) at the Technical University of Munich. The 2017-18 project was titled *Interactive preCICE Online Tutorial* and was conducted and supervised in a cooperation between TUM and the University of Stuttgart.

Acknowledgments

First of all, we would like to thank our supervisors, Dr. rer. nat. Benjamin Uekermann and Friedrich Menhorn M.Sc. (hons) from TUM for their constant guidance and support. We are also grateful to Prof. Dr. rer. nat. habil. Miriam Mehl and Dipl.-Ing. Florian Lindner from the University of Stuttgart for their valuable help and cooperation. Finally, we would like to thank the Bavarian Graduate School of Computational Engineering and, in particular, its director Prof. Dr. rer. nat. habil. Hans-Joachim Bungartz for providing us the opportunity to work on this project.

Contents

Outline and Overview

Purpose of The Document

The purpose of this document is to describe the main outcome of our project - the *Interactive preCICE Online Tutorial* website - along with the technology it relies on.

Document Overview

The document provides a timeline view of the project. This means that we will lead you through the evolution of the project, from the basic architecture and technologies we employed to the multiple design iterations that led to the final product. The contents of the individual chapters are summarized below.

CHAPTER 1: INTRODUCTION
To begin with, we provide motivation for the project and introduce the concrete objectives that we set out to achieve along with key organizational details.

CHAPTER 2: WEB TECHNOLOGY STACK AND SERVER ARCHITECTURE
This chapter describes the technology stack we used for creating the website. We provide a survey of available web technologies, discuss our design decisions in light of the requirements, and finally present key features of the technologies we used and the architecture we settled on.

CHAPTER 3: HOSTING AND OTHER PRODUCTION CONSIDERATION
Here we list potential hosts for the website and the licenses for the packages we used for the website implementation. Furthermore, we discuss where the current backend is hosted and provide our thoughts on whether this might need to be revised in the future.

CHAPTER 4: PRECICE COUPLING LIBRARY
Before delving into the concrete details of the project, we present a short description of the coupling library preCICE. Here we only discuss aspects of preCICE that are relevant to our tutorial.

CHAPTER 5 : TUTORIAL TEST CASE
This chapter introduces the test case for the interactive tutorial and describes the used simulation software and the corresponding installation processes. Here, we lead you through how we developed the test case and provide a list of problems that we encountered while installing the required software.

CHAPTER 6 : SIMULATION: FIRST ITERATION
The first version of the FSI simulation is introduced in this chapter. This includes separate single-physics fluid- and structure simulations as well as the coupled simulation that was built on top of the former.

CHAPTER 7 : SIMULATION: SECOND ITERATION

A completely redesigned version of the FSI simulation is introduced to counteract runtime and convergence issues with the first version. We describe the approaches we explored to decrease the runtime of the simulation and measures we took to achieve a convergent simulation.

CHAPTER 8 : SIMULATION: THIRD ITERATION

In this chapter we describe the final version of the simulation that was used for the interactive tutorial. This version builds on the work introduced in Chapter 7.

CHAPTER 9 : USER STORY

The user story was developed in conjunction with the different versions of the simulation. Here we talk about the multiple versions of the user story we came up with and implemented on the website.

CHAPTER 10 : OFFLINE TUTORIAL

As a first step towards an interactive tutorial, a static tutorial was developed and added to the *github-wiki* of the preCICE project. This chapter describes the evolution of this offline tutorial.

CHAPTER 11: WEBSITE: FIRST ITERATION

The first version of the website consisted of mock web pages which were meant to serve as a proof-of-concept. We came up with a basic architecture for the website which persisted through all three design iterations. This chapter describes this work in detail.

CHAPTER 12: WEBSITE: SECOND ITERATION

The second version of the website added interactivity, such as browser based consoles and a user story based on playing with the Aitken relaxation parameter. This chapter takes as foundation the work described in Chapter 11.

CHAPTER 13: WEBSITE: THIRD ITERATION

Here we describe the final version of the website along with the interactive tutorial. We talk about major bug fixes, the final user story, and features developed in the second iteration that did not make the cut.

CHAPTER 14: USER TESTING

To test the final version of the website, we conducted extensive user testing. In this chapter, we present our user testing strategy and summarize the data collected from the users. We further discuss how this feedback led to improvements on the website.

CHAPTER 15: CONCLUSION

Last but not least, we reflect on the things we achieved and mention major directions for future work on the project.

1 Introduction

1.1 Motivation

preCICE is an open-source library that allows users to couple existing simulation codes and run multiphysics simulations. It is currently being developed at the Technical University of Munich (TUM) and the University Stuttgart.
One of the current objectives of preCICE developers is to introduce the software to a wider range of customers from both industry and academia. In order to reach this goal, potential users of preCICE need to get a simple general overview of the software and a first idea about how to use the library. This is where this honours project comes into the picture. The goal of this project was to create an interactive online tutorial that allows users to get familiar with preCICE and demonstrates its capabilities. The tutorial aims to help attract new users to the library and capture their interest.

1.2 Project Structure

Let us begin by presenting the structure of the project. We define the goals that we set out to achieve and the expected outcome of the project, and provide the organizational outline – from a team oriented as well as a time oriented perspective.

1.2.1 Aims and Goals

In the beginning, the client provided us with a list of requirements that the project had to fulfill. While our focus shifted over the course of the project, the core requirements remained the same.

Essential Deliverables

The following requirements had to be satisfied:

- **Website:** Develop a minimal interactive online tutorial that realizes all stages of the simulation pipeline – namely setup, simulation, and visualization. In addition, the user has to be able to step back and forth between the stages in a smooth way.
- **Content of tutorials:** At least one classical fluid-structure interaction case that demonstrates various capabilities of the coupling library has to be realized.
- **Evaluation:** Success of the developed tutorials will be measured by click statistics after completion of the project.

Nice-To-Have Deliverables

In addition, the project specification contained a list of additional nice-to-have-deliverables:

- **Website:** visualization of simulation results can be improved in multiple ways (e.g. representation, additional user options).
- **Content of tutorials**: learning experience can be enhanced by embedding YouTube videos or adding further scenarios.

1.2.2 Team Organization

Since the project involved two separate streams of work, development of the website and preparation of the tutorial, we decided to split into two teams of three during the initial phase. The first group was responsible for development of the website, whereas the second group focused on simulations and worked with preCICE. To ensure that the two teams were working in the same direction, we held weekly meetings involving both teams.

1.2.3 Timeline

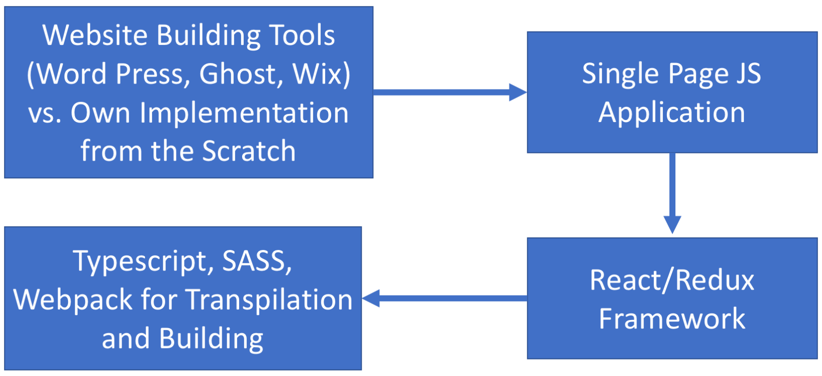
The project was executed in three separate sections or milestones, each with its own objectives. The milestones represented scheduled contact points with the client where we updated them on the progress. The deliverables for each milestone represented the starting point for the next milestone, with a degree of iterative development that incorporated feedback from the client gathered through the milestone meetings and informal contact. A list of the deliverables for each milestone can be found below.

1. Website mock-ups and offline version of the first tutorial are presented.
2. Beta version of the interactive tutorial is available online.
3. Final version of the tutorial is online. See also Section 1.2.1.

2 Web Technology Stack and Server Architecture

2.1 Overview and Main Components

A website is a collection of web pages, which can be accessed over the internet under some communication protocol. We can broadly divide a website into the frontend, code running in the client browser which is essentially what can be seen on the screen and how it is realized; and the backend, the code running on a server that manages the data and serves web pages on request. In the sections to follow, we will go through the history of web development, take a look at our technology decisions for the frontend and the backend, and discuss certain communication protocols. First, however, let us take a look at Figure 2.1 which provides a broad overview of how we decided the essential technology questions in setting up our web application. In the following subsections, we will deal with each part in detail.



Website Building Tools (Word Press, Ghost, Wix) vs. Own Implementation from the Scratch
Single Page JS Application
React/Redux Framework
Typescript, SASS, Webpack for Transpilation and Building

Figure 2.1: Technology Choice Rationales for the Frontend

2.2 Past and Present

There are various technological frameworks for designing the frontend and the backend of a website. In this section, we will take you on a guided tour of the history of web development. We will motivate this discussion by talking about broad technological decisions we made before drilling down into the specifics in the next section.

2.2.1 Website Building Tools versus Own Implementations

There are two main options for building a website. We can either use website building tools, such as WordPress[1], or implement our own website from scratch. WordPress provides an online editor, which allows us to easily create static webpages. However, if we want to add dynamic content, we need to use WordPress Hooks. For example, scripts must be added in ”functions.php”. This is not a sustainable architecture for a website with a lot of dynamic content and it can get quite complicated to maintain the code as the project proceeds. Since an interactive tutorial primarily relies on dynamic content, we decided to implement a website from scratch.

2.2.2 Single Page JavaScript Application

A popular way to implement a website from scratch currently is to create a Single Page Application (SPA). An SPA is quite similar to a desktop application in that the logic of the application runs on the client. This solves the page reload problem that comes with delegating the logic to the server. The code is usually fetched once and any changes, for example due to user action, are incorporated into the page dynamically. This is also why we now talk about ”web applications”.

There are many ways to create a web application. However, before we get into that, we will take a brief detour and take a look at the history of web development. To start with, take a look at Figure 2.2 which depicts the progression of web pages from static to dynamic.

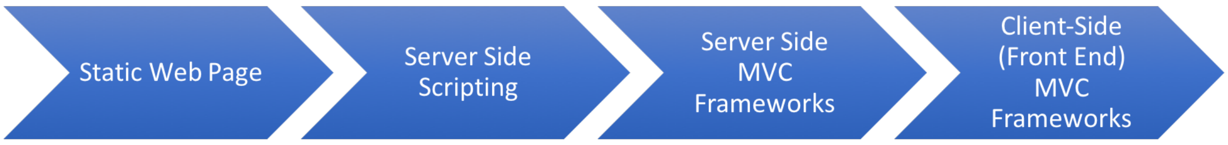

Static Web Page
Server Side Scripting
Server Side MVC Frameworks
Client-Side (Front End) MVC Frameworks

Figure 2.2: Flow Chart of Development Towards Single Page Apps

Static Web Page The first website in history was created at CERN and consisted of static HTML pages. Figure 2.3 illustrates how such a website works. Looking at the source code for such a website, we can see the content wrapped in HTML tags without any styling. The communication model is also very simple. A client sends a request for a particular page to the server which then retrieves the page from the disk and sends it to the client. In this case, the URL of a page simply mirrors the local file system on the server.

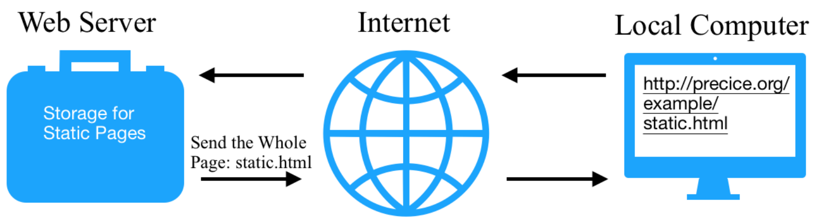

Web Server
Internet
Local Computer
Storage for Static Pages
Send the Whole Page: static.html
http://precice.org/example/static.html

Figure 2.3: Illustration for Static Web Page

It ought to be obvious that static web pages are insufficient for creating an interactive tutorial. So how do we realize dynamic content? There are two options: server-side scripting and client-side scripting. Let us take a look at each in turn.

Server Side Scripting Server-side scripting refers to putting the logic of the application on the server. When a user interacts with a web page, for example she enters her name into a text field, the data is sent to the server. Depending on the program that handles the text on the server, the server might render some html, which could just be a custom greeting for this particular user, and send it back to the client. This allows us to customize the website for each user and hides the code from the user. With server-side scripting we can often take advantage of page caching and reduce load times. There are many server-side scripting languages such as Java, PHP, and Python. Figure 2.4 illustrates how server-side scripting works.

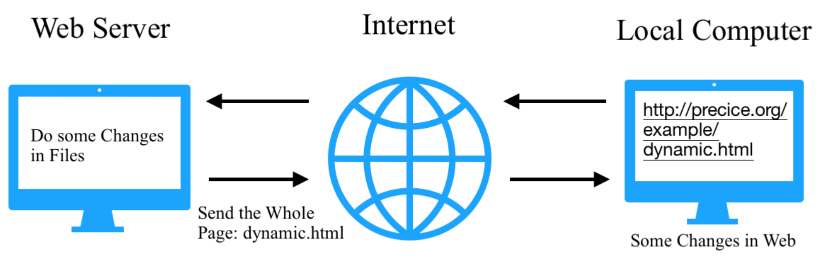


Figure 2.4: Illustration for Server Side Scripting

Client-Side Scripting Client-side scripting, as should be evident from the name, refers to executing the logic of the application on the client. This means that when we open a website, we will get the entire source code of the application which then runs in the browser. The server in this scenario is reduced mostly to a database as in Figure 2.5. If we have a website with user accounts, we might use the server to store user credentials, for example. Client-side scripting is almost always performed with JavaScript since that is the only language supported by all web browsers.

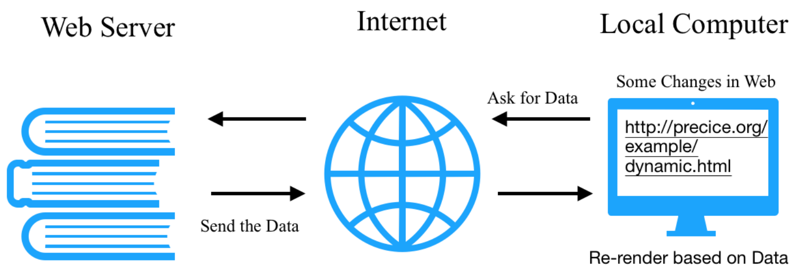


Figure 2.5: Illustration for Client Side Dynamic

Model-View-Controller Framework No matter which type of scripting we use, we still need a way to organize our code in order create a Single Page Application. The most popular pattern for this purpose is Model-View-Controller (MVC) depicted in Figure 2.6.

The MVC design pattern comes from desktop graphical user interfaces and has become quite popular in the design of web applications. MVC divides the application into three largely independent entities which can be developed separately. The model handles the data and the logic of the application, the view is a visual representation of the information in the model, while the controller acts as a bridge between the user and the other two components, updating the model and/or view as necessary. MVC makes it easy to organize code by attributing it to one of the three entities and separates the logic of the application from the view. It is important to realize that it can be incredibly difficult to write maintainable code for web applications without using an MVC-like framework.

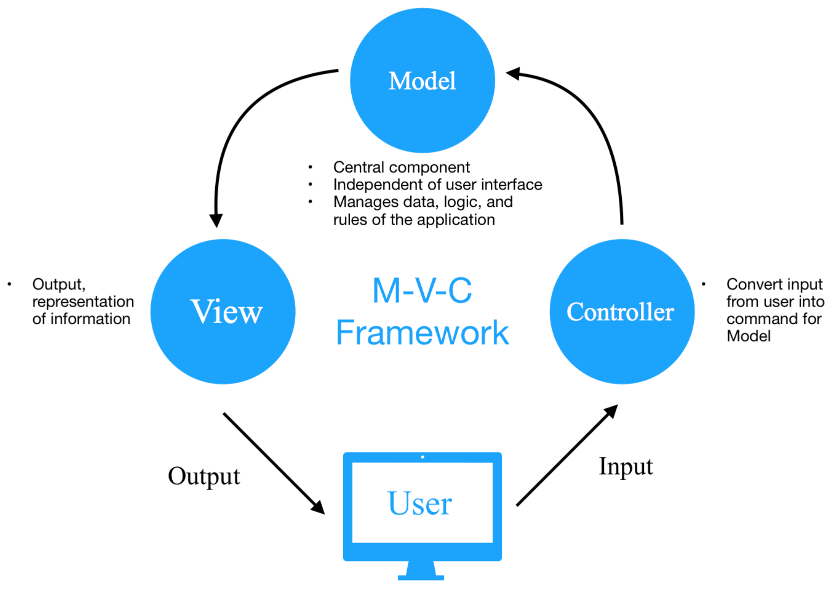

Model
• Central component
• Independent of user interface
• Manages data, logic, and rules of the application
• Output, representation of information
View
M-V-C Framework
Controller
• Convert input from user into command for Model
Output
User
Input

Figure 2.6: Illustration for MVC Framework

Server-Side Model-View-Controller Frameworks Server side MVC frameworks go hand in hand with the idea of server-side scripting since the server is responsible for rendering HTML views. Most server-side frameworks provide solutions for routing, templating, object relational mapping (ORM), middleware, and lots of helper functions. Since these are important concepts, let us talk about them in a bit more detail.

As mentioned before, in the early days of web development, the URL mirrored the directory structure on the web server. This is no longer true and we are free to define the structure of the URL through routing, regardless of what the actual directory structure looks like. Furthermore, templated views allow us to adapt our application to user actions. For example, imagine a user logs in to a website and receives a welcome message with her name on it. This is a trivial example of what we can do with a view template. Finally, ORM solves the problem of storing objects, in the object oriented sense, in a database by creating a ”virtual object database”. Most databases do not allow us to directly store non-scalar values and since OOP objects are usually non-scalar, we cannot directly store them in a database. However, a virtual object database translates the object into an atomized form and thus allows us to store non-scalar objects.

There are many other features that server-side MVC frameworks provide and there are a lot of frameworks out there. Table 2.1 lists popular server-side MVC frameworks.

Provider	Language
ASP.NET	C#
Play Spring	Java
Zend CakePHP Laravel	PHP
Django	Python
Rails	Ruby

Table 2.1: Providers of Server Side MVC Frameworks

Client-Side Model-View-Controller Frameworks (frontend) The difference between client-side and server-side MVC frameworks is depicted in Figure 2.7.

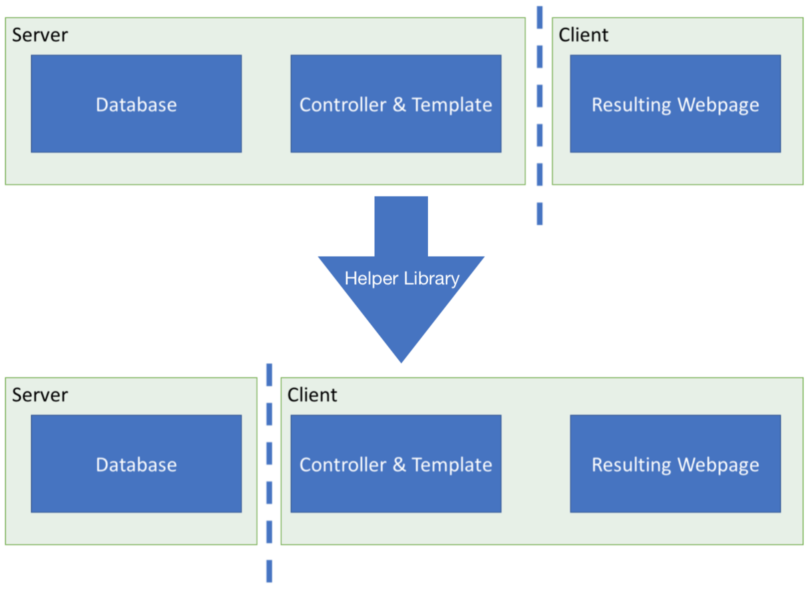


Figure 2.7: Progressing from Server Side to Client Side

In order to create a website with a highly interactive UI, we want to use a client-side framework so that UI elements can be changed on the fly. The features that client-side MVC frameworks provide, for instance, routing, templating, and additional features such as a virtual DOM in React, help us achieve client-side scripting. Take a look at Table 2.2 for a list of popular client-side frameworks.

Provider	Maintainer or Developer/Original Author
ANGULAR	Google
BACKBONE.JS	Jeremy Ashkenas
Knockout	Steve Sanderson
React	Facebook, Instagram, and community/Jordan Walke
Redux	Dan Abramov
Vue.js	Evan You

Table 2.2: Providers of Client Side MVC Frameworks

2.3 Frontend

Having looked at the history of web development and settled on the preliminaries, let us now delve into the more concrete technology decisions we made. The application that we want to build relies on interactivity. The user can take multiple paths through the tutorial and at any given moment, the user will be faced with a task that will depend on her task history. This means that parts of the UI will be constantly changing and we know that this problem is well suited for client-side scripting. The problem then reduces to choosing a client-side framework. We settled on React-Redux.

2.3.1 React and Redux Framework

Why did we settle on React[2] and Redux[3]? Let us discuss what they offer and how that makes them suitable for our interactive tutorial.

React React is a popular JavaScript library for creating interactive UIs. It was created by Jordan Walke, a software engineer at Facebook, and is currently maintained by Facebook and Instagram. In terms of the MVC pattern, React takes care of the views and the logic that goes into creating them. It provides developers the ability to build components that can then be composed to create complex UIs. These components are commonly written in JSX which is transformed to JavaScript and that finally emits html that can be show in a browser.

React has an internal representation of the document object model (DOM), a virtual DOM, and only updates those parts of the actual DOM that differ from the virtual DOM. There are no full page reloads; only components that need to be changed are reloaded. Furthermore, we can also use React to do routing. While its components usually have their own state, we do not use this and instead rely on Redux for managing the state of our application.

Redux Redux is a JavaScript library that is used to manage the state of a web application. It provides a central data manager, the "Redux store", which can be accessed through JavaScript functions called selectors. The Redux store is useful since it saves us from distributing the state of our application over a dozen or more React components. The central store is a basically a giant JavaScript object and is read-only. Furthermore, Redux can be easily used with other JavaScript frameworks.

React coupled with Redux makes it easy to design responsive UIs that change with the state of the application. For example, suppose we have a progress bar in our tutorial application.

The progress bar has has multiple views depending on which stage of the tutorial the user is currently at. This involves examining the Redux store to determine the current route/location, passing that to the progress bar component written with React. React should then automatically update the progress bar component when the user moves to a different stage and the state, that is, route in this example, changes.

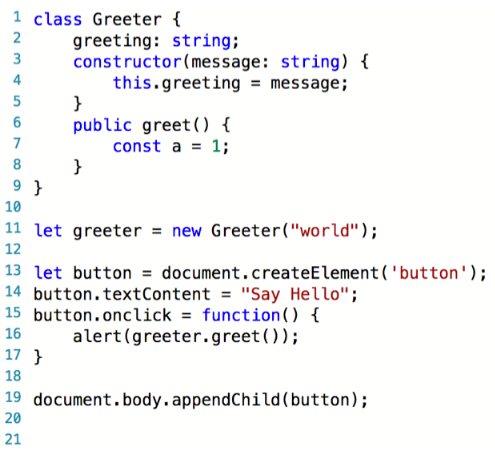
2.3.2 TypeScript, SASS, Webpack for Transpilation and Building

While JavaScript is the only language supported by most browsers, it has some serious shortcomings:

- JavaScript does not have static typing.
- Across different browsers, JavaScript supports inhomogeneous features.
- Delayed browser support for new JavaScript features.
- No coherent module import system.

