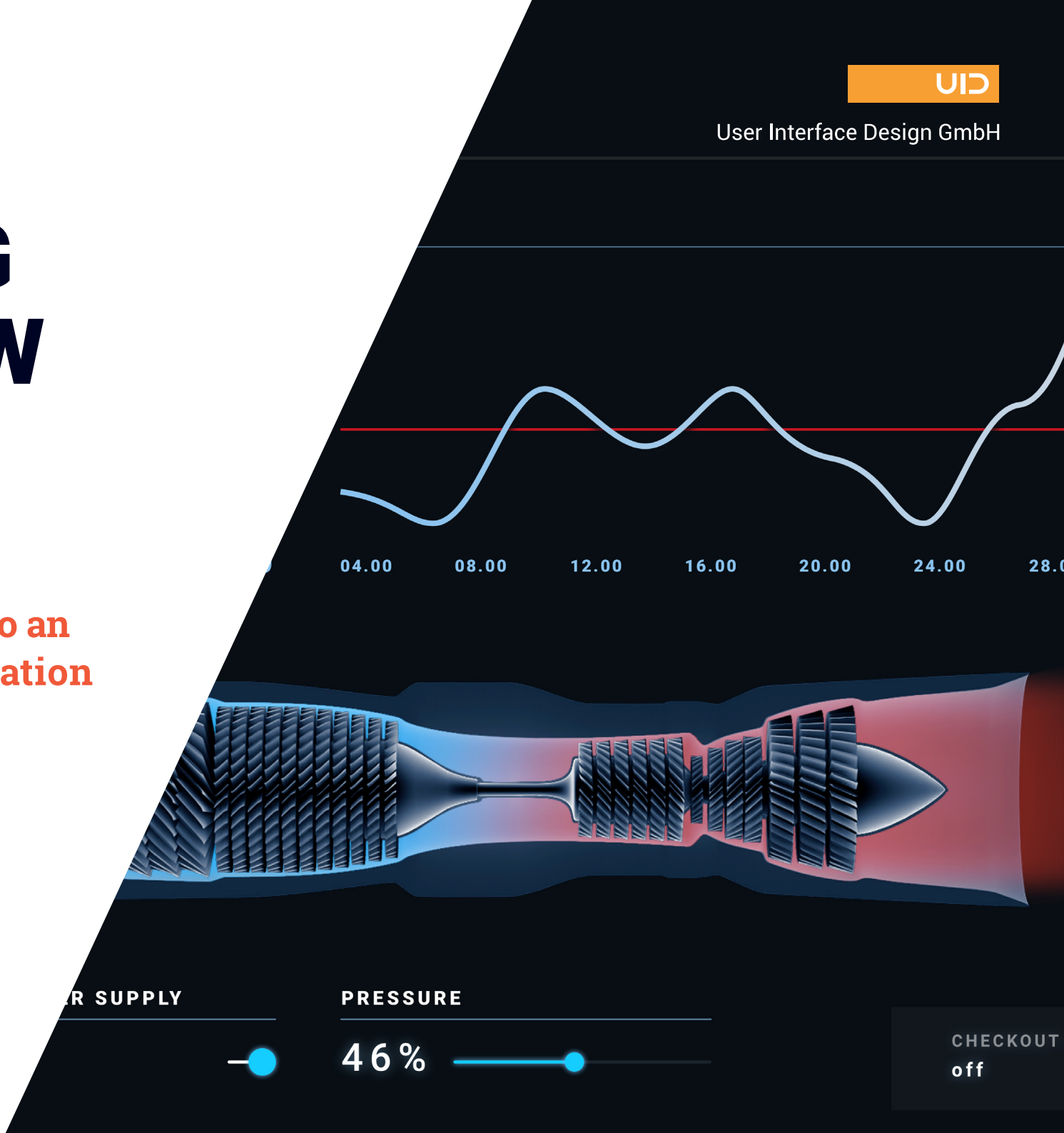
HEADING TO A NEW WORLD

How to migrate your desktop application to an HTML5-based application (example .NET)



1 INTRODUCTION

Self-explanatory, aesthetic and cross-device inspiring – these are standards which an HMI today will be measured against. With web technologies, you create the technological basis to meet these challenges. Because web technologies have a particular strength: they enable immediate responsiveness and platform independence. No wonder that more and more automation solutions like ctrlX AUTOMATION from Bosch Rexroth consequently rely on web technologies.

