



LAUREA
UNIVERSITY OF APPLIED SCIENCES
Together we are stronger

Creating software component using atomic design and test-driven development

Le, Nghi

2017 Laurea

Laurea University of Applied Sciences
Leppävaara

**Creating software component using atomic design and test-driven
development**

Nghi Le Vinh
Degree Programme in
Business Information Technology
Bachelor's Thesis
January, 2017

Le, Nghi Vinh

Creating software component using atomic design and test-driven development

Year	2017	Pages	61
------	------	-------	----

We are living in the age of hyperconnectivity where millions of people are using digital communities to exchange information with each other. People are radically changing the rules of business. They demand effortless, smooth, and personalized experiences at every touch point of the customer journey. To make the most of the opportunities ahead, businesses must go beyond hardware devices and traditional ways of serving their customers.

Nordea is keenly aware of these opportunities. The bank is building new digital solutions to offer its customers the best-in-class experience. This thesis project concerns the building of a common software component that can be used by developers to facilitate their development process in one of Nordea's digital project. In the financial sector, each change must be done in the way that it is secure, compliant, yet scalable and maintainable. This thesis project serves as a proof of concept of why and how using common components helps developers to complete their development faster, as well as making software well-organized, scalable and maintainable.

The thesis report offers a brief introduction to Nordea, as one of the leading banks in Northern Europe. The theoretical section discusses the concepts underlying the thesis project, such as good practices in software development. In addition to that, it briefly explains various development methodologies in the project implementation process, such as atomic design, test-driven development, and code coverage. The objective was to build a common component that was scalable, maintainable and could later be used by other developers to build their software features.

In this thesis project, atomic design is mentioned since it is the methodology that is used in system designs which lead to the design of common components. In the project implementation, test-driven development (TDD) is used. TDD is an increasingly popular and practical development methodology in today's software industry. It depends heavily on the repetition of a succinct cycle of development iterations. Tests cases are converted from a breakdown of functionalities/features requirements. Using TDD, developers feel more productive, the development process results in more tests, fewer defects with better software design and maintainable code. In addition to that, code coverage is used as the metrics to measure how efficiently the tests in TDD have been exercised.

As a result of this thesis project, a common component is developed as a proof of concept of how common components can be developed using test-driven development. This component is also published on Node Package Management as an open source software project where the bank, as well as other developers, can freely enjoy accessing and using it as a third party dependency.

Keywords: TDD, common component, software development, web, Agile, finance, bank, Nordea

Table of contents

1	Introduction.....	8
	1.1 Case organization: Nordea	8
	1.2 Project background	9
	1.3 Goals and objectives.....	10
	1.4 Project scope and limitations	10
2	Methodology	11
	2.1 System development life cycle.....	11
	2.2.1 Agile software development	11
	2.2 Atomic design	10
	2.3 Test-driven development (TDD).....	10
	2.4 Code coverage	10
3	Theory	18
	3.1 The automated test pyramid	18
	3.2 Common component.....	19
	3.2.1 Definition	19
	3.2.2 Benefits of common component.....	19
	3.3 Web component	20
	3.4 Repository	20
	3.5 Software development good practices.....	20
	3.5.1 Code refactoring.....	21
	3.5.2 Design pattern	21
	3.5.3 Unit testing	21
	3.6 Tools.....	21
	3.6.1 Webstorm	21
	3.6.2 Git.....	22
	3.6.3 Github	22
	3.6.4 NPM	23
	3.6.5 Gulp.....	24
	3.6.6 Browserify	25
	3.6.7 browserify-istanbul	25
	3.7 Programming languages and frameworks.....	25
	3.7.1 JavaScript.....	25
	3.7.2 MomentJs	26
	3.7.3 Lodash	26
	3.7.4 NodeJs.....	26
	3.7.5 AngularJs.....	26
	3.7.6 CSS	26
	3.7.7 Little Calendar CSS	26
4	Project implementation.....	27
	4.1 Atomic design of Date picker common component	27
	4.2 Moment-calendar-2 component	30
	4.3 Moment-calendar-2 requirements.....	30
	4.4 Implementation plan.....	31
	4.5 Implementation process.....	31
	4.5.1 Set up repository and build system	31
	4.5.2 Build system	32
	4.5.3 API Design and structure	34
	4.5.4 Implementation using TDD	37
	4.5.5 Code coverage	40
	4.5.6 Documentations and releases.....	42
5	Evaluation	47
	5.1 Evaluations of TDD in development	47
	5.1 Evaluations of developing common software component using TDD	47
6	Conclusion.....	49
	References	51
	Figures.....	54

Tables.....55
Appendixes.....56

Terms and abbreviations

FinTech	Financial technology, an industry composed of companies that use new technology and innovation to compete in the market-place of traditional financial institutions
Financial services	services provided by the finance industry including credit unions, banks, insurance companies, stock brokerages, investment funds etc.
Financial institution	an institution that provides financial services for its clients or members
Compliance	adherence to standards, regulations, and other requirements
Big data	data sets that are so large or complex that traditional data processing application softwares are inadequate to deal with them
UI	User Interface
UI Test	tests include functional testing of the UI controls. They let you verify that the whole application, including its user interface, is functioning correctly
UX	User experience
TDD	Test driven development
HTML	Hypertext Markup Language
Front-end	Short for Front-end web development. Also known as client-side development is the practice of producing HTML, CSS and JavaScript for a website or Web Application so that a user can see and interact with them directly
ECMAScript	Also named ES. A trademarked scripting-language specification standardized by the European Computer Manufacturers Association in ECMA-262 and ISO/IEC 16262. It was created to standardize JavaScript, so as to foster multiple independent implementations
NPM	the default package manager for the JavaScript runtime environment Node.js
Customer journey	the complete sum of experiences that customers go through when interacting with your company and brand
Test case	a set of conditions under which a tester will determine whether an application, software system or one of its features is working as it was originally intended
Stubs, mocks, fakes	a piece of code used to stand in for some other programming functionality which mimics the behavior of real objects in controlled ways
Test harness	Also called automated test framework. A collection of software and test data configured to test a program unit by running it under varying conditions and monitoring its behavior and outputs. It has two main parts: the test execution engine and the test script repository
Nondeterministic algorithm	an algorithm that, even for the same input, can exhibit different behaviors on different runs, as opposed to a deterministic algorithm. An algorithm may behave differently from run to run due to it is a concurrent, probabilistic, time-dependent one.
Asynchronous	the state of not being in synchronization.
Chronometer	An instrument for measuring time, especially one designed to keep accurate time in spite of motion or variations in temperature, humidity, and air pressure
High-level programming language	a programming language with strong abstraction from the details of the computer. It may use natural language elements, be easier to use, or may automate (or even hide entirely) significant

areas of computing systems

Dynamic programming language	class of high-level programming languages which, at runtime, execute many common programming behaviors that static programming languages perform during compilation
Untyped programming language	Languages without static type systems
Interpreted programming language	a programming language for which most of its implementations execute instructions directly, without previously compiling a program into machine-language instructions
Service-oriented architecture	A service-oriented architecture (SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network.

1 Introduction

The advent of the internet has over the past decades dramatically transformed the way people work, socialize and share information. The internet alone has contributed to the creation of millions of jobs and billions of dollars in economic activity, which has a profound impact considering the bleak economic outlook the economy has suffered in recent years.

Living in such an exciting landscape of the digital age, most executives from banking and financial service are keenly aware of the fact that this can be an opportunity or a threat. Hence, it is not about whether should they join the trend and engage their business in digital transformation but rather how they can achieve it while maintaining their current legacy IT infrastructure, regulations compliance, a healthy sustainable business which brings positive impacts on society.

As one of the largest and leading banks in Europe, Nordea has already been in the game. The bank has put tremendous efforts in leverage existing IT investments and innovate its digital strategy. It knows too well that digital transformation is now not only a nice thing to have, but also a matter of survival. Nordea is in the process of creating a new digital solution that changes and enhances the way it serves customers. As a large enterprise, creating a solution like this does not come easy for Nordea. The bank needs to think ahead of time carefully how to build it solution as quickly as possible to shorten the time to market yet ensure its security, scalability, and maintainability.

With the experience of working in one of Nordea's largest software development project, this thesis covers the process of building common software component using TDD to facilitate the development process, making sure that the project being developed, delivered on time yet ensure its scalability and flexibility. The thesis is divided into 6 main sections. The introduction section introduces about Nordea, the goals and objectives of this thesis and describes the development project background. It also lays out the scope and limitations of what is written. The methodology part describes in depth the frameworks used in the research part of the project. The theory section builds the necessary knowledge base for later ease of comprehension. This is where all the terms and concepts is explained more thoroughly, what tools and dependencies the project is developed upon, as well as what programming languages and frameworks are used in the project implementation. A development of a common component using TDD will be demonstrated step by step in the subsequent section. Finally, the last two sections are about the evaluation of the project and conclusion of the thesis.

1.1 Case organization: Nordea

With more than 11 million customers, 1400 branch offices and 30000 employees, financial services Nordea (publ) is one of the leading and largest financial services company in the Nordic and Baltic region. It offers a wide range of services and products for its household and corporate customer. The organization has operations in Finland, Sweden, Denmark, Norway and some Baltics countries. It has a good reputation in its transparency and its eagerness in improving its current digital financial platform with the vision to become a great European bank.

The bank is the result of the successive mergers and acquisitions of the Finnish, Danish, Norwegian and Swedish banks. Therefore they had a huge fragmented legacy system including different stacks of technologies that are needed to be modernized, unified, simplified and maintained to make them more lean and agile. This process of unification and simplification faces challenges especially when it comes to digital services, which is a strategic area, where Nordea as well as other banks are striving to further enhance to provide the best customer experience for their users.

1.2 Project background

The project that currently being developed is one of many projects that Nordea is building in an effort to transform itself to become one of the leading banks in digital innovation who offers best-in-class services for its customers.

Since the project is huge, and the demand for fast delivery is high, the development of a common foundation framework where all the teams across business areas can reuse and implement in a rapid and agile fashion is needed. This thesis will examine one of the common components that is used by teams to implement their feature in the project's application. This can also be served as a proof of concept on how a common component is developed using TDD, which is one of the strategies that is heavily concentrated by Agile methodologies. Development common component using TDD predominantly optimizes development time and enhance the quality of the deliverables.