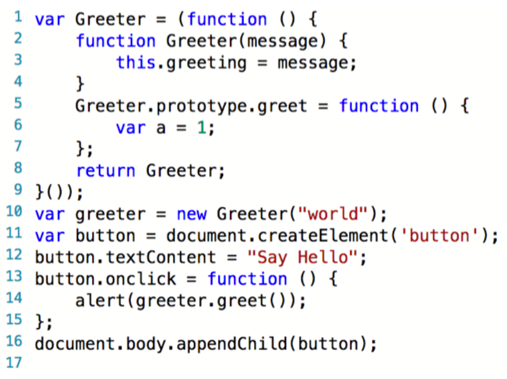
TypeScript Some of the shortcomings listed above can be addressed by using TypeScript. It is a free open-source programming language with static typing and is a superset of JavaScript. Since it is a superset, all existing JavaScript programs are valid TypeScript programs.

Transpiled JavaScript In order to run in a browser, our TypeScript code has to be transpiled to plain JavaScript. Most browsers currently comply with ECMA5 standards and for this reason, we transpile TypeScript to ECMA5 JavaScript. We can also use JavaScript features from ECMA6 in TypeScript since these features can be implemented by additional ECMA5 code. Figures 2.8 and 2.9 show TypeScript code before and after transpilation. ECMA5 does not have classes but we still get the same functionality through additional code as in Figure 2.9.



```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    public greet() {
        const a = 1;
    }
}

let greeter = new Greeter("world");

let button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function() {
    alert(greeter.greet());
}

document.body.appendChild(button);

```

Figure 2.8: TypeScript code



```
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        var a = 1;
    };
    return Greeter;
}());
var greeter = new Greeter("world");
var button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function () {
    alert(greeter.greet());
};
document.body.appendChild(button);

```

Figure 2.9: TypeScript transpiled to ECMA5

SASS The natural choice for styling in web applications is CSS. However, as with JavaScript, there are certain shortcomings in CSS that can make the code hard to maintain as the application grows. We decided to use Syntactically Awesome Styleheets (SASS), a CSS extension. SASS provides additional features like variables and nested styles and is compiled to CSS. For details on implementation, please refer to section 11.1.

Webpack A typical web application written in JavaScript can have hundreds of JavaScript modules as dependencies. If we were to leave our application distributed over several files, with hundreds of module dependencies, our application would be incredibly slow since we would need to perform multiple HTTP requests to fetch modules in order to run code that depends on them. To circumnavigate this problem, we use a module bundler which will put our JavaScript code along with the dependencies in a single JavaScript file. Webpack is widely used for this purpose and is a standard choice.

2.4 Backend Webserver Implementation in NodeJS

Over the past few years, NodeJS has emerged as a leading server scripting language. It is based on the Google V8 JavaScript interpreter engine and therefore follows the same coding standards as Google Chrome and Opera web browsers. One of the main advantages of NodeJS, compared to traditional backend systems implemented for instance in PHP, Perl or Java, is that NodeJS was developed with modern web application patterns in mind. For instance, NodeJS servers assume fully semantic routing as opposed to PHP interpreters that are still following the very old idea of mirroring the file system to the clients. Due to the rapid growth of the developer community, thousands of open-source libraries have been developed and are actively maintained. Furthermore, the node package manager (npm) is a great tool that helps fetch these JavaScript modules from a central repository. We made extensive use of npm in our project.

2.5 Communication Protocol Between Frontend and Backend

Having settled on the technologies for the frontend and the backend, all we need to do now is define the communication protocol between the two.

2.5.1 REST API for the Visualizations of Config Files

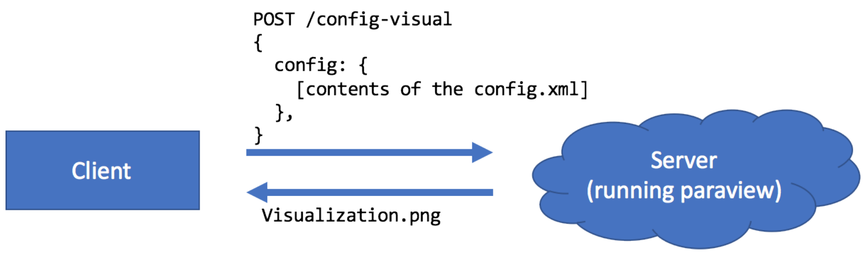

```
POST /config-visual
{
  config: {
    [contents of the config.xml]
  },
}
```

Figure 2.10: Request to obtain the visualization of the config.xml file

In order to obtain visualizations of the current preCICE configuration, a simple Representational State Transfer (REST) API offers a robust solution. REST offers a few properties which are particularly advantageous:

- *Stateless*: the server does not need to maintain a session for different users
- *Clientside caching*: the client does not need to send a new request if the config.xml file does not change, even across different browser sessions. It would be also possible to implement this behavior using localStorage for example, but it is good practice to utilize the built in browser caching
- *Serverside caching*: if multiple clients request visualizations for the same config file, the visualization does not need to be regenerated

All the points above lead to a horizontally (distribute the request load to multiple servers) and vertically (handle many requests on one server) scalable solution. The concrete implementation of such an API can be seen in Figure 2.10. It should be mentioned that semantically, a GET request would make more sense in 2.10, but since the payload (the contents of the config.xml file) is too big to be encoded as part of the URL, a POST request must be used.

When the project concluded in March 2018, this feature was not implemented since the user story did not require customized XML files any more, cf. section 6. However this part of the architecture is still a valid choice and can be implemented in the future.

2.5.2 Websockets for Shell IO

One of the key requirements of the tutorial involves the user being able to run a simulation in the browser. Naturally, the user should be able to use an emulated terminal in his browser that sends input to and receives output from the server, where the commands are actually running. This requirement does not allow us to implement this feature using a REST API since the underlying protocol, HTTP(S), does not support two way communication. Furthermore, terminal sessions must be maintained on the server which is not possible with a stateless REST API.

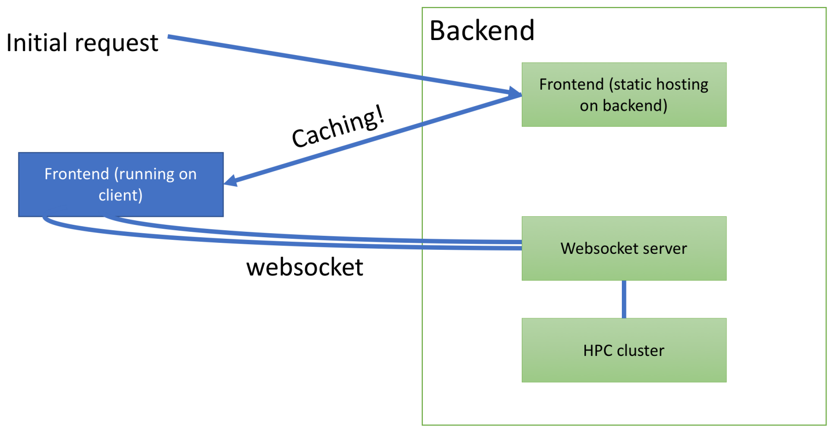


Figure 2.11: Architecture between frontend and backend

Websockets are an established solution for two-way data transfer between a client browser and a web server. Figure 2.11 shows the basic architecture. In contrast to the request approach in HTTP, websocket communication is event-based over a persistent connection. Events can be emitted and listened to from both sides. A sample implementation for the terminal communication problem discussed above can be seen in Listing 2.1 and Listing 2.2.

In reading these listings, keep in mind that there are several events the server has to listen for:

- *connect*: A client has connected to the server
- *spawn_consoles*: The client wants to spawn two terminals
- *console_cmd*: The client sends a command.

```
socket.listen("connect", function() {
 // log connection
});

socket.listen("spawn_consoles", function() {
 // allocate resources for two consoles
 // create two console ids
 socket.emit("spawn_consoles_success", consoleIds);
});

socket.listen("console_cmd", function(console_id: consoleId, cmd) {
 // check if console_id is valid.
 // check if the cmds are allowed (have a whitelist of
 // commands) and execute them. Bind the "console_output"
 // event like this:

 const proc = spawn(cmd);

 proc.stdout.on("data", (data) => {
  socket.emit("console_ouput", {
   console_id: consoleId,
   stream: "stdout"
   data,
  })
 });

 // same for proc.stderr

 proc.on("close", (code) => {
  // emit "console_clode" event
 });
});
```

Listing 2.1: Server websocket

```
socket.connect();

socket.emit("spawn_consoles");


```

```
// emit this event when the user has typed a command
socket.emit("console_cmd", {
 console_id: "XYZ",
 cmd: "./StructureSolver ./config.xml N"
})


socket.listen("spawn_consoles_success", function([ consoleIds ]) {
 // save the console ids locally and assign them
 // to the two forntend terminals. This makes it possible
 // to place the server output in the right frontend
 // terminal.
});

socket.listen("console_ouput", function({ console_id: consoleId, data, stream }) {
 // write the output in the frontend terminal
 // which has the assigned identifier consoleId
 // stream can either be stdout or stderr
});

socket.listen("console_exit", function({ console_id: consoleId, exit_code }) {
 // show also in the frontend that the program has terminated
});
```

Listing 2.2: Client websocket

It is also possible to cache the console output and send the output read from a file instead of real console output to the frontend. Since the frontend is agnostic to the backend implementation and its only touch point to the frontend is the websocket interface, no changes in the frontend are necessary. With the implemented adapter-like pattern in the backend, it is also possible to define new data sources besides the stdout stream of a program or a file of recorded output.

3 Hosting and Other Production Considerations

3.1 Cloud Server Provider

Currently we are hosting our website on the virtual machine provided by the chair. However, in the future we might want to consider moving to a proper web service provider. There are several cloud servers that are easy to use and affordable. Below we list some of them and provide an estimate of the cost for Amazon Web Services in table 3.1.

- Amazon Web Services
- Google Cloud Platform
- IBM Bluemix
- Microsoft Azure

Amazon Web Services	frontend	
Cloudfront CDN	per GB (data transfer)	\$0.085
S3	per GB per month(storage)	\$0.0245

	backend	
Elastic Beanstalk	per month	\$20
L Certificate	per month	\$3
(Possibly) Additional Servers on Demand(AWS HPC)	per month	Depends on specs

Table 3.1: Cost estimate for Amazon Web Services

As of March 2018, when the project concluded, we are running the frontend and backend on the chair machine while redirecting all frontend traffic through the http://run.coplon.de domain using a so-called *A* DNS record. The backend does not need a domain name since its URL is only used internally in the Redux/React application and is not exposed to the user. Therefore, the IP address can be used as the URL.

An *A* record is used when a domain name or subdomain (such as *run.coplon.de*) is assigned to a fixed IP address, in our case the IP adress of the VM provided by the chair. Other kinds of DNS records include *CNAME* which assigns a domain name another domain name (instead of an IP address), *ALIAS*, which is internally the same as *CNAME* but is a more common expression when root domain names are created and a *URL* redirection which shows similar behavior to the user but is actually an http redirect and not a DNS record.

Since the code can resolve all its dependencies by itself using the node package manage, the only requirement is nodeJS on the server. This makes the code portable and provides the possibility to deploy it on a different server in the future without much hassle. In doing so continuous integration must be set up accordingly. For even faster and more reliable deployment, it could be useful to bundle the webservers for the frontend and backend into a docker image. Docker is a container software that adds another layer of abstraction between the OS and the actual application and thus makes it more portable and isolated from other processes. This enables multiple running instances across multiple machines and eliminates mostly possible incompatibilities (for instance the exact version of nodeJS is always guaranteed). It can be thought of as a virtual machine but with less overhead since the Docker VM is just a regular process in the hosting OS.

However, it is arguable if a docker deployment is worth the effort to introduce automated docker bundling and deployment since user load is expected to stay comparably low. Therefore horizontal scalability is not a requirement. Also, the current infrastructure is simpler, only consisting of a centralized backend and frontend and not multiple distributed services, for which docker would be the natural choice.

3.2 Licenses

Since we are using a lot of external packages, we also need to take a look at the licensing for these packages. Currently we are using 763 packages in total, and almost all are either open source or distributed under the MIT License.

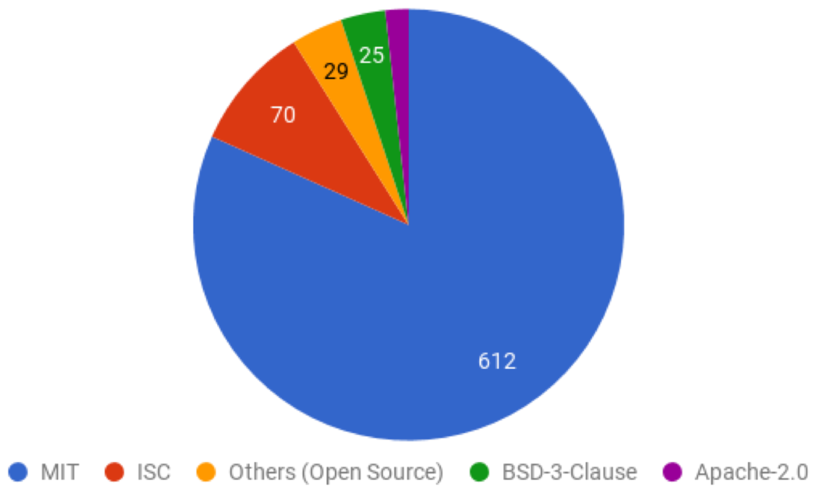


Figure 3.1: Packages and Where They are from

The MIT License, as well as most other Open Source licenses like ISC, BSD-3-Clause and Apache-2.0, only requires an acknowledgement to the original project. Other than that, they allow free use of the corresponding software if the project is not intended to be patented or similar. Since these rules do not apply in our context, the licences should not be a problem factor.

However, there are licenses like GNU GPLv3 that require the disclosure of the source code and therefore the corresponding packages are not suitable for our project.

4 preCICE Coupling Library

As already mentioned in Section 1, the goal of preCICE is to provide all functionality required for the realization of a multi-physics simulation environment based on reuse of existing single-physics solvers. Multi-physics simulations involve problems that contain several types of equations. Examples are fluid-structure interaction, fluid-solid thermodynamics, porous-free flow, etc.

In the following, we provide a brief overview of the functionality implemented in preCICE and its features. A detailed and thorough description is provided in [4].

preCICE provides three main ingredients that are necessary for a partitioned multi-physics simulation:

- **Iterative methods for solving an interface fixed-point equation:** explicit and various implicit coupling schemes are realized.
- **Data mapping for interpolating between non-matching grids at the coupling surface:** preCICE allows the user to choose between nearest-neighbor, nearest-projection and radial basis function methods.
- **Data communication between several solvers:** communication methods are based on MPI ports or TCP/IP sockets.

All above functionalities are provided in a single library API.

It is clear that for coupling arbitrary (including black-box and parallel) single-physics solvers, preCICE needs high flexibility. Therefore, the latest version of the library works solely with input and output of the involved solvers. Another important feature of preCICE is the ability to work efficiently with parallel solvers. The library provides fully parallel point-to-point communication between the involved solvers based on the analysis of the mesh decomposition. This is a significant advantage as multi-physics simulations are generally run to obtain higher accuracy compared to single-physics cases and require fine meshes and large amounts of computations.

5 Tutorial Test Case

In order to create the online tutorial, we came up with an FSI simulation designed to show the capabilities of preCICE. In this chapter, we will briefly present the software we used for this purpose and the simulation set up. The software includes preCICE, two solvers and solver adapters required for coupling.

5.1 Used Software

- **CalculiX**
  A 3D finite element structure solver [5]
- **SU2**
  An open-source CFD solver developed at Stanford [6]
- **preCICE**
  A coupling library for partitioned multi-physics simulations [4]

5.2 Installation of Solvers

In installing the solvers, SU2 and CalculiX, and preCICE, we ran into some issues. These observations might help the preCICE development team in their efforts to make the library more popular.

SU2 We installed SU2 through the official github repository using the standard GNU build system (./configure and make).

CalculiX We downloaded binaries for both CCX (structural solver) and CGX (graphical interface) from the official website.

preCICE As direct dependencies for the preCICE library, Eigen, Boost, MPI and optionally PETSc had to be installed. We used SCons as our build system. We observed several issues during this process. For instance, we had trouble installing preCICE on Mac OS systems due to incorrect behavior of environment variables and non-transparency of the SCons configuration script. Moreover, the library requires Boost 1.60. This version of Boost is more recent than the version that is packaged with popular Linux distributions. Therefore, we had to install it Boost 1.60 solely for preCICE, even though an older version was already present on the system. Another issue was related to the non-header based Boost libraries. The configuration script did not check their location and if Boost related environment variables had not been not specified

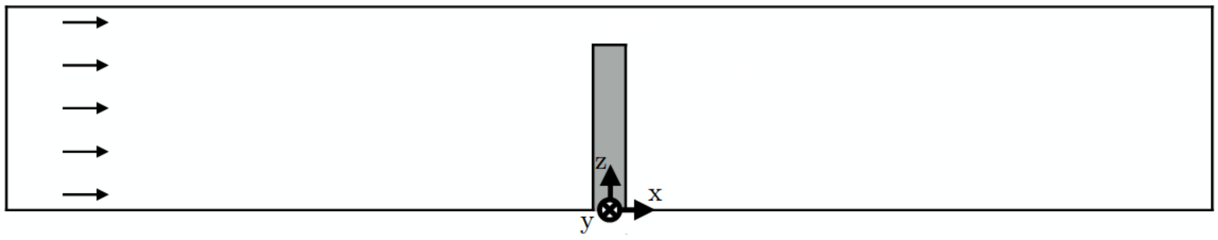


Figure 5.1: FSI scenario: fluid flows through the channel from left to right, the solid flap oscillates due to fluid pressure; Figure adapted from [7]

properly, this led to a linking error on the very last stage of compilation. These issues greatly extended the installation time.

SU2 adapter Installation of the SU2 adapter [7] required modification of the SU2 source code. We had to include additional functions for the adapter, recompile the whole code together with the adapter files and link it to preCICE. The modification of SU2 code had to be done by copying functions from the pdf supplement [7]. We had difficulties with this step since some required functions were missing, their exact location was not specified and the adapter was not compatible with the latest version of SU2. For potential users, the SU2 adapter might therefore present serious challenges. One alternative would be to fork the SU2 code and introduce needed changes as new versions of SU2 are released.

CalculiX adapter Installation of Calculix adapter [8] required downloading and building additional solvers, such as Arpack and Spooles. The absence of a unified build system and the legacy nature of the software naturally lead to tedious manual modification of makefiles for each solver in order to produce a working build. After this we built CalculiX from source and linked it with preCICE. As configuration files for CalculiX are provided in YAML format, we had to additionally install a C++ based YAML parser.

5.3 Scenario

We simulate a typical example of fluid-structure interaction (FSI) where a fluid flows through a channel and interacts with an elastic flap that is fixed to the floor of the channel. The geometry of the setup is depicted in Figure 5.1. The fluid – in this case air – enters the channel from the left, flows over the flap, and leaves the domain through the outlet on the right. The flap oscillates due to the fluid pressure building up on its surface. We use SU2 to simulate the fluid flow and CalculiX for the structure/flap.

The coupling – performed with preCICE – involves the communication of values at the interface between the fluid solver and the structure solver. The pressure exerted by the fluid on the interface is sent from the fluid to the structure solver, so that the displacement of the flap can be calculated. The displacement of the flap surface is then sent back to the fluid solver and used as the new boundary. For more details on the definition of the case and its parameters, see [7].

6 Simulation: First Iteration

In this chapter, the simulation of the scenario described in Section 5.3 is presented. We deal with the first version of the test case here. Chapter 7 and 8 introduce improved versions. First, we discuss two simple uncoupled simulations: one for the fluid and one for the structure part. In the following we call them *single-physics* simulations in order to differentiate them from the *multiphysics* case described in Section 5.3. The last part of this chapter describes the coupled multiphysics simulation.

6.1 Fluid Simulation

To begin with, we ran a standard, single physics, computational fluid dynamics (CFD) simulation in order to get familiar with the fluid solver. For running such a simulation with SU2, a configuration file and a mesh file are needed. Since these mesh files have to be in SU2's own mesh format, the conversion of meshes generated by different meshing tools can be problematic. We encountered problems when converting meshes generated with Hypermesh and OpenFOAM's blockMesh. In each case, the boundary markers were not converted correctly. For a simple flow setup like this, the problem can be overcome by writing a python script that directly outputs the mesh file – without the use of a meshing software.

The configuration file can be adapted from one of the many SU2 tutorials. We chose to solve the incompressible Navier-Stokes equations with water as the participating fluid. After trying a few possible settings, the simulation converged and generated a velocity field like the one depicted in Figure 6.1.

6.2 Structure Simulation

Similarly, to get a basic understanding required for working with CalculiX, we performed several simple FEM-simulations. Using meshing tools provided by CalculiX, we created a beam model consisting of one-dimensional line elements. The beam is fixed on one of its ends, and is subject to a distributed force acting from the side. The resulting displacement is shown in Figure 6.2 which was generated by using the visualization tool provided with CalculiX. Using this set up, we were able to accomplish all the steps required for running simulations with CalculiX: mesh generation, specification of configuration file and visualization.