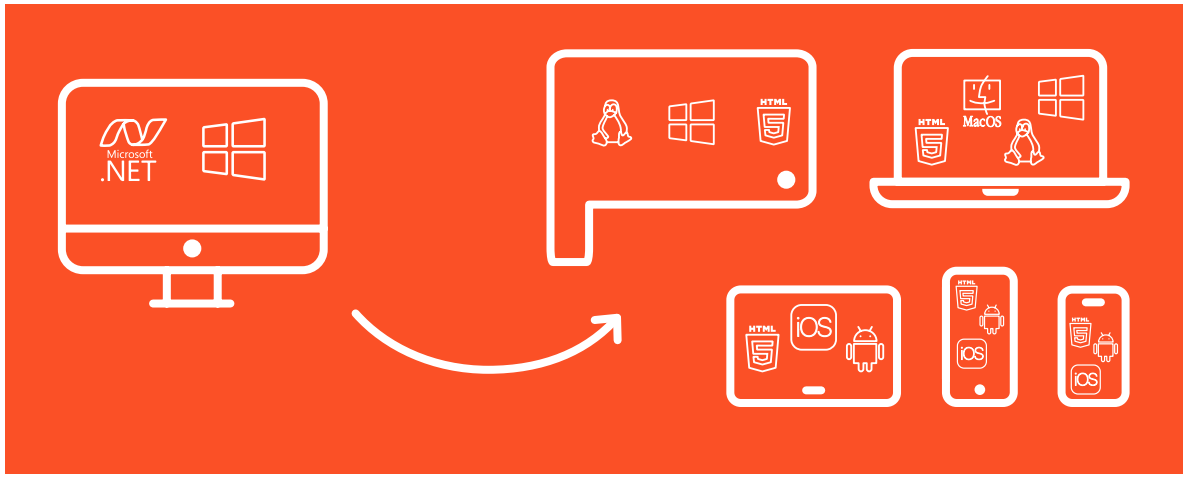
You have an existing application in .NET, Qt or some other technology, however, want to take advantage of the web world? Our whitepaper shows you how to migrate an established application to HTML5. In the following example, we assume .NET/WPF as an example. The demonstrations can, however, be transferred to any other technology.

We provide you with four modules you can use to work out a strategy for resuing as many components as possible and therefore migrate your application as quickly and efficiently:

1. Basics: Lay out a clear objective as the basis for a successful migration.
2. Analysis: Determine the requirements for the migration of design and layout, UI controls and user experience as well as application code.
3. Good practices: There are already best practices to migrate software from one technology to another. Make use of these to benefit from them.
4. Build know-how: Make your team fit for the new technology.

2 MIGRATION BASICS

Before you start migrating, make sure to answer the following questions: What are your motives for the migration? What is the origin, and what is the target technology? How extensive should the migration be? Which special requirements regarding functionality, interaction or structure do you need to consider?



Web technologies enable immediate platform independence.

2.1 MOTIVATION: WHY MIGRATE?

In mechanical and plant engineering there is one clear trend: away from the individual machines towards comprehensive machines and cross-device ecosystems made of locally distributed and connected hardware, software and value-added services. In the past, applications were mostly designed for a specific single standalone system. Today, customers expect them to be available on different systems – for example on permanently installed displays on a machine or desktop workstations – as well as support for mobile devices.

Unfortunately, this does not work with every technology just like that. A classic .NET application, for example, does not reach all potential users: There are operating systems that do not support .NET applications. In addition, such applications aren't responsive and therefore do not work for all display resolutions. By moving your application from .NET to HTML5, you can reach a larger group of users.

Web technologies come with solutions and tools for even the most complex responsive designs. In addition, they can be displayed on any operating system.

Web HMIs that run in the browser or in the web view of an application that has not been developed with HTML5 can easily be used on different operating systems and devices – e.g., Android smartphone, Apple tablet, Windows Industrial PC or a Linux-based control center. Responsive web HMIs adapt to any screen resolution and offer an optimal user experience on all devices.

This platform independence and responsiveness but are not the only advantages of a migration: They also offer you the opportunity to tackle long-planned changes and to implement – either to add new features, optimize processes or update UX and design.