The mentioned component is named moment-calendar-2 which is used to build Date picker components where end users can interact with and pick a date of their choice. Date picker is a common UI component that is developed based on Nordea UI/UX designers. It is a large and complicated component which includes a user interface (UI) whereas moment-date-2 can be seen as its core engine, where all the date logic is taken charge of, without any UI. Below is also an example of how a banking application can make use of Date pickers and moment-calendar-2:

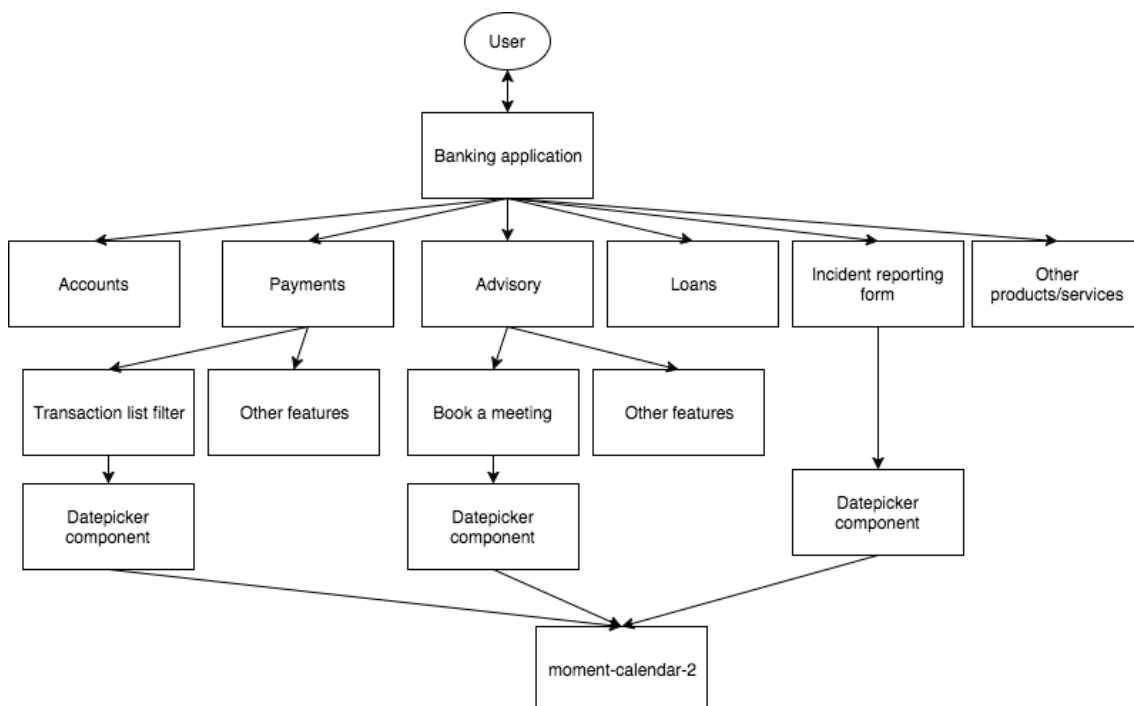


Figure 1: Architecture design of a general banking application where moment-calendar-2 is used

As can be seen in the diagram, a typical banking application has its UI to offer its products and services such as Accounts, Payments, Advisory etc., for its customers. Each product and service is a composition of several features, and each feature is built based on templates, modules and common components (in the Atomic design section, more information about how this whole system model is created). Each common components can be built based on other smaller common components. In this case, the Date picker is an UI common component which is developed based on moment-calendar-2, the common component that will be discussed in the project implementation section.

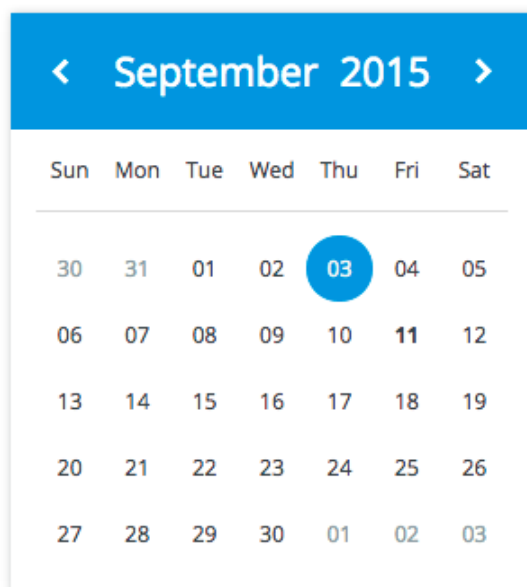


Figure 2: Date picker UI example

The name moment-calendar-2 comes from the fact that it is build based on MomentJs library, a library that alleviates parsing, validating, manipulation and displaying dates in JavaScript. '2' is the version number since there has been another component with the same name. The component is a super lightweight, easy-to-use yet powerful Node module to create month calendar where each date is a MomentJs date. Its capability can be leveraged further with date operations as well as configuring localization through MomentJs.

The component is developed with scalability in mind and can be in another project, different business context, and different technology stack. The demo application will give a glimpse on how a small, non-UI piece of software can bring lots of power by demonstrating how to use moment-calendar-2 together with angular to build a Date picker component. A screenshot of the demo application can be found at appendix 3.

1.3 Goals and objectives

The goal of this thesis is to create a common software component using Test-driven development that is used to build Date picker or other date/time-related component in the company project.

With those goals in mind, the followings are objectives, which are also the major requirements of this common component, that this thesis strives to achieve:

- Build a demo application to illustrate how the common component is used.
- Document and publish the component to Github so that the community can further develop and enhance the component.
- Release the component to NPM to make it available to the open-source market allowing it to be used by any party or developer.

1.4 Projec scope and limitations

The thesis mentioned about Atomic design to give readers a better view of how the idea of a common component is developed from in the project implementation part. It, however, does not cover the design process of the component (i.e how it is designed using Photoshop etc.) or how the component requirements are defined at Agile methodology level or at a higher level of the development project in the system development life cycle.

2 Methodology

In this section the methods used in the research and development process of the project are described. It first introduces about systems development life cycle (SDLC) as the development process that the application project employed at the highest level. Following that, this section covers the methodology of Atomic design for approaching system design and how this leads to the development of common component. It also discusses about Test-driven development deeper in theory which later will be implemented in practices in later section.

2.1 System development life cycle

The systems development life cycle (SDLC), also referred to as the application development life-cycle, is a term used in systems engineering, information systems and software engineering to describe a well defined process for planning, creating, testing, and deliver a software product. A system that is produced by SDLC can be equivalent to a product that is manufactured on an assembly line. The aim is to produce high-quality systems that meet customers expectations, based on their requirements, by delivering systems that move through each distinct work phase, within scheduled time frames and cost estimates. A number of SDLC models have been created to manage the complexity of computer systems, especially with the recent trend of service-oriented architecture, whose basic principles are independent of vendors, products and technologies. Those are "waterfall"; "spiral"; "Agile software development"; "rapid prototyping"; "incremental"; and "synchronize and stabilize". At a higher level, the project employs Agile as its main project development methodology.

2.1.1 Agile software development

Agile is one of the big buzzwords nowadays not only in the IT sector but also in some other areas as well, ranging from financial services sector to marketing and designing. The word "Agile" derives from the agile manifesto which was made by a small group of people who got together in 2001 to exchange their awareness and concerns about the traditional approach in managing software development projects. They realized that the current way was vulnerable too often, and there had to be a better more sustainable way. They conclude with the agile manifesto, which describes 4 important core values that are still relevant today:

"Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan"

Ever since that time then, the use of Agile methods that support these values has become more and more demanding.

Simply put, agile development is a different way of managing development teams and projects. In software development area, it is an approach to development that iteratively builds software in a small time frame for delivering after each development cycle instead of trying to deliver it all at one go in the end. It works by breaking projects down into smaller features where each of them includes user functionality or user stories, where they will be prioritized, and then continuously delivered in short time-box cycles called iterations.

2.2 Atomic design

Atomic design is methodology, a unifying theory when it comes to designing systems and common components and not just web pages. This is not necessarily a new approach but is a tried and true one and definitely an interesting way of looking at systems design in general and web design in particular. This is especially useful when it comes to massive large soft-

ware application where maintaining styles and functionalities consistent is crucial and in high demand.

Take web design for example. There have been a lot of discussions about how to organize and establish style sheets structure where foundations of color, typography, grids etc. are laid out. Those aspects concentrate heavily in detail on how the UI looks. Atomic design, on the other hand, is more about how design system is constructed in a methodical way. The name "Atomic" is inspired by the idea that matter is composed of atoms. The atomic units combine to form particles, which in turn bond together to become more complex molecules and ultimately create all matter.

In the same manner, systems are made up of smaller individual components. This is the fundamental idea of Atomic design. In web development, there are five clear levels in atomic design: Atoms, Molecules, Organisms, Templates, Pages.

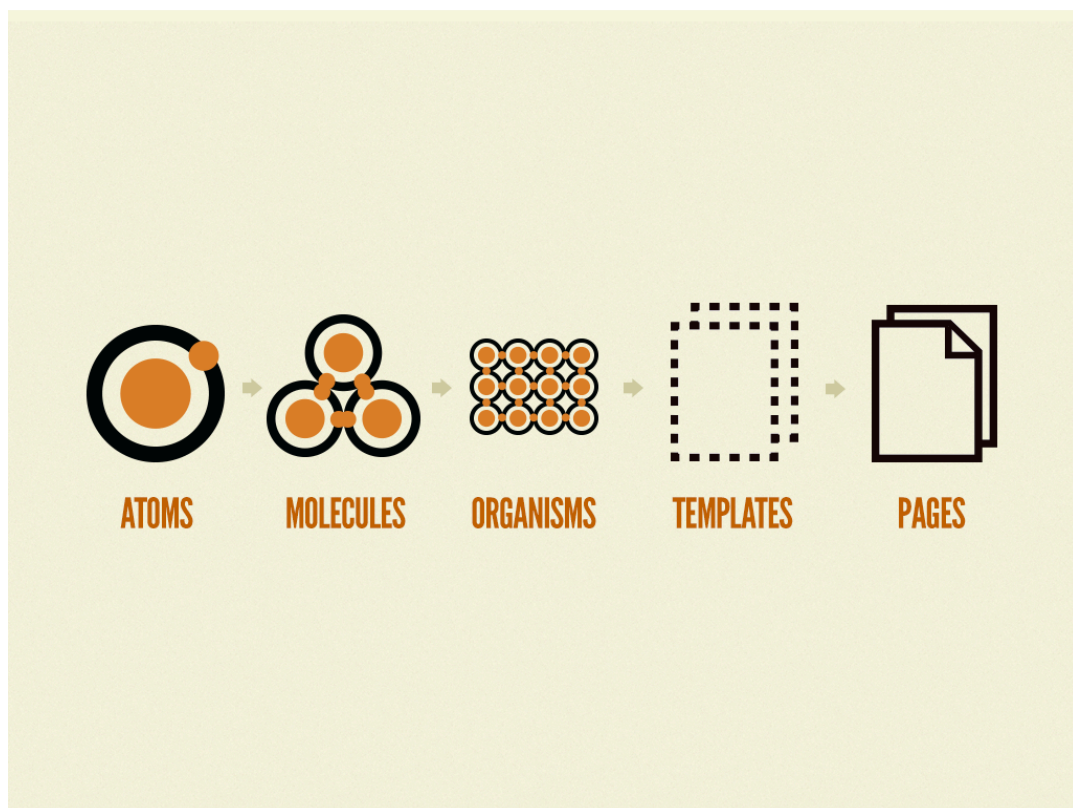


Figure 3: Five levels in atomic design

Atoms: atoms are the basic building blocks of everything. In the case of web UI, they are the native HTML tags such as button, select, input, label etc. They can also be something more abstract like color, fonts, or even animations. As their name, atoms alone do not bring many values at all. They are, however, good for context referencing and maintaining when a specific variable needed to be justified. In programming, atoms can be variables, constants etc.