6.3 Coupled Simulation

In the following, we give a brief overview of the steps required to conduct the coupled simulation for the scenario described in 5.3. More details can be found in the github-tutorial (see Section 10). To reiterate, we use SU2 for fluid simulation, CalculiX as the structure solver and

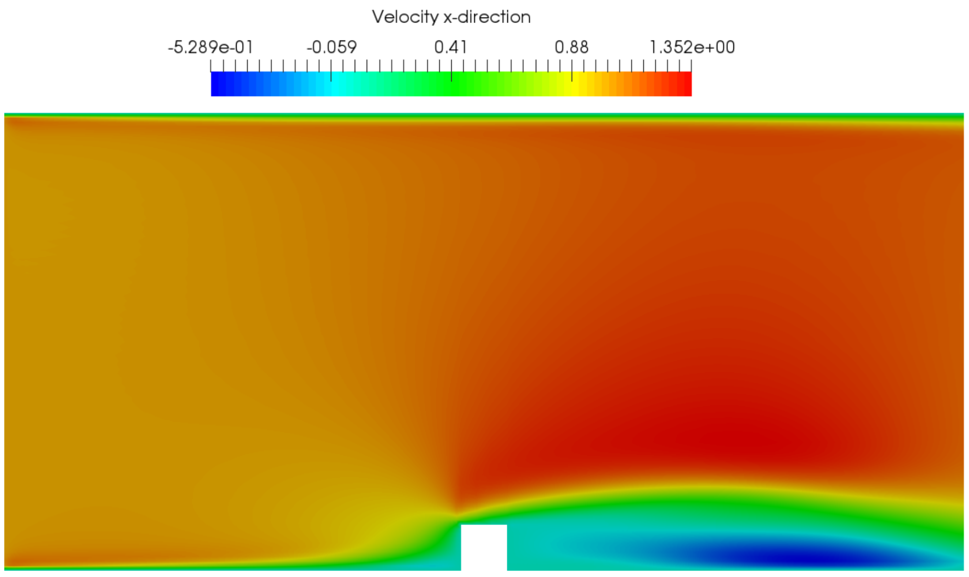


Figure 6.1: Single physics SU2 simulation of incompressible flow

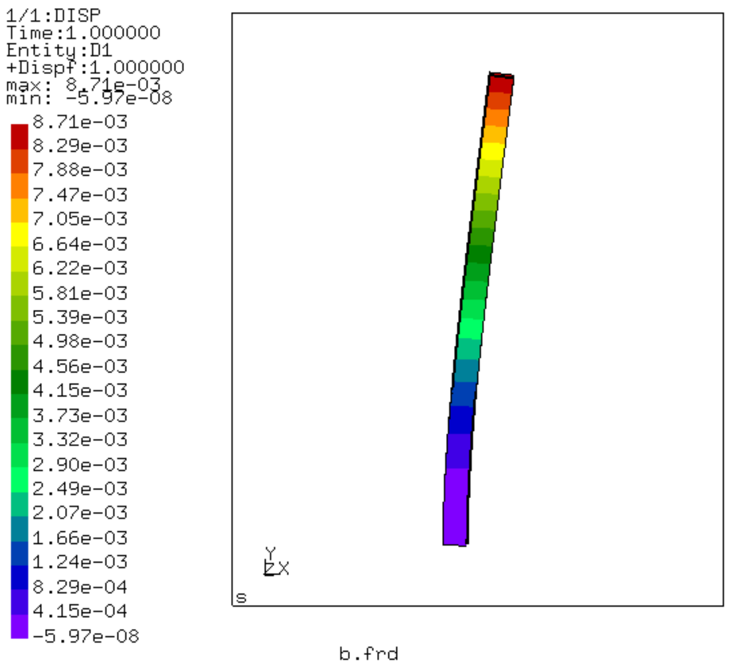


Figure 6.2: Result of FEM simulation with CalculiX, displacement of the beam elements due to a force acting from the side

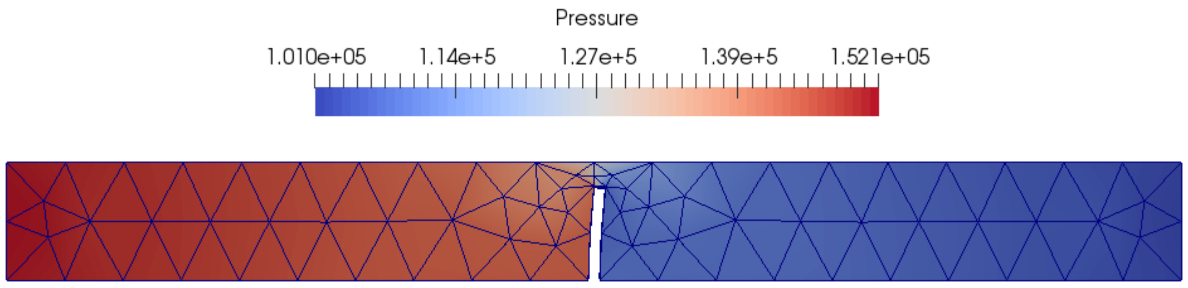


Figure 6.3: Coupled FSI simulation with SU2 and CalculiX

preCICE as the coupling tool between the two. Although this test case is two-dimensional, we run a three-dimensional simulation with a constant width in y-direction (quasi 2D). Therefore, we provide 3D meshes to both structure and fluid solvers. This constraint is imposed by CalculiX which only deals with 3D scenarios.

The fluid flow is simulated by solving the incompressible Navier-Stokes equations for laminar viscous flow. A constant velocity profile is assumed on the left input of the channel and a no-slip condition is prescribed on the walls of the channel. Before starting the coupled simulation, we run a single physics fluid simulation to get a good initial velocity field. Afterwards a coupled simulation is started that uses the precomputed results as its initial condition. Therefore, two configuration files are used. The file for the coupled simulation contains the specifications required for coupling with preCICE.

The elastic flap is modelled as a three-dimensional elastic structure. The set-up of the configuration file barely differs from the configuration used for the single-physics FEM-simulations. The coupling with preCICE is done by providing an additional xml-file. There we define the participating solvers and the interfaces where data is exchanged. We also specify the algorithms used for data mapping between the meshes and the coupling scheme in the xml-file.

After the preparation described in the previous paragraphs, the coupled simulation can be run by starting both participating solvers independently. A sample result of the simulation is shown in Figure 6.3.

7 Simulation: Second Iteration

The first version of the tutorial presented in Chapter 6 is to a large extent based on the configuration described in [7]. The simulation clearly demonstrates the capabilities of preCICE for fluid-structure interaction tasks, but has several problems that prevent us from using it for the interactive – web-based – tutorial. Most importantly, the runtime is about 30 minutes. Furthermore, the simulation is very sensitive to variations of the parameters. Both issues impede the construction of a good user story as potential users can be discouraged by the long runtime and a very limited choice of simulation parameters.

In the following, we present a new simulation setup for the online tutorial, which is still based on the scenario discussed in Section 5.3. This time, the setup was developed from scratch - we created new meshes for the solid and fluid parts as well as new configuration files for both solvers and preCICE. We tried to reduce the complexity of the setup, as our main goal is to produce a robust and fast simulation that serves well as a basis for the online tutorial.

In order to increase the stability of the setup, we decided to change the geometry of the channel and the flap. Now, the gap between top of the flap and top of the channel is much wider. This simplifies the solution of the flow equations. Moreover, the flap is shifted left from the center. This results in an enhanced flow at the outlet and can lead to more stable simulations. The updated setup is schematically sketched in Figure 7.1

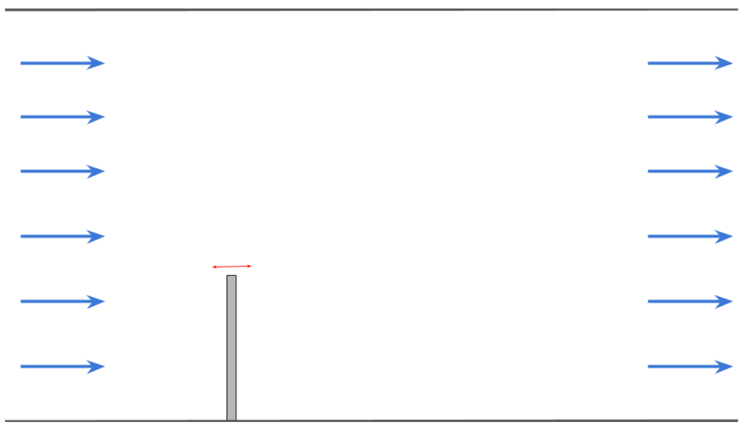


Figure 7.1: Sketch of the geometry for the new setup.

7.1 Fluid Simulation

As a starting point for the new fluid solver setup, we used the configuration file of the SU2 tutorial that deals with an inviscid bump in a channel [9]. Is is important to note that we decided to solve the Euler equations instead of the Navier-Stokes system that we previously considered.

For our setup, the Euler equations lead to more stable simulations and allow the use of less complicated numerical methods. This decreases the runtime.

In addition, we created several adaptive and uniform meshes with different resolutions using gmsh [10]. We started from a two-dimensional case with a simple geometry, and iteratively increased the geometric complexity while trying to keep the mesh resolution as coarse as possible. This is reflected in Figures 7.2–7.4.

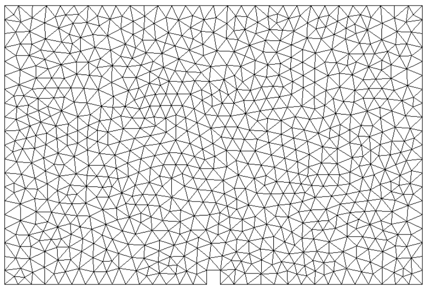


Figure 7.2: Simple geometry with uniform mesh.

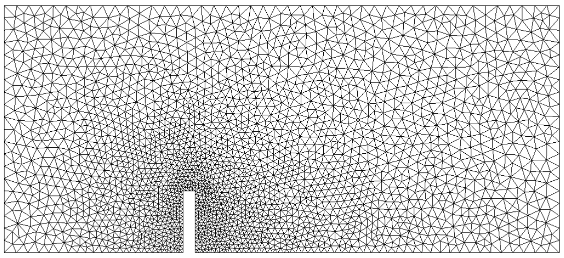


Figure 7.3: Fine adaptive mesh.

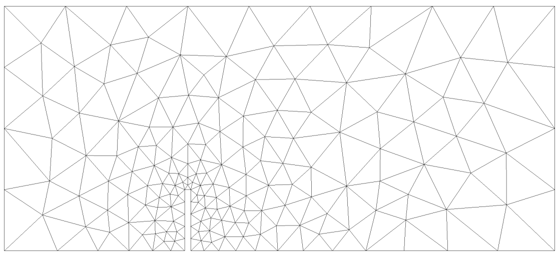


Figure 7.4: Coarse adaptive mesh (used for the final setup).

Finally, we obtained a converging steady-state fluid simulation on a rather coarse mesh. The resulting velocity field is shown in Figure 7.5. The two-dimensional mesh was then extruded to a quasi three-dimensional one, so that the obtained result can be used as the initial condition for the unsteady coupled simulation. See Section 8.3 for more details.

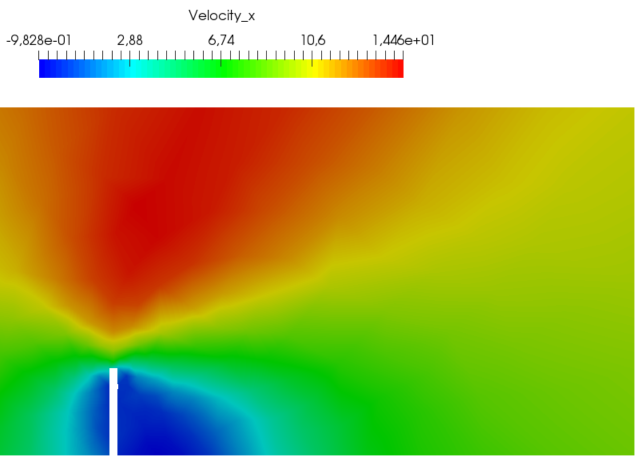


Figure 7.5: Fluid simulation, velocity in channel direction.

7.2 Structure Simulation

Similarly, we used the CalculiX meshing tool to generate various meshes for the solid structure. We evaluated their performance for a simple distributed traction, as well as a coupled simulation with SU2. For the final version of the setup we chose a mesh with 5 high-order brick elements with 20 nodes each – the so-called C3D20 elements [11]. After analyzing the output of several simulations, we saw that these elements can nicely capture the large deformations observed during the coupled simulation. In addition, the runtime is moderate as only 68 nodes are required to represent the entire flap. Figure 7.6 shows the deformation of the flap during a coupled simulation.

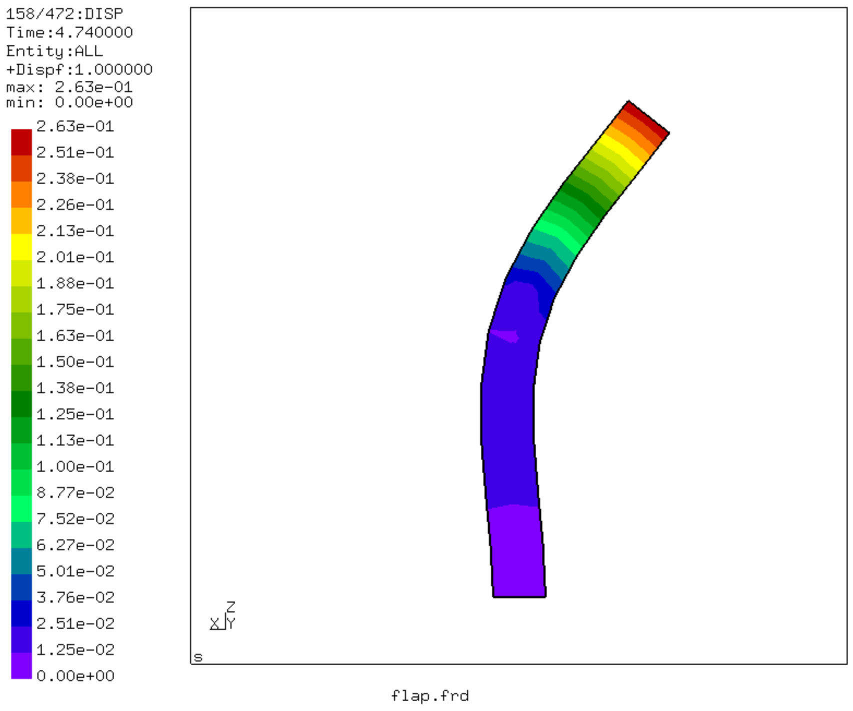


Figure 7.6: Flap deformation during the coupled simulation.

7.3 Coupled Simulation

In the following, we give a brief overview of the main parameters that were used to couple SU2 and CalculiX. A detailed explanation of the configuration of preCICE and both solvers can be found in Chapter 10 and in the github-tutorial.

The two solvers were coupled via TCP/IP sockets in an implicit manner. Implicit coupling was based on fixed point iteration. In order to improve the coupling relaxation, and thus, the simulation runtime, the Aitken relaxation method was chosen; see Chapter 9. The number and length of the timesteps was chosen to be the same for both preCICE and SU2 – 160 timesteps of length 0.03 – as this configuration leads to the fastest and most stable results. This corresponds to coupling on every timestep.

We tuned various configuration parameters, such as numerical methods and convergence criteria for both solvers and the preCICE coupling library in order to further speed up the simulation. Most importantly, we relaxed the convergence criterion (residual) for the fixed point iteration. This can be justified by the fact that we only aim to provide a qualitatively and visually insightful simulation. Any simplification of the case, such as solving Euler equations in SU2 and using weaker convergence criteria, makes sense as long as the results are meaningful. In our setup the results look physical even though the residuals are not decreasing in every timestep. We also added a watchpoint to the preCICE configuration so that the displacement of the top of the flap is recorded by preCICE and written to an output file. The resulting displacement can be visualized using **gnuplot** [12]. This can be seen in Figure 7.7. As a result, we obtained a coupled simulation that runs in under 4 minutes and provides qualitative insight into the nature of fluid structure interaction. The velocity field at the last time step of the coupled simulation is given in Figure 7.8.

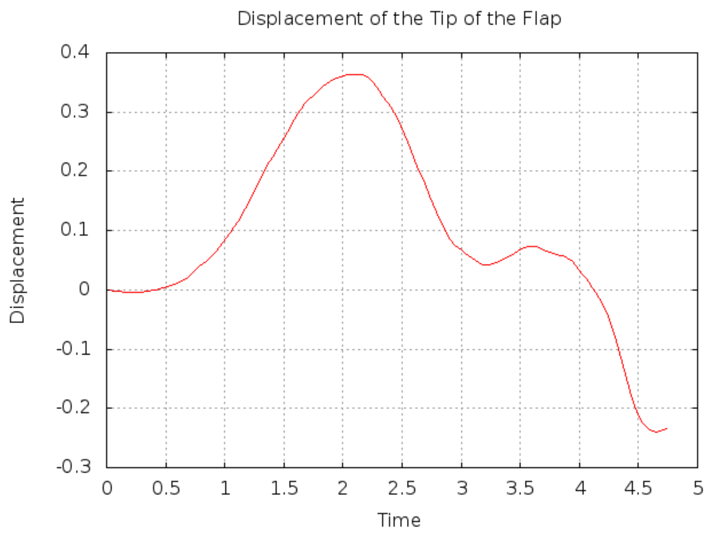


Figure 7.7: Displacement of the top of the flap.

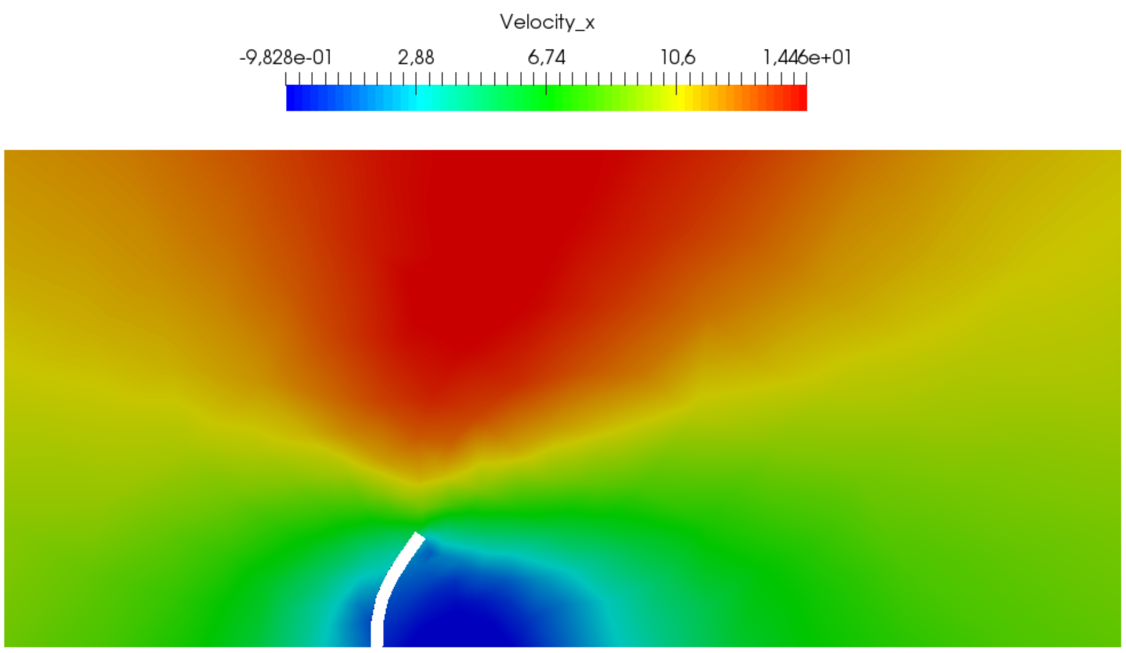


Figure 7.8: Result of the coupled simulation: velocity in channel direction after 4.6 seconds.

8 Simulation: Third Iteration

The third and final version of the simulation builds upon the setup described in 7. As we had experienced some convergence issues with that setup, the main focus was on getting a simulation that converged nicely. The second challenge was preparing a series of similar setups that can help the user learn about preCICE and thus, make for a good user story for the interactive tutorial. For more details on the user story, please refer to 9.

8.1 Fluid Simulation

While the simulation we came up with during the second iteration provided optically appealing results, it did not converge. The residuals did not decrease in most timesteps but rather, showed oscillatory behaviour. The results of the simulation were still plausible – possibly due to the highly accurate initial solution – but for the final setup to be displayed on the website, we wanted a simulation with decreasing residuals. This was achieved by adjusting the convergence criteria on all the involved numerical solvers. The criterion for the maximum residual of linear solvers was set to 10^{-4}, and for the fluid solver to $10^{-3.5}$. This led to improved convergence as the residuals now decrease in every timestep.

The fluid mesh was also changed slightly. On the right hand side, where the fluid flows out, a part of the mesh was cut off so that the solid flap was positioned in the center of the channel. This slightly decreased the number of cells and thus had a positive effect on the runtime. The updated mesh is shown in Figure 8.1

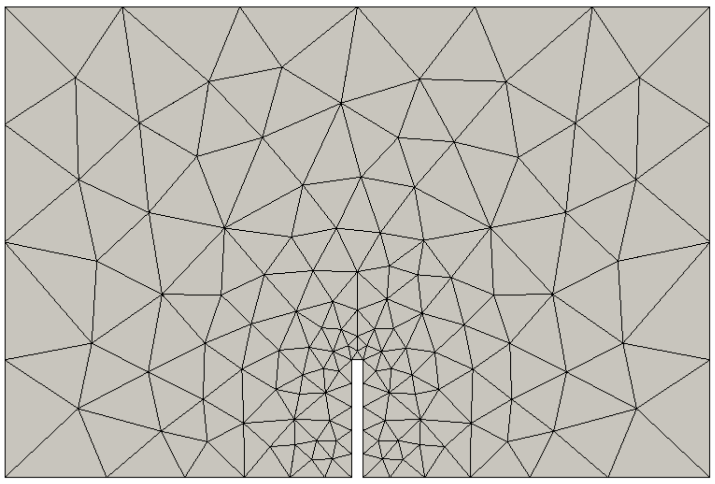


Figure 8.1: Final Fluid Mesh

8.2 Structure Simulation

The mesh for the solid flap was not changed after the Second Iteration; see 7.2 for the details. We only played with the material parameters and chose a combination of density and stiffness that led to results with appealing oscillations. For the final setup, Young's modulus was chosen to be $200000\frac{N}{m^2}$ and the Poisson ratio was set to 0.3. The density of the material is now $3000\frac{kg}{m^3}$.

8.3 Coupled Simulation

As the convergence criteria for the fluid solver were updated, the criterion for the implicit coupling had to be adjusted as well – we set it to 10^{-3}. It is best practice to set it to a value slightly larger than the maximum residual for the single-physics simulations, as solving the coupling equations more accurately than the single-physics ones would unnecessarily increase the runtime.

We ended up with a setup that fulfilled all our requirements. The runtime is now in the order of one minute, the simulation converges nicely, and the results look physical. The capabilities of preCICE are well demonstrated in this test case. Figure 8.2 shows the result of our final simulation setup after 200 out of 400 timesteps.

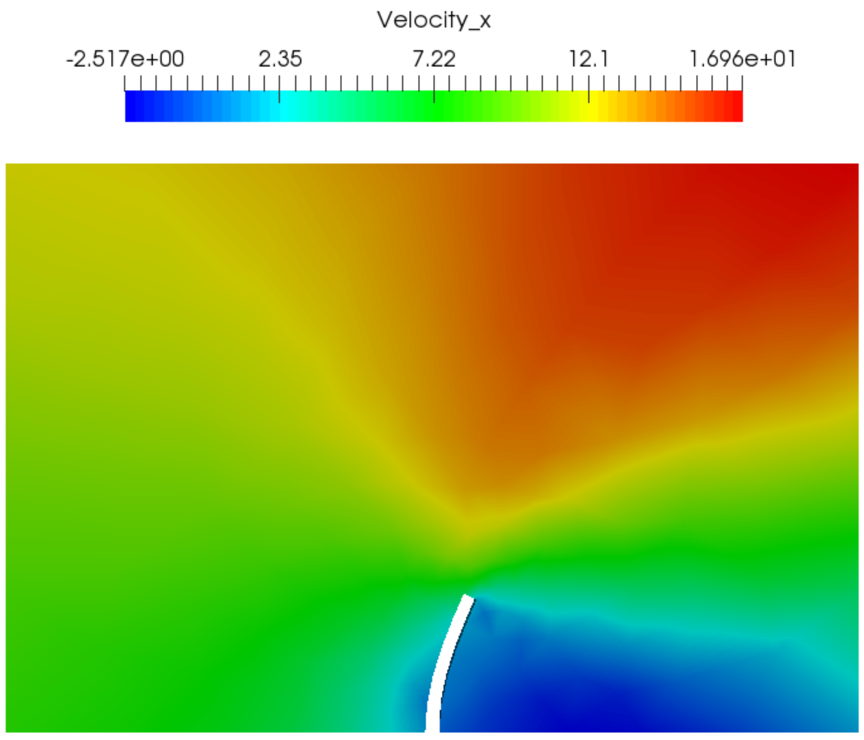


Figure 8.2: Output of the final simulation

9 User Story

Having come up with an adequate simulation setup, we now want to create a user story out of it that demonstrates the capabilities of preCICE. The user story represents a typical user's journey through the website where they learn, step by step, how to start using the coupling library. Naturally, this requires a great deal of thought and planning in order to keep the user's interest and educate him or her about the possibilities and the usage of preCICE.

9.1 First Idea

Our first idea for the user story was to allow users to modify the value of the Aitken relaxation parameter [13]. This was implemented on the website during the second iteration as mentioned in Section 12.4. For this, we used the fluid-structure simulation from Chapter 7 with implicit coupling via fixed point iteration. Such simulations can sometimes suffer from slow convergence, especially if interaction between fluid and structure is strong, as in our scenario. In order to change the runtime, we can change the value of the Aitken relaxation parameter which adapts the relaxation rate in each iteration based on the previous iterations. Changing its value thus gives users the opportunity to play around with the simulation and see how the parameter affects the runtime. For example, after changing the value of the Aitken relaxation parameter from 0.4 to 0.9, the computation time decreases by roughly 10 %. To demonstrate the influence of a particular Aitken value on the runtime, we implemented a table of highscores, lowest runtimes, which was shown at the end of each simulation. For details, see Chapter 12.

