Whitepaper

How We Build Web Applications

Buildable

Table of Contents

About Buildable

Our nimble, versatile team of full-stack engineers, creatives, and developers come to work every day with the knowledge we're making the stuff that will change our clients' work lives for the better. It's why we attract and keep some of the best technical talent in the industry, and it's how we deliver the highest quality work on every project.

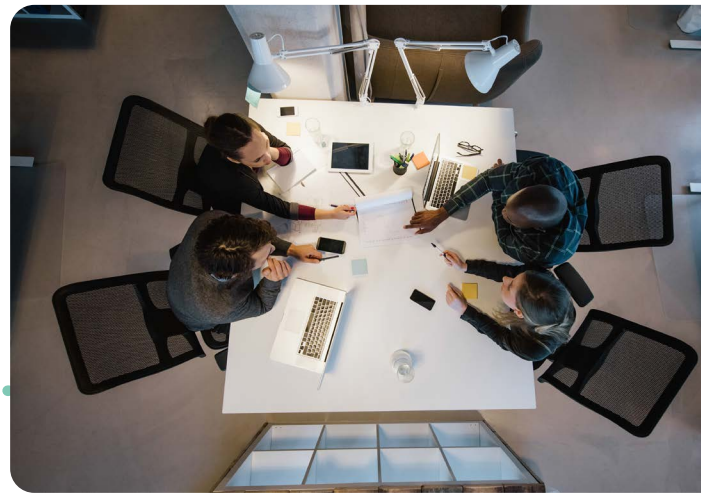
Collaboration: Our Agile mindset ensures communication, transparency, and buy-in at every stage of a project.

Pragmatism: We build things that solve problems now and work for years into the future.

Humanity: Our people are important because our clients' people — and their experiences with our technology — are important.

Spark: We help clients imagine a better way to get their jobs done. And then we build it.

Introduction



In this whitepaper, we describe what a software web application does, how it is built, and why we build the way that we do. Throughout this book, it will become clear to you that Buildable software engineering is cutting-edge. We are masters of design and development, and we'll take the time to get to know your business as well as we know ours.

Software is built as a series of programs that are stored in computer memory. A software web application, then, uses the computer system to perform special functions beyond the basic operation of the computer itself. It is designed to perform a group of coordinated functions, tasks, or activities for the benefit of the user. We have built applications to assist in diagnosing medical conditions, quoting purchases, and testing water quality.

Distinct from other software is the web application, which is written in HTML, CSS, JavaScript, and other web-native technologies. These applications typically require an internet connection, although today most offline versions that can selectively synchronize with a web server.

Single-page applications (SPAs) are web applications that fit on a single web page and provide a user experience like that of a desktop application. All HTML, CSS, JavaScript, and other necessary code files are retrieved with a single page load. Oftentimes, appropriate resources are dynamically loaded and added to the page as necessary in response to user interaction. These interactive functions require dynamic communication with the web server.

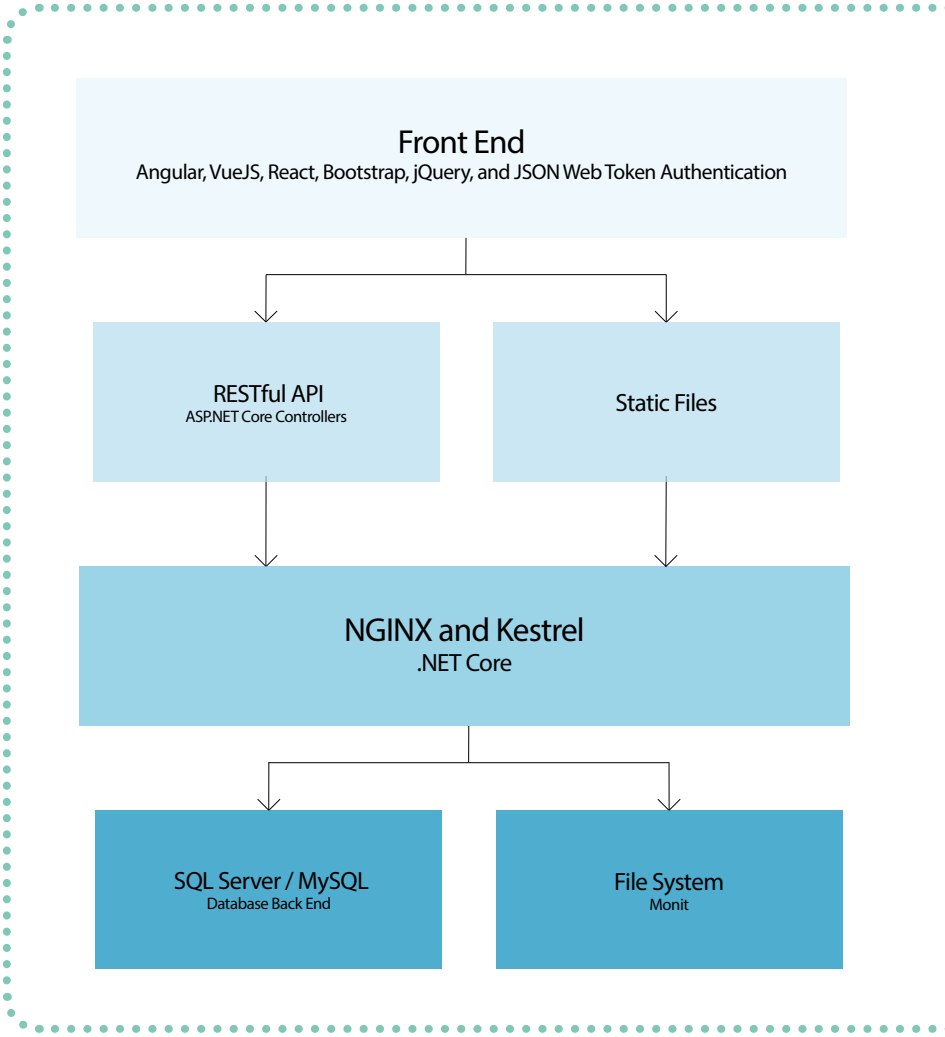
Throughout this whitepaper, we describe our process of web application development. We note key elements of this process in this way so that potential clients can become more familiar with the technical jargon we will use to describe features of our applications.