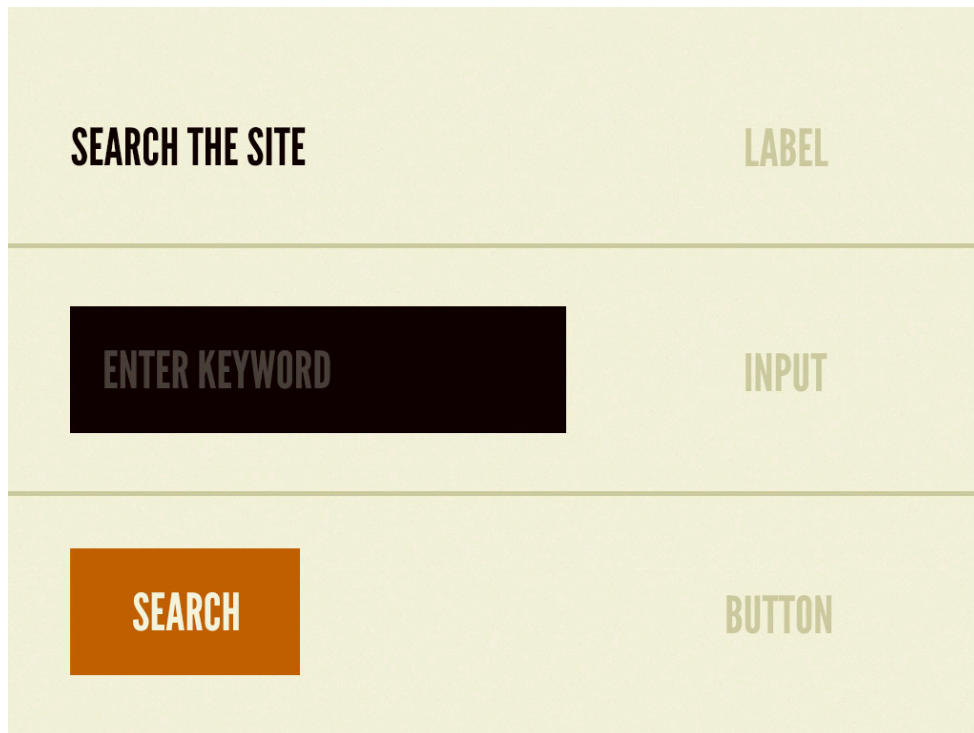


Figure 4: Atom level in atomic design

Molecules: when atoms are combined together, they form molecules. This is where interesting things happen. These molecules can have their own properties and functionalities and serve as the backbone of the system design. In UI case, it can be any elements ranging from a form, a panel with question texts and confirmation buttons etc. In web programming implemented by JS, molecules are components like directives, SASS functions etc. In fact, the component that this thesis will later focus on developing is a molecule component which will be discussed deeper in the later sections.

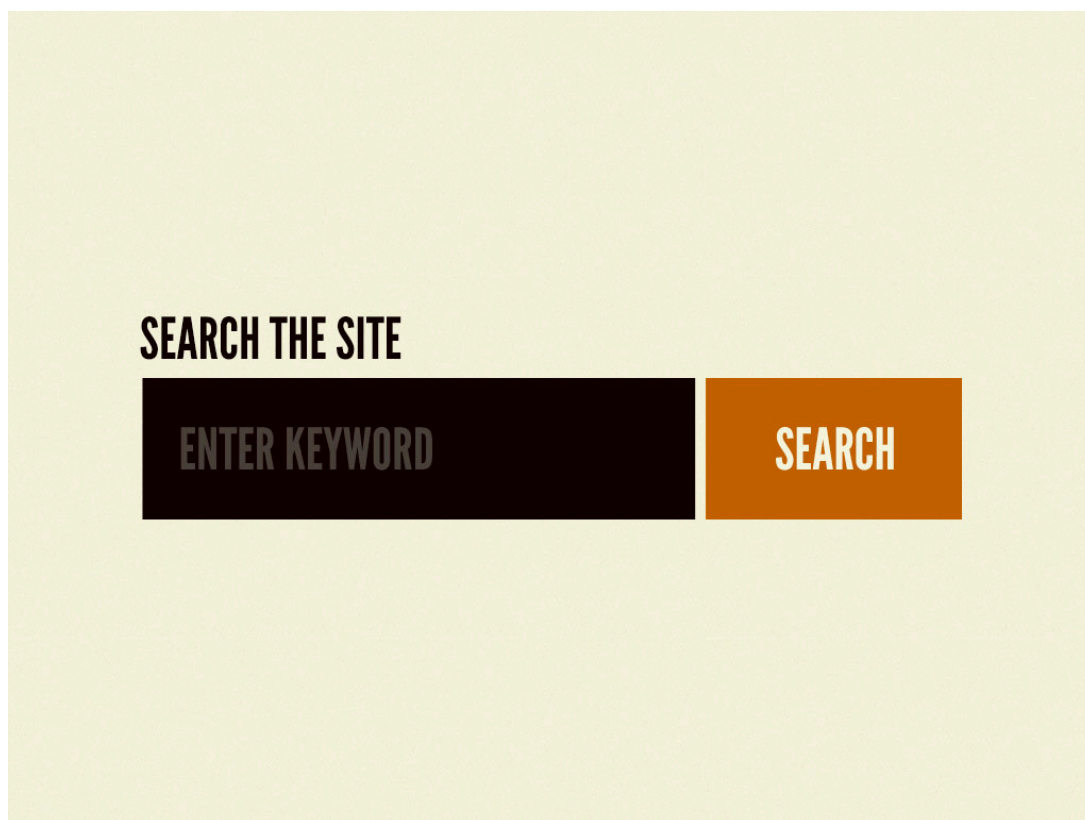


Figure 5: Molecules level in atomic design

Organisms: Organisms are groups of molecules bound together to established a more sophisticated unique section. In this thesis, we briefly discuss Date picker component, which based on this theory, can be considered as an organism, since Date picker contains functionalities that it requires from molecules like date and timer formatting module, text and number styles component, date and time calculation, navigation buttons, even possibly input fields, etc.

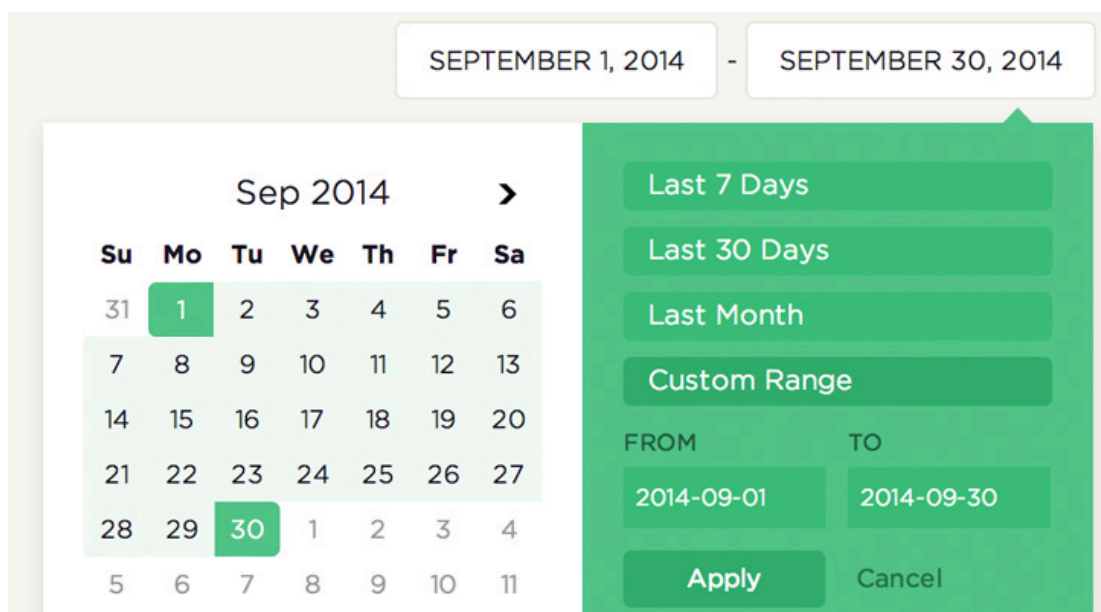


Figure 6: Organisms level in atomic design - Date picker with input field

Templates: in UI design, templates are groups of organisms collaborated together to form pages. At this stage, we no longer use chemistry terms but instead, we get into the customer language. This is where a layout is formed and something concrete and visible for the end users.

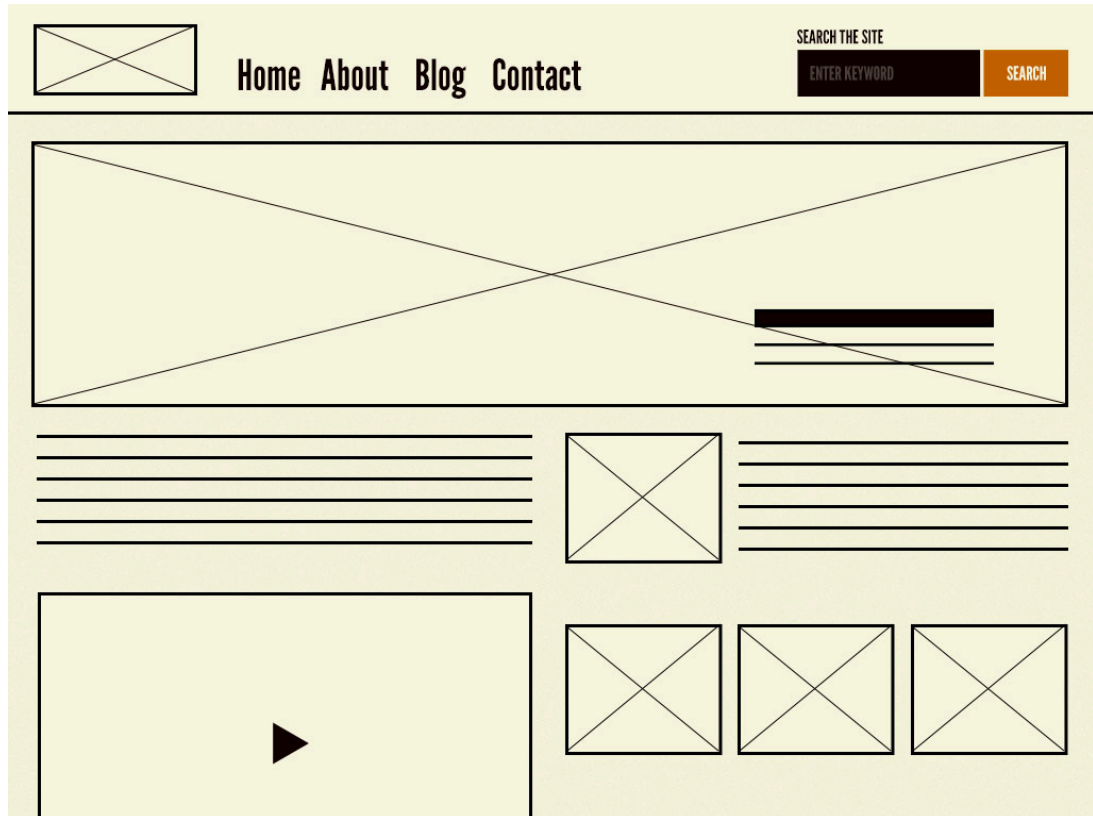


Figure 7: Template level in atomic design

Pages: pages are unique instances of templates. It can be a composition of different templates. At this stage user will be able to see the final content laid out in front of their eyes, with real texts and images, not just a placeholder. It is the highest level of fidelity and tangibility. This is where most time is spent on reviewing.