9.2 Final Version

Changing the Atiken relaxation parameter is a good start, but the user might get a better understanding of preCICE if he could adjust more than one single parameter. The simulation setup derived in Chapter 8 was a good basis for developing a more sophisticated user story that presented the most important features of preCICE. This change was precipitated by the fact that the first version of the user story, based on Aitken, did not expose the user to the core features of preCICE such as different coupling schemes that can radically change the stability and convergence properties of a simulation. Thus, for the final version of the website, we came up with a five step, sequential user story. In each step, the user runs the simulation developed in Chapter 8 with a different coupling scheme or post-processing method and can observe how these changes affect the convergence, stability and runtime of the simulation. We start with a basic setup and gradually introduce the user to more sophisticated coupling schemes motivated by some problem with the current setup. For example, we introduce implicit coupling schemes after our first simulation diverges. Let us take now take a look at each step in detail:

Step 1: Serial Explicit Coupling
The first setup features simplest way of coupling two solvers: explicit coupling. In this scheme, each solver computes the solution in each timestep independently and the results are simply mapped between the meshes. This is quite fast and works for FSI simulations if the deformations are very small. In our case, this provides physical results for a very limited number of timesteps. So, for the first simulation we use explicit coupling and compute a solution for 20 timesteps.

Step 2: Serial Explicit Coupling – Longer Simulation
After getting our feet wet with the first simulation, the second step allows the user to run the same simulation for 50 timesteps. This leads to a divergent simulation due to conjunction of our explicit coupling approach and larger deformations. This is used as a motivation for the introduction of implicit coupling approaches.

Step 3: Serial Implicit Coupling
After presenting the diverging results of the second step, we introduce the user to implicit coupling. Here, both the solvers iteratively solve each timestep multiple times until the results on both sides of the interface match. This leads to a stable simulation and we simulate for 400 timesteps. The results thus obtained show a full oscillation of the flap.

Step 4: Serial Implicit Coupling with Post-processing
Having achieved a stable simulation in the third step, we now try to make it faster. Here we introduce Quasi-Newton post-processing in order to speed up the solution of the coupling equations. This simulation runs slightly faster than the previous one and produces the same results.

Step 5: Parallel Implicit Coupling with Post-processing
In order to significantly improve the runtime, we finally introduce the user to parallel coupling. In this scheme, both solvers compute each timestep at the same time in parallel rather than in a serial, alternating fashion. This has a significant impact on the runtime and the results are still qualitatively the same as for the previous two steps. The user thus ends up with a stable and fast simulation setup.

After completing the online tutorial, the user is presented a link to a github version of our tutorial. This offline version is the subject of Chapter 10.

10 Offline Tutorial

10.1 First Iteration

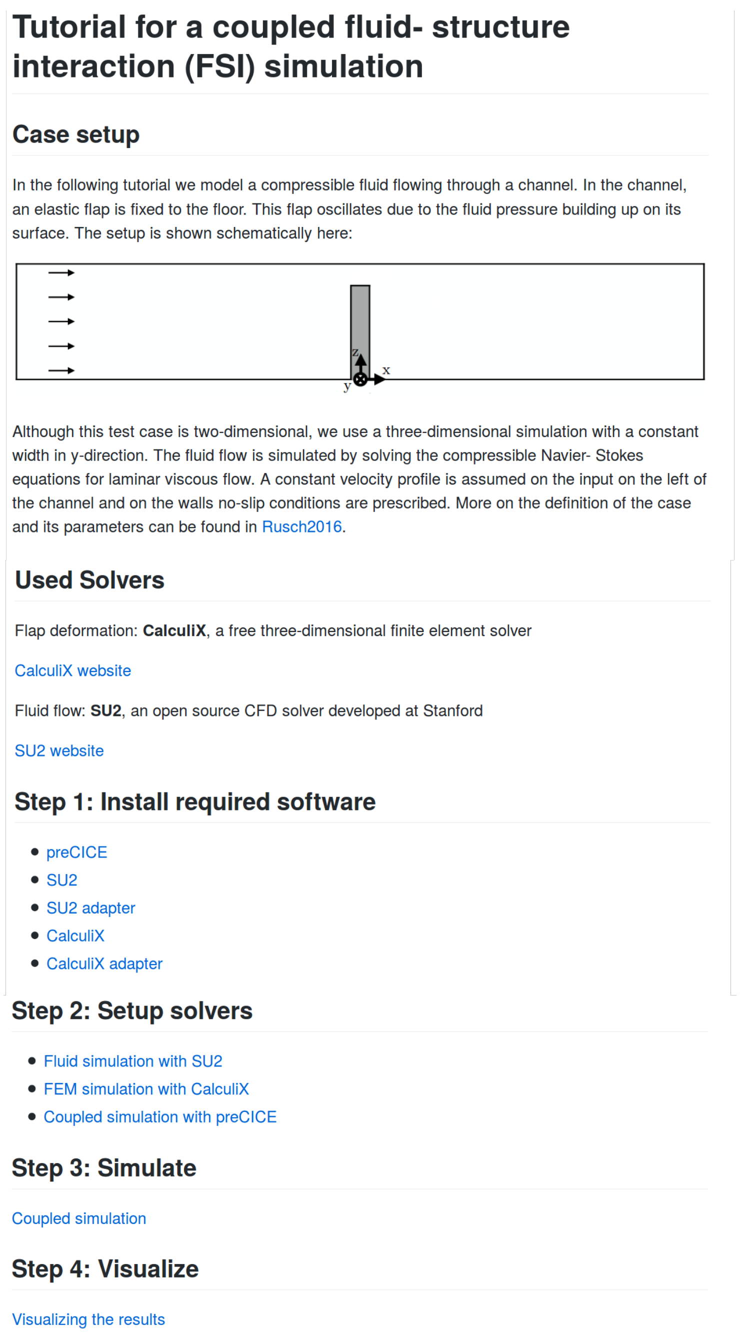
Before creating an interactive online tutorial, we created a static version of our tutorial which describes a typical simulation setup with preCICE. This offline version evolved along with our simulation setups as described in Chapters 6-8. To put this tutorial on the web, we created a "Wiki" page on a forked version of the preCICE gitHub repository. It consists of a home page that includes the general description of the test case, similar to the one given in Section 5.3, as well as presenting the software needed for running the simulation. The four key steps of the simulation pipeline are described in detail on different pages of this wiki. The wiki home page provides links to these, as well as some external websites. The first step consists of the installation of all the software; see Section 5.1. After installing the software, we set up the solvers and preCICE. This constitutes step two. Step three is the simulation itself and step four includes the visualization of the results. To get an impression about this tutorial, one can look at the screenshot in Figure 10.1.

10.2 Second Iteration

After developing the new version of the coupled scenario, as presented in chapter 7, we updated the github-tutorial and included more information about the preCICE configuration file and solver parameters. Additionally we provided scripts for plotting the displacement as well as a handy bash script, based on **tmux** [14], that allows users to run simulations easily without the need to explicitly split the terminal. The script also filters out all the unnecessary output. All used files and the tutorial description are provided on the official preCICE github page.

10.3 Third Iteration

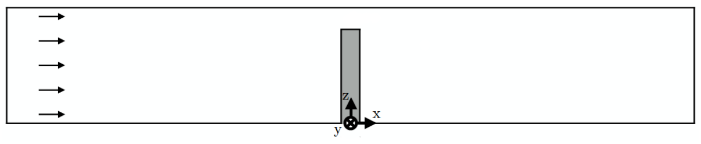
The github tutorial was adapted to the new setup derived in Chapter 8. Only one version of the configuration is provided, instead of all five versions included in the website. All the pages on the wiki were revised and have been updated to the latest version of the test case. A few changes to the github tutorial were also made by people from the preCICE community and they will actively maintain the tutorial in the future.



Tutorial for a coupled fluid- structure interaction (FSI) simulation

Case setup

In the following tutorial we model a compressible fluid flowing through a channel. In the channel, an elastic flap is fixed to the floor. This flap oscillates due to the fluid pressure building up on its surface. The setup is shown schematically here:



Although this test case is two-dimensional, we use a three-dimensional simulation with a constant width in y-direction. The fluid flow is simulated by solving the compressible Navier- Stokes equations for laminar viscous flow. A constant velocity profile is assumed on the input on the left of the channel and on the walls no-slip conditions are prescribed. More on the definition of the case and its parameters can be found in Rusch2016.

Used Solvers

Flap deformation: **CalculiX**, a free three-dimensional finite element solver

CalculiX website

Fluid flow: **SU2**, an open source CFD solver developed at Stanford

SU2 website

Step 1: Install required software

- preCICE
- SU2
- SU2 adapter
- CalculiX
- CalculiX adapter

Step 2: Setup solvers

- Fluid simulation with SU2
- FEM simulation with CalculiX
- Coupled simulation with preCICE

Step 3: Simulate

Coupled simulation

Step 4: Visualize

Visualizing the results

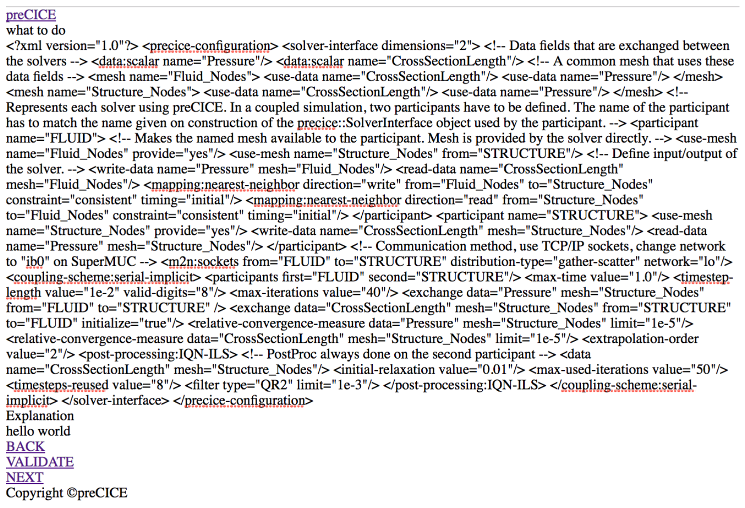
Figure 10.1: Home page of the non-interactive Wiki tutorial. URL: https://github.com/sltmyr/precice/wiki

11 Website: First Iteration

Armed with the knowledge of the different simulation setups, we can now use them to put the development of the website in context. As discussed previously, we used a React-Redux framework to implement the website. During the first iteration of the website, we settled technology decisions, as discussed in Chapter 2, and created mock web pages for the website. In the following sections we will present the mock-ups and take a look at the implementation of various elements.

11.1 The Elements of Style

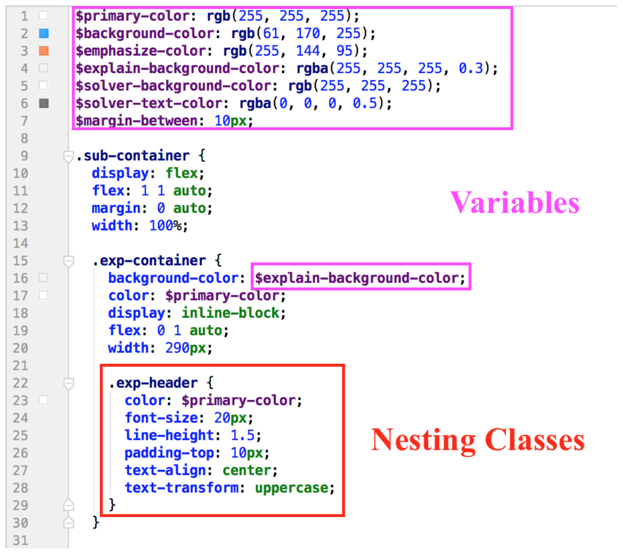
While we can use HTML to divide our screen into different regions, it is style that makes a web page truly come alive. A web page without style is boring and most users would leave as soon as they find themselves stranded on a page similar to the one in figure 11.1. Even though this page has the same content as the one in figure 11.7, the latter obviously looks much better.



```
preCICE
what to do
<?xml version="1.0"?> <precice-configuration> <solver-interface dimensions="2"> <!-- Data fields that are exchanged between
the solvers --> <data:scalar name="Pressure"/> <data:scalar name="CrossSectionLength"/> <!-- A common mesh that uses these
data fields --> <mesh name="Fluid_Nodes"> <use-data name="CrossSectionLength"/> <use-data name="Pressure"/> </mesh>
<mesh name="Structure_Nodes"> <use-data name="CrossSectionLength"/> <use-data name="Pressure"/> </mesh> <!--
Represents each solver using preCICE. In a coupled simulation, two participants have to be defined. The name of the participant
has to match the name given on construction of the precice::SolverInterface object used by the participant. --> <participant
name="FLUID"> <!-- Makes the named mesh available to the participant. Mesh is provided by the solver directly. --> <use-mesh
name="Fluid_Nodes" provide="yes"/> <use-mesh name="Structure_Nodes" from="STRUCTURE"/> <!-- Define input/output of
the solver. --> <write-data name="Pressure" mesh="Fluid_Nodes"/> <read-data name="CrossSectionLength"
mesh="Fluid_Nodes"/> <mapping:nearest-neighbor direction="write" from="Fluid_Nodes" to="Structure_Nodes"
constraint="consistent" timing="initial"/> <mapping:nearest-neighbor direction="read" from="Structure_Nodes"
to="Fluid_Nodes" constraint="consistent" timing="initial"/> </participant> <participant name="STRUCTURE"> <use-mesh
name="Structure_Nodes" provide="yes"/> <write-data name="CrossSectionLength" mesh="Structure_Nodes"/> <read-data
name="Pressure" mesh="Structure_Nodes"/> </participant> <!-- Communication method, use TCP/IP sockets, change network
to "ib0" on SuperMUC --> <m2n:sockets from="FLUID" to="STRUCTURE" distribution-type="gather-scatter" network="lo"/>
<coupling-scheme:serial-implicit> <participants first="FLUID" second="STRUCTURE"/> <max-time value="1.0"/> <timestep-
length value="1e-2" valid-digits="8"/> <max-iterations value="40"/> <exchange data="Pressure" mesh="Structure_Nodes"
from="FLUID" to="STRUCTURE" /> <exchange data="CrossSectionLength" mesh="Structure_Nodes" from="STRUCTURE"
to="FLUID" initialize="true"/> <relative-convergence-measure data="Pressure" mesh="Structure_Nodes" limit="1e-5"/>
<relative-convergence-measure data="CrossSectionLength" mesh="Structure_Nodes" limit="1e-5"/> <extrapolation-order
value="2"/> <post-processing:IQN-ILS> <!-- PostProc always done on the second participant --> <data
name="CrossSectionLength" mesh="Structure_Nodes"/> <initial-relaxation value="0.01"/> <max-used-iterations value="50"/>
<timesteps-reused value="8"/> <filter type="QR2" limit="1e-3"/> </post-processing:IQN-ILS> </coupling-scheme:serial-
implicit> </solver-interface> </precice-configuration>
Explanation
hello world
BACK
VALIDATE
NEXT
Copyright ©preCICE
```

Figure 11.1: Web Page Without Style

As we discussed in section 2.3.2, we use SASS for styling instead of CSS. To be more specific, we use SCSS, the latest version of SASS, which is compiled to CSS3. SCSS supports CSS since it is a CSS superset and provides additional features such as variables and nested styling. Figure 11.2 presents a sample code segment from our implementation of the mock-ups and it shows how we used these features.



```
$primary-color: rgb(255, 255, 255);
$background-color: rgb(61, 170, 255);
$emphasize-color: rgb(255, 144, 95);
$explain-background-color: rgba(255, 255, 255, 0.3);
$solver-background-color: rgb(255, 255, 255);
$solver-text-color: rgba(0, 0, 0, 0.5);
$margin-between: 10px;

.sub-container {
  display: flex;
  flex: 1 1 auto;
  margin: 0 auto;
  width: 100%;

  .exp-container {
    background-color: $explain-background-color;
    color: $primary-color;
    display: inline-block;
    flex: 0 1 auto;
    width: 290px;

    .exp-header {
      color: $primary-color;
      font-size: 20px;
      line-height: 1.5;
      padding-top: 10px;
      text-align: center;
      text-transform: uppercase;
    }
  }

```

Figure 11.2: Sample Code for SCSS

Variables CSS does not allow us to use variables but SCSS does. We can use variables to store styles such as background color and border width. With these variables in hand, we can rapidly change the style on multiple pages.

Nested Styling As we said before, we divide our screen space into several regions. These regions might be further subdivided in order to accommodate certain components. With nested classes, we can set general styling features in the outer class, and set the specific ones in the nested class.

11.2 Composing Structures with React

As mentioned in Section 2.3.1, we use React-Redux as our frontend framework. In what follows, we will discuss how we used this framework to create the mock-ups. Figure 11.3 shows the general structure of a mock-up page based on the React framework, with React components nested inside other React components. This composed structure allows making local dynamic changes to components that only trigger a local component update.

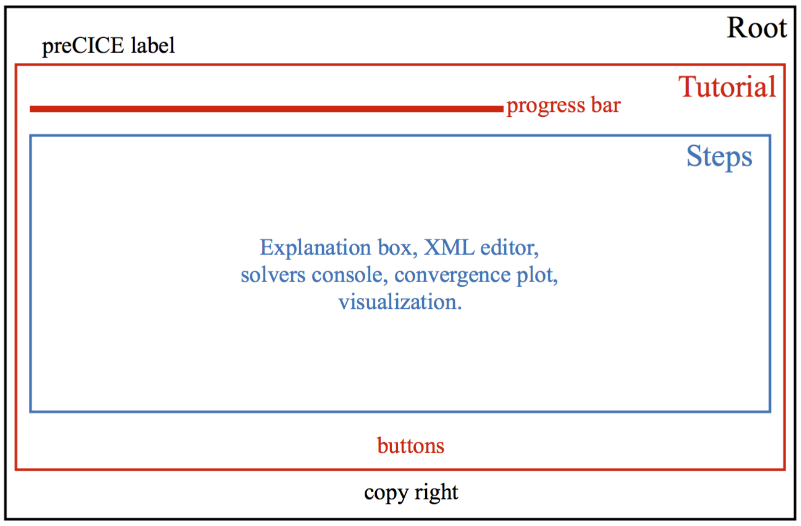

Root
preCICE label
Tutorial
progress bar
Steps
Explanation box, XML editor, solvers console, convergence plot, visualization.
buttons
copy right

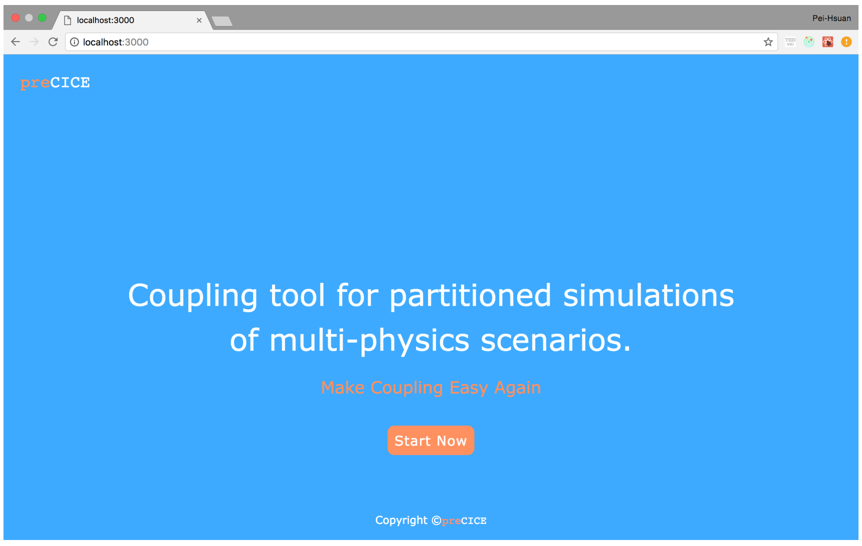
Figure 11.3: Structure of the Website

With React we only re-render the components that have been changed. This is made possible by the Virtual DOM provided by React. Referring to figure 11.3, the copyright and preCICE label will never change during the the tutorial. Therefore, we placed them in the Root layer. The progress bar and the buttons will be present on all web pages for the duration of the tutorial and therefore they were placed in the Tutorial layer. The ”Steps” layer represents the four basic steps in tutorial and it will have a different layout for each of these steps.

11.3 Tutorial Mock-Ups

With the general structure of the web pages having been defined, we shifted our attention to deciding the actual structure of the website. For this, we looked at the typical simulation setups in the offline tutorial. Thus, we came up with five different pages: Landing Page, Step One for introducing the test case, Step Two for the setup, Step Three for the simulation itself, and Step Four for visualization of the results. Since the basic structure of the final website is the same as this version, most of what is said below also applies to the final iteration.

Landing Page This is the first page users see when they open the website. After clicking the ”Start Now” button they can begin the tutorial. Figure 11.4 shows the Landing page of our website.

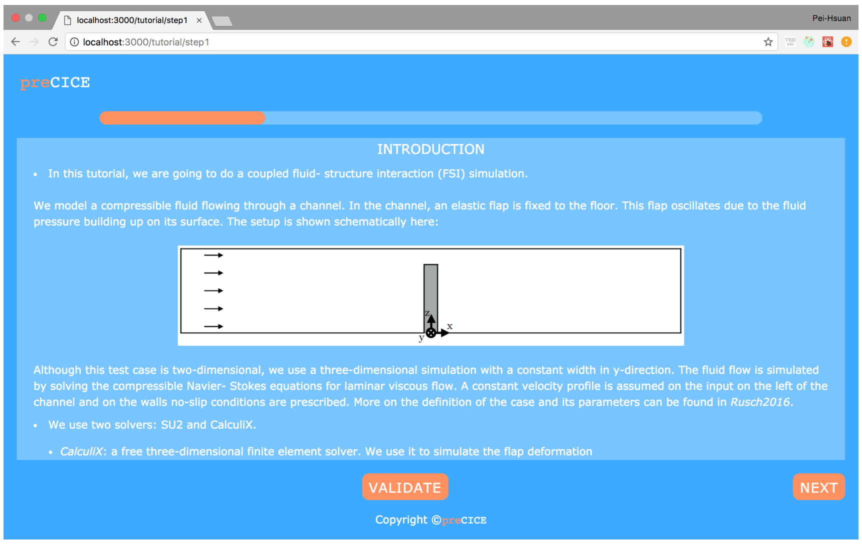


Coupling tool for partitioned simulations of multi-physics scenarios.

Make Coupling Easy Again

Figure 11.4: Landing Page of the Website

Introduction Page Before the users start the tutorial, we want to give them a brief overview of the tutorial. This page presents the simulation scenario, mentions the solvers, and explains how we use preCICE to couple the solvers. Figure 11.5 shows a screenshot of this page.



INTRODUCTION

- In this tutorial, we are going to do a coupled fluid- structure interaction (FSI) simulation.