2.2 THE ORIGINAL TECHNOLOGY

To identify which components need to be migrated, and what factors affect migration, clarify how your current application is implemented: Is the current application, for example based on a client-server structure? If so, does it use a thin or fat client? Or is it a standalone solution without a central data source? The better data is separated from the UI, the easier it is to migrate the UI. If data and UI are closely linked, you should consider separating them first or during migration at the latest.

Determine if your UI is developed with a high-level UI language (.NET / WPF, Qt / QML, ...) or with UI toolkits (INOSOFT, VisiWin, COPA-DATA zenon, etc.). Migration can only be successful with enough knowledge of the original technology.

2.3 THE TARGET TECHNOLOGY

To migrate your application, you can take different approaches. One of them leads directly through the basic web technologies HTML5, JavaScript and CSS.

HTML5 (Hypertext Markup Language) is a markup language that describes which elements (UI controls, text, images) in a user interface are used. CSS determines the styling and positioning of these elements, as well as their responsive behavior. This separation of markup and styling enables an efficient development. The script language JavaScript changes a static site to make it interactive.

In addition to manual, direct programming in HTML5 / CSS / Javascript, it is recommended to use application frameworks such as Angular, React or vueJS. These enable easy access to web technologies and make it easier for developers to create complex web applications. They come with tools and solutions specifically for the development of applications.

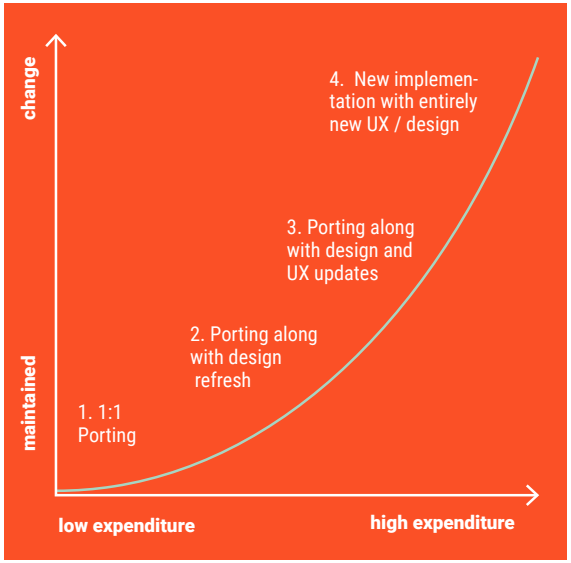
Also, UI frameworks like Google Material, Bootstrap or Web IQ for example can make it easier especially to migrate design and layout (see chapter 4.3 and 4.4). With WebIQ, for example, you can use the integrated WYSIWYG layout editor without CSS knowledge using drag & drop web HMIs.

The basic web technologies HTML5, JavaScript and CSS

2.4 REVIEW: WHAT NEEDS TO BE MIGRATED?

During the review process, the existing application comes into focus: Get a comprehensive idea of functionality and structure. Only in this way can the migration be planned. The following questions should be kept in mind:

- Purpose: For which use cases will the application be used for, or what kind of problems is it supposed to solve?
- Functionality: Which functionality does it include?
- Structure: How is the application classified and structured? What does the architecture look like? Are there any hidden functions, such as certain service or administration menus, that are accessible via keyboard shortcuts?
- Interaction: How do users interact with the application (touch, mouse, keyboard and / or hard key operation)?
- Modes / states: Are there modes or states for which certain functions are not available? Possible examples are start or maintenance processes or error states: For example, if the front door of a machine is open, certain functions are not available for safety reasons.
- Role system: Do users have different permissions? Do users have different types of functions and / or user-centered views?
- System and hardware access: Does the application use functions of the operating system, for example to send e-mails, to print documents or to access files or databases? Does it access sensors and actuators of the hardware directly?



The four possible levels of migration

2.5 CONCRETE: DETERMINE THE SCOPE OF THE MIGRATION

Compared to a new development, you won't have to start at the beginning when migrating. You already know how the overall result will look like. This makes project planning and product design easier. You don't have to develop the whole application down to the last detail, but instead you can look at the old application for guidance.