Figure 8: Page level in atomic design - Nordea front page

Atomic design facilitates a scalable and portable way of developing system application. Components are later easier to be developed or extended if they have already a solid common base. It is definitely a good approach that well played with Agile methodology strategy where it is developed in close, collaborative environment including designers, developers and other stakeholders.

Though this thesis does not have any UI implementation, Atomic design still worths mentioning here since this is the backbone philosophy to develop web common components in our project. Later in the common component implementation section, we will come across Date picker component, which is an UI component that is also developed based on this philosophy.

2.3 Test-driven development

Test-driven development (TDD) is an increasingly popular, and practical, development methodology in today's software industry. It heavily depends on the repetition of a succinct cycle of each development iterations. Tests cases are converted from requirements or small broken down pieces of functionalities/features. If further development is needed, then the software is enhanced to pass the new tests set.

According to Ken Beck, who is considered as the father of TDD, TDD encourages simplicity and confidence in systems design. TDD is also relevant and consistent with test-first programming philosophies of extreme programming, which was founded in 1999, but now has gained more popularity under its own name.

The idea of test-driven development is developer write a few tests first, make them fail, before proceeding with the implementation to make them pass. Following each test will be discussed more specifically:

First, test is written. Everything starts with a test being written. The test serves not only as a verification step but also as a step to function definitions and enhancements. Each test should be succinct and concise to the feature it strives to cover. Doing this way, it forces the developers to keep his focus on the requirements. This step is special, which makes it stand out from writing standard unit tests after the code is implemented.

Second, some tests are run just to fails. This step is a way to prove that the test mechanism is working correctly, even though the tests does not pass. The new tests shall not pass just to validate that the new feature has not been implemented. This ensures there is no possibility that the new test is flawed and passed. The tests fail but they will bring a confidence boost for developers who are implementing them.

Third, code is implemented. Next step is to implement the code that helps the test to pass. The implementation can be just inelegant, even useless, as long as it can make the test passes since it will be enhanced and fully developed later in Refactoring step.

Fourth, run tests again. The tests are later run again and should pass. If tests do not pass, clearly the code implementation stage has not been handled correctly. This again gives a necessary confidence boost for developers.

Fifth, code refactoring. This is where real code is implemented. During this phase, growing code based should be cleaned up frequently to reduce wastes and duplications. All programming entities (functions, variables, classes etc.) should be clearly defined and represents their single purpose and responsibility to enhance maintainability and readability.

Continuing repeating these steps, the software implementation is enhanced incrementally after each cycle. Continuous integration also plays a role as checkpoints during this process. When developing with the help of external libraries, it is crucial to make small increments and avoid testing the library itself.

When it comes to software development and writing code, TDD is unfortunately not that ubiquitous. Many developers still do development first and probably write tests later. This practice overrides the benefits of test-driven development. Why it is so crucial to write tests first and why test-driven development plays a big role in Agile development? Section 5 will strive to answer those questions.

TDD is all about changes and therefore it is expected that the code will be re-written several times. Every time code is re-written, the developer improves the code, and improves himself - his programming skills in a significant way. Though TDD is just one possibility of doing Agile development, these mentioned above are hidden charms of using TDD, and why it is so well-suited with Agile and in fact, an art of Agile development.

2.4 Code coverage

When it comes to testing, test coverage is a methodology in software programming to measure how much a software application source code has been covered by tests. It is a metric that help determine how much thorough the tests have been exercised. There are various kinds of test coverage:

- code coverage
- feature coverage
- scenario coverage
- screen item coverage
- model coverage

Each of these coverage has a different baseline defining the system under test. Test coverage types, therefore, varies depending on how the system is defined. The test coverage system that is used in the implementation part of this thesis is code coverage. It has the following baselines or rather than questions that the code coverage will strive to answer:

- has a particular statement been executed?
- how many times has a statement been executed?
- have all the statements in a program been executed?
- have all the decision points in the code/branches been exercised?

3 Theory

This section covers theoretical concepts that are needed to build a strong knowledge base for comprehending the upcoming sections of this thesis. It first dives deeper into the automated test pyramid, and explains which test phase is important, as well as how it is done during the development process. It explains some of the main concepts of the thesis, and also introduces some best practices that are used in the process of software development.

3.1 The automated test pyramid

Despite all the differences in Agile methods, they all share the same principles as listed above. This thesis will particularly focus on discussing deeper about principle number 9: Testing is integrated throughout the project lifecycle - test early and often. This principle has a huge impact both from technical and business point of view since it plays an extremely important role in managing the quality outcome of the final product.

In his book *Succeeding with Agile*, developed by Mike Cohn, the concept of test pyramid is first introduced. The main idea of this model is the higher level of the test pyramid, the more expensive it costs for business and more complicate it is for engineering.

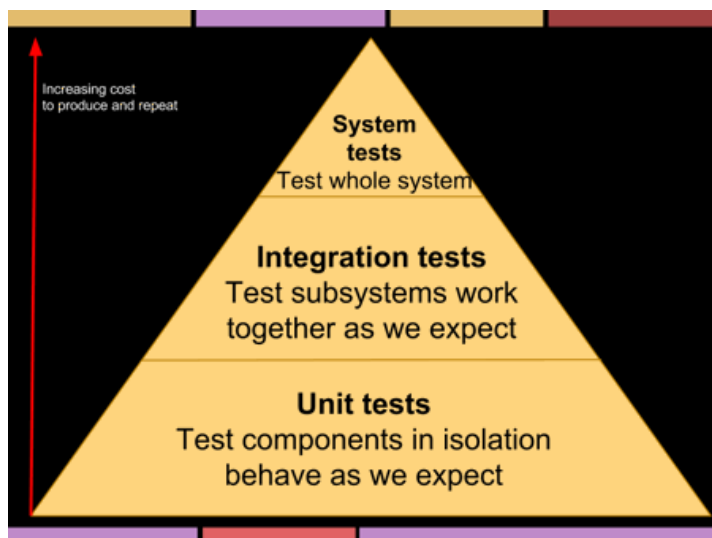


Figure 9: Test pyramid model

In the test pyramid visualization above, there are 3 levels of testing: system tests, integration tests, and unit test. A good test strategy is to focus on each type of differently, or in other words, testing mix. A good approach would be testing mix with a majority of unit tests (70%), some integration tests (20%), and a small number of acceptance tests (10%). The percentage figure in brackets is just for reference. In reality, there can be many combinations of these that should be selected appropriately since they need to be easily maintained. Here are the reasons:

Acceptance tests are the most expensive tests to be done. This is why it is crucial to write, run and maintain minimally these types of tests. The reason is acceptance tests consider other third parties and system to be taken into account and not mocking them. They answer a question of "Is the System working?". Sometimes, at this level, it would be better for manual testing in case there are so many systems being involved hand hard to coded. Acceptance test can not be fully trusted because they tend to non-determinism problems. They are fragile, expensive to write, and very time consuming to run at the same time. Indeed what the pyramid tells is trying to say is that it is important to focus much more on automated testing through unit tests than GUI based testing.

Integration tests are harder and complicate to maintain process compared to unit tests but still, are far easier and manageable than Acceptance test. Integration can be something like UI tests, where mocked backend and other service dependencies are allowed.

At the bottom, the final layer is the one is the most fine-grained, extremely fast to implement, easy to maintain, and run very quickly. For that reason, unit tests are super powerful in Test-driven development, which will be explored more in the later section.

3.2 Common component

3.2.1 Definition

Testing is an important part of development process. However knowing what and how to develop and organize software is important too, especially in large software development project. The best strategy is striving to develop common reusable components that can be used later in the project. Talking about common software component, first, we need to know a bit about Component-based software engineering (CBSE), or component-based development (CBD). It is a branch of software engineering that concentrates on the separation of concerns principle. The idea of a component is splitting the source code into smaller reusable chunks of code. Doing that way, not only developers can reduce complexity by limit their focus into a smaller area at a time but also they can focus on developing and delivering solutions that are reusable and easy to maintain.

"An individual software component is a software package, a web service, a web resource, or a module that encapsulates a set of related functions (or data).

All system processes are placed into separate components so that all of the data and functions inside each component are semantically related (just as with the contents of classes). Because of this principle, it is often said that components are modular and cohesive." (Wikipedia: Component-based software engineering)

The idea of the component is all about splitting down systems into smaller pieces which can benefit us since it brings more unity between design and development. The next section will say more about why this pattern is beneficial not only for developers but also for business as it will help later when it comes to maintenance and further development.

3.2.2 Benefits of common component

Regardless of front or backend development, develop software the common component way provides some clear major benefits:

- **Consistency:** implementing reusable components helps keep the consistency in software design and make code much more clear and organized compared to non-reusable ones.

- **Maintainability:** common components increase code maintainability hugely. Whenever a piece of common component's functionalities needed to be refactored or updated, it will be applied wherever the component is used in the application.
- **Scalability:** Having a library of components can clearly speed up the development process.
- As the project scales, making sure namespacing the components correctly will help to avoid styles and functionality leaking into the wrong place.

3.3 Web component

Front-end web development is an area that develops extremely fast. When the web application grows, it brings more and more complexities and challenges for developers to upgrade and maintain. Therefore writing code in a modularized manner is very important both for the technical and business point of view since it will drive cost down when it comes to maintenance and further development. If one has been working on a large scale web application project, he may have already created components of some kind from Angular, React, Ember etc. even though each one "thinks" of components in its own way. If in the future, there needs to be a change in framework, it would be extremely costly and unwise to rewrite our application from scratch. Instead, it would be nice if what developers build are component-based, using the same native core technologies, and reusable regardless of frameworks the web application will depend on. Web components, therefore, make this possible. It also offers the ability for the component to interact and communicate to each other under the same application. It is a collection of technologies that enable developers to efficiently define the current existing HTML implementation (Developing Web Components: IO from JQuery to Polymer). Regardless of frameworks, one common rule though is if the component's name has more than 1 word then it must have a dash (-). This is important because there needs to be a way to distinguish custom HTML component from standard ones and prevent conflicts in naming.