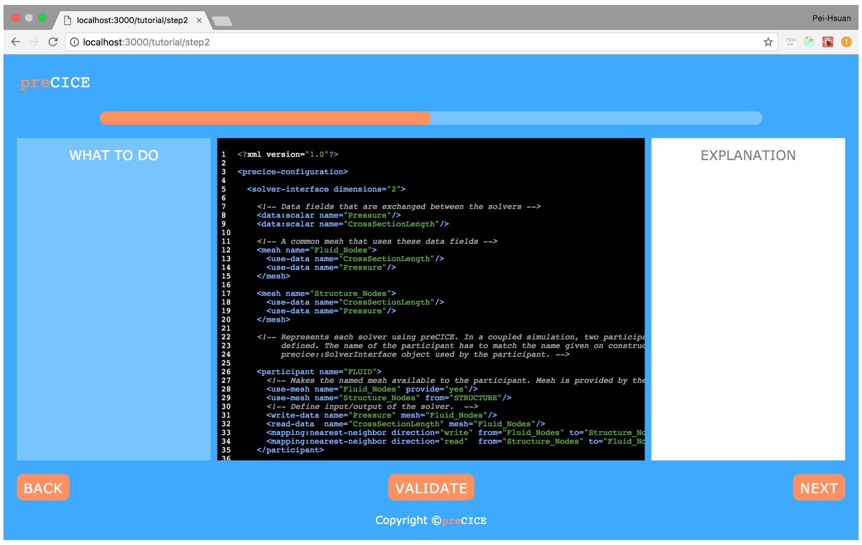
We model a compressible fluid flowing through a channel. In the channel, an elastic flap is fixed to the floor. This flap oscillates due to the fluid pressure building up on its surface. The setup is shown schematically here:

Although this test case is two-dimensional, we use a three-dimensional simulation with a constant width in y-direction. The fluid flow is simulated by solving the compressible Navier- Stokes equations for laminar viscous flow. A constant velocity profile is assumed on the input on the left of the channel and on the walls no-slip conditions are prescribed. More on the definition of the case and its parameters can be found in *Rusch2016*.

- We use two solvers: SU2 and CalculiX.
  - *CalculiX*: a free three-dimensional finite element solver. We use it to simulate the flap deformation

Figure 11.5: Introduction of the Tutorial

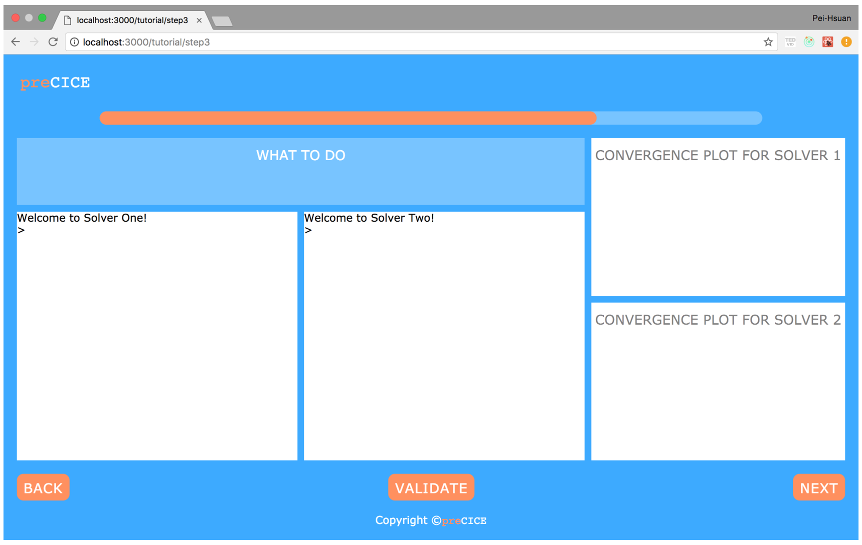
Step Two Page In this step the users are taken on a guided tour of the preCICE XML configuration file. The page is divided into three region as in 11.6. The users first get a general idea of the role the XML file plays in coupling solvers using preCICE in the ”What To Do” section. The XML editor is in the middle and it shows the configuration file for the simulation scenario presented on the previous page. In the ”Explanation” section, the users might be instructed to change certain parameters in the file and how this will affect the simulation. At this stage of the development, we thought this step might be later divided into several sub-steps, depending on the tutorial stories we would come up with. As you go through the next few chapters, you will see how these forecasts were realized.



```xml
<?xml version="1.0"?>

<precice-configuration>

  <solver-interface dimensions="2">

    <!-- Data fields that are exchanged between the solvers -->
    <data:scalar name="Pressure"/>
    <data:scalar name="CrossSectionLength"/>

    <!-- A common mesh that uses these data fields -->
    <mesh name="Fluid_Nodes">
      <use-data name="CrossSectionLength"/>
      <use-data name="Pressure"/>
    </mesh>

    <mesh name="Structure_Nodes">
      <use-data name="CrossSectionLength"/>
      <use-data name="Pressure"/>
    </mesh>

    <!-- Represents each solver using preCICE. In a coupled simulation, two participa
         defined. The name of the participant has to match the name given on construc
         precice::SolverInterface object used by the participant. -->

    <participant name="FLUID">
      <!-- Makes the named mesh available to the participant. Mesh is provided by the
      <use-mesh name="Fluid_Nodes" provide="yes"/>
      <use-mesh name="Structure_Nodes" from="STRUCTURE"/>
      <!-- Define input/output of the solver.  -->
      <write-data name="Pressure" mesh="Fluid_Nodes"/>
      <read-data  name="CrossSectionLength" mesh="Fluid_Nodes"/>
      <mapping:nearest-neighbor direction="write" from="Fluid_Nodes" to="Structure_No
      <mapping:nearest-neighbor direction="read"  from="Structure_Nodes" to="Fluid_No
    </participant>
```

Figure 11.6: Step Two of the Website

Step Three Page In this step, the users will run the two solvers simultaneously in two separate consoles as in figure 11.7. Again, in the "What To Do" section, the users can get a general idea of the tasks they have to perform in this section. Here our plans for the future included convergence plots for the solvers and visualization of other relevant metrics.


localhost:3000/tutorial/step3
Pei-Hsuan
localhost:3000/tutorial/step3
preCICE
WHAT TO DO
CONVERGENCE PLOT FOR SOLVER 1
CONVERGENCE PLOT FOR SOLVER 2
BACK
VALIDATE
NEXT
Copyright ©preCICE

```
Welcome to Solver One!
>
```

```
Welcome to Solver Two!
>
```

Figure 11.7: Step Three of the Website

Step Four Page The last part of the tutorial is the visualization step. The users will be able to see the results of the simulation on this page. The "What To Do" here explains the results.

Figure 11.8 shows the Step Four page.

b

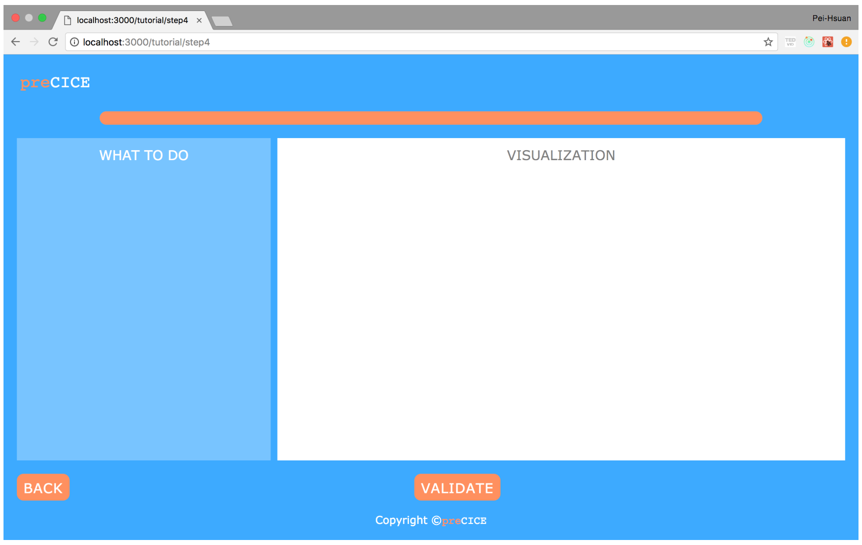


Figure 11.8: Step Four of the Website

Progress Bar and Buttons We added a progress bar to the tutorial in order to provide the user with an estimate of how much time it might take her to finish the tutorial. The buttons at the bottom of the page are obviously for navigation. The ”NEXT” button can be used to skip the current step. The ”Validate” button did not make it to the final version of the website and was meant to function simultaneously for input validation and as a ”submit” button.

11.4 The Redux Store

As described in Chapter 2, we use Redux to manage the state of our application. Figure 11.9 illustrates the Redux work flow.

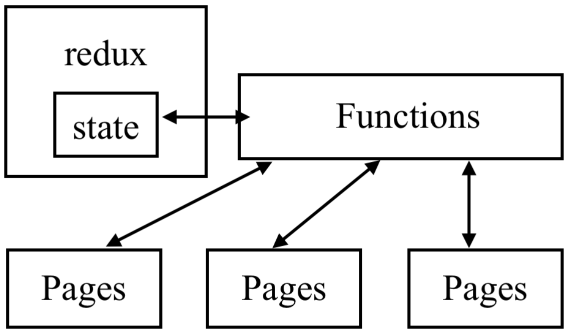


Figure 11.9: Redux Work Flow

From Figures 11.5 through 11.8, we can see that the progress bar changes depending on which step we are at currently. This is one case where we make use of the Redux store to re-render the progress bar component written using React. We extract the route from the Redux store, use it to determine our current location, and based on that we define how the progress bar should look like. We use a similar technique to decide what the buttons should do on each step. We then map the Redux state to the React component props(which stands for property in a React component) for our components and then whenever the state changes, our React components that should be affected by this state change are automatically re-rendered.

11.5 Data Visualization

During the first iteration of the website, data visualization was still an open question. We hadn't yet decided what to visualize and how to visualize it. We came up with some ideas that are listed in table 11.5 along with how much effort we thought it would take to implement these ideas. We include these relics to present an accurate historical record of the development of the project.

Which Data	Source	Concurrent /Afterwards	Visualization Tool	Effort
Console Output	Console Output	Concurrent		Moderate
Current Steps (Progess of Simulation)	Console Output	Concurrent	Progress Bar	Medium
Coupling Iterations per Time Step	Console Output	Concurrent	JS Chart Library	Medium
Coupling Residual	Console Output	Concurrent	JS Chart Library	Medium
Domain (Velocity Field, Heat Field, Structural Deformations)	Output Files	Afterwards	(1) Paraview Scripting Mode (2) Pre-rendered Image for Web Viewer (3) Fully Functional Paraview Instance on Server with Realtime Communication to Front	(1) Medium (2) High (3) Very High

Table 11.1: Analysis of Data Visualization

These decisions were much easier to make once we had settled on a preliminary version of the user story.

12 Website: Second Iteration

The second iteration of the website differed from the initial version mainly by the features added for the second milestone. These additional features together with the first iteration, constituted a base working version of the tutorial website.

12.1 General Settings and Landing Page

In implementing the new features, we made certain improvements that were not immediately visible on the website. Let us first take a look at those.

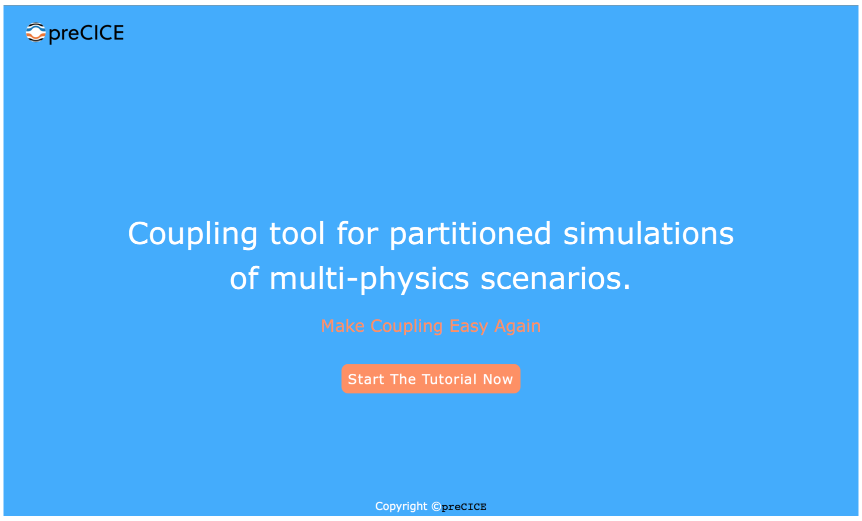


Figure 12.1: Landing Page

- **Deployment on the virtual machine**

  After running the first prototype in a development environment on the virtual machine, we changed to the robust, production ready node process manager pm2. pm2 is used widely in production environments by many companies including IBM, Microsoft and PayPal.

- **Continuous integration**

  In order to make our development process faster and more agile, we set up continuous integration. Whenever someone pushes changes on the master branch of either the frontend or the backend repository, the new version is deployed on our VM. This allowed us to make quick adaptations when reacting to user feedback later.

- **preCICE logo**

  The new preCICE logo was released in November 2017 and we included it in the website during the second iteration. Additionally, clicking on the logo now took you back to the landing page.

- **The figure rendering problem**

  When it comes to rendering figures for the website, there are multiple concerns that we need to keep in view. First, we are using React to render the figures. This means that figures, such as the preCICE logo, should be imported as static elements. Otherwise the figures might disappear when the user moves between pages.

  Another concern is that the figures take up a huge amount of space which makes the rendering slow. When the website is on the virtual machine, users need to download all the figures before rendering them and this can take a lot of time. We compressed the figures and that mitigated the problem to a degree.

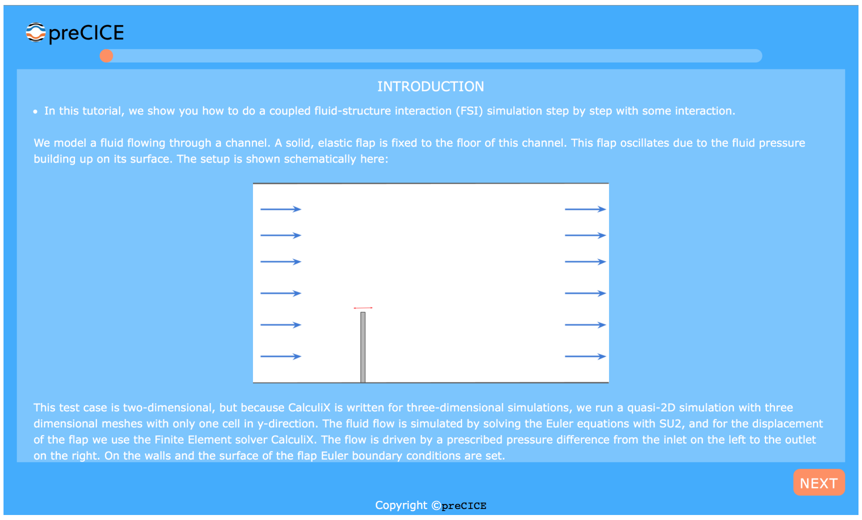
- **Information and explanation updated to new setup**

  As you probably know by now, the simulation setups changed in conjunction with the website. The second iteration of the website, therefore, presented the second iteration of the tutorial. This required us to adapt the explanations for the first two steps.

- **Overlays to help users navigate**

  To help the users navigate the website, we added overlays to steps two and three. For further detail please see section 12.3.

12.2 Step One: Introduction



INTRODUCTION

- In this tutorial, we show you how to do a coupled fluid-structure interaction (FSI) simulation step by step with some interaction.

We model a fluid flowing through a channel. A solid, elastic flap is fixed to the floor of this channel. This flap oscillates due to the fluid pressure building up on its surface. The setup is shown schematically here:

This test case is two-dimensional, but because CalculiX is written for three-dimensional simulations, we run a quasi-2D simulation with three dimensional meshes with only one cell in y-direction. The fluid flow is simulated by solving the Euler equations with SU2, and for the displacement of the flap we use the Finite Element solver CalculiX. The flow is driven by a prescribed pressure difference from the inlet on the left to the outlet on the right. On the walls and the surface of the flap Euler boundary conditions are set.

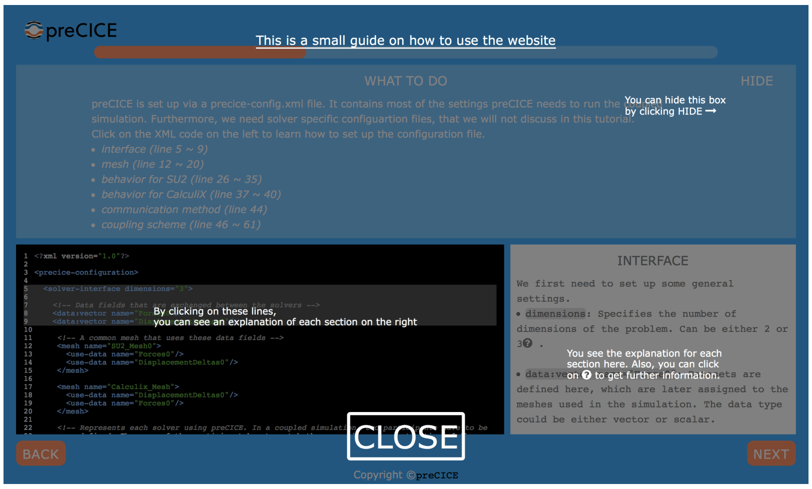
Copyright ©preCICE

Figure 12.2: Introduction page

The only change to the page for step one concerned the explanation regarding the new simulation setup.

12.3 Step Two: Setup

- **Text Overlays**


preCICE
This is a small guide on how to use the website
WHAT TO DO
HIDE
You can hide this box by clicking HIDE ⟶
By clicking on these lines,
you can see an explanation of each section on the right
You see the explanation for each section here. Also, you can click on ? to get further information.
CLOSE
BACK
NEXT
Copyright ©preCICE

preCICE is set up via a precice-config.xml file. It contains most of the settings preCICE needs to run the coupled simulation. Furthermore, we need solver specific configuartion files, that we will not discuss in this tutorial.
Click on the XML code on the left to learn how to set up the configuration file.
- *interface (line 5 ~ 9)*
- *mesh (line 12 ~ 20)*
- *behavior for SU2 (line 26 ~ 35)*
- *behavior for CalculiX (line 37 ~ 40)*
- *communication method (line 44)*
- *coupling scheme (line 46 ~ 61)*

```
1  <?xml version="1.0"?>
2
3  <precice-configuration>
4
5    <solver-interface dimensions="3">
6
7      <!-- Data fields that are exchanged between the solvers -->
8      <data:vector name="Forces0"/>
9      <data:vector name="DisplacementDeltas0"/>
10
11      <!-- A common mesh that uses these data fields -->
12      <mesh name="SU2_Mesh0">
13        <use-data name="Forces0"/>
14        <use-data name="DisplacementDeltas0"/>
15      </mesh>
16
17      <mesh name="Calculix_Mesh">
18        <use-data name="DisplacementDeltas0"/>
19        <use-data name="Forces0"/>
20      </mesh>
21
22      <!-- Represents each solver using preCICE. In a coupled simulation, two participants have to be
```

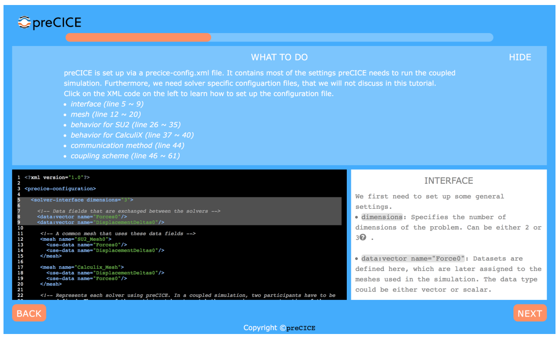
INTERFACE

We first need to set up some general settings.
- dimensions: Specifies the number of dimensions of the problem. Can be either 2 or 3? .
- data:vector name="Force0": Datasets are defined here, which are later assigned to the meshes used in the simulation. The data type could be either vector or scalar.

Figure 12.3: Overlay

When users arrived at the step two page, the first thing they saw was the overlay. This pointed out the location of the hide button, explained how to interact with the configuration file, and pointed out where they can find explanations on the configuration file. This feature was left out of the final version of the website.

- **New layout and hide function**



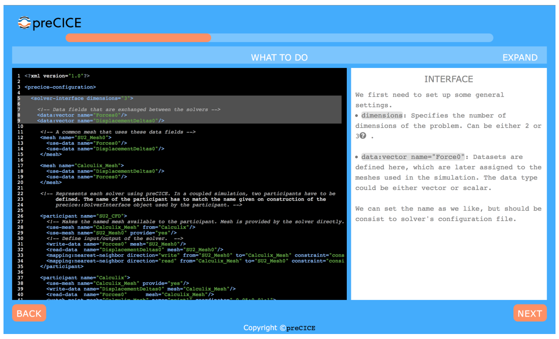
preCICE is set up via a precice-config.xml file. It contains most of the settings preCICE needs to run the coupled simulation. Furthermore, we need solver specific configuartion files, that we will not discuss in this tutorial.
Click on the XML code on the left to learn how to set up the configuration file.
- *interface (line 5 ~ 9)*
- *mesh (line 12 ~ 20)*
- *behavior for SU2 (line 26 ~ 35)*
- *behavior for CalculiX (line 37 ~ 40)*
- *communication method (line 44)*
- *coupling scheme (line 46 ~ 61)*

```
1  <?xml version="1.0"?>
2
3  <precice-configuration>
4
5    <solver-interface dimensions="3">
6
7      <!-- Data fields that are exchanged between the solvers -->
8      <data:vector name="Forces0"/>
9      <data:vector name="DisplacementDeltas0"/>
10
11      <!-- A common mesh that uses these data fields -->
12      <mesh name="SU2_Mesh0">
13        <use-data name="Forces0"/>
14        <use-data name="DisplacementDeltas0"/>
15      </mesh>
16
17      <mesh name="Calculix_Mesh">
18        <use-data name="DisplacementDeltas0"/>
19        <use-data name="Forces0"/>
20      </mesh>
21
22      <!-- Represents each solver using preCICE. In a coupled simulation, two participants have to be
```

INTERFACE

We first need to set up some general settings.
- dimensions: Specifies the number of dimensions of the problem. Can be either 2 or 3? .
- data:vector name="Force0": Datasets are defined here, which are later assigned to the meshes used in the simulation. The data type could be either vector or scalar.

Figure 12.4: Step 2 new layout



```
1  <?xml version="1.0"?>
2
3  <precice-configuration>
4
5    <solver-interface dimensions="3">
6
7      <!-- Data fields that are exchanged between the solvers -->
8      <data:vector name="Forces0"/>
9      <data:vector name="DisplacementDeltas0"/>
10
11      <!-- A common mesh that uses these data fields -->
12      <mesh name="SU2_Mesh0">
13        <use-data name="Forces0"/>
14        <use-data name="DisplacementDeltas0"/>
15      </mesh>
16
17      <mesh name="Calculix_Mesh">
18        <use-data name="DisplacementDeltas0"/>
19        <use-data name="Forces0"/>
20      </mesh>
21
22      <!-- Represents each solver using preCICE. In a coupled simulation, two participants have to be
23           defined. The name of the participant has to match the name given on construction of the
24           precice::SolverInterface object used by the participant. -->
25
26      <participant name="SU2_CFD">
27        <!-- Makes the named mesh available to the participant. Mesh is provided by the solver directly.
28        <use-mesh name="Calculix_Mesh" from="Calculix"/>
29        <use-mesh name="SU2_Mesh0" provide="yes"/>
30        <!-- Define input/output of the solver.  -->
31        <write-data name="Forces0" mesh="SU2_Mesh0"/>
32        <read-data  name="DisplacementDeltas0" mesh="SU2_Mesh0"/>
33        <mapping:nearest-neighbor direction="write" from="SU2_Mesh0" to="Calculix_Mesh" constraint="cons
34        <mapping:nearest-neighbor direction="read" from="Calculix_Mesh" to="SU2_Mesh0" constraint="consi
35      </participant>
36
37      <participant name="Calculix">
38        <use-mesh name="Calculix_Mesh" provide="yes"/>
39        <write-data name="DisplacementDeltas0" mesh="Calculix_Mesh"/>
40        <read-data  name="Forces0"      mesh="Calculix_Mesh"/>
```

INTERFACE

We first need to set up some general settings.
- dimensions: Specifies the number of dimensions of the problem. Can be either 2 or 3? .
- data:vector name="Force0": Datasets are defined here, which are later assigned to the meshes used in the simulation. The data type could be either vector or scalar.