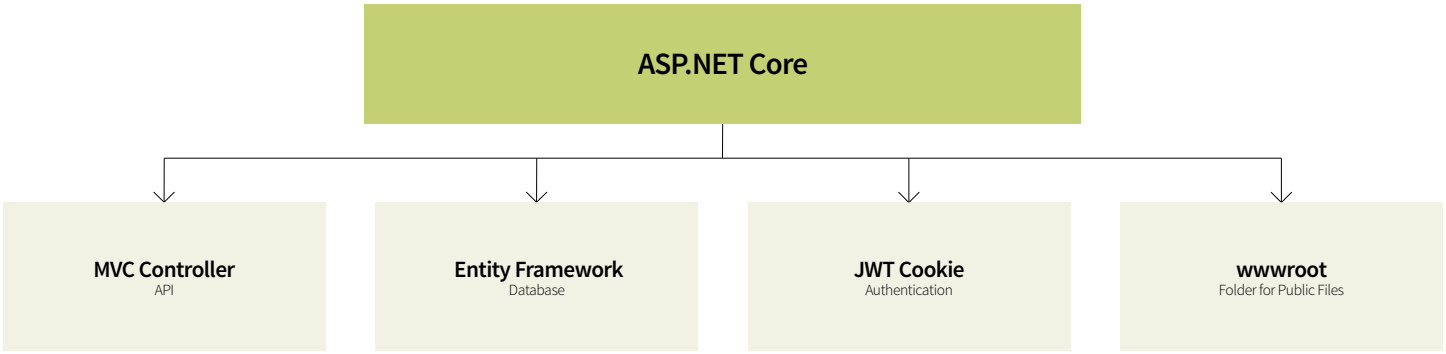
We look forward to discussing ideas with you.

Application Architecture

Before getting into detail, it may be helpful to have an idea of the structure of a Buildable web application.

The frontend of the application is built with Angular, Vue, or React–written in TypeScript, HTML, and CSS–and delivered to the browser as a single JavaScript file. Bootstrap provides mobile responsiveness and a friendly user interface. jQuery is used wherever the application requires functionality beyond what Angular can offer, for example, with time fields and input controls. The frontend communicates exclusively with the backend through a RESTful API layer and stores a JSON Web Token cookie for authentication.





The RESTful API is entirely written in C# with Visual Studio as an ASP.NET Core single-file web application. The ASP.NET Core application provides an embedded web server, Kestrel, which can be hosted through IIS on Windows servers or through NGINX on Linux machines. The database access is entirely abstracted using Entity Framework, which maps database models to database tables and provides a zero-SQL approach to building a web application.