Having a native way to build component is the holy grail of web application development since it helps us to achieve the following:

- **Composability:** ability to reuse and group components together
- **Encapsulation:** capability to group markup and style into isolated group
- **Reusability:** ability to reuse components across applications and ease of functionality extension

Even though web components are men to be a standard of web development and have a potential to be a major shift in how developer develop the web, at the time of this writing, web component is still not widely supported. Only Chrome browser supports the components specifications. However, as said before, in web development, we can think of web common component as strategies that are spread in various places from CSS methodologies to JavaScript frameworks, design patterns and style guides. Developing in the component way is similar to playing Lego. Common components can be thought as building blocks that can be combined to form a larger, more robust structure. Those components can be detached, reattached, rearranged and combined in multi different ways. Whether one is writing code in JavaScript or adjusting some SASS/CSS styles there are common patterns of what a component is. Those patterns are independent, clearly defined, encapsulated and reusable. In the scope of this thesis, later we will analyze an example of how a non-UI web component is developed from start to end using test-driven development.

3.4 Repository

In a context of software development, a repository is a location where software packages and module are stored and can be retrieved or installed on a computer. There are many online services that are used for hosting repositories such as Bitbucket, GitLab, Stash, Github etc.

3.5 Software development good practices

Software best practice is a technique that has been widely approved or agreed as a superior solution as it brings many benefit as well as produces results that are better than other alternative methods. It has become a standard that all the developers should know and following. This section focus on some of the best practices that were used in the project implementation of this thesis.

3.5.1 Code refactoring

Code refactoring is the technique of restructuring current computer source code in order to enhance maintainability and the software's nonfunctional attributes without alter its external behavior. Other advantages are code readability enhancement and complexity reduction.

3.5.2 Design pattern

Software design pattern is a reusable solution in software engineering to tackle popular recurring issues within a certain software design context. Design pattern is not a complete design that can be converted directly into code that the computer can execute. It, however, plays a role as a template for how to resolve problems that can be utilized in various situations and contexts. It is amongst the best practices that developer can use when it comes to design software application.

3.5.3 Unit testing

Unit testing is a method to test software from an units level of the program source code, or arrays of one or more modules. The purpose is to decide whether the source codes are appropriate for use. A unit can be considered as the smallest part of a program. In procedural programming, a unit could be an entire module or an individual function. In OOP, a unit can be an interface, a class, but can also be an single method. Each test case should be independent of others. To achieve testing module in isolation, which is something makes unit test stands out from UI test or integration test, substitute such as method stubs, mock objects, fakes, and test harnesses are used.

Since the goal of the unit test is to test the code in isolation and validate its correctness, it can be done in an automated fashion or manually. Regardless of how it is performed, the unit test helps to find issues early in the development cycle. It also helps developers in refactoring the source code or further development. Doing correctly, the unit test also serves as a living documentation and a design of the application.

Unit test, however, has its own drawbacks too. It will not guarantee catching every issue or defect of the program. It can not be a replacement for integration testing. Writing unit test can take time especially when it comes to an area of source code where it has to rely on a Boolean decision, nondeterministic problems or asynchronous issues. The unit test code is likely to be buggy as the testing code itself. In this case, unit test could be skipped somehow. In *The Mythical Man-Month*, Fred Brooks wrote that: "Never go to sea with two chronometers; take one or three." Meaning, if two chronometers contradict, it would give you more headaches.

3.6 Tools

During the development implementation process, several applications and tools are used to create, build, and publish the software component. This section represents what are these tools, and why they were chosen i.e over others.

3.6.1 Webstorm

WebStorm is a delicate, agile yet robust JavaScript integrated development environment (IDE), which is perfectly supplied for client-side development and complex server-side devel-

oment with Node.js. It is branded as The Smartest JavaScript IDE by JetBrains, a software development company whose tools are targeted towards software developers and project managers.

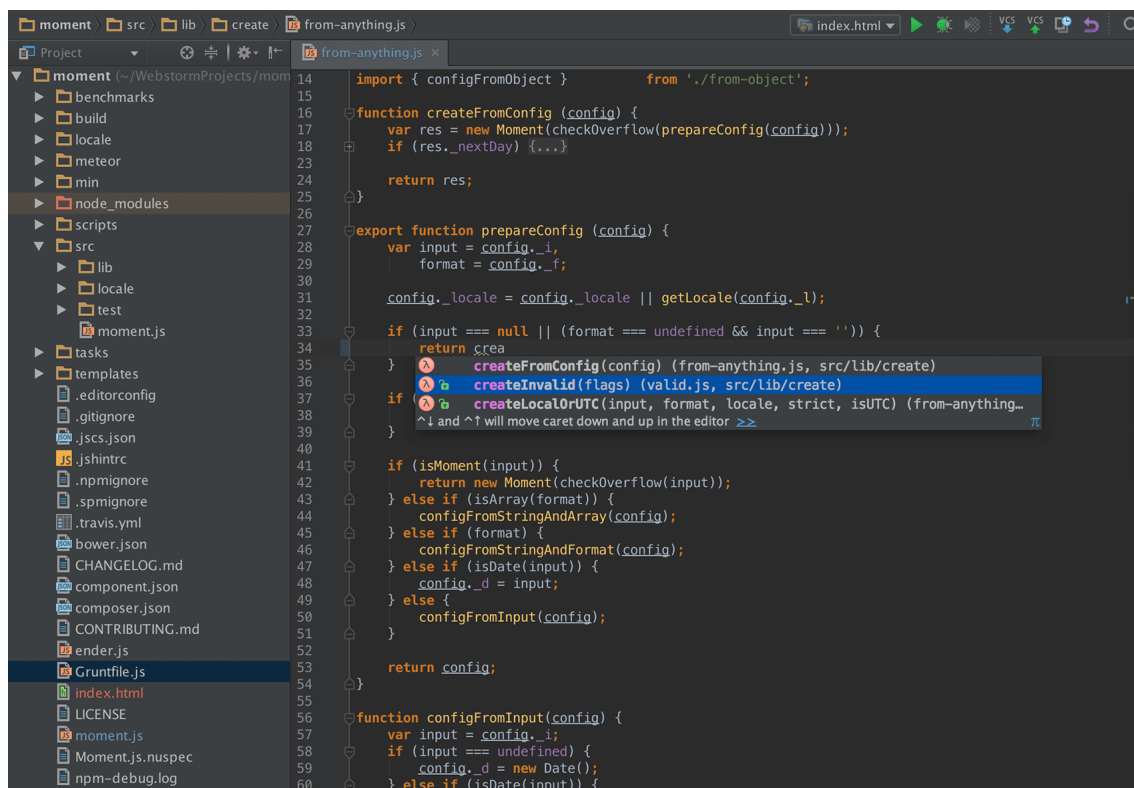


Figure 10: Webstorm UI

The tool is selected due to its specialized capabilities for front end development using JavaScript, as well as other front end technologies. It has a lean UI which creates a convenient workflow for JavaScript developers. It also has a built-in terminal environment where command lines can be input and executed, which will greatly help developers during the development process where they need to run tests, and build the source code in real time.

3.6.2 Git

Git is a version control system (VCS) for tracking modifications in computer files and coordinating work on those files among several people. It is mainly used for software development, but it can also be used to monitor alterations in any files.

Git was chosen due to its popularity and ease of use. Git is well-documented, available online and the syntax is familiar to all the developers in the team.

3.6.3 Github

GitHub is a web-based Git or VCS repository as well as online hosting service. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features. It supplies several collaborations and management features such as bug tracking, feature requests, task management, and wikis for every project. On the same account, GitHub provides both plans for private and free of charge repositories which are commonly used to host software projects.

The screenshot shows the GitHub repository page for 'moment-calendar-2' by user 'vinhnghi223'. The repository is described as 'A super lightweight, easy-to-use yet powerful Node module to create month calendar where each date is a momentJS date.' It has 19 commits, 2 branches, 4 releases, 1 contributor, and is licensed under MIT. The commit history table is as follows:

Commit	Description	Time Ago
coverages/Chrome 56.0.2924 (Mac OS...	Fixed test coverage	19 days ago
demo	Updated readme	7 months ago
dist	Updated readme	7 months ago
src	Added changelog and update specs for getInstance	3 months ago
.gitignore	Added keywords	7 months ago
.jshinttrc	Init commit	7 months ago
.travis.yml	Init commit	7 months ago
CHANGELOG.md	Updated CHANGELOG	3 months ago
LICENSE	Init commit	7 months ago
README.md	Release v0.0.3	7 months ago
gulpfile.js	Finalized	7 months ago
karma.conf.js	Fixed test coverage	19 days ago
package.json	Fixed test coverage	19 days ago
README.md		

Figure 11: Github UI

Github was chosen due to its popularity and public availability, good web-based UI and cost-effective for the open-source project. It also has a nice graph UI as well as other useful features that help developers keep track of the project development history in an easy manner.

3.6.4 NPM

Npm is written completely in JavaScript and was developed by Isaac Z. Schlueter as a result of having "seen module packaging done terribly" and with inspiration from the limitations of other identical projects such as PHP (PEAR). It is the default package manager for the JavaScript runtime environment Node.js. Unlike Bower, which is also another dependency management tool, npm can also be used for installing Node.js module, whereas Bower is only used for managing front-end components such as HTML, CSS, etc.

Neural Prediction Model npm Enterprise features pricing documentation support

npm find packages sign up or log in

★ moment-calendar-2 public

npm package 0.0.4

A super lightweight, easy-to-use yet powerful Node module to create month calendar where each date is a momentJS date. You can leverage further with date operations as well as configuring localization through MomentJS.

You can use this component with other frameworks and libraries with ease. This [Demo](#) demonstrates how to use moment-calendar-2 together with angular to build a date picker component.

Getting Started

NPM install moment-calendar-2

```
npm install moment-calendar-2
```

Notes that this is a stand-alone version. moment-calendar depends on momentJS, so make sure you have also include that in your app.

Include the script in your html:

```
<script src="node_modules/moment-calendar-2/dist/moment-calendar-2.js">
```

You can also include 'moment-calendar-2' by using 'require':

```
var MomentCalendarFactory = require('./node_modules/moment-calendar-2');
```

In your JS, use 'moment-calendar' like so:

Manage permissions for the whole team

Manage developer teams with varying permissions and multiple projects. [Learn more about Private Packages and Organizations...](#)

npm 1 moment-calendar-2 [how? learn more](#)

★ [vinhngchi223](#) published 7 months ago

0.0.4 is the latest of 3 releases

github.com/vinhngchi223/moment-calendar-2

MIT

Collaborators [list](#)

★

Stats

0 downloads in the last day

6 downloads in the last week

45 downloads in the last month

No open issues on GitHub

Figure 12: NPM UI

NPM was chosen due to its popularity as a platform where a developer can publish their JavaScript components/libraries. Moment-calendar-2 is a Node Js component and does not require any UI part. This makes it fit perfectly with NPM. Besides that, NPM also has its own tracking system where it keeps track of how many components a developer has published as well as how many users downloads the component gets.

3.6.5 Gulp

Gulp is a toolkit for the automated build system. Leveraged by node's streams file manipulation, gulp helps developer streamline their tedious, time-consuming build process in their development workflow. Build tools such as GNU Make is not something new to developers, however, tasks runners that are made for web-specific are still quite new experience. Grunt and Gulp are two most popular task runners/Node's automated build tool nowadays.