We can set the name as we like, but should be consist to solver's configuration file.

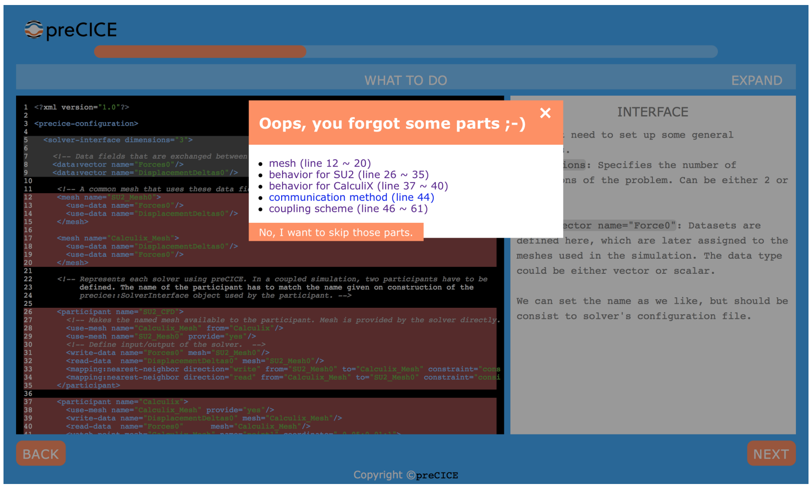
Figure 12.5: Step 2 when explanation is hidden

The mock-up layout for this step was too crowded. In order to solve this problem we moved the explanation for step two to the top. We also added the "Hide" button to the top right of the page to allow the users to collapse the explanation.

- **Interaction with the configuration file**

  One of the goals of the tutorial is to provide information on the configuration file. To achieve this, we divided the configuration file into sections depending on the function of the relevant tags. By clicking on these sections, users can get more information about the current section in the panel on the right.

- **Modal box**



```
1  <?xml version="1.0"?>
2
3  <precice-configuration>
4
5    <solver-interface dimensions="3">
6
7      <!-- Data fields that are exchanged between
8      <data:vector name="Forces0"/>
9      <data:vector name="DisplacementDeltas0"/>
10
11      <!-- A common mesh that uses these data fi
12      <mesh name="SU2_Mesh0">
13        <use-data name="Forces0"/>
14        <use-data name="DisplacementDeltas0"/>
15      </mesh>
16
17      <mesh name="Calculix_Mesh">
18        <use-data name="DisplacementDeltas0"/>
19        <use-data name="Forces0"/>
20      </mesh>
21
22      <!-- Represents each solver using preCICE. In a coupled simulation, two participants have to be
23          defined. The name of the participant has to match the name given on construction of the
24          precice::SolverInterface object used by the participant. -->
25
26      <participant name="SU2_CFD">
27        <!-- Makes the named mesh available to the participant. Mesh is provided by the solver directly.
28        <use-mesh name="Calculix_Mesh" from="Calculix"/>
29        <use-mesh name="SU2_Mesh0" provide="yes"/>
30        <!-- Define input/output of the solver.  -->
31        <write-data name="Forces0" mesh="SU2_Mesh0"/>
32        <read-data  name="DisplacementDeltas0" mesh="SU2_Mesh0"/>
33        <mapping:nearest-neighbor direction="write" from="SU2_Mesh0" to="Calculix_Mesh" constraint="cons
34        <mapping:nearest-neighbor direction="read" from="Calculix_Mesh" to="SU2_Mesh0" constraint="consi
35      </participant>
36
37      <participant name="Calculix">
38        <use-mesh name="Calculix_Mesh" provide="yes"/>
39        <write-data name="DisplacementDeltas0" mesh="Calculix_Mesh"/>
40        <read-data  name="Forces0"      mesh="Calculix_Mesh"/>
```

Oops, you forgot some parts ;-)

- mesh (line 12 ~ 20)
- behavior for SU2 (line 26 ~ 35)
- behavior for CalculiX (line 37 ~ 40)
- communication method (line 44)
- coupling scheme (line 46 ~ 61)

No, I want to skip those parts.

INTERFACE

need to set up some general

ions: Specifies the number of
ns of the problem. Can be either 2 or

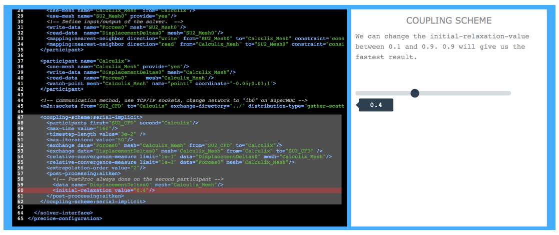
ector name="Force0": Datasets are defined here, which are later assigned to the meshes used in the simulation. The data type could be either vector or scalar.

We can set the name as we like, but should be consist to solver's configuration file.

Figure 12.6: Step 2 with modal box

If a user clicks the "NEXT" button before going through the configuration file, a modal box shows up informing them about the sections they missed. The modal box allows you to jump directly to one of the sections that you missed or to skip to the next step. This feature was only included for the first simulation in the final design.

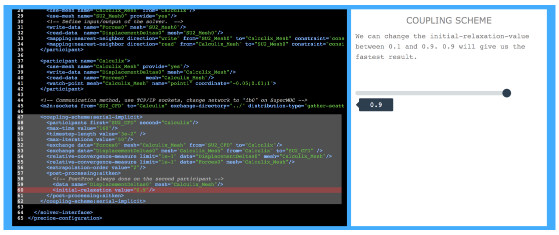
- **Slider for Aitken relaxation parameter**



COUPLING SCHEME

We can change the initial-relaxation-value between 0.1 and 0.9. 0.9 will give us the fastest result.

Figure 12.7: with relaxation value 0.4



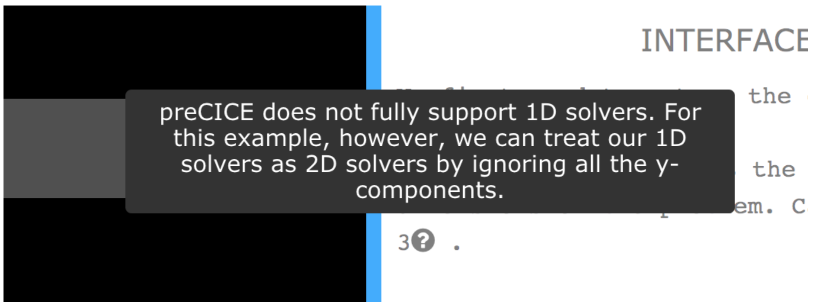
COUPLING SCHEME

We can change the initial-relaxation-value between 0.1 and 0.9. 0.9 will give us the fastest result.

Figure 12.8: with relaxation value 0.9

As described in chapter 9, our first version of the user story relied on allowing the users to change the Aitken relaxation parameter after they had performed their first simulation. When users came back to Step Two after their first simulation, the explanation panel transformed into a slider that allowed them to change the value of the parameter.

- **Tooltips for additional information**



preCICE does not fully support 1D solvers. For this example, however, we can treat our 1D solvers as 2D solvers by ignoring all the y-components.

Figure 12.9: Step 2 with a tooltip

Some users might want to know more about the configuration file. They can do this by using the ”?” tooltips scattered throughout the text in the explanation panel. Pointing at one of these symbols will bring up a a box with further details.

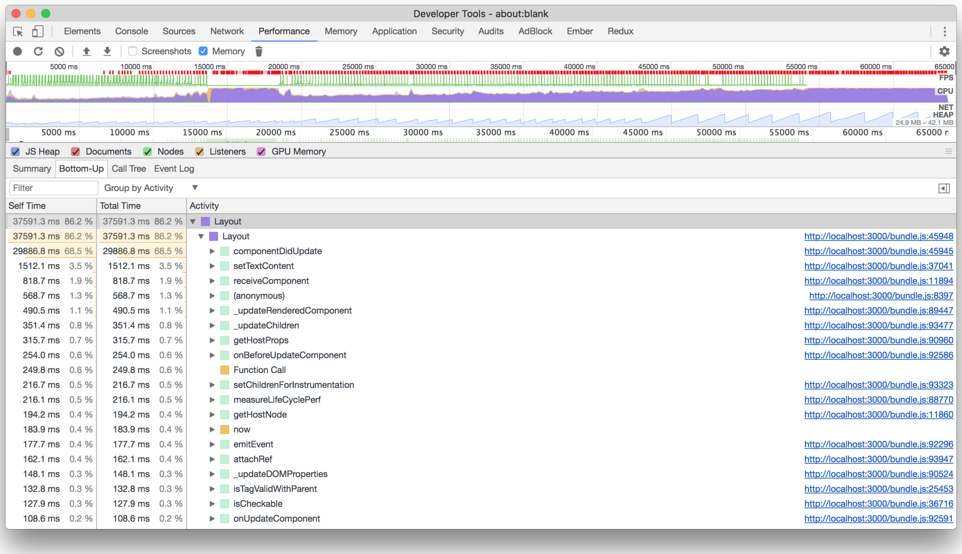
12.4 Step Three: Simulation

- **UTF-8 encoding for socket communication**

  We switched from Array Buffers to utf8 encoding for backend communication through websockets. Without the additional overhead of converting between encodings, the code becomes leaner. The loss in transmission speed is negligible. The biggest performance penalty lies in layouting and rendering in the browser

- **Architectural changes to prevent browser from crashing**

  Our first prototype had a major issue with the consoles. When running the simulation, the browser would become unresponsive after about 20 seconds. Using the chrome developer tools for profiling JavaScript performance, it became clear that the problem was caused by frequent re-rendering. Figure 12.10 provides a screenshot that depicts this. Every time the browser receives a piece of console output from the backend, it has to scroll the console window down and thus it needs to rerender parts of the window. Furthermore, every piece of console output is put into a separate element. Since console output can arrive several times per second, the browser crashed when it had to recalculate the position of too many elements. This usually happened when the browser had already received a large part of the console output.



Self Time	Total Time	Activity	
37591.3 ms 86.2 %	37591.3 ms 86.2 %	▼ Layout	
37591.3 ms 86.2 %	37591.3 ms 86.2 %	▼ Layout	http://localhost:3000/bundle.js:45948
29886.8 ms 68.5 %	29886.8 ms 68.5 %	▶ componentDidUpdate	http://localhost:3000/bundle.js:45945
1512.1 ms 3.5 %	1512.1 ms 3.5 %	▶ setTextContent	http://localhost:3000/bundle.js:37041
818.7 ms 1.9 %	818.7 ms 1.9 %	▶ receiveComponent	http://localhost:3000/bundle.js:11894
568.7 ms 1.3 %	568.7 ms 1.3 %	▶ (anonymous)	http://localhost:3000/bundle.js:8397
490.5 ms 1.1 %	490.5 ms 1.1 %	▶ _updateRenderedComponent	http://localhost:3000/bundle.js:89447
351.4 ms 0.8 %	351.4 ms 0.8 %	▶ _updateChildren	http://localhost:3000/bundle.js:93477
315.7 ms 0.7 %	315.7 ms 0.7 %	▶ getHostProps	http://localhost:3000/bundle.js:90960
254.0 ms 0.6 %	254.0 ms 0.6 %	▶ onBeforeUpdateComponent	http://localhost:3000/bundle.js:92586
249.8 ms 0.6 %	249.8 ms 0.6 %	Function Call	
216.7 ms 0.5 %	216.7 ms 0.5 %	▶ setChildrenForInstrumentation	http://localhost:3000/bundle.js:93323
216.1 ms 0.5 %	216.1 ms 0.5 %	▶ measureLifeCyclePerf	http://localhost:3000/bundle.js:88770
194.2 ms 0.4 %	194.2 ms 0.4 %	▶ getHostNode	http://localhost:3000/bundle.js:11860
183.9 ms 0.4 %	183.9 ms 0.4 %	▶ now	
177.7 ms 0.4 %	177.7 ms 0.4 %	▶ emitEvent	http://localhost:3000/bundle.js:92296
162.1 ms 0.4 %	162.1 ms 0.4 %	▶ attachRef	http://localhost:3000/bundle.js:93947
148.1 ms 0.3 %	148.1 ms 0.3 %	▶ _updateDOMProperties	http://localhost:3000/bundle.js:90524
132.8 ms 0.3 %	132.8 ms 0.3 %	▶ isTagValidWithParent	http://localhost:3000/bundle.js:25453
127.9 ms 0.3 %	127.9 ms 0.3 %	▶ isCheckable	http://localhost:3000/bundle.js:36716
108.6 ms 0.2 %	108.6 ms 0.2 %	▶ onUpdateComponent	http://localhost:3000/bundle.js:92591

Figure 12.10: Layouting after scrolling (triggered by componentDidUpdate) is the biggest performance penalty, especially when there is already a lot of text

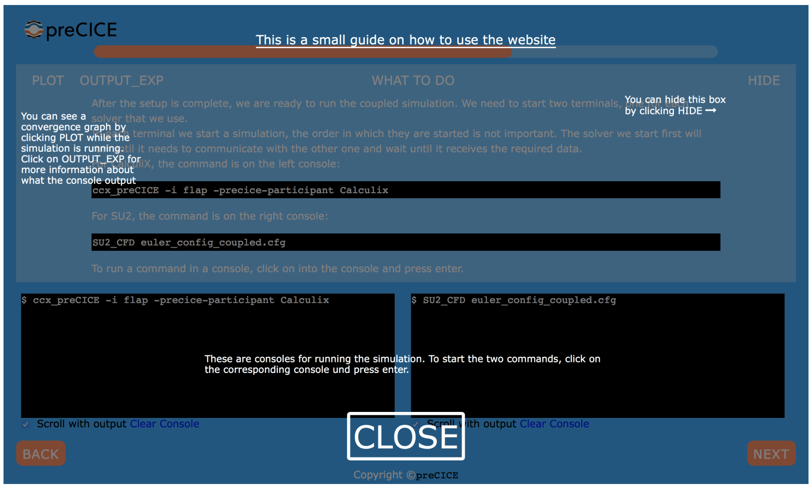
In order to solve this issue, a complete rewrite of the console component was necessary. We were using a third party library in the first prototype. Now we are using our own implementation. The new implementation strikes a good balance between creating too many elements (which is the case if a new element gets created for each line of console output) and creating bigger elements (which is the other extreme, putting all the text into one element). New elements are now created for each 1000 lines of code and there is always only one element that undergoes mutation. This reduces the amount of computation required when rerendering. All unchanged elements (their content size based dimensions) do not have to be recalculated, only their positions when scrolling. Furthermore since we now have fewer elements, repositioning when scrolling down happens in a reasonable amount of time.

- **Cached console output**

  To test our website with a local backend without a preCICE installation, we started to develop interchangeable output sources for the console output. In addition to actually running the simulation and working with real time output, it was now also possible to read the output from a text file and send its contents, within a given timespan, to the frontend. While this was only implemented for local development purposes, this might open the door to highly resource friendly backend hosting. If we only have a discrete set of simulation configurations, it makes sense to cache the output for each in a text file and use that on request.

- **Overlay**

  As in Step Two, we also added an overlay with explanations for Step Three. This can be seen in Figure 12.11.


This is a small guide on how to use the website

After the setup is complete, we are ready to run the coupled simulation. We need to start two terminals, one for each solver that we use.
In each terminal we start a simulation, the order in which they are started is not important. The solver we start first will run until it needs to communicate with the other one and wait until it receives the required data.
For CalculiX, the command is on the left console:

```
ccx_preCICE -i flap -precice-participant Calculix
```

For SU2, the command is on the right console:

```
SU2_CFD euler_config_coupled.cfg
```

To run a command in a console, click on into the console and press enter.

You can see a convergence graph by clicking PLOT while the simulation is running. Click on OUTPUT_EXP for more information about what the console output

You can hide this box by clicking HIDE ⟶

```
$ ccx_preCICE -i flap -precice-participant Calculix
```

```
$ SU2_CFD euler_config_coupled.cfg
```

These are consoles for running the simulation. To start the two commands, click on the corresponding console und press enter.

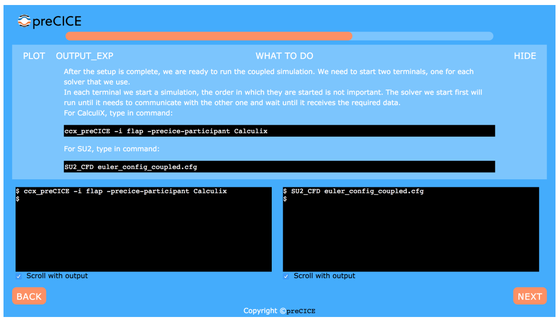
✓ Scroll with output Clear Console

✓ Scroll with output Clear Console

Copyright ©preCICE

Figure 12.11: Step 3 with Overlay

- **Layout**


After the setup is complete, we are ready to run the coupled simulation. We need to start two terminals, one for each solver that we use.
In each terminal we start a simulation, the order in which they are started is not important. The solver we start first will run until it needs to communicate with the other one and wait until it receives the required data.
For CalculiX, type in command:

```
ccx_preCICE -i flap -precice-participant Calculix
```

For SU2, type in command:

```
SU2_CFD euler_config_coupled.cfg
```

```
$ ccx_preCICE -i flap -precice-participant Calculix
$
```

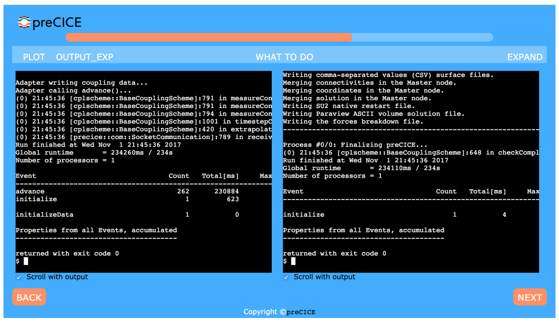
```
$ SU2_CFD euler_config_coupled.cfg
$
```

✓ Scroll with output

✓ Scroll with output

Copyright ©preCICE

Figure 12.12: Step 3 new layout


```
Adapter writing coupling data...
Adapter calling advance()...
(0) 21:45:36 [cplscheme::BaseCouplingScheme]:791 in measureCon
(0) 21:45:36 [cplscheme::BaseCouplingScheme]:791 in measureCon
(0) 21:45:36 [cplscheme::BaseCouplingScheme]:794 in measureCon
(0) 21:45:36 [cplscheme::BaseCouplingScheme]:1001 in timestepC
(0) 21:45:36 [cplscheme::BaseCouplingScheme]:420 in extrapolat
(0) 21:45:36 [precice::com::SocketCommunication]:789 in receiv
Run finished at Wed Nov  1 21:45:36 2017
Global runtime       = 234260ms / 234s
Number of processors = 1

Event                                 Count   Total[ms]     Max
---------------------------------------------------------------
advance                                 262      230884
initialize                                1         623

initializeData                            1           0

Properties from all Events, accumulated
---------------------------------------

returned with exit code 0
$ █
```

```
Writing comma-separated values (CSV) surface files.
Merging connectivities in the Master node.
Merging coordinates in the Master node.
Merging solution in the Master node.
Writing SU2 native restart file.
Writing Paraview ASCII volume solution file.
Writing the forces breakdown file.
---------------------------------------------------------------

Process #0/0: Finalizing preCICE...
(0) 21:45:36 [cplscheme::BaseCouplingScheme]:648 in checkCompl
Run finished at Wed Nov  1 21:45:36 2017
Global runtime       = 234110ms / 234s
Number of processors = 1

Event                                 Count   Total[ms]     Max
---------------------------------------------------------------

initialize                                1           4

Properties from all Events, accumulated
---------------------------------------

returned with exit code 0
$ █
```

✓ Scroll with output

✓ Scroll with output

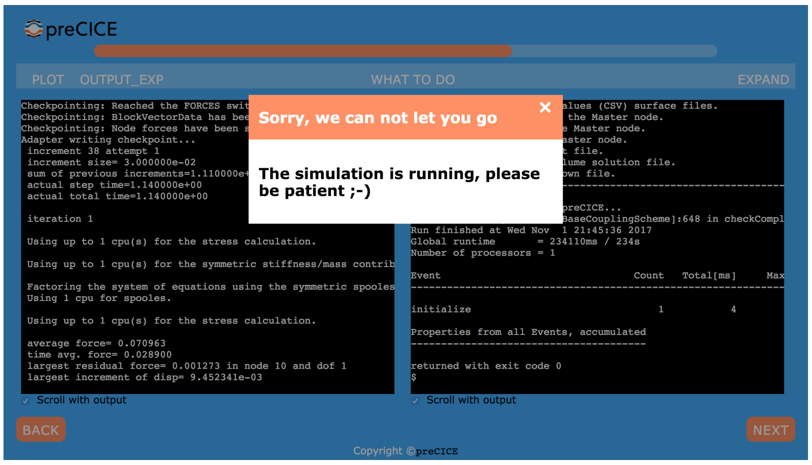
Copyright ©preCICE

Figure 12.13: Step 3 with a running simulation

Like Step Two, the mock-up for Step Three was too crowded. When users arrive at Step Three now, they only see the two consoles and instructions for the current step. The coupling iteration plot and console output explanation could now be accessed by using the buttons on the top left of the header. This last feature was slightly changed in the final iteration of the website.

- **Buttons disabled while simulation is running**

While the simulation is running, the ”BACK” and ”NEXT” buttons were disabled. If a user clicked on them during this time, they would be greeted by a modal box as in figure 12.14.



Sorry, we can not let you go

The simulation is running, please be patient ;-)

```
Checkpointing: Reached the FORCES swit
Checkpointing: BlockVectorData has bee
Checkpointing: Node forces have been s
Adapter writing checkpoint...
 increment 38 attempt 1
 increment size= 3.000000e-02
 sum of previous increments=1.110000e+
 actual step time=1.140000e+00
 actual total time=1.140000e+00

 iteration 1

 Using up to 1 cpu(s) for the stress calculation.

 Using up to 1 cpu(s) for the symmetric stiffness/mass contrib

 Factoring the system of equations using the symmetric spooles
 Using 1 cpu for spooles.

 Using up to 1 cpu(s) for the stress calculation.

 average force= 0.070963
 time avg. forc= 0.028900
 largest residual force= 0.001273 in node 10 and dof 1
 largest increment of disp= 9.452341e-03
```

```
alues (CSV) surface files.
 the Master node.
e Master node.
aster node.
t file.
lume solution file.
own file.
-----------------------------------------

preCICE...
BaseCouplingScheme]:648 in checkCompl
Run finished at Wed Nov  1 21:45:36 2017
Global runtime        = 234110ms / 234s
Number of processors = 1

Event                           Count   Total[ms]     Max
-----------------------------------------------------------

initialize                          1           4

Properties from all Events, accumulated
---------------------------------------

returned with exit code 0
$
```

Figure 12.14: Step 3 with modal box

- **User story**

  With the possibility for the user to change the relaxation value, we implemented the first real interaction of the user with the simulation setup. After a user had performed their first simulation with the default configuration, they could go back to Step Two and change the value of the Aitken relaxation parameter. The configuration value was then sent to the backend where the simulation was started with a configuration file that contained the specified numerical value for the relaxation parameter.