Additionally, you can add new functionalities or improve certain areas for which you already received user feedback. Functions or screens, however, like settings, you can most often migrate unchanged.

One can basically differentiate between 4 levels of migration, where 1 would take the least amount of effort and 4 the most:

1. You migrate the existing application 1:1 as it currently is. The user won't notice a change, simply the technology in the background has changed.
2. The existing interaction logic (for example classification and screen structure) remains unchanged on migration, however, visual design can be modernized or adjusted to new corporate design specifications.
3. You have received ideas or user feedback that would improve the user experience of your application. As a result, you add new functionalities, new screens and pop-ups or restructure existing screens.
4. You don't migrate existing functionality, design or structure, but instead redesign the application anew based on existing requirements.

2.6 DETERMINE TEAMS FOR MIGRATION

Depending on which level of migration you decided on, your development team needs to consist of different people:

If you want to improve user experience, you should bring UX experts into the team. If your need to optimize the visual design, team members with appropriate design skills are required. If there are functions that cannot or aren't allowed to run in the client, you need a backend developer.

In any case, the team should definitely have a HTML5 and a .NET developer on board. Ideally, you have front-end developer familiar with both technologies who can extract algorithms from the old code as well and style and interactions.

3 ANALYSIS: DEFINE MIGRATION

To define the scope of the migration you need to analyze which elements you can build on you need to keep in mind the following questions and topics:

3.1 DESIGN

To achieve a uniform, cross-application appearance, you should have a good look at existing design style guides. Design requirements can come from different sources:

- Are there corporate design guidelines you need to adhere to?
- Can you build on existing HTML UI toolkits developed in other projects?
- Are there any requirements from marketing (for example through the company websites)? If you are using a UI toolkit such as WebIQ: Which elements can be used "out of the box"?
- Which ones do you need to add? What do you need to adjust visually?

3.2 UI CONTROLS

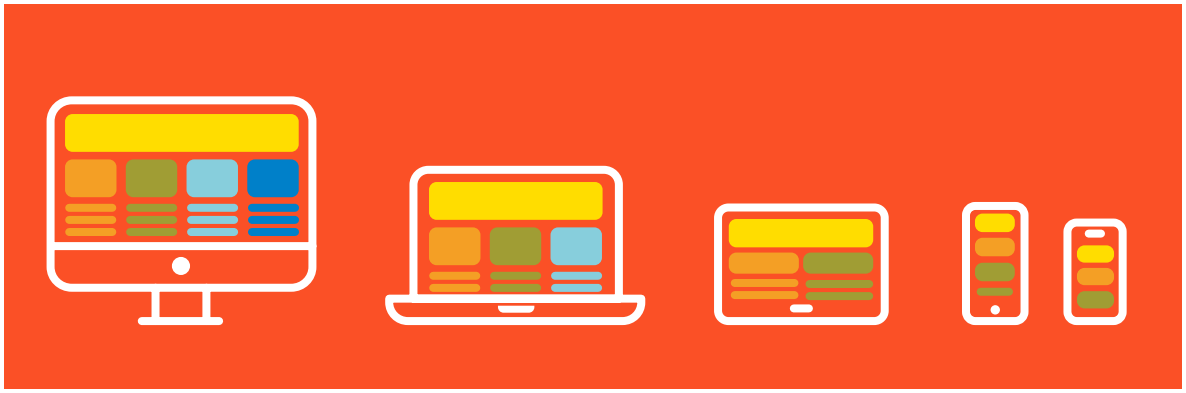
Document which UI controls you need:

- Which one can you use unchanged from the existing application?
- Which UI controls do you want to change because you noticed need for improvement?
- Which ones are new to integrate new functionalities, for example?
- Which ones can be disposed of?
- If you use UI frameworks, you can use predefined widgets?
- Which ones can you use "out of the box"?
- Which ones would need to be adjusted to meet your individual requirements?

3.3 LAYOUTS

Similar to the UI controls and the design, you should identify which layouts need to be adopted, newly developed or adjusted:

- Which layouts (screens/templates) are there in the current application?
- Which layouts (screens/templates) must be adopted?
- Which layouts do you need to restructure to enable responsive design or improve interaction, for example?
- Which new features are required to integrate new functionalities or optimize vulnerabilities during migration?