Unlike Grunt, Gulp does not require a handful of plugins to achieve basic functionalities such as file watching. Even a simple grunt task needs to be configured in a sophisticated manner whereas gulp syntax is straight forward, and less verbose. It leverages the use of NodeJs's stream to pass data through a series of piped plugins and, hence, processes data much faster.

3.6.6 Browserify

browserify is a development tool that allows developers to write node-flavored CommonJs modules that compile for use in the browser. It can be used to organize source code and third-party libraries even if these code is not written in the node way. To do that, browserify recursively analyzes all the require() calls in the application in order to build a bundle that can be served up to the browser in a single <script> tag.

Since moment-calendar-2 is developed using NodeJs, which is a server-side JavaScript environment, it needs a way to make it available for the web front end if necessary. For example, in our demo application's case, there is a demonstration of a Date picker UI component. The dDate picker in the demo application is an AngularJs directive which is dependent on moment-calendar-2. Therefore, moment-calendar-2 needs to be loaded first before Date picker when the browser is opened. Browserify is a perfect tool for just that. It can also be easily integrated with Grunt and Gulp.

3.6.7 browserify-istanbul

One of the most powerful features of Browserify is source transform. It is the capability that allows a stream to be injected between the module source code and the final returned content. For example, if the source code is written in CoffeeScript rather than JavaScript, the developer usually needs first to convert CoffeeScript to JavaScript before continuing with bundling. This phase is called precompilation phase. However, with source transform, there is no longer need for this step of the process.

Browserify-istanbul is a source transform for the istanbul code coverage tool. It helps Browserify to generate test coverage from the source code based on its unit tests.

3.7 Programming languages and frameworks

In this section, the programming languages and frameworks used in the project are described. Firstly, JavaScript and its libraries, MomentJs, NodeJs, AngularJs are discussed. Followed that, the thesis walks through what makes the demo application looks visually attractive by explaining about the use of CSS and a premade template for the calendar.

3.7.1 JavaScript

JavaScript is a high-level, dynamic, untyped, and interpreted programming language. When it is applied to an HTML page, it can bring interactivity and functionality on the web. Together with HTML and CSS, JavaScript is one of the most important technologies of the world wide web. It has been standardized in the ECMAScript language specification as well as being supported by all modern web browsers without the help of any plugin.

Due to its importance and necessity in the web, JavaScript is the language used in this thesis for the implementation of web common component.

3.7.2 MomentJs

MomentJs is a JavaScript library for parsing, validating, manipulating, and formatting dates. It was designed to work both in the browser and in Node.js. MomentJs can be used as following:

In NodeJs:

```
npm install moment
var moment = require('moment');
moment().format();
```

On the browser:

```
<script src="moment.js"></script>
<script>
  moment().format();
</script>
```

In the above script, the variable `moment` holds the moment object instance, where its API can be accessed. MomentJs can be used in a different way, with different front-end toolkits/plugins such as Require.js, Browserify, Typescript, etc.

MomentJs plays an important role in the project implementation since all date, time, validation, formatting logics are operated thanks to it.

3.7.3 Lodash

Lodash is a modern JavaScript library that contains a bundle of utility functions for processing, operating calculations on arrays, objects etc.

Together with MomentJs, Lodash facilitates all data processing and calculation operations of `moment-calendar-2`.

3.7.4 NodeJs

Node.js is an open-source, cross-platform JavaScript runtime environment for developing a multiple varieties of server tools and applications. Many of NodeJs fundamental modules are written in JavaScript, even though it is not a JavaScript framework. By using Google's V8 JavaScript engine, the runtime environment interprets JavaScript and, hence, developers can continue to write new modules in JavaScript.

NodeJs is used in both a develop and build phase of the project. The project final output is a NodeJs module which is packaged and published to NPM (Node Package Management)

3.7.5 AngularJs

AngularJS is one of the most popular web frontend frameworks at the time of this writing. It lets developer extend the use of native HTML by the use of Angular directives and provided ways to make a dynamic web application with ease using two ways binding.

AngularJS directive is a forerunner of Web Components. By default, native HTML elements are quite primitive, with the same set of properties that applied to it. It can not keep up with the rate of changes and demand of a dynamic web application development easily. Using AngularJS directive feature, I quickly made an example calendar component to serve the purpose of demonstrating how quick and easy `moment-calendar-2` can help build a date/time-related UI component as well as how easy the module can be integrated with other front-end frameworks.

3.7.6 CSS

Cascading Style Sheets (CSS) is a stylesheet language used for describing the presentation of a document written in a markup language.

Just like JavaScript, CSS is, nowadays, a technology that a web page can not live without. Even though `moment-calendar-2` is a non-UI, and purely functional component, CSS adds styles and defines page's elements positions to the demo application using Date picker, a component that is built based on `moment-calendar-2`.

3.7.7 Little Calendar CSS

Little Calendar is an open-source project, a premade HTML template with predefined CSS style sheets used for Date picker component by CSSFlow.

Like AngularJs, this premade template is used to quickly construct a nice-looking demo application of Date picker showing how quick and simple it can be to use `moment-calendar-2` in order to build a complex UI component like Date picker.

4 Project implementation

This section describes the project implementation of the aforementioned theoretical concepts and technologies to create moment-calendar-2, a common component that Date picker component to consume. It first briefly describes how Date picker is designed from U/UX design phase using Atomic design. It then demonstrates how moment-calendar-2, which is the non-UI part of Date picker component, is developed, step by step, using TDD. In addition to that, it also covers how the component is documented, published and released to the market so that other developers can join hands to further enhance its features and enjoy using the component.

4.1 Atomic design of Date picker common component

In the context of banking application, Date picker is one of the most (if not the most) complicated component yet very important throughout the entire application.

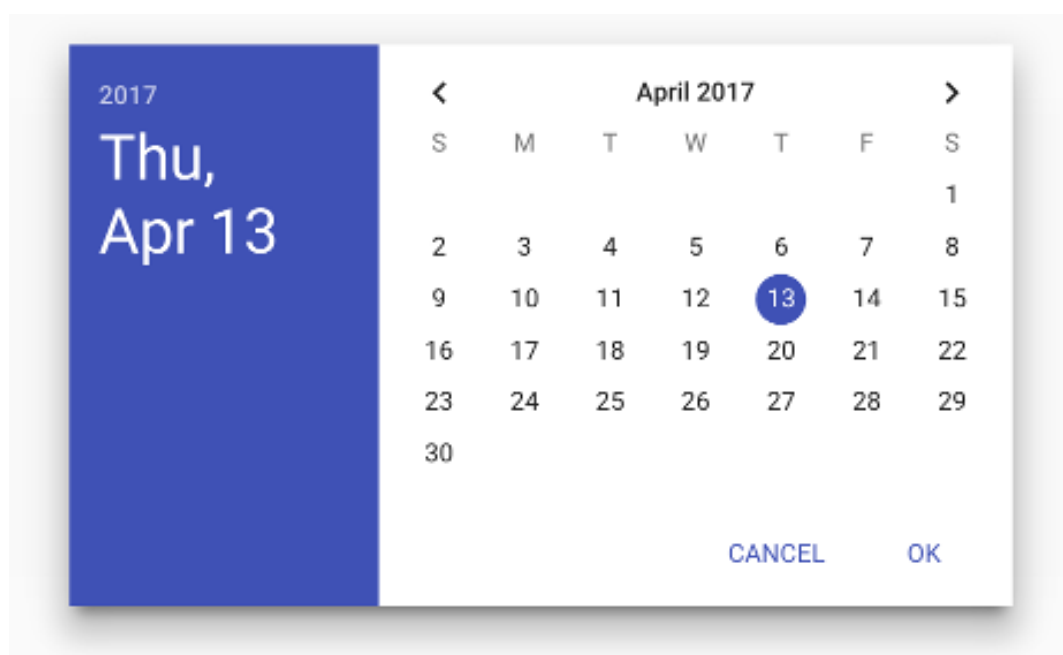


Figure 13: A typical UI of Date picker

One of the reason is Date picker can comes in with different shapes and functionalities. A Date picker can be used as it is, but can also used with other components like dropdown list or input field (where user can type in a date so that the date is selected on the Date picker and vice versa).

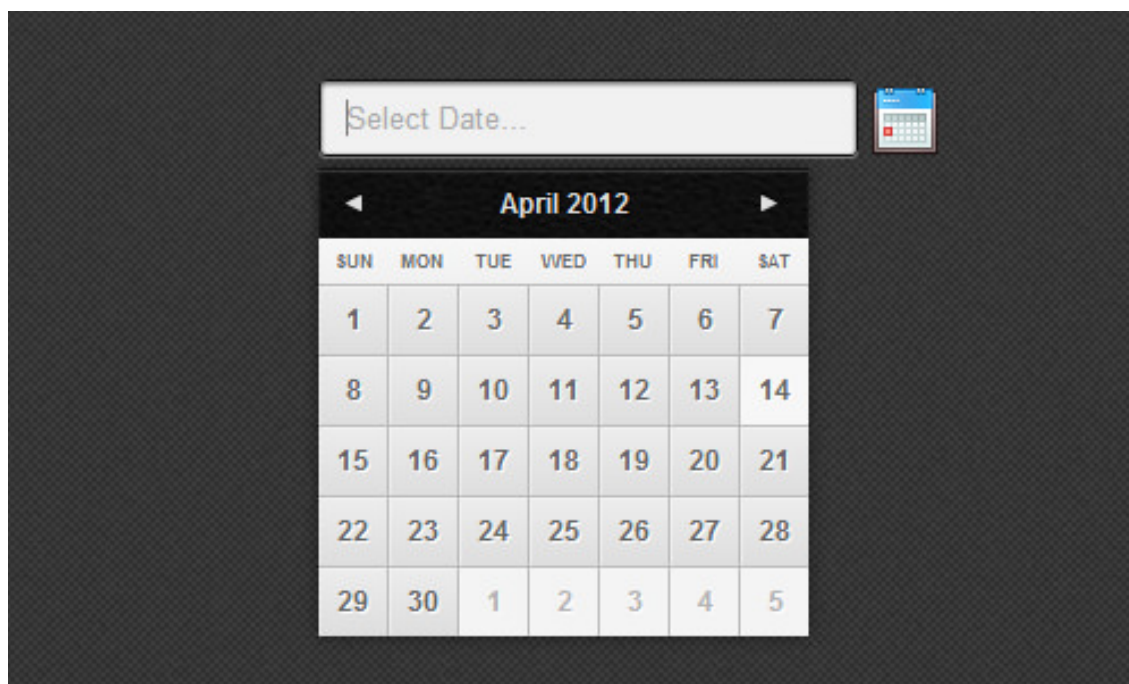


Figure 14: Date picker with input field

A Date picker can also trigger backend request calls that fetch dates from the server. One use case is a Date picker component locating in book meeting area where it executes non banking dates service call every time user navigates to another month.