- **Convergence plot and progress bar**

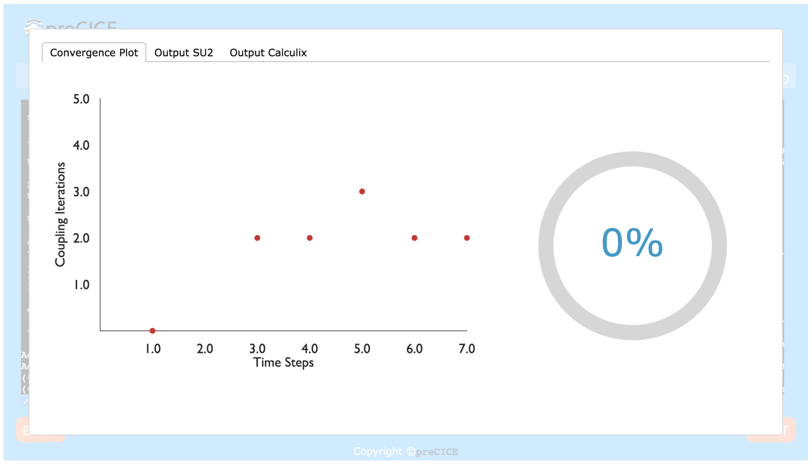
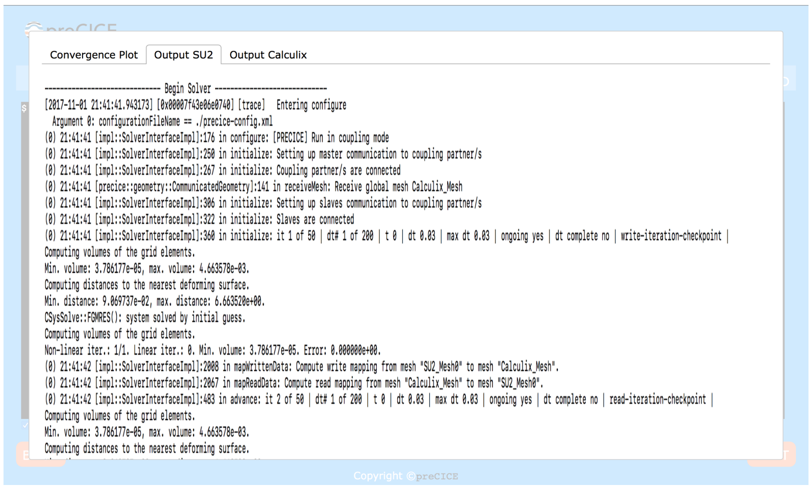


Figure 12.15: Step 3 with convergence plot

Starting the simulation brought up the coupling iteration plot to the foreground. This window also contained a progress bar for the simulation. Users could also access this from the buttons on the top left, as described above. These features were also redesigned in the final iteration of the website.

- **Console output explanation**

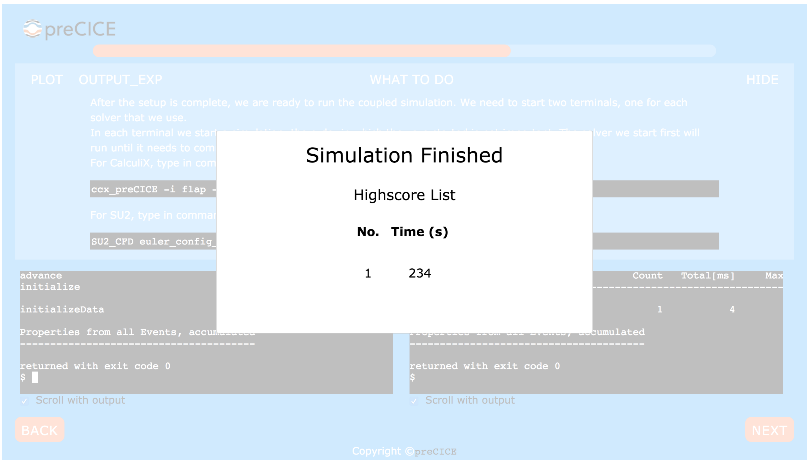


```
----------------------------- Begin Solver ----------------------------
[2017-11-01 21:41:41.943173] [0x00007f43e06e0740] [trace]   Entering configure
  Argument 0: configurationFileName == ./precice-config.xml
(0) 21:41:41 [impl::SolverInterfaceImpl]:176 in configure: [PRECICE] Run in coupling mode
(0) 21:41:41 [impl::SolverInterfaceImpl]:250 in initialize: Setting up master communication to coupling partner/s
(0) 21:41:41 [impl::SolverInterfaceImpl]:267 in initialize: Coupling partner/s are connected
(0) 21:41:41 [precice::geometry::CommunicatedGeometry]:141 in receiveMesh: Receive global mesh Calculix_Mesh
(0) 21:41:41 [impl::SolverInterfaceImpl]:306 in initialize: Setting up slaves communication to coupling partner/s
(0) 21:41:41 [impl::SolverInterfaceImpl]:322 in initialize: Slaves are connected
(0) 21:41:41 [impl::SolverInterfaceImpl]:360 in initialize: it 1 of 50 | dt# 1 of 200 | t 0 | dt 0.03 | max dt 0.03 | ongoing yes | dt complete no | write-iteration-checkpoint |
Computing volumes of the grid elements.
Min. volume: 3.786177e-05, max. volume: 4.663578e-03.
Computing distances to the nearest deforming surface.
Min. distance: 9.069737e-02, max. distance: 6.663520e+00.
CSysSolve::FGMRES(): system solved by initial guess.
Computing volumes of the grid elements.
Non-linear iter.: 1/1. Linear iter.: 0. Min. volume: 3.786177e-05. Error: 0.000000e+00.
(0) 21:41:42 [impl::SolverInterfaceImpl]:2008 in mapWrittenData: Compute write mapping from mesh "SU2_Mesh0" to mesh "Calculix_Mesh".
(0) 21:41:42 [impl::SolverInterfaceImpl]:2067 in mapReadData: Compute read mapping from mesh "Calculix_Mesh" to mesh "SU2_Mesh0".
(0) 21:41:42 [impl::SolverInterfaceImpl]:483 in advance: it 2 of 50 | dt# 1 of 200 | t 0 | dt 0.03 | max dt 0.03 | ongoing yes | dt complete no | read-iteration-checkpoint |
Computing volumes of the grid elements.
Min. volume: 3.786177e-05, max. volume: 4.663578e-03.
Computing distances to the nearest deforming surface.
```

Figure 12.16: Step 3 with output explaining section

Using the tabs on the coupling iteration window, users could shift between the explanation for Calculix and SU2 output. At that point, we were planning on developing this feature further in the next iteration of the website. However, as we will describe in Chapter 13, the user story was completely redesigned and therefore, this feature did not make the final cut.

- **Simulation High Score**



Simulation Finished

Highscore List

No.	Time (s)
1	234

After the setup is complete, we are ready to run the coupled simulation. We need to start two terminals, one for each solver that we use.
In each terminal we start

```
ccx_preCICE -i flap -
```

For SU2, type in comma

```
SU2_CFD euler_config_
```

```
advance
initialize

initializeData

Properties from all Events, accumulated
---------------------------------------

returned with exit code 0
$
```

```
                    Count   Total[ms]     Max
------------------------------------------------
                       1           4

Properties from all Events, accumulated
---------------------------------------

returned with exit code 0
$
```

Figure 12.17: Step 3 with high score table

The runtime time for each simulation was stored in the Redux Store in the frontend. Once the simulation ended, a highscore list was displayed with the simulation runtimes in ascending order. This was designed to help the users see how changing the Aitken parameter affected the runtime. Since we rethought our user story for the last iteration, this feature was also not included in the final version.

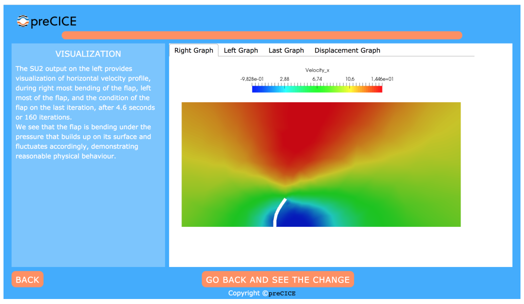
12.5 Step Four: Visualization

- **Button for going back to step 2**

  As described before, the user story for the second iteration revolved around changing the Aitken relaxation parameter. Once a user had performed their first simulation, they could go back to Step Two using a button on the Step Four page.

- **Visualization for the output**

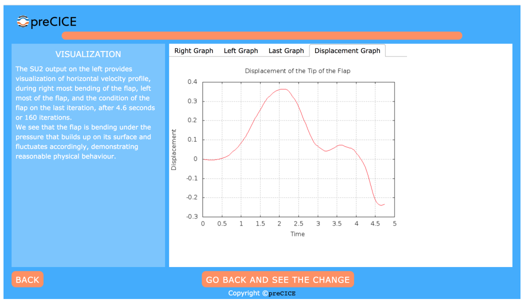
  During the second iteration, we had tabs for the various visualization graphs we included in this step, as in Figures 12.18 and 12.19 . Furthermore, without running the simulation, the user wasn't able to visualize the results as shown in Figure 12.20. However, this part was again completely redesigned for the final iteration.



VISUALIZATION

The SU2 output on the left provides visualization of horizontal velocity profile, during right most bending of the flap, left most of the flap, and the condition of the flap on the last iteration, after 4.6 seconds or 160 iterations.
We see that the flap is bending under the pressure that builds up on its surface and fluctuates accordingly, demonstrating reasonable physical behaviour.

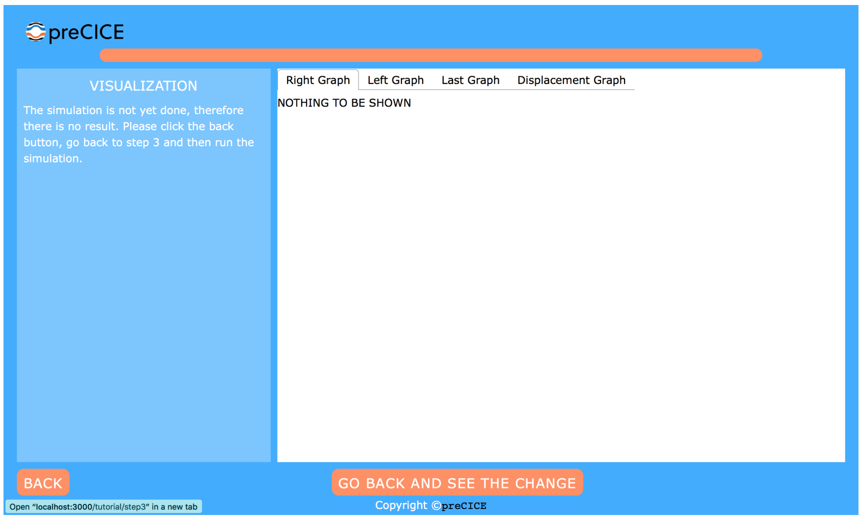
Figure 12.18: Visualization of the velocity field



VISUALIZATION

The SU2 output on the left provides visualization of horizontal velocity profile, during right most bending of the flap, left most of the flap, and the condition of the flap on the last iteration, after 4.6 seconds or 160 iterations.
We see that the flap is bending under the pressure that builds up on its surface and fluctuates accordingly, demonstrating reasonable physical behaviour.

Figure 12.19: Visualization of the displacement of the flap



VISUALIZATION

The simulation is not yet done, therefore there is no result. Please click the back button, go back to step 3 and then run the simulation.

NOTHING TO BE SHOWN

Figure 12.20: Nothing simulated

13 Website: Third Iteration

The final iteration of the website is based on the user story described in Section 9.2. This required a complete redesign of certain features of the website. Furthermore, the rerendering problem described in Chapter 12 persisted and we spent a great deal of time on solving it. In order to improve user experience, we also incorporated user feedback, gathered primarily in Stuttgart as described in Chapter 14.

13.1 General Structural Changes

During the second iteration, we had implemented most of the functional features of the website. However, as described in Section 9.2, we completely redesigned the user story for the final iteration and extended it to five cases. In order to present this on the website, we needed to modify the structure of the website. Additionally, some features, such as image compression and explanations for preCICE configuration, had to be tweaked to make them work as desired.

13.1.1 Structure: Iterative or Serial?

For the second iteration of the website, we only had one test case. The structure of the website was thus straightforward. As illustrated in figure 13.1, users were guided through the first four steps and at the last step, they had the opportunity to go back to step two, modify the Aitken parameter, and run a new simulation. Since we now have five similar simulations in the user story, each test case should have a similar user interface for the four basic steps for running a simulation. Thus, the structure of the website was changed from the iterative version in the second iteration to serial – as in a story.

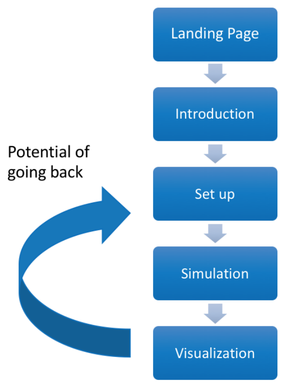


Figure 13.1: Iterative structure for the second iteration of the website

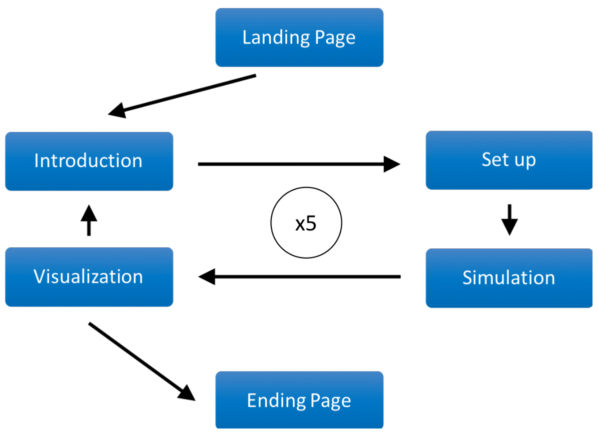


Figure 13.2: Serial structure for the third iteration of the website

The new structure divides the tutorial into five parts and is illustrated in Figure 13.2. The users first arrive at the landing page and are guided through the four steps of the first simulation, and can then go to the second part or chapter of the tutorial. The transition between different steps in different parts is transparent – that is, at any stage of the tutorial, the users can go back to an earlier step or skip a step. Once a user has finished the tutorial, they arrive at the final page which provides links to the offline tutorial described in Chapter 10, preCICE, and Coplon.

Implementing the new structure was made easier by the technology decisions we had made in the beginning of the project. We reused the React components we had created earlier for the four steps of a simulation and populated them with the appropriate content for the five parts of the new user story. Once we have these components in hand, we can use Redux to obtain the current location of the user and render the component with the right content. This structure will allow developers to easily add further test cases to the website in the future.

13.1.2 Text

Since the main focus of the project was creating a tutorial, the text on the website is therefore the main medium through which users can be educated about preCICE. It should be detailed enough to provide information necessary to understand the tutorial test case, but at the same time it should be succinct enough to not distract the user from the main thrust of the learning material.

The written material on the website can be divided into two broad categories: information pertaining to the test cases and explanations for the preCICE xml configuration file. Since a lot of information is needed for the configuration file, we explained all the tags briefly, especially the ones that differ from chapter to chapter. We used the tooltips in figure 12.9 to present additional information that did not fit into the normal flow of the text.

Information relating to the test cases was revised multiple times to include necessary information and present a coherent user story that is consistent across all five parts of the final user story. This required a great deal of deliberation and thinking and both the simulation team and the website team participated in this task. We consulted papers on preCICE and numerical methods to write the final version of the text and where needed, point the users to the source material.

13.1.3 Performance Optimizations

After the second iteration, the real time console output still had performance issues both on the backend and the frontend. Therefore, on the backend side, caching was dramatically intensified. Since the backend is now just sending recorded output to the frontend, this implementation is quite slim and would scale nicely.

On the frontend side, more work was needed. The simulation was running smoothly on Mac machines, but especially on low-end linux computers, the simulation still got stuck. The final solution we came up with dramatically reduced the amount of console output shown in the browser: now we only display the last 1000 lines. Moreover, the number of synchronizations between backend and frontend was reduced to four per second from ten per second in the second iteration. We arrived at these numbers by testing our website systematically on various computers.

13.1.4 Housekeeping

The second iteration of the website had several features that were removed or modified for the final iteration. These changes were primarily motivated by the new user story. In the following, we list the changes that were made for the final iteration.

Things we deleted:

- **Slider for Aitken relaxation parameter**

  Since the user story was changed, we no longer allow the user to change the relaxation parameter. We thus deleted the slider and the respective storage for this feature from the Redux store.

- **Overlay in step 2 and step 3**

  The overlay was designed to guide the user to important features of the user interface. However, it was deemed that the overlay made the website complicated and clunky. Therefore, in the last iteration, we tried to make the interface more ″natural″ and intuitive and as a consequence, removed the overlay.

- **Explanation of the output**

  The console output is mostly related to the two solvers, SU2 and CalculiX. Since this tutorial focuses on almost exclusively on preCICE, the output explanation was deemed superfluous.

Things we improved:

- **Portablility across browsers**

  Some CSS features produce different results on different browsers. To provide a consistent user interface, we changed these to more neutral options that led to a consistent user interface on different browses.

- **New progress bar**

  Since the new user story has five distinct chapters, we added five circles to the progress bar to indicate the current chapter. The current chapter is indicated by a hollow orange circle. After the user has completed a chapter, the corresponding circle is painted orange.

- **Videos for visualization**

  The second iteration included multiple visualization plots separated into tabs. While these plots indicated a moving flap, they were static. We decided it would be much better to do away with the plots and provide a video which showed the flap actually moving since a video is much more interesting and intuitive than a static plot.

- **Compression of figures**

  Some of the figures on the website were too large and led to significant loading times for the website. In order to reduce network traffic and make the website faster, we decreased the resolution and compressed these images.

We believe these changes made the website slicker and improved the user experience. Feedback from users, presented in Chapter 14, concurs with this assertion.

13.2 Changes to the User Interface

Taking into account feedback from the customer, internal quality assurance, and external user testing we decided to redesign several components and focus on accessibility and an intuitive interface. This was a multi-step process where we first incorporated feedback from the client. Our internal quality assurance team then evaluated these changes and came up with suggestions to further improve the website. The last step was then executed after user testing in Stuttgart.

13.2.1 Major Changes

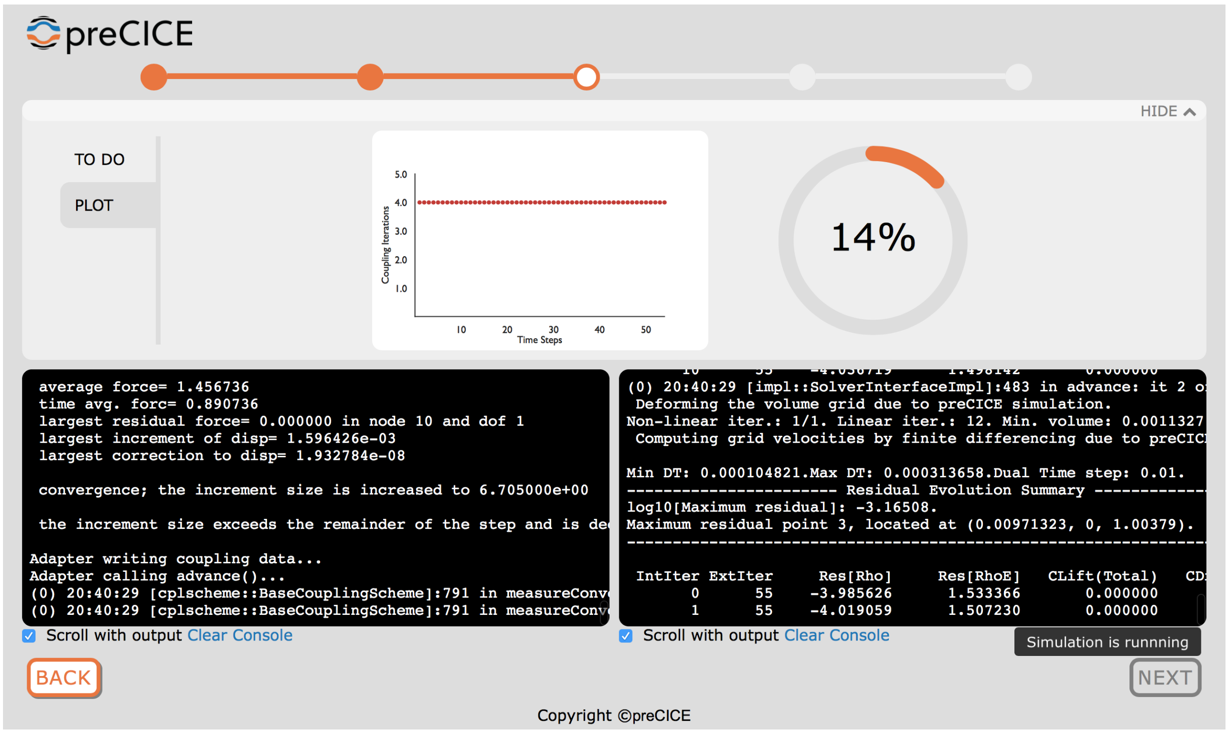
Major changes to the user interface were motivated by the feedback from the client after the second milestone and the new user story. Below, we discuss these changes in detail.

- **New color scheme**

  The website now has a new color scheme with a more moderate background color. Many other colors were adjusted, providing a distraction free interface with only important components marked in shiny colors. See, for example, Figure 13.3.

- **The coupling iteration plot**

  Now we introduce the coupling iteration plot in the third chapter of the tutorial since the first two simulations use explicit coupling. In order to introduce this to the user, the simulation page for chapter three thus introduces the user to the coupling iteration plot. This is achieved by using a tabbed display that allows the user to switch between the explanation and the plot. This feature is displayed in Figure 13.3. Additionally if the user is especially interested in the coupling iteration plot, she can click on it and zoom in on the plot, as show in Figure 13.4.