Responsive HMIs adapt to different screen sizes

3.4 RESPONSIVE DESIGN

Identify whether your company already implements responsiveness in other applications, for example, and if so, what rules it follows:

- Are breakpoints (usually at certain screen resolutions) already defined, for example, where the layout breaks and content realigns?
- Are there different layouts for different resolutions? Do typography and images adapt to the size of the screen?
- Are there rules according to which criteria information is prioritized on smaller screens?
- How is text overflow handled on small screen sizes? Are they cut off or replaced by shortened text alternatives?

If there are no corresponding rules for these or other questions about responsiveness, you should define them before migration and lay the foundation for a uniform responsive design of your applications.

3.5 APPLICATION CODE

In the old application, identify the code components that you can or should adopt. Which code components will you implement in the client, which one in the server?

In particular, the code components in which you have invested a lot of development time, company know-how and special knowledge, such as recipe editors, should or must be "encapsulated" as a block on the server.

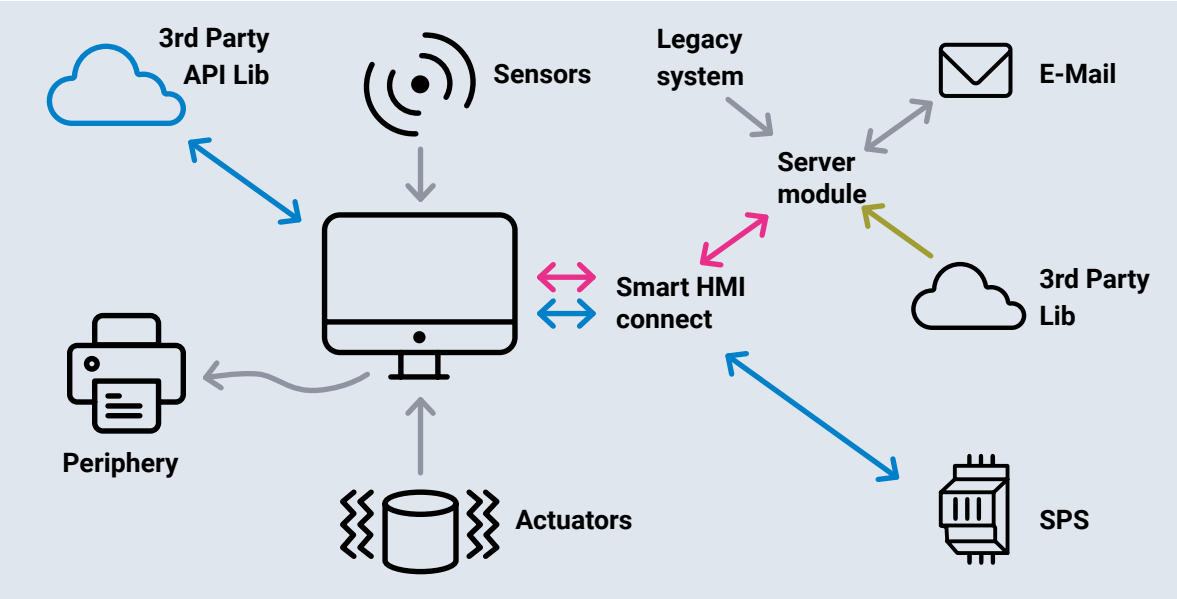
You can import these server components with minor adjustments to the interfaces – from the .NET code and connect them to the front-end via your visualization solution (see chapter 4.5).

If you use WebIQ, for example, this is easily possible via Connect API. But also front-end components such as algorithms (for example for plausibility queries or for the validation of text field entries) can also be migrated from the old code. You cannot use .NET code 1: 1, but must adapt the syntax to JavaScript and HTML5. However, this is easier and faster than completely re-designing and implementing the logic.

3.6 INTERFACES

Analyze your existing interfaces and identify where adjustments are required or where problems may arise: Which backend connections are currently available?

- Which will remain?
- Which solutions are replaced by solutions such as CtrlX AUTOMATION or WebIQ?
- Are third-party systems such as external databases connected?
- How is the web server configured? For example, certain security-related configurations can prevent functions from running properly.
- Do you want to connect hardware such as hand scanners, cameras, actuators and/or sensors? Do you want to access the operating system via interfaces to integrate functions such as printing or sending mail? These hardware or OS-related functions cannot be implemented easily via the web client. They must be implemented on the server side instead(see chapter 4.5).