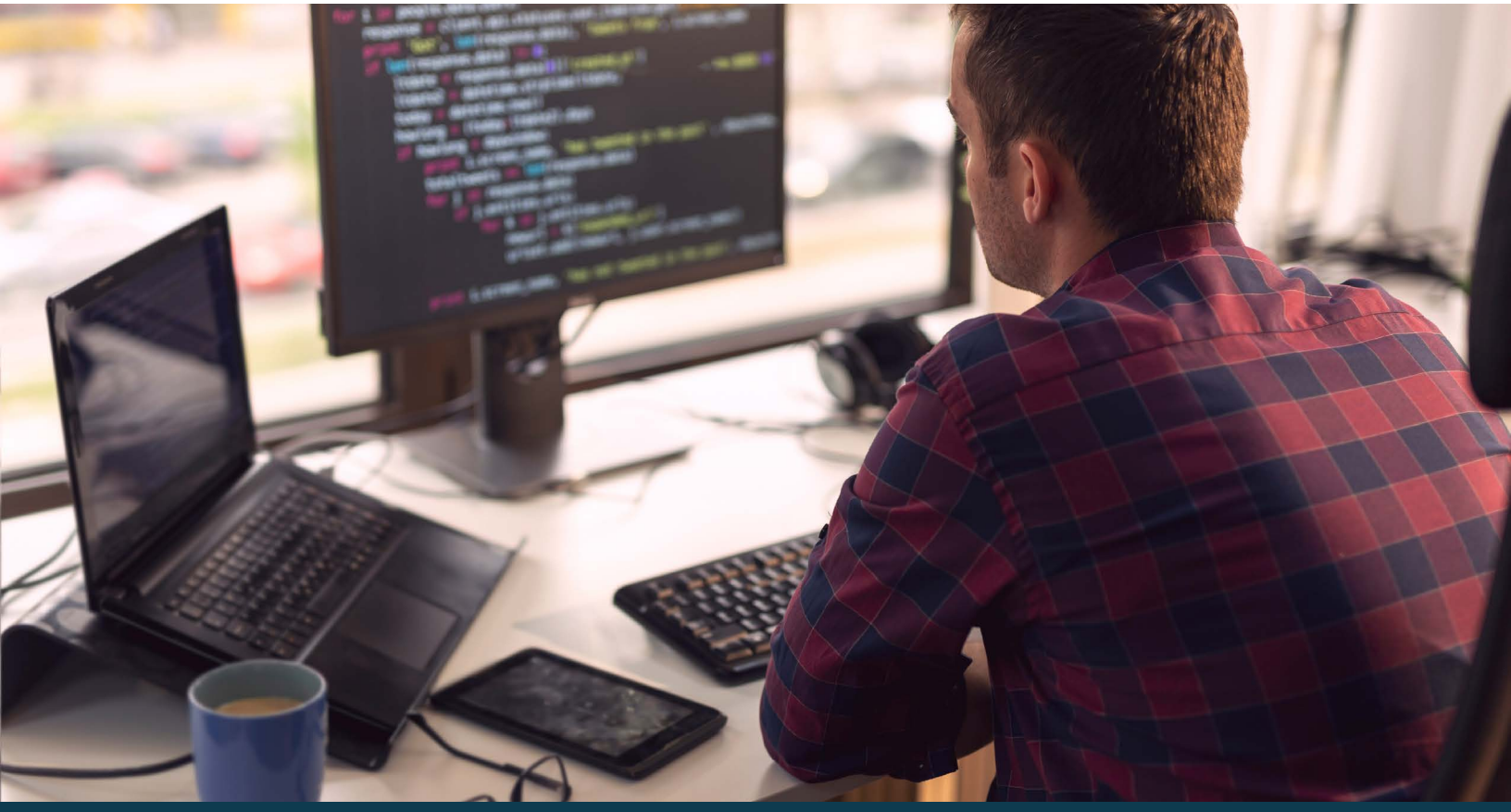
Application Architecture Overview

The web server technology is two-fold. The .NET Core web application runs with Kestrel, which is a highly efficient, embedded web server. Kestrel invokes the JSON Web Token (JWT) handlers and API controllers. NGINX is the actual web server frontend, which provides the initial connection-handling, Geo-IP filtering, and SSL handshake before requests are sent to Kestrel. NGINX provides SSL (Secure Sockets Layer, for authenticated information exchange) and access to other static files, such as mobile tablet hashes. It acts as a proxy, forwarding requests to Kestrel.

The backend is connected to any database. Our preference is either Microsoft SQL Server or MySQL. All strings in the database are encoded in Unicode (UTF-8). SQL Server or MySQL can be hosted locally (for development) or on a secondary server (for production) on Linux, Windows, or in the cloud, and we create twice-daily backups.

Static files required by the single-file application are all located under the web application wwwroot folder and are made of third-party files (such as CSS for styling) or images required by the application. Some static files, such as mobile tablet hashes, are stored elsewhere and are accessible directly through NGINX. Static, frequently accessed files are generally better served by NGINX, avoiding the request, and proxying overhead of accessing Kestrel.

Lastly, the web application is hardened with Fail2ban, which operates by monitoring log files for potential intrusion and banning potential intruders for 24 hours by adding rules to the firewall. In addition, the application is monitored with Monit, a software configured to send email alerts when logs contain a pattern, or the machine performance breaks a specific threshold.



Our Process

The Buildable Process for software development is sophisticated, modern, and clean. Here's how we do it.

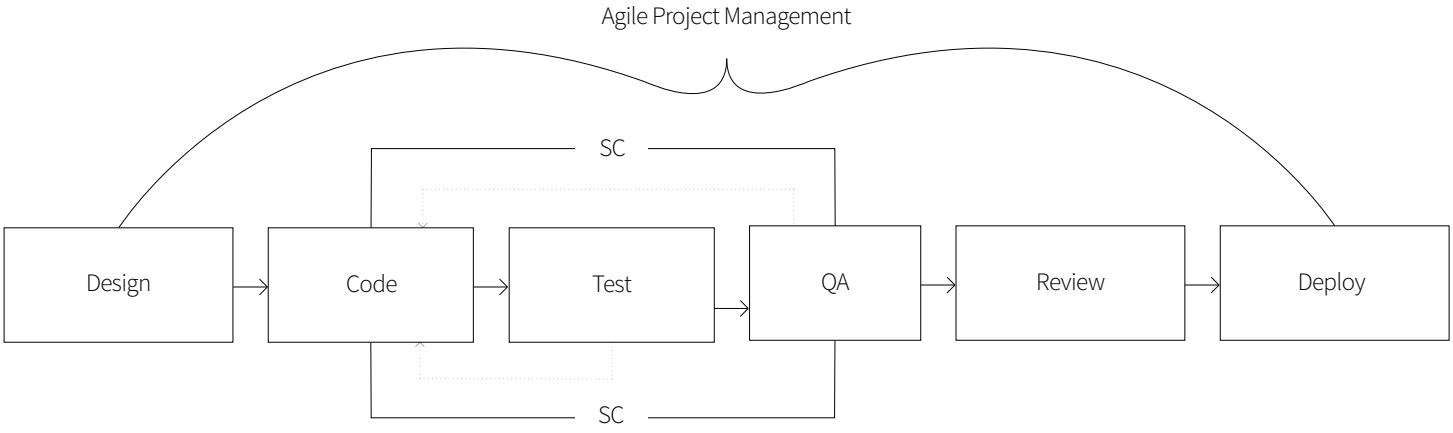
Application Architecture Guidelines

To understand the Buildable software engineering process, it is important that we present a quick overview of our application architecture guidelines, which have been designed to provide a solid foundation for any software we build.

Every project is built through multiple Sprints and each Sprint begins in the design phase and ends in a deployment. Upon approval of the design, the Buildable software engineers create a functional edition of the design. Throughout the trilogy of coding, testing, and assuring quality, source code (SC above) is carefully controlled with Git so all changes can be tracked and audited. Then the project is brought to the client for Sprint review. When the client is fully satisfied with the Sprint build, it is deployed. Depending on the size of the project, the final Sprints will include end-to-end testing called UAT and after the result of the which the product will be deployed live.

Developers build applications by following a pattern that makes sense for each individual project.

We apply strict design patterns to provide code separation, code reuse, and bug mitigation. Maintaining these principles results in intelligent, manageable code. Design patterns are methods of organizing an application's source code into layers, or folders. This process is called code separation. There are many variations of design patterns, and software engineers fit patterns to each individual project as they build.



Application Architecture Guidelines (cont.)

Code reuse is a critical advantage that developers gain when utilizing design patterns. Once a set of functions has been written for one project, due to the process of code separation, those same functions can be reused in a future project.

Interaction amongst the layers creates the system, something higher-level than the working parts would be in and of themselves.

Each layer of an application's codebase, the entirety of an application's source code, is assigned a specific kind of task which should not require any of the other layers. Rather, a task accomplished in one layer should ultimately offer something particularly useful to the other layers and, consequently, the system.