Another use case of Date picker is when user needs to filter out a list of transactions in a certain date range. Typically in this case there will either be a date range picker or two Date pickers, one for from-date and another for to-date.

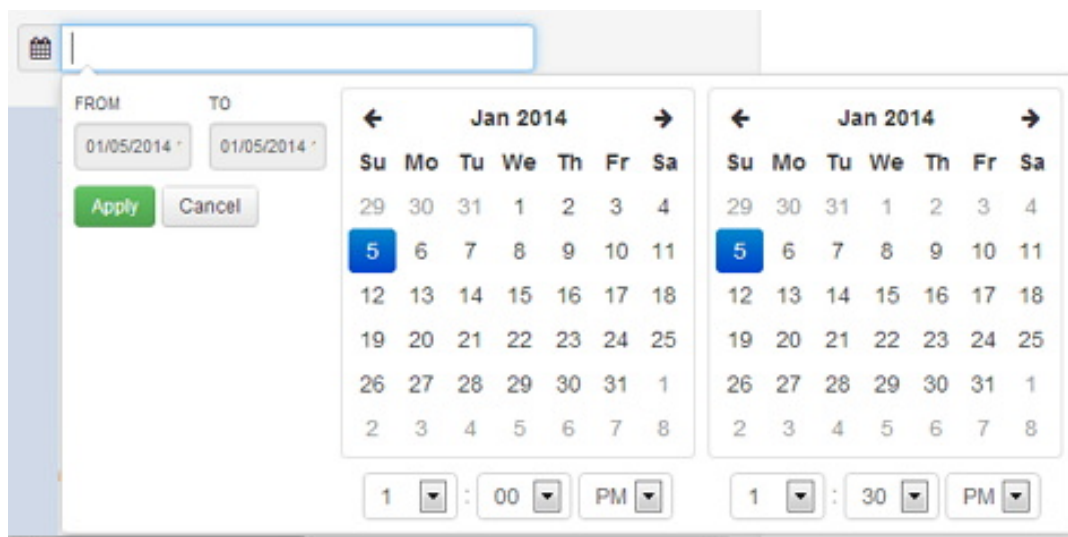


Figure 15: Date range picker

In form such as payment or issue report, Date picker is the must. This is where user must pick a date, indicating the date that the issues are found, or, in the case of payment, Date picker is used to specify the due date.

Date picker also needs to be available in different languages and localization. This is not only because the week date names are different from language to language but also the order of weekdates can be a factor that changes the calendar layout.

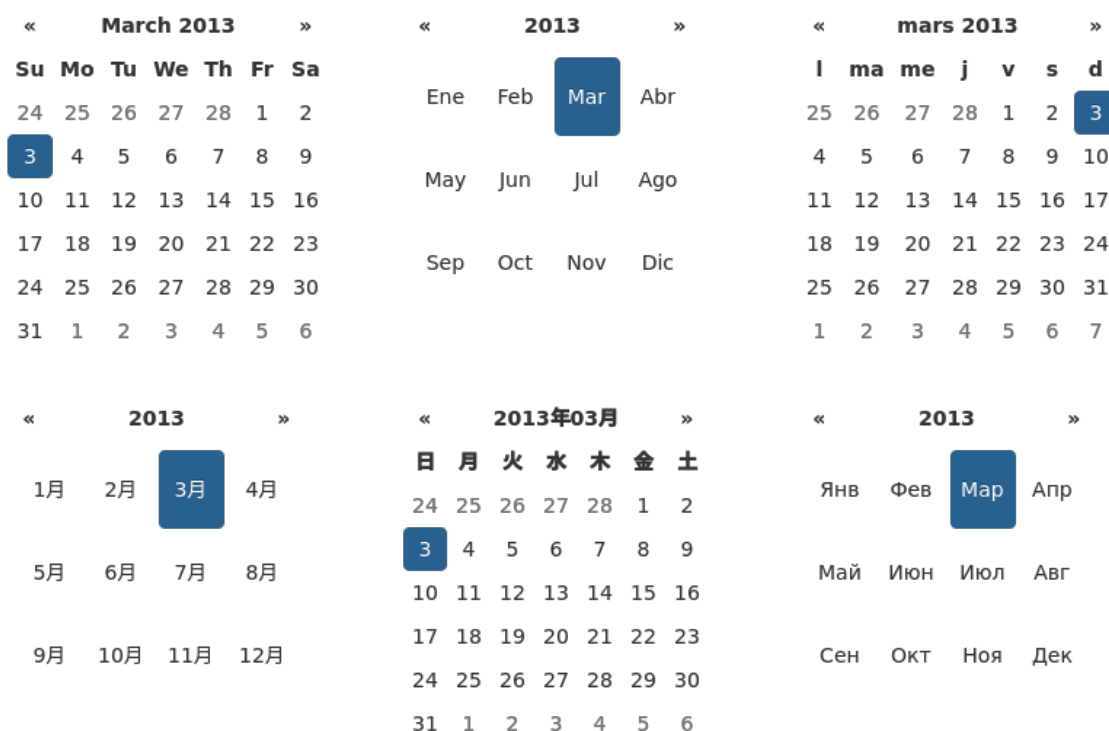


Figure 16: Date picker with different localizations

In addition to that, Date picker should also be flexible in colors based on theme. If the whole page is in dark theme, the Date picker should "know" how to pick an appropriate color scheme to make it stand out from the background. Some Date picker behaves a bit different from others even if they all look the same from UI perspective. For example, sometimes, user only needs to select a date and right after that Date picker will be automatically closed. Another time, user needs to close the Date picker explicitly. Date picker also needs to support screen reader device, keyboard navigations, and should have some interesting animations as well.

The design of Date picker should take all those criteria into account. Using Atomic design the Date picker can be viewed as an organism where it comprises of mocules components.

From an UI perspective, these mocules are Date picker's opener icon, header, calendar view (month table), confirmation buttons, input area etc.

Each of these mocules will be the composition of atomic elements like navigation button icons, labels, texts, buttons, input field etc.

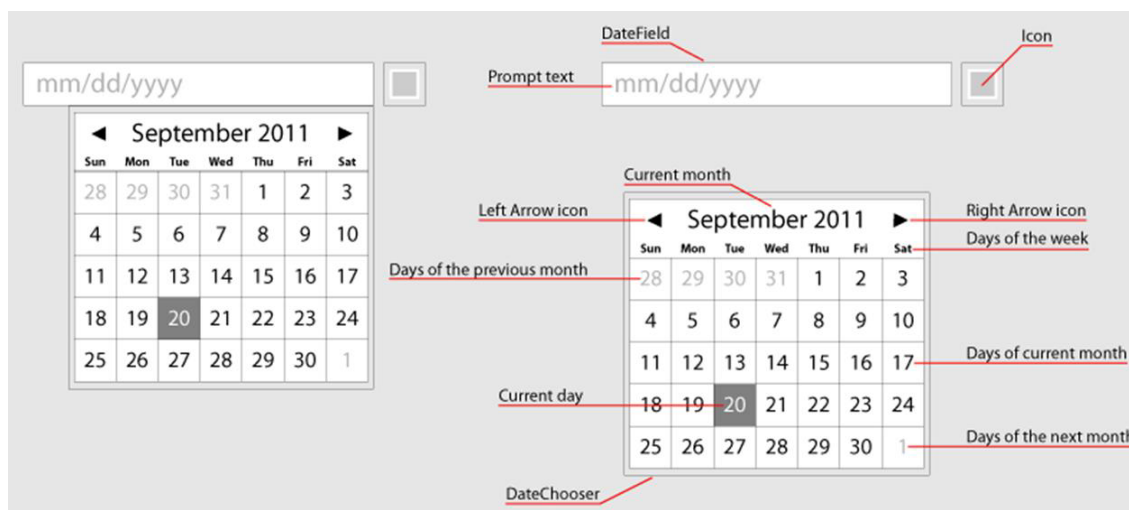


Figure 17: Date picker design with atomic design

Based on these knowledge, UI and UX designers started working on the UI of Date picker and provided several scenarios such as how Date picker looks like when it is open, when it is closed, when it has disabled dates, or when the user start to interact on the calendar UI etc.

After that, I as a developer started to work on how the Date picker is built. From a functional perspective, Date picker is dependent on moment-calendar-2, the core engine that calculates dates and months operations, date formatter, date validators etc. Along the way of this development process, I also carried out informal queries session wit UX/UI designers and product owners to learn more about different business scenarios that require the use of Date picker as well as how Date picker should behave in each of those cases. In the next section of the project implementation, the thesis will go deeper on how moment-calendar-2, the common that is employed by the aforementioned Date picker.

4.2 Moment-calendar-2 component

Date and time are complex concepts to deal with in software programming, yet they are very crucial and sensitive in a banking application. Due to its overly huge complexity, and for security and confidential reason, this thesis will not be about Date picker but focus more on Date picker's sub-common component which is moment-calendar-2. As introduced in the first section, for a quick recap, the name moment-calendar-2 is chosen because it describes how calendar date logics and values are calculated based on, and since there is another component named moment-date, '2' is just a version number added to that.

Even though moment-calendar-2 is a much smaller component and does not require an UI at all, it plays a role as the brain of the Date picker component. Without it, Date picker can be just simply a good looking template with static text strings, which are primitive and quite useless from a user perspective. Moreover, being a common component means that moment-calendar-2 can be used else where then just for Date picker. It can be used to build more complicated date and time-related UI component. It can also be used together with other software stacks and frameworks such as NodeJs, JQuery, AngularJS or just simply plain old JavaScript.

4.3 Moment-calendar-2 requirements

As described above, moment-calendar-2 is a purely functional component and does not require an UI at all. The target customers, or to be exact, the consumers of this component are developers who want to build date and time-related software programs/components (such as Date picker). With that in mind, here is a list of general moment-calendar-2 requirements:

- The developer needs a common component where they can reuse to build date and time-related software components/programs.
- The developer should be able to use its API to achieve the following:
- The developer can create a calendar object which can return the current date (today).
- The developer can create a calendar object which can set the current date (today). For example, he can set the value of today to a totally different date as he wishes.
- The developer can create a calendar object which can update a date object with its new information, i.e developer can assign a value and also later update that value to a specific date. For example, the developer can assign 8th of March 2017 a value of "International women day".
- The developer can retrieve the value of a specific date. In the example above, the developer can retrieve "International women day" from date 8th of March 2017.
- The developer can get a weeks table where each row will represent a week in that current month. The current month is calculated based on the current date that the developer can get and set as stated above. Each row cell is a specific moment date object.
- The developer can get all the dates in a certain month. For example, the developer can get an array of date objects representing 1 to 28 from February 2017.
- The developer can get previous month overlapping dates. Overlapping dates are understood as dates that are in the same week with other current month dates. For example, if the current month is March 2017, then previous overlapping dates will be 27th and 28th February 2017.
- The developer can get next month overlapping dates. Same as above except that now they are the next following month instead of previous one. For example, in the case of March 2017, those would be 1st and 2nd April 2017.

After a draft estimation, a plan has been scheduled on taking into account how the practical implementation of the component should be done.