Analysis of the interfaces

3.7 OPTIMIZE PERFORMANCE

To ensure the optimal execution of your application, you should check the hardware of your web server: is it sufficient for optimal performance? To avoid performance problems regarding the hardware, you should first clarify which of the planned functions are better implemented on the server or client side: Functions such as the calculation of charts and images can be solved more efficiently in the client. You implement resource-critical processes such as complex calculations on the server so that, for example, animations or interactions in the UI are executed smoothly.

3.8 DEFINE PROCEDURES AND IDENTIFY PROBLEMS

Identify and classify which components are easy to migrate or can possibly even migrate automatically. Where does the migration become more complex or even requires special workarounds to be successful. This allows you to define how you want to approach it for the individual areas of your application and and specify a sequence.

3.9 DEFINE USER ACCEPTANCE

In addition to the technical challenges, you should not lose sight of your users: how do they react to announced changes? What emotions do these evoke? What do they expect from the update? You need to incorporate this initial situation of the users into your decisions about what you change in the context of the migration. With a good release strategy (see Chapter 4.2) as well as open and transparent communication regarding the procedure, the planned changes and the associated benefits, you can secure a high level of acceptance of your users in advance.

Engaging customers at an early stage can increase acceptance.

4 GOOD PRACTICES: EXPERIENCES FROM SUCCESSFUL MIGRATIONS

To migrate software from one technology to another, there are already best practices you can draw from. We introduce you to some established good practices with which the migration will be successful.

4.1 TECHNICAL CUT

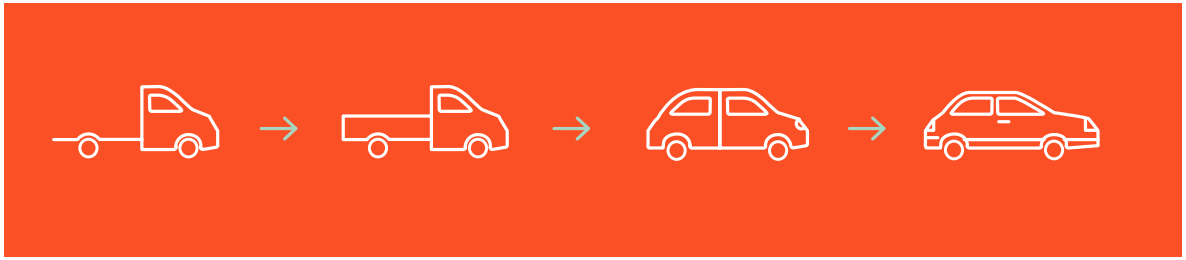
We recommend that you start with a technical proof of concept. By implementing a manageable scenario of your application, you evaluate whether your migration strategy is feasible. This allows you to identify possible stumbling blocks and pitfalls early and cost-effectively, and can make adjustments to your strategy if necessary.

4.2 RELEASE-PLANUNG: BIG BANG OR STEP BY STEP

In addition to the actual planning of the migration, you should decide how to introduce the innovation for the user. There are basically two possible methods for the release:

1. In the sense of a big bang, you can first finalize the migration and make the complete new application accessible to the user in one go.
2. You migrate individual areas, views and functions of the application step by step and make them gradually accessible to users. Users can thus get used to the new look and functions. There is even the option that individual new and old areas co-exist for a certain period of time.

None of the two variants can be recommended as the "right" one. Rather, both approaches have their advantages and disadvantages, which have to be weighed up individually for the respective project: are users used to regular updates? Does the existing application have weaknesses that make it difficult for users to work and that could be remedied in a timely manner through step-by-step updates? Or is there currently no need for action? Is there a risk of competitors gaining early insight into their innovations, copying them, and thus losing a competitive advantage through the gradual introduction? These are just a few questions that play a role in the decision for one or the other process.



The Minimum Viable Product (MVP) already meets the customer's needs in basic terms.