Code separation, in addition to enhancing flexibility and efficiency, also serves the purpose of bug mitigation. This is the isolation of errors in layers of code, rather than having to debug an application by working through its entire codebase.

Code is organized into layers per different design patterns. Teams work together more efficiently with these practices and can complete projects much more quickly as they are able to reuse powerful code. Following is a selection of the layers, design patterns, frameworks, and other principles implemented by the Buildable software team.

The Model-View-Controller (MVC) and Model-View-View Model (MVVM) are the two design patterns for building user interfaces most frequently followed by the Buildable software engineers. These require the use of behavior mechanisms (events), reactive mechanisms (observables), use of tasks referenced from the Task Parallel Library (TPL), and models.

Models hold data (information), but don't perform services that manipulate it. The View is the way data is displayed. The Controller reacts to the user's interactions with the View.

An application programming interface (API) is a set of principles and tools for building application software. APIs help standardize the way developers build applications. This way, software is consistent with the user's operating environment.

APIs specify how software components should fit together. By following these principles, software is more consistent across platforms.

The Data Access Layer (DAL) contains classes, drivers, and models that primarily provide access to the database system. This separates low-level data-accessing APIs or operations from high-level business behaviors, which are kept in the BOL.

The DAL provides a solid base for Entity Framework and LINQ queries, streamlining data access.

It is expected that the BOL classes interact with outside systems, such as Amazon Web Services, databases, email servers, and file systems.

The Business Object Logic (BOL) layer is built exclusively to accomplish repetitive tasks related to the business the application will serve. Examples of BOL are EmailBOL, ConfigBOL, and AmazonS3BOL.

The BOL typically contains helpers. These are static classes with static functions, or behaviors. BOL helpers consume database models, execute queries with LINQ, update database records, send messages to log files, send emails, process payments, and more.

Error handling is performed in all layers, using "try/catch" statements to trap errors. Each is logged with information about the error, including the data, time, thread ID, class, and method. All errors are centralized to a single file and graded from Debug, Info, Warning, Error, to Fatal. Error logs are monitored on the production servers, which the Buildable software engineers are automatically alerted to when errors and fatal errors occur.

The Agile Process

The Buildable team utilizes CI/CD Software (GitLab). GitLab is built for Agile project management, an iterative approach to managing software projects centered on sprints and client feedback. An Agile approach is best for building most highly complex software web applications and encouraging client communication.

Sprints are fixed-length iterations of work that guide and summarize a team's operations. Tasks are assigned to individual team members, who can then request feedback from and send updates back to the team. Priority is assigned to each task, as well as a status within the workflow of the project. Workflow generally follows in succession from To Do, In Progress, In Review, to Done status. Having an Agile workflow means that the team can stay focused and adapt quickly to changing guidelines and deadlines.

Every morning, the Buildable team has a standup, a 15-minute meeting optimized for efficiency. This allows for the software team to synchronize their efforts without losing development time. Each team member speaks to their progress on projects from the workday before and their goals for the day at hand. The project manager also oversees the scrum board, which visualizes the entire workflow of a sprint. Individual tasks are filed under columns by their status in the workflow. As team members accomplish tasks, they are moved through the workflow. Through this process, the project manager can easily monitor progress and forego problems before they arise.

Assigning team members to work on different layers of code is critical for adhering to project deadlines.

Powerful applications are built by the exchange of knowledge and sharing of workload among software teams.

A project manager's responsibilities, among other things, are to ensure that teams are working most efficiently and creating the best possible product for the client.

The Agile Process (cont.)

Source control, also understood as version or revision control, is a practice of software configuration management whereby changes to code are tracked, drafted, and reviewed before being applied to the final edition of a project. Software engineers often utilize source control systems, such as Git, to track and coordinate file changes using a repository.

Buildable hosts a private, local GitLab repository for source control. The Buildable tool chain, development environment, and publishing scripts integrate with source control. Access to the central Git repository is controlled through specific per-developer credentials. The Git repository is backed up on a daily schedule, as well as a per-project basis.

Perhaps most essential to Buildable source control is our granularity rule: All change sets committed to Git must be atomic in nature (i.e. centered solely around one feature request or one bug fix) and submitted with a descriptive commit message (which may or may not contain an associated Gitlab ticket for project management purposes). Version numbers are committed to source control as well, further enhancing traceability of our code.

In addition, Buildable software engineers follow an incremental development cycle. Changes are worked on locally, committed locally, pushed to the central repository, deployed on a test server, and tested again before other tasks can be addressed.

Source control transparency is implemented for the development team. The Git central repository sends automated and detailed email each time changes are submitted (i.e. committed and pushed). The entire development team receives these emails.

Only the software development team has access to the central Git repository.

Each team member works on localized versions of the code, leaving the final edition untouched until changes are approved.

Before changes to a final edition are official, each is developed locally and submitted for approval. Code is tracked this way throughout development.

Data put in source control includes the following components...

- Source code
- Database scripts
- JSON data
- XML data
- Binary images

All other files in source control are otherwise required to build and deploy a project.

Developers use JSON as an efficient and dependable manner of encoding, transferring, and embedding information into a database.

Anatomy of a Web Application

A web application consists of several interconnected parts that depend on each other for performing complex functions and displaying vast amounts of information. An application depends critically on the backend for functional programming and on the frontend for appealing, usable form.

In addition, user confidentiality should be a primary concern for all businesses. Best practices for authentication and security are crucial to producing a successful application.

User Experience Design

For Buildable, user experience (UX) design is more than just a concept or simply wireframing. It is a dependable framework for exquisite design and engineering–and a satisfied, productive user.

This UX state of mind is especially active throughout the frontend development stage of Buildable software development. This practice further ensures that we are building products of excellence. At the forefront of our design and engineering is the enhancement of users' experience.