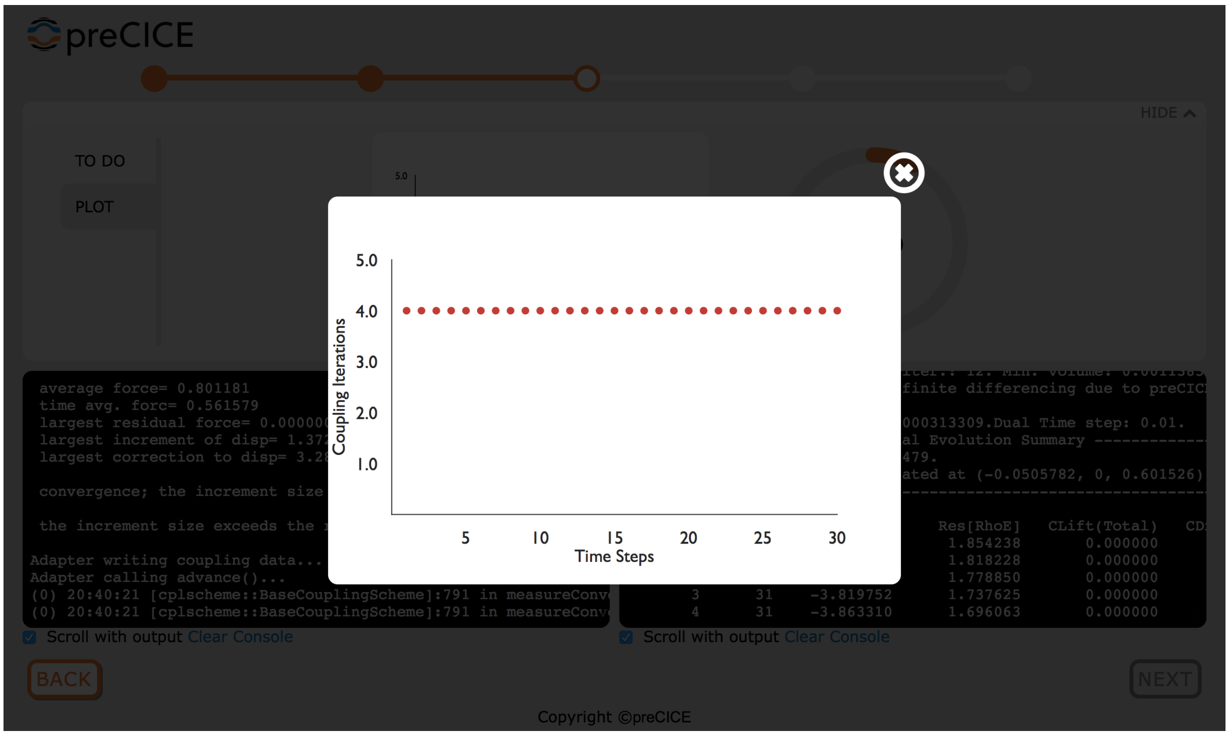
```
 average force= 1.456736
 time avg. forc= 0.890736
 largest residual force= 0.000000 in node 10 and dof 1
 largest increment of disp= 1.596426e-03
 largest correction to disp= 1.932784e-08

 convergence; the increment size is increased to 6.705000e+00

 the increment size exceeds the remainder of the step and is de
Adapter writing coupling data...
Adapter calling advance()...
(0) 20:40:29 [cplscheme::BaseCouplingScheme]:791 in measureConv
(0) 20:40:29 [cplscheme::BaseCouplingScheme]:791 in measureConv
```

Scroll with output Clear Console

```
(0) 20:40:29 [impl::SolverInterfaceImpl]:483 in advance: it 2 o
 Deforming the volume grid due to preCICE simulation.
Non-linear iter.: 1/1. Linear iter.: 12. Min. volume: 0.0011327
 Computing grid velocities by finite differencing due to preCIC

Min DT: 0.000104821.Max DT: 0.000313658.Dual Time step: 0.01.
----------------------- Residual Evolution Summary ------------
log10[Maximum residual]: -3.16508.
Maximum residual point 3, located at (0.00971323, 0, 1.00379).
---------------------------------------------------------------

 IntIter ExtIter     Res[Rho]      Res[RhoE]    CLift(Total)   CD
       0      55    -3.985626       1.533366       0.000000
       1      55    -4.019059       1.507230       0.000000
```

Scroll with output Clear Console

Figure 13.3: Updated layout of the page during simulation



```
 average force= 0.801181
 time avg. forc= 0.561579
 largest residual force= 0.000000
 largest increment of disp= 1.372
 largest correction to disp= 3.28

 convergence; the increment size

 the increment size exceeds the

Adapter writing coupling data...
Adapter calling advance()...
(0) 20:40:21 [cplscheme::BaseCouplingScheme]:791 in measureConv
(0) 20:40:21 [cplscheme::BaseCouplingScheme]:791 in measureConv
```

Scroll with output Clear Console

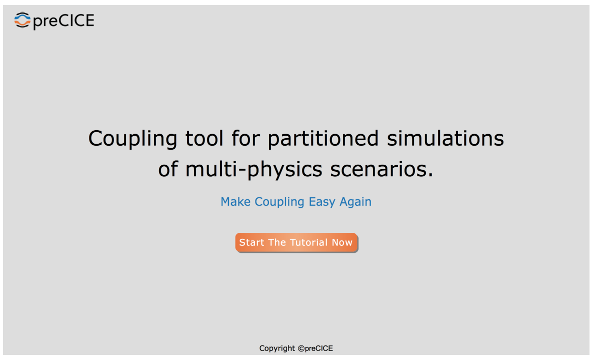
```
finite differencing due to preCIC

000313309.Dual Time step: 0.01.
al Evolution Summary ------------
479.
ated at (-0.0505782, 0, 0.601526)
---------------------------------

                               Res[RhoE]    CLift(Total)   CD
                                1.854238       0.000000
                                1.818228       0.000000
                                1.778850       0.000000
       3      31    -3.819752   1.737625       0.000000
       4      31    -3.863310   1.696063       0.000000
```

Scroll with output Clear Console

Figure 13.4: Ability to focus on plot

- **Redesigned landing and final pages**

  The landing page is an important component of the website since it is the first point of contact of potential preCICE users with the tutorial. Therefore, we tweaked the landing page and added a ”Voices of customers” section that can reassure users about the usability of the software. Similarly, we designed a final page that provides links to the github tutorial and the preCICE webpage as potential resources that can help the user learn more about preCICE. The updated pages are shown in Figures 13.5 and 13.6.

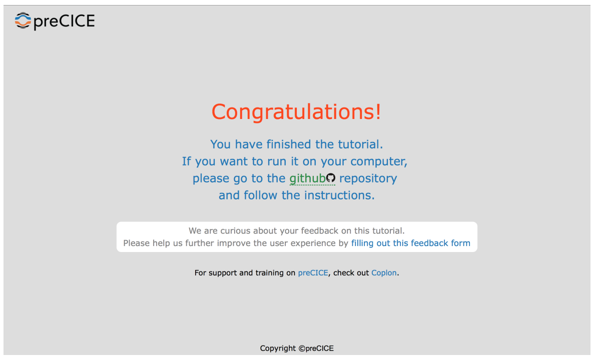


Coupling tool for partitioned simulations
of multi-physics scenarios.

Make Coupling Easy Again

Copyright ©preCICE

Figure 13.5: Landing page



Congratulations!

You have finished the tutorial.
If you want to run it on your computer,
please go to the github repository
and follow the instructions.

We are curious about your feedback on this tutorial.
Please help us further improve the user experience by filling out this feedback form

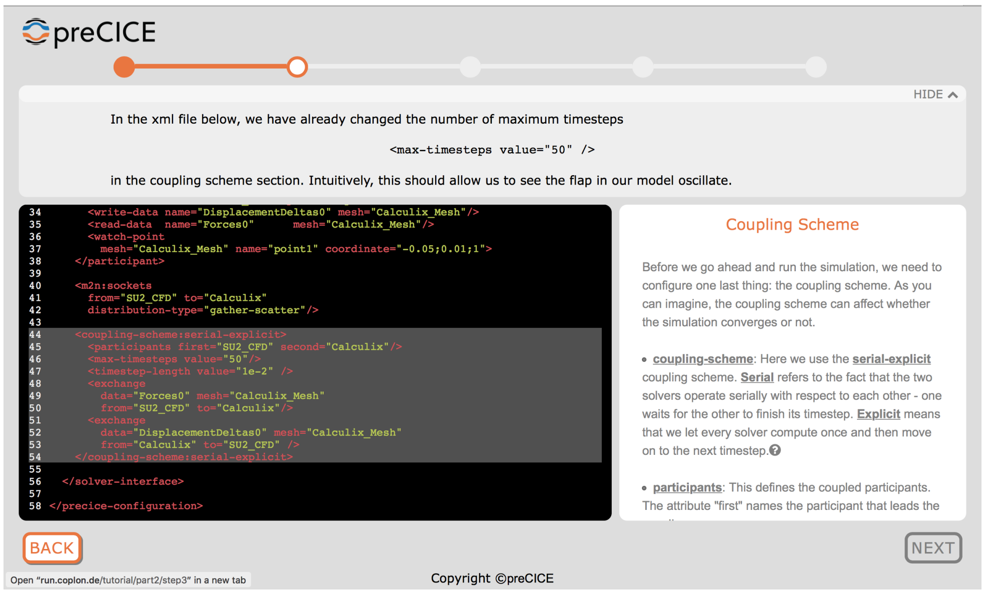
For support and training on preCICE, check out Coplon.

Copyright ©preCICE

Figure 13.6: Final page

- **XML explanations**

  A large part of the time a user spends on the website is devoted to reading and understanding parts of the preCICE configuration file. Therefore, considerable attention was devoted to adjusting the look and feel of the configuration file and the corresponding explanation text. The new user interface for this part is presented in Figure 13.7.



In the xml file below, we have already changed the number of maximum timesteps

`<max-timesteps value="50" />`

in the coupling scheme section. Intuitively, this should allow us to see the flap in our model oscillate.

```
      <write-data name="DisplacementDeltas0" mesh="Calculix_Mesh"/>
      <read-data  name="Forces0"      mesh="Calculix_Mesh"/>
      <watch-point
        mesh="Calculix_Mesh" name="point1" coordinate="-0.05;0.01;1">
    </participant>

    <m2n:sockets
      from="SU2_CFD" to="Calculix"
      distribution-type="gather-scatter"/>

    <coupling-scheme:serial-explicit>
      <participants first="SU2_CFD" second="Calculix"/>
      <max-timesteps value="50"/>
      <timestep-length value="1e-2" />
      <exchange
        data="Forces0" mesh="Calculix_Mesh"
        from="SU2_CFD" to="Calculix"/>
      <exchange
        data="DisplacementDeltas0" mesh="Calculix_Mesh"
        from="Calculix" to="SU2_CFD" />
    </coupling-scheme:serial-explicit>

  </solver-interface>

</precice-configuration>
```

Coupling Scheme

Before we go ahead and run the simulation, we need to configure one last thing: the coupling scheme. As you can imagine, the coupling scheme can affect whether the simulation converges or not.

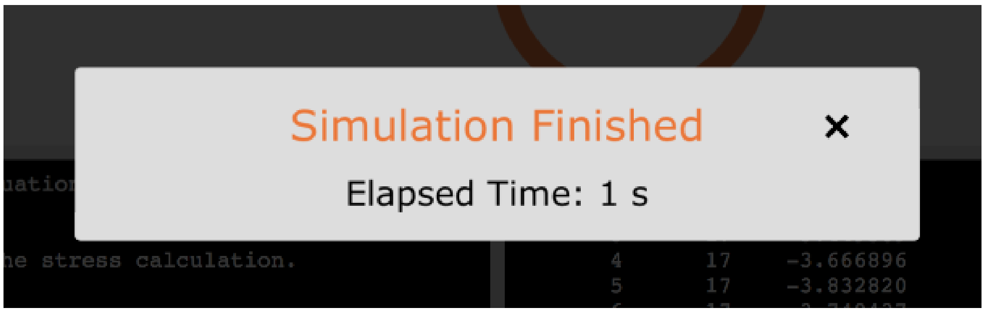
- **coupling-scheme**: Here we use the **serial-explicit** coupling scheme. **Serial** refers to the fact that the two solvers operate serially with respect to each other - one waits for the other to finish its timestep. **Explicit** means that we let every solver compute once and then move on to the next timestep.
- **participants**: This defines the coupled participants. The attribute "first" names the participant that leads the

Open “run.coplon.de/tutorial/part2/step3” in a new tab

Copyright ©preCICE

Figure 13.7: Updated page with the description of the xml-file

- **High score**


Simulation Finished

Elapsed Time: 1 s

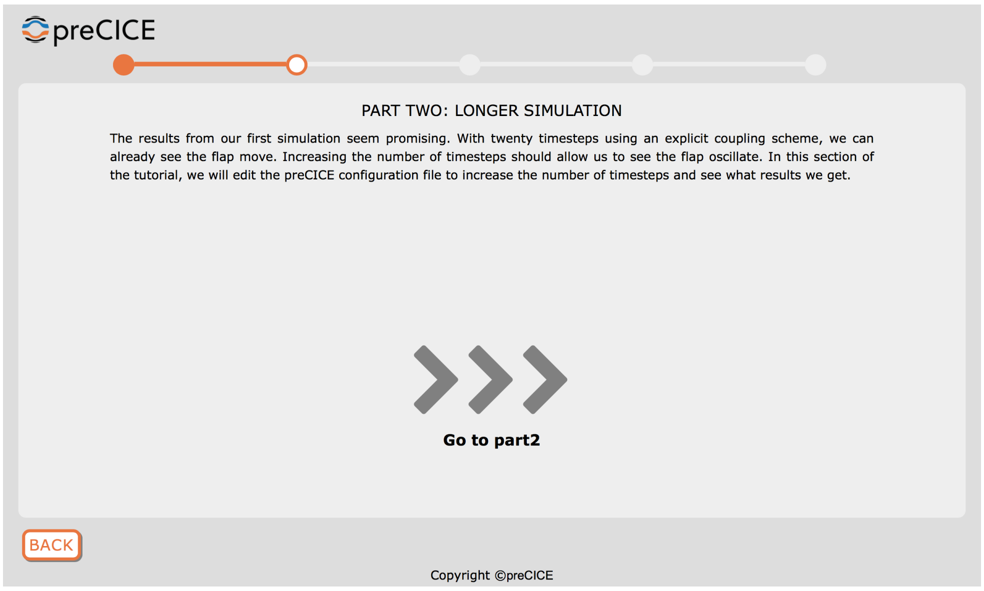
```
uation
he stress calculation.          4      17     -3.666896
                                5      17     -3.832820
```

Figure 13.8: Pop-up with elapsed upon finishing the simulation

In the new user tutorial, we have a fixed number of chapters, five, and do not allow user to explicitly change any parameters. Therefore, now there is no need to display a list of best running times with respect to some parameter. Instead, we just display the total simulation time for each part since they usually differ significantly. We assume that the user will be able to notice the difference and based on it, understand which coupling scheme is better. To further stress the difference, we point out the reduced simulation time in the text.

- **Movement restrictions**

During the second iteration, we greyed out the navigation buttons during the simulation. However, during user testing a common remark was that the user should be allowed to go back while the simulation is running. Since a significant number of users mentioned this in the feedback, users can now go back while the simulation is running. Now we also allow the user to skip the simulation completely and still see prerecorded results of the simulation in the form of a video. However, if the user has already started the simulation, we do not allow her to go to the visualization page in order to preserve the integrity of the consoles in different chapters of the tutorial. An illustration of this functionality is shown in Figure 13.3. Since the user story is now separated into five distinct chapters, we introduced introduction pages for each chapter. These include discussions of preCICE features that we demonstrate in each chapter and provide rationale for using these features. An example can be seen in Figure 13.9.



PART TWO: LONGER SIMULATION

The results from our first simulation seem promising. With twenty timesteps using an explicit coupling scheme, we can already see the flap move. Increasing the number of timesteps should allow us to see the flap oscillate. In this section of the tutorial, we will edit the preCICE configuration file to increase the number of timesteps and see what results we get.

Copyright ©preCICE

Figure 13.9: Intermediate page between different part of the story

13.2.2 Miscellaneous Improvements

In addition to the major visible changes described above, a lot of minor changes were also introduced. Although these fixes are not discussed here in detail, they are essential for a smooth user experience. These include things like adjusting fonts and spacing, carefully designed margins, making certain buttons more prominent, providing links to the preCICE and Coplon websites on the final page, and opening all links in a new tab by default.

13.3 Analytics

An interactive website cannot be considered complete without the facility to to respond to user behaviour by improving the design. To achieve this, we incorporated a basic version of Google Analytics into the website. With this basic implementation, we can get statistics about which pages a user visited, demographics, and how much time a user spent on a certain page. This basic implementation can be extended to also capture certain events triggered by the user. For example, we can log every time a user runs a simulation and see whether the users are actually running the simulations. Improving the analytics capability can be a particularly fruitful direction for further development of the website.

14 User Testing

One the biggest issues in developing a user interface is that it is hard to predict a priori what users expect from a particular website. Sometimes even the users themselves do not know what they actually want before using the interface. A trial-and-fix strategy is thus the only way to improve the website. Of course, the trial step need not be random but can be guided by the vast amount of knowledge that web developers have gathered over the last three decades.

Even though we had tested the website internally, we recognized that it was very important to conduct real world trials. Often times developers in a team can acquire a particular view of the website that is not shared by ordinary users. This is a common bias among people who spend a great deal of time developing a product. Fortunately, we were provided the chance to perform user testing at the University of Stuttgart with students attending a course on numerical simulation. This was particularly fruitful since these students belong to a part of the audience this tutorial is targeting.

We decided to acquire user feedback over two dimensions: quantitative and qualitative. For quantitative feedback, we designed a questionnaire with Google Forms. This allowed us to get hard numbers on what users thought of particular features of the website. Since we designed this questionnaire ourselves, we recognized that it would be subject to the same biases we mentioned above. Therefore, besides the questionnaire, we also decided to interview the users after they had finished the tutorial. This qualitative feedback was quite helpful since the users mentioned certain problems and provided suggestions which were not reflected in the quantitative feedback. Below, we summarize the feedback we received from the users. Note that we also categorize open ended questions in the questionnaire as qualitative feedback.

14.1 Quantitative Feedback

To date, 41 participants have filled out the questionnaire. Table 14.1 and 14.2 present the questions, results and an analysis of the responses. The results seem to be promising. After finishing the tutorial, most of the users said they understood the test case and were curious to learn more about preCICE.

Table 14.1: Questions and Result from the questionnaire

General Questions		
How many years of experience do you have with CFD?	1 year : 35.7% 0.5 years : 21.4%	More than half of the users had no more than a year of experience in CFD. Most of the users were students who might have first gotten in touch with CFD in the lecture.
How many years of experience do you have with computational numerics?	3 years : 35.7% 4 years : 28.6%	Even though the majority of the users were beginners in CFD, most of them already had experience in computational numerics.
From 0 to 3, how familiar are you with Fluid-Structure-Interaction (FSI)?	1: 64.3%	Most of the users were not very familiar with FSI.
Which of the following numerics software packages do you know?	SU2: 14.3% Calculix: 14.3%	Most of the users did not know the solvers we are using for the simulations.
Simulation Scenarios		
Is the main difference between the simulations clear?	yes: 100%	All the users understood the main difference between the simulations
Do you think you would be able to adapt the preCICE configuration to a similar setup?	yes: 92.9%	Most of the user were confident they can adapt the preCICE configuration to a similar setup.
Do you understand the physical setup of the simulated case?	yes: 100%	In light of this response, the explanation provided on the website is adequate.
Was the given information about preCICE features sufficient?	yes: 87.5%	Most of the users thought the information was sufficient. We will talk more about this in Chapter 15.
User Interface		
From 0 to 3, how did you like the design?	1: 20% 2: 26.7% 3: 53.3%	Most users like the design of the website.
From 0 to 3, how did you like the structure of the website?	1: 20% 2: 26.7% 3: 53.3%	Most users like the structure of the website.
From 0 to 3, how well did you understand what to do?	1: 6.7% 2: 50% 3: 42.9%	Most of the users understood what to do which implies that the user interface is easy to follow.
From 0 to 3, how much text did you skip?	0: 14.3% 1: 64% 2: 21.4% 3: 0%	Here 0 represents no text skipped. As you can see most users only skipped a small amount of the text.

Table 14.2: General Impression of questionnaire takers

General Impression		
From 0 to 3, how curious are you now to find out more about preCICE?	1: 20% 2: 57.1% 3: 21.4%	The result is promising. 78.5% of users are curious about preCICE.
From 0 to 3, how confident do you feel that you can run a preCICE simulation?	0: 7.1% 1: 50% 2: 21.4% 3: 21.4%	Most of the users were not confident to run a preCICE simulation by themselves. However, this might be due to the fact that most of them did not have much experience in FSI.
From 0 to 3, how much would you like to use preCICE for a university project?	0: 0% 1: 35.7% 2: 35.7% 3: 28.6%	Most users showed interest in using preCICE for a university project. This is encouraging.

14.2 Qualitative Feedback

After the user testing session in Stuttgart, we interviewed most of the students and talked to them about the website personally. We received several helpful suggestions for the website and we decided to implement some of them. However, we also decided to disregard some suggestions since incorporating them would have led to a bloated website. Below, we provide a sampling of these suggestions.

Things we have implemented:

- “I would like to go back to the config file explanation while running the test case.”
- “Third simulation does not run well on my Linux machine.”
  This problem was fixed before the final presentation.
- “I would like to get more information on Aitken underrelaxation.”
  To satisfy this request, we added a link to a paper that introduces Aitken underrelaxation.

Things we decided not to implement:

- “I am not sure if I need to change my own simulation code or configuring preCICE through the xml file is enough.”
  We included a link to the wiki which discusses the setup in detail.
- “If I didn’t know what slip conditions are, I would not understand the test case. Maybe you should explain them.”
  We assume that users interested in preCICE would possess the theoretical background required to understand the test case. Explaining slip conditions would require us to go into the mathematical theory of boundary conditions which would clutter up the website.

15 Conclusion

This is where our journey ends. We reflect on the objectives we accomplished and discuss possible improvements and extensions for the website.

15.1 Summary

The main goal of our project was to provide a tool that would provide a simple introduction to preCICE software to potential users. For this purpose, we developed an interactive online tutorial. To give a gentle introduction and provide a brief overview of the coupling library, we prepared a series of simple multiphysics fluid-structure interaction (FSI) simulations. The simulations are organized in the form of a user story that demonstrates various ways to set up a coupled simulation with preCICE. Each configuration brings its own advantages - such as increased stability or decreased runtime.

The user story was implemented in the form of an online tutorial that takes users through all steps of a simulation, from an introduction of the test case to visualization of the final results. The tutorial is highly interactive and requires active participation on each stage. User experience was one of the most significant points for our project. Therefore, we actively collected feedback from users and implemented the suggested improvements during the final stages of the project.

During the project we employed an iterative development approach for both the website and the tutorial content. In this way, the overall quality of the project was gradually improved over ten months. For the final milestone, we provided a user story with five stable and fast FSI simulations, and delivered a website that exhibits consistent behavior across various operating systems and browsers.

15.2 Outlook to Future Work

Our project creates a solid basis for possible future development. One possible direction is the extension of the tutorial. It seems appealing to provide an opportunity to choose among several solvers. This option gives users the freedom to work with their preferred solvers and, thus, further simplifies introduction to preCICE from the user's perspective. Taking this idea to the limit, it might be possible to implement a ”sandbox mode” that allows the user to run arbitrary coupled simulations from the browser. Another possible field of work is refinement of the web analytics that we provided. Fine-grained analytics often suggest further ways for potential improvement of the user experience.

Bibliography

[1] Wordpress.com: Create a free website or blog. https://en.wordpress.com. Accessed: 12.07.2017.

[2] React - a javascript library for building user interfaces. https://facebook.github.io/react/. Accessed: 12.07.2017.

[3] Read me Â· redux. http://redux.js.org. Accessed: 12.07.2017.

[4] Hans-Joachim Bungartz, Florian Lindner, Bernhard Gatzhammer, Miriam Mehl, Klaudius Scheufele, Alexander Shukaev, and Benjamin Uekermann. precice – a fully parallel library for multi-physics surface coupling. *Computers and Fluids*, 141:250—-258, 2016.

[5] Calculix: A three-dimensional structural finite element program. http://www.calculix.de/. Accessed: 12.07.2017.

[6] Su2, the open-source cfd code. http://su2.stanford.edu/. Accessed: 12.07.2017.

[7] Alexander Rusch. Extending su2 to fluid-structure interaction via precice. Bachelor's thesis, Munich School of Engineering, April 2016.

[8] Lucia Cheung Yau. Conjugate heat transfer with the multiphysics coupling library precice. Master's thesis, Faculty of Informatics, TUM, April 2016.

[9] Su2 turorial - inviscid bump in a channel. https://github.com/su2code/SU2/tree/master/TestCases/euler/channel. Accessed: 30.10.2017.

[10] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 2009.

[11] L. Lapidus and G.F. Pinder. *Numerical solution of partial differential equations in science and engineering*. John Wiley and Sons, 1982.

[12] gnuplot. http://gnuplot.info/. Accessed: 28.11.2017.

[13] Robert C. Tuck Bruce M. Irons. A version of the aitken accelerator for computer iteration. *International Journal for Numerical Methods in Engineering*, 1:275—-277, 1969.

[14] Tmux - terminal multiplexer. https://github.com/tmux/tmux. Accessed: 28.11.2017.