4.3 PORTING THE DESIGN

Visualization solutions such as WebIQ from Smart HMI facilitate migrating the design, the layouts, but also the application logic (see 4.5). You can save the manual migration of the design to CSS. Rather, your HMI design can be quickly and elegantly transferred via the WYSIWYG editor. Whether font, color, size or behavior – with the comprehensive parameterization dialogs from WebIQ you can adapt the look & feel of the prefabricated widgets to your design. Only if your desired adjustments are not possible via the parameterization dialogs of your framework, or you want to create individual widget libraries, you have to convert them manually into CSS. The individually tailored libraries, but also other UI libraries such as Google Material, can be integrated into your visualization solution as CSS.

To ensure a consistent look & feel, there are different ways to prepare the migration – depending on which existing documentation you can access:

1. The simplest and most effective way leads over an existing style guide, in which font, color world, layout etc. are already specified. Based on this specification, you can transfer the design.
2. If there is no style guide, a designer can derive the design of the existing application. Based on this new specification, you migrate your HMI design. At first glance, this seems an expensive and expensive way. However, if you migrate or redevelop further applications, you can fall back on this style guide in the future.
3. It is cheaper – but also more prone to errors – if you do without design specifications. You have the styling information read out of the existing application and migrated directly – without going through the process of creating a specification. You can more rapidly progress in the concrete project this way. The disadvantage: If design elements are implemented inconsistently in the old application, these inconsistencies are taken over into the migrated application. There is also no design specification available for future developments and migrations.

   You have to read out the design information from the existing application again. In cases 2 and 3, the reading of the design information such as distances or color values can be done either manually by examining the old design or the old code. Or tools such as Live Visual Tree from Visual Studio 2019 or Snoop automatically export them from the old WPF or XAML code.
4. If you want to use the migration as an opportunity to optimize your application, the creation of the specification can effectively and cost-efficiently be linked to a design improvement. The designer creates a new specification by following the design requirements of the old application, but optimizing it at the same time.

4.4 MIGRATING OF LAYOUTS

Similar to the design, you can also manually transfer layouts from the old application or the new concept using the WYSIWYG editor of visualization solutions such as WebIQ and implement any planned optimizations.
To do this, it is important to identify which of the layouts need to be migrated, modified or recreated (see chapter 3.3). In order for the user experience to remain identical on different devices, a responsive website layout is important (see chapter 3.4).

4.5 PORTING THE APPLICATION LOGIC

When porting the application logic, it is primarily important to identify which components you implement in the client and which ones you implement on the server.

4.5.1 Server-side implementations

You should implement complex business logic as well as hard - and operating system – related logic on the server side.

Complex business logic

JavaScript is always visible for end users and thus for competitors. However, application logic usually contains mission-critical or security-relevant components, such as a recipe generator or other complex algorithms. A lot of know-how and development time has been invested in the development of these modules, in some cases they even make up the USP of a company. To ensure that competitors cannot read this business logic from JavaScript, they should implement it with their web solution on the server and thus secure it in a “black box“ (capsule away).

Operating system-related logic

The same applies to functions that access the operating system or hardware, such as sending an e-mail, printing documents, or controlling actuators, sensors, or hardware. These processes are either very resource-intensive and heavily burden the performance of the client. Or they cannot be solved on the client side, but always require communication with the server.

If you use UI frameworks such as Web IQ, you can integrate a .NET code module isolated from the old code into the HTML5 UI via the WEBIQ Connect API.

4.5.2 Implementation in the client

Simple application logic such as input validations as well as navigation logic in the client are implemented in JavaScript/HTML5. Reprogramming is not complicated, as you can import such algorithms and regular expressions directly from the old .NET code.

5 BUILD KNOW-HOW

To migrate the existing application to a web technology and then maintain it, you need the appropriate skills within the company.

5.1 UPSKILL AND EXPAND THE DEVELOPMENT TEAM

If these skills are not yet available in the company, you can make your development team fit for the new technology on-the-job training, self-study, seminars, webinars and conferences. To make the transition easier, it is advisable to bring an experienced HTML5 developer into the team – permanently or temporarily as a coach. You can also easily outsource individual tasks to external experts.

5.2 BUILDING WEB COMPETENCE

For the migration, you need competences in each of the three web technologies HTML5, CSS and JavaScript. In HTML5 and CSS, basic knowledge is often sufficient to work with HTML templates, change font and color in CSS, or to collaborate with external service providers.