User experience (UX) design adds much to the quality of an application. The role of UX is often overlooked, but that isn't necessarily negative. In fact, it can be considered a compliment if someone does not directly notice a design. Design is often successful by fitting right in with the "flow" of an interaction. It is easier to find examples of bad design because great design is invisible, so to speak. In some excellent cases, though, UX design is so successful that the user is delighted and tells all their friends about the application's relevance, ease-of-use, and aesthetic appeal. Word-of-mouth advertising is the most effective campaign strategy. So, it is justified on many levels to pay due attention to the experience of your product.

Buildable UX work begins by reviewing the client's business model, thoroughly understanding the purpose of the application, and planning for the greatest impact on the goals of the user. The beauty of UX is that it form-fits as necessary to each project. The influence of UX in Buildable web applications can be traced through a timeless, six-step process: discovery, wireframing, design, review, development, and deployment.

The primary goal of UX design is to provide a satisfying product experience to the user.

UX designers craft application experiences to fit the industry it will serve.

User Experience Design (cont.)

During the discovery stage, the Buildable team meets with the client, discussing fundamental goals and expectations, as well as core usability requirements. These requirements are the barebones of a software web application: what it must ultimately accomplish to be successful. In addition, the designer will inspect the client's brand materials. If the client has written copy, such as that in brochures or a website, the designer will also review these to find a voice for the design. Sometimes the voice isn't used solely for the copy of an application. The voice can, for example, be expressed in the way its information is organized. An application used in a medical office will be structured in a way that makes sense to, say, a surgeon. An application used by a department store will be organized differently, in a voice that will be helpful for salespeople or their consumers.

Once the usability requirements, brand materials, and voice have been set, the designer can get to work on structuring the site in the wireframing stage. The wireframe will express how the application's information and functions will appeal to the user (though it won't yet have aesthetic appeal, which is decided in the design stage). Wireframing is dedicated to ensuring that the user interface (UI), the way software is structured for interaction, of the application makes sense.

Equipped with the wireframe, the designer proceeds to the design stage. Generally, clients have an existing logo or color scheme for the designer to work off. If not, Buildable designers are happy and qualified to create them. While a company likely wants to stand out from their competition, they also want to make sense to their user. For example, users expect environmental organizations or financial businesses to use green in their branding.

Choosing a color scheme based on its audience's expectations should feel less like strict boundaries and more like helpful guidelines, useful in the case that there isn't a specific purpose for including some color. Therefore, color schemes–the main color(s) and accent color(s) used to identify a company–must be carefully chosen, as they play an important role in the overall success of an application. Whether we acknowledge it daily or not, colors have hierarchical effects on us in every moment of our lives.

The Buildable software team meets with the client to discuss application requirements. Designers can start brainstorming look and feel throughout discussion.

UX designers strategize before designing, following the timeless adage: Form follows function.

UX designers ensure that an application's UI is conducive to the user's workflow.

There are a wide variety of color tools, such as digital color pickers, to help designers choose colors that work well together.

User Experience Design (cont.)

Choosing specific colors to convey meaning in a web application is one of the clearest ways for the designer to communicate a message to and instill emotions in the user. For example, a warning message should be red, orange, or yellow, which conveys a message of "danger" to the user, prompting them to be careful when making their next decision. More serious messages should be red, while less serious alerts can be yellow. The designer recognizes the common connotations that people have with color and uses

Informing users of the status of processes makes a huge impact on the application experience. For example, confirmation of a successful process lets the user know that there is nothing to worry about.

these to make it easy for new users to adapt to a product. When a color scheme makes sense and conveys deeper meaning to the user, a business and its web application(s) are even more successful because the application goes beyond satisfying usability requirements.

From the UX designer's perspective, the most important consideration of all is the user's experience, including how positively the user views their experience with the product after its purpose has been served. An application must be designed to be easy to use for beginners yet efficient for experienced users, and leave all users feeling successful and in control. The best outcome is when the application, no matter its goal, is engaging and delightful for the user.

User Experience Design (cont.)

Good design allows for a product's affordances–what the product can do and how these actions are performed–to be easily discovered by the user. Signifiers are used by designers to flag affordances. One of the most common signifiers in web applications is the button. For example, buttons can be used to differentiate important links from common links or reveal a hidden menu bar to the user. A particular kind of button is the call to action (CTA) button. These call for extra attention from the user. The goal of a CTA button stems from the web application's overarching goal. For example, for an ecommerce application, a CTA button will draw the user toward purchasing actions. For a medical diagnosis application, the user will be drawn toward completing the form.

Developers work closely with designers to ensure that the structure and interface of an application match the design approved by the client.

Users like to be in control. When an application is consistent, users know what to expect and can learn how to use the product most efficiently. Fonts, dimensions, icons, buttons, colors, and all other aspects of a design must have consistent appearance and intent throughout the application. People like patterns because they minimize the cognitive effort required to use a product. When some piece of a consistent design goes rogue, the user will notice. If it is not, like a CTA button, designed to distract from the rest of the design, that piece of design will disrupt the user's flow, resulting in a negative experience associated with the application.

A design can be perfect to a designer, but nothing matters more than the client's opinion. During the review stage, the designer has received approval from the project manager to send the design to the client. At this point, the client has full license to request new features or changes, and the designer responds accordingly. It is critical to a project's deadline that the design does not get passed to production before review. Consider a situation where a client decides that they do not want a certain feature in their design, after the production may have spent hours implementing it. Only once the design has received the go-ahead from the client does the design get handed off to the development team, for the sake of efficiency.

Only once the client approves of the design does it get passed to production.

The development team starts the production stage by fitting the application's IA into the wireframe constructed by the designer. Once that is settled, the team references the design to lay styling over the structure. Communication amongst team members is imperative at this stage, as the rendered project must echo the design. Therefore, Buildable makes persistent use of Gitlab project management and Git source control described earlier in this whitepaper.

A project resulting from the Buildable process is a fully functional web application, designed and developed with the user in mind.