4.4 Implementation plan

Considering the fact that this is a complicated component, the author divided the development process into the following main phases:

- Set up repository and build system
- API Design and structure
- Implementation using TDD
- Test coverage
- Documentations and release

4.5 Implementation process

4.5.1 Set up repository and build system

A repository can be created by just simply click on the New repository option under Github top menu:

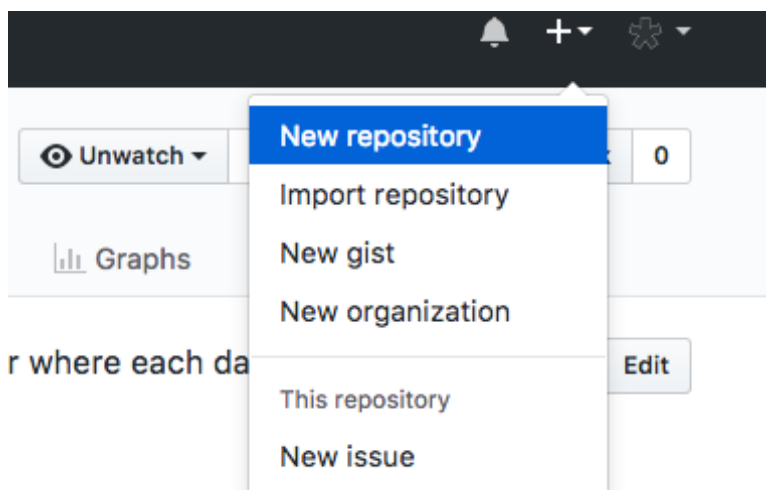


Figure 18: Create a new repository

Follows along the instructions on Github, a new blank repository is created.

4.5.2 Build system

The very first step is to install dependencies. Usually, when the project starts, developers start to install a few dependencies first. Since this project will later be packaged and published to NPM, it is wise to install a few dependencies first such as gulp plugins (which will be explained later). Later on, in the development phase, developers can decide on which dependencies he needs to install or whether he wants to install more dependencies or not. Below is the list of dependencies in this project's package.json file where all the dependencies for this project are listed:

Development dependencies:

<pre>"angular": "^1.5.8", "angular-mocks": "^1.5.8", "browserify": "^13.1.0", "browserify-istanbul": "^2.0.0", "del": "^2.2.2", "gulp": "^3.9.1", "gulp-concat": "^2.6.0", "gulp-load-plugins": "^1.2.2", "gulp-ng-annotate": "^2.0.0", "gulp-release-easy": "^1.0.5", "gulp-rename": "^1.2.2", "gulp-sass": "^2.3.2", "gulp-sequence": "^0.4.5", "gulp-uglify": "^2.0.0", "gulp-webserver": "^0.9.1",</pre>	<pre>"gulp-wrap": "^0.11.0", "http-server": "^0.9.0", "istanbul": "^0.4.5", "jasmine-core": "^2.4.1", "karma": "^1.5.0", "karma-browserify": "^5.1.0", "karma-chrome-launcher": "^0.2.3", "karma-coverage": "^1.1.1", "karma-jasmine": "^0.3.8", "karma-requirejs": "^1.0.0", "requirejs": "^2.2.0", "vinyl-source-stream": "^1.1.0", "vinyl-transform": "^1.0.0", "watchify": "^3.7.0"</pre>
--	---

Dependencies:

```
"moment": "^2.14.1"
```

The number on the left of each dependency name is the dependency's version being used at the time of this writing. There are two types of dependencies. The first one is dependencies that are used for the development of the project. It includes dependencies for build process and As can be seen in the list above, while the development dependencies list is quite extensive, there is only one single dependency that this component relies on, which is MomentJs. This is actually a very good thing because it will keep the final built package lightweight. Also

the fewer real dependencies we have, the less maintenance and upgrade follow-up needs to be done later.

The next step is to implement a build system. The two most popular build system for front-end development at the time of this writing are Grunt and Gulp. Gulp is the chosen option for this project due to its simplicity, and functional way of programming.

All the gulp tasks will be put inside one file named gulp file, which can be viewed in appendix 4.

Here the build process is exposed through a set of tasks and subtasks. The most used task is the 'default' task, which you can run by just simply issuing 'gulp' in your command prompt or terminal window:

```
gulp.task('server', ['build'], function () {
  return gulp.src(['demo', 'node_modules', 'dist'])
    .pipe($.webserver({
      port: 8080,
      livereload: true,
      open: true
    }));
});

gulp.task('default', ['server'], function () {
  gulp.watch('src/**/*.js', ['browserify']);
});
```

Figure 19: Basic gulp tasks

This code snippet describes what happens when you run the default task. The task will trigger two preceding tasks, one is 'server, and another is 'watch'. A description of some of these mentioned tasks are listed in the table below:

Task name	Definitions
Default	Run this task to start "server" and then "watch" task.
Watch	Run this task to make gulp "keeps an eye on" source files changes and triggers web browsers to reload/rerender the pages so that changes are reflected and developers can see them.
Server	Run this task to initiate the local web server at a certain port. This way developer can enter local host address with its port number to the web browser to access the demo application.

Table 1: gulp tasks definitions

There are some other gulp sub tasks, but those tasks above are the most important ones that one should learn in case one wants to start contributing the project.

Since later it is super important to run tests and generate coverage reports, it is also good to refine those configurations in this stage. This project will use karma to launch a web server which can then load our project resources for testing, building our source code and showing our demo application. Its main goal is to create a productive testing environment for developers. Karma runs based on what it is configured. Those configurations are set in karma config file. Without this file, or if been set improperly, then there may be problems later with

steps like testing, building, coverage, continuous integration, etc. Below figure how the project is configured.

```

1 //jshint strict: false
2 module.exports = function(config) {
3   config.set({
4     basePath: '.',
5     autoWatch: true,
6     frameworks: ['jasmine', 'browserify'],
7     files: [
8       'node_modules/moment/moment.js',
9       'src/*.js'
10    ],
11    browsers: ['Chrome'],
12    reporters: ['progress', 'coverage'],
13    preprocessors: {
14      'src/*.js': ['browserify']
15    },
16    browserify: {
17      debug: true,
18      transform: ['browserify-istanbul']
19    },
20    coverageReporter: {
21      reporters: [
22        {"type": "text"},
23        {"type": "html", dir: 'coverages'}
24      ]
25    },
26    singleRun: true,
27    plugins: [
28      'karma-coverage',
29      'karma-browserify',
30      'karma-chrome-launcher',
31      'karma-jasmine'
32    ]
33  });
34 };

```

Figure 20: Karma configuration file

Since the component in this thesis is a non-UI one, so far most of the configuration here is testing related purposes. In a more involved project/component, it may require also some configuration on how to bundle styles files (SASS/CSS files), inject environment variables, login credentials, security level and so on.

4.5.3 API Design and structure

Even though in this thesis, the requirements (design and structure) and unit tests are presented separately but in reality, both should be developed at the same time.

Since the component has been developed and ready to be used at the time of this writing, it is easier to just navigate to its repository and see how the structure was defined. From the project's github page, click on 'src', a list of component files will be shown. The component is divided into files/modules:

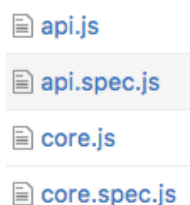


Figure 21: moment-calendar-2 sub modules

As can be seen in the screenshot. The structure of the component is minimal. It does not contain any UI related files (HTML templates and/or Stylesheets files) and is purely functional. There are two sub-components: one is API and another is core. The suffix 'spec' simply means that these files are unit tests or, in other words, specifications files.

API file plays a role as the public interface of the component, where it exposes all of its methods for the outsider to use. Core file in the other hand acts as a helper class where all the intricate internal logics are implemented. It is included by the API file by simply using Node require function like this:

```
var core = require("./core.js");
```

Figure 22: How another module is included

This code simply means a module named 'core' from file core.js will be assigned to a variable name core which will later be used in the API module. There is one public method at the end of the component which is called 'getInstance'.

```
// Public
function getInstance(date) {
  var monthCalendarInstance = Object.create({
    setCurrentDate: setCurrentDate,
    getCurrentDate: getCurrentDate,
    updateDate: updateDate,
    getDate: getDate,
    getWeeksTable: getWeeksTable,
    getDatePosition: getDatePosition
  });
  monthCalendarInstance.setCurrentDate(date);
  return monthCalendarInstance;
}

module.exports = {
  getInstance: getInstance
};
```

Figure 23: getInstance factory method

The idea of a getInstance function is it will create and return an instance object of Moment-Calendar where all of its API is exposed. This snippet below shows how it is used:

```
var calendar = MomentCalendarFactory.getInstance();
calendar.getWeeksTable();
```

Below is a brief description of how each of the methods is used:

Methods	Arguments	Description
setCurrentDate	a date (eg. '2017-01-01')	Set the current date for month calendar. If the argument is left blank or is an invalid date then current time being will be used.
getCurrentMonthDate		Get the current date of month calendar. Return will be a moment object.
getDate	a number or date (eg. '1', '2017-01-01' etc.)	Get a moment object corresponding to the date you want to get from month calendar.
updateDate	(date, obj) (eg. ('2017-01-01', {isHoliday: true}))	Extend a date object of the calendar with an object defined by your own. This is good when you need to attach a certain customized properties/methods that are not defined by MomentJs. If you need to override those properties, just simply pass another object with same property keys again. Later when you call getDate, you can access those properties from the returned object.
getWeeksTable	boolean (false by default if argument is missing)	Return an array of weeks table, the one that is similar to a month calendar view in Date picker (with the first element is an array of weekdays ('S', 'M', 'T' etc.)). Each date is a moment date. If you want a simplified version of weeks table where each date is an integer (eg. from 1 to 31), pass in true as the argument.
getDatePosition	a number or date (eg. '1', '2017-01-01' etc.)	Get back date position in the weeks' table mentioned above. The returned object has 2 properties 'row' and 'column' that indicate where the date is located in the weeks' table.

Table 2: monthCalendar instance's API

Later in phase three - unit tests, there will be a deeper look into these functions.

As mentioned before, the Core module plays a role as a helper class where all the intricated internal logics are implemented. Below are all the helper functions and properties that Core has to offers:

```

module.exports = {
  currentMonthDate: currentMonthDate,
  extend: extend,
  isValidDate: isValidDate,
  isDateInMonth: isDateInMonth,
  initWeeksDay: initWeeksDay,
  populateWeeksTable: populateWeeksTable,
  getDaysOfMonth: getDaysOfMonth,
  getPreviousMonthOverlappingDays: getPreviousMonthOverlappingDays,
  getNextMonthOverlappingDays: getNextMonthOverlappingDays
};

```

Figure 24: moment-calendar-2's interface

Below table will show brief descriptions of how each of the method/property is used (the only method will have arguments):

Methods	Arguments	Description
currentMonthDate	(not applicable)	Value for current month date. This is a momentJs object, therefore it can be formatted using ISO 8601 date string format, which is the standard uses the Gregorian calendar, serving as an international standard for civil use.
extend	Source object, destination object