These basics can usually be learned quickly and easily. However, for more complicated tasks such as implementing sophisticated designs, layouts and animations in CSS, it often takes years of experience. But don't worry: this CSS part, but also the programming of challenging HTML5 components or templates can be outsourced easily so that you can hand over this work to external experts. In addition, visualization solutions such as WebIQ with their WYSIWIG editors facilitate the migration of the HMI design.

It is more important to build JavaScript competence in your team. Due to the complexity and lack of intuitiveness, pure JavaScript is difficult for .NET developers to learn. With TypeScript, however, a modern alternative is available. A compiler converts the TypeScript code into JavaScript. Since the grammar is similar to that of .NET, TypeScript makes it easier for .NET developers to switch. TypeScript brings even more advantages: it enables consistency in code, high productivity, maintainability and modularity, as well as early detection of errors by using types for variables.

5.3 MAKING IT EASIER TO GET STARTED WITH SUITABLE FRAMEWORKS

.NET builds on the so-called Model View View-Model (MVVM), which separates the display and logic from the user interface. .NET developers are used to think and work in these structures. To make .NET developers feel more at home in web technologies, you can use application frameworks such as Angular. Angular supports working according to the MVVM pattern, but does not require it. The .NET-like syntax makes it easier to switch to web technologies.
At the same time, Angular is compatible with the automation solution CtrlX AUTOMATION or the visualization solution WebIQ.

6 CONCLUSION

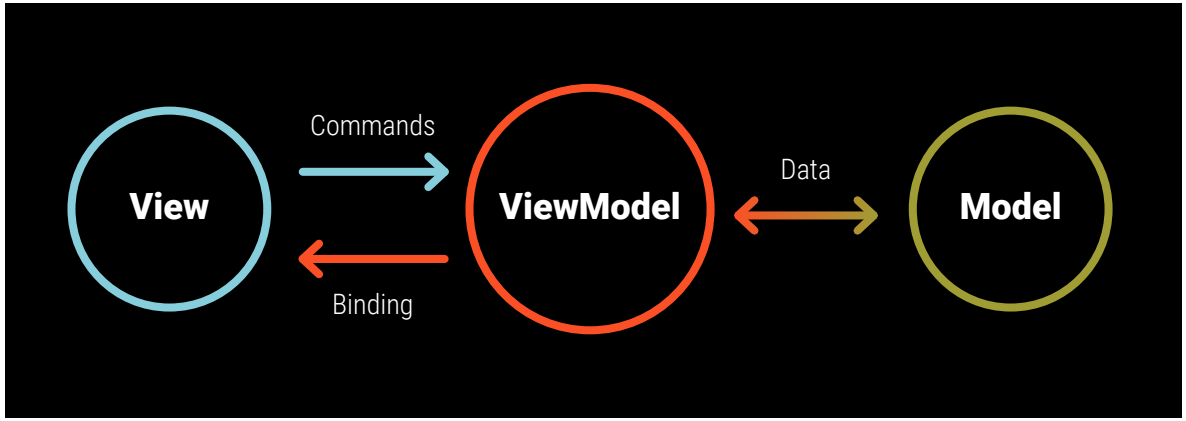
HTML5 has established itself as the technology of the future. When migrating your system to the web world of HTML5, anything is possible: with the right strategy, there are no features and designs that cannot be migrated to HTML5 from .NET, Qt or any other technology or framework.

With the right strategy, the migration is elegant and efficient. You benefit from the responsiveness and platform independence that enables you to move to HTML5.

This allows you to address new user groups without excluding your existing ones.
At the same time, porting offers you the opportunity to tackle long – planned changes, implement new features and improve the user experience-in order to enable your customers to work more satisfactorily, more efficiently and more easily.

YOU WANT TO LEARN MORE?

If you have any questions or comments, please feel free to contact us at any time:
info@uid.com | www.uid.com



Model View

User Interface Design GmbH

Ludwigsburg
Berlin
Dortmund
Mannheim
Munich

THE AUTHORS

DANIEL KURZ
Lead Software Engineer

DR. ALEXANDER SEILER
Director Sales and
Business Development

HANS-GERD SODERMANNS
Director Business Support

YOU WANT TO LEARN MORE?

If you have any questions or comments,
please feel free to contact us at any time:
info@uid.com
www.uid.com

HELLO, TECHNOLOGY