Build and Deployment

An application can be roughly divided into two parts: the frontend and the backend.The frontend is the aspect of the application that the user sees, created with the HTML, CSS, and JavaScript languages and their frameworks (such as Bootstrap and jQuery). The backend deals in executing functions that manipulate data from the database. Users can access data because backend developers use tools like REST API, C#, Entity Framework, and LINQ.

Clients are only able to communicate with servers when they are connected to the Internet.

Whether or not a design is responsive, adaptable to screens on different devices, is one task of a frontend developer.

The construction of a web application can be roughly divided into two parts. The frontend deals with the client-side of a client-server relationship in a computer network, whereas the backend deals with the server-side. The client is run on a user's computer or mobile device. When an application is run, the code is downloaded, run, and displayed by the browser. Server-side code, on the other hand, is run on the server, then its results are downloaded and displayed in the browser.

The frontend is the aspect of the application that the user sees. Frontend developers build an application to display information in a structured context that makes sense to the user. Applications are viewed in some sort of browser, whether it be on a desktop or mobile device. Frontend developers must ensure that applications are functional and maintain appealing form in the face of differing browser interpretations of code which is called cross-browser testing. Mobile applications are generally less information-dense than desktop browsers because vast amounts of information or detail don't scale well to small screens.

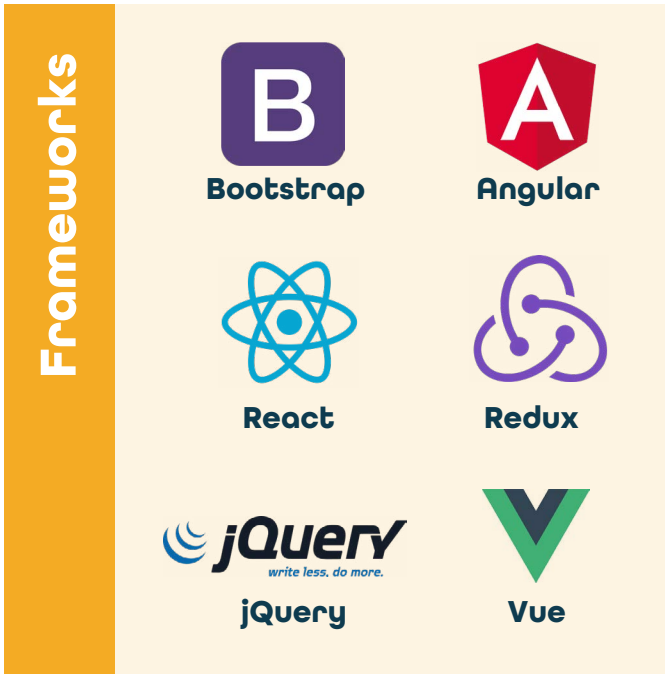
User behaviors occur based on, for example, a task they are trying to accomplish or the information they are seeking. For this reason, among others, it is beneficial for frontend developers to work with user experience (UX) designers, who are trained professionals in ensuring a satisfying experience for users. UX is in the same family as user-interface (UI) design and human-computer interaction (HCI); however, UX places greater emphasis on the user's emotions throughout an experience. In the digital world, it's critical to consider how a user feels because this determines the overall success of an application. An application can have a beautiful exterior and a bulletproof interior, but it will nonetheless fail if the user can't accomplish their tasks or finds a more pleasing visual interface elsewhere.

User interaction is the basis of an application's codebase.

Users come first. Applications have no purpose if they are unusable, or else do not help the business they are meant to serve.

Backend developers deal in the server-side of an application. A user interacts with the application in the frontend, and these interactions are processed in the backend. Information (data) is added, adjusted, or removed in the database, which is stored on a local or remote server.

Frontend Tools



To lay a framework for and fill in the content of an application, frontend developers use HTML (hypertext markup language). Styling can be applied in HTML, but this is rarely done because it makes far more sense to separate style from structure. The reasoning for this is especially evident in practice when developers are building large-scale applications. These might link to several stylesheets and scripts, which could not be accomplished with HTML alone.

Stylesheets are written in SCSS (cascading stylesheets) and compiled to CSS, and linked to the HTML file. SCSS allows programmers to use variables in CSS and fosters CSS templating and reuse. For example, Bootstrap theming is done through SCSS. Since certain style rules are very common, developers have created CSS frameworks like Bootstrap to make styling more streamlined.

With the framework and styling set, an application is presentable. However, HTML and CSS don't lend much to user interaction, aside from basic effects like rotation or dropdown navigation on hover. These pages, with no dynamically updating content, are referred to as static. For an application to have powerful interactive potential, it must involve JavaScript (JS). JS is used to create dynamically updating content, control multimedia, animate images, and perform other complex behaviors that cannot be accomplished in markup or styling languages.

Designers specify text styles in CSS. There are many aspects to the look and feel of text, and even subtle changes can make a big difference.

Bootstrap allows frontend developers and designers to design for mobile responsiveness.

JavaScript is essential for creating dynamic websites, which reveal different content under different circumstances.

Build and Deployment (cont.)

JavaScript also allows for API integration, further empowering developers to create impactful web applications. JavaScript is used to create dynamic applications, which can selectively reveal content. Server-side JavaScript dynamically generates new content on the server, whereas client-side JavaScript dynamically generates new content inside the browser. For example, client-side JavaScript might create a new HTML data, insert data requested from the server into it, then display the table in a page shown to the user. JavaScript gives front and backend developers great power to create dynamic applications.

Developers build tools and share their ideas to promote the production of better software.

Programmers like to use frameworks to be able to access time-tested building blocks of code. An application is typically built with 8-10 different frameworks and dozens of common libraries, if not more.

Using these blocks, programmers can offer powerful applications to their clients in much less time than if they had attempted to write them in vanilla code, without using frameworks. Buildable programmers are highly qualified to write vanilla code, but they also know exactly the right times to use frameworks. They can do more with fewer lines of code, and get the job done well and fast.

There are many frameworks in use in the world of web development, and more are rising all the time. It takes a trained eye to select combinations of frameworks that will be used in tandem to create the best possible application. Over years in the software development business, Buildable has become extremely familiar and proficient using frameworks like Bootstrap, jQuery, Angular, React, VueJS, and RxJS.

With frameworks and other tools, production is incredibly fast.

Buildable applications are created with the power of several foundational technologies and frameworks.

Build and Deployment (cont.)

The jQuery framework is nowadays mostly used for DOM manipulation as frameworks are moving away from depending upon jQuery. For example, Bootstrap 5 and Angular no longer include jQuery but jQuery still has its place for compatibility and executing browser-independent functionality.

Utilizing JavaScript frameworks, developers can write less code, yet perform more tasks. They allow programmers to use templating for component reuse and they simplify RestAPI, Ajax interactions and DOM manipulations, for rapid web development. Ajax is used to refresh data, make quick, incremental updates to the user interface without reloading the entire browser page. The DOM (Document Object Model) is a browser API that represents and interacts with any HTML or XML file. It allows code running in a browser to access and interact with every node (each representing an element) in the document.

Angular, VueJS, and React are full-fledged website frameworks that allow us to design web applications that are responsive and consistent with underlying business logic. The elements on the page react and evolve with a user's interaction, according to fundamental processes that are driving the purpose of the application itself.

Redux is a tool for managing both data-state and UI-state in JavaScript applications, generally used in tandem with Redux. Developers can write applications that behave consistently, run in client, server, and native environments, and are easily tested. It's ideal for use in SPAs, where managing state over time can be complex. Redux is also framework-agnostic so, while it was created with React in mind, it can be used together with Angular and VueJS.

Using each of these tools described and some others, Buildable frontend developers create effective, efficient, and aesthetically appealing application interfaces.

Web applications are a sum of layers, from structure to design, content to functionality. The frontend is what the end-user sees. What powers the application, though, the user does not see. The technology and programming behind frontend interfaces are called the backend.

The backend consists of the server and the database. Sometimes called the brain of the application, the backend is the machine that runs a site. It is always running in the background, delivering smooth functionality, and drawing information from the database right into the browser. Backend code adds utility to everything the frontend designer creates.

The DOM is a complex document loaded in the browser. jQuery makes it easier to write DOM manipulations.

Angular makes it possible to design web applications that make sense with business logic.

The backend can be thought of as layers that provide functionality and information to the application's frontend.

Build and Deployment (cont.)

When a user accesses a web application, they are requesting information from a server via the Internet. As the user interacts with the dynamic web page, they make requests for information. Server-side APIs structure how data is exchanged, then server-side scripts and frameworks take that information and process the request, pulling what they need from the database. Then, the server sends information back over the Internet to be delivered to the frontend of the application. Anything you see on a dynamic web application is made possible by back-end code, which exists on and is powered by a server.

Responsibilities of the backend include:

- Database access (SQL Server, MySQL, SQLite, MongoDB, etc.)
- Web server technologies (Apache, NGINX)
- Cloud computing integration (Amazon Web Services, Azure)
- Server-side programming languages (Python, PHP, JavaScript, Node.js)
- Content management system development, deployment, and maintenance
- API integration
- Security settings and hack prevention
- Reporting analytics
- Backup and restore technologies

In this whitepaper, we will cover just a few of the powerful technologies implemented by Buildable backend software engineers.

Backend developers use a wide array of programming languages and frameworks when building server-side software and choose these on a per-project basis. Applications may have specific requirements that call for the use of a particular language or framework, or a certain set of frameworks might work together to satisfy requirements in a powerful way.

When there is an interaction on the frontend, the backend must communicate through a series of layers to access the necessary information.

Backend software engineers must ensure that databases are created, integrated, and managed effectively.

Analytics are important for gauging the effectiveness of your web application.

Build and Deployment (cont.)

ASP.NET Core is an open-source and cross-platform framework for building modern, cloud-based, Internet-connected applications, such as web applications and mobile backends. It was created to optimize development frameworks for applications that are deployed to the cloud or run locally. Key features of ASP.NET Core include:

- Unified UI and API build stories
- Integrated frameworks and workflows
- Built-in dependency injection
- A lightweight HTTP request pipeline

The .NET Framework has its runtime environment, which allows code to be run on any hardware on which .NET is supported.

ASP.NET Core builds on ASP.NET, which was released over 15 years ago. The redesign consists of architectural changes that have resulted in an open-source framework that can be run not only on Microsoft operating systems but also on Linux and OSX. Since developers only need to include the packages they need, they are left with a smaller application surface area. This entails tighter security, reduced servicing, improved performance, and decreased costs.

Entity Framework (EF) is an object-relational mapper enabling .NET developers to work with relational data using domain-specific objects. EF eliminates the need for most of the data-access code that developers usually need to write.

PHP code is executed on the server, then returned to the browser as plain HTML.

Language-Integrated Query (LINQ) allows a single, general-purpose declarative query facility to be applied to all in-memory information, not just information from external sources. Both LINQ and EF allow for compilation testing on database access.

C# is the main programming language of the Microsoft .NET framework. It is a modern, typesafe, object-oriented, and comprehensive language with advanced features. Features that distinguish C# from other languages are its portability, strong typing, metaprogramming, memory access, and polymorphism.

PHP (Hypertext Preprocessor) is an open-source scripting language. PHP files can contain text, HTML, CSS, JavaScript, and PHP code, although the code is executed on the server and returned to the browser as plain HTML. PHP generates dynamic page content, handles files on the server, collects form data, sends and receives cookies, manages data, controls user-access, and encrypts data. It runs on various platforms, is compatible with most servers used today, and runs efficiently on the server-side.

We've already discussed client-side JavaScript, the version of JS extended to enhance and manipulate web pages and client browsers. It is also available as server-side JavaScript (SSJS), which extends JS to enable backend access to databases, file systems, and servers. SSJS builds more scalable, event-driven, and non-I/O (input/output)-blocking applications.

Backend Tools

The REST (Representational State Transfer) API is a resource-based architectural style that passes representations between the client and the server. REST APIs are used to interact with database or backend systems, are stateless, and authenticated. They allow us to Create, Retrieve, Update, and Delete (CRUD) entities from a data store. (Ex. database, file system, etc.)

REST operates off six constraints, which define the RESTful style of programming:

- Uniform interface
- Stateless
- Cacheable
- Client-server
- Layered system
- Code on demand (optional)

Because Buildable software engineers comply with REST constraints, our applications are scalable, simple, modifiable, visible, portable, and reliable.

SQLite empowers developers to update content continuously and atomically, so that little or no work is lost in a power failure or crash.

SQLite is a self-contained, high-reliability, full-featured, and open-source relational database management system contained in a C programming library. SQLite stores the entire database as a single cross-platform file on a host machine. Since it is a compact library, even with all features enabled its size can be less than 500KB. It can be made to run in minimal stack space. SQLite responds gracefully to memory allocation failures and disk I/O errors. All transactions are ACID (Atomic, Consistent, Isolated, and Durable), even if interrupted by system crashes or power failures.

MySQL is secure at the enterprise-level, highly reliable, and trusted by the world's leading brands. Its extensible, modern architecture at every layer in the database allows for a flexible configuration that supports both traditional and emerging enterprise use cases.

SQL Server is trusted by the enterprise market and MySQL is trusted by companies such as HP and Wikipedia.

SQL Server is the gold standard on Windows implementations. It runs on both Linux or Windows, on-premises or in the cloud. It provides noSQL and JSON capabilities which makes for an excellent choice for any web application.

With these resources at the ready, Buildable software engineers are prepared to develop sophisticated web applications for your business.

Build and Deployment (cont.)

When a new release must be built, source code is fully checked out from source control, the version number is incremented (automatically or manually, at the maintainer's discretion), the build process is launched, executable code is bundled, and the release candidate is deployed to a test server for integration and quality assurance (QA) testing.

All Buildable software is rigorously tested in QA before deployment.

Once QA testing completes, our development team submits the release candidate to the client, who is expected to conduct a QA process on their end. At the end of client testing, they can confirm whether the software is operating as expected and all functionality interpretation differences have been resolved. If not, then the development and release cycle will repeat.

Each version of the software is clearly labeled to avoid confusion and help the user know whether they are up-to-date.

Webpack bundles software into one large file and is then securely transferred to the client's system for installation.

Version numbers are usually displayed in the footer, in the About section, or on the Login page of applications. Regardless of application type, the build process will always output code in what is called release mode, which is optimized for speed. Web projects will add a few extra steps: JavaScript syntax validation; typescript compilation check; CSS, JavaScript, and HTML minification; and bundling all these elements into the fewest number of files possible.

The build process (web pack) will output the bundled file and transfer it to a network folder to be archived, called the Release Publishing Destination (RPD). For Windows desktop applications, the software will typically be signed and built into a self-installing package (MSI or Setup.exe, for instance). In other cases, such as with UNIX services, the software will be built with self-installing command-line capabilities. For web applications, the software will be zipped into a self-containing archive.

The deployment script is always built to execute the following tasks:

- Backup of the current system as a rollback point, by creating a .zip file of the current web application
- Installation, by downloading the release from the RPD and unzipping and restarting services which require it (for example, NGINX and Kestrel)
- Rollback, which takes place in such a case where the installation fails, by removing installed software, unzipping backup, and restarting services

Once the deployment is executed, the new version should be displayed on the home page of the web application. Our process provides a reliable mapping between published release and installed release, as well as providing a rollback mechanism whereby a prior release can be quickly restored. All servers hosted by Buildable are backed up twice daily, to ensure a safety net in the case of unexpected challenges.

Authentication and Security

Among the many challenges that software engineers face, authentication and security are of extreme importance. Yet, they are often under appreciated.

On the contrary, Buildable pays dutiful attention to developing secure applications that protect your business' and your clients' information alike, on top of being created with beautiful interfaces and superb functionality. With Buildable technology, your business can focus on what it does best.

Buildable provides exclusive access to parts of applications with JWTs, which cannot be replicated.

In today's world, where so much personal information is hidden within digital halls, software engineers must ensure that they are developing with authentication and security in mind.

When an application gives select users access to restricted resources, processes of authentication ensure that users are who they say they are. Buildable handles this with JSON Web Token (JWT)s, which are signed tokens that contain the roles users have access to. They are forgery-proof, meaning that a hacker cannot forge a token of their own unless they have a private key, which only exists on the server in a secure folder. The token is created upon login and sent back to the browser. This method of access is session-less. Therefore, it is easy to load balance–called scaling–to a farm of web servers, if each server can extract the token properly.

We instill security in our applications through the following methods:

- Ensure all communication happens over SSL (Secure Socket Layer), a protocol for transmitting private documents that use two keys, one private and one public, to encrypt data
- Restrict access to SSH (Secure Shell)–a program to connect with a remote computer over a network–only allowing entry with certificate authentication
- Open only Port 443 on the server
- Use Fail2ban and Monit for monitoring and detecting suspicious activity

Using these technologies, Buildable can provide the best in software security.

Web applications are monitored with Monit, software configured to send email alerts when logs contain a pattern or when the machine breaks a specific threshold. Monit comes packed with tons of useful application checks, including memory usage, HTTP requests, and CPU usage.

Buildable applications are also strengthened with Fail2ban, which operates by monitoring log files for the potential intrusion. Upon such an attempt, Fail2ban blocks potential intruders from the system for 24 hours by adding rules to the firewall.

NGINX logs are also kept. NGINX is a web server that can be used as a reverse proxy, load balancer, and HTTP cache. Its modular, event-driven architecture provides more predictable performance under high loads. NGINX leaves a low memory footprint, which keeps your system fast and secure.

Buildable

FOR MORE INFORMATION
www.BuildableWorks.com | sales@BuildableWorks.com
+1 (503) 468–4880 | 620 NE 3rd St, Suite A, McMinnville, OR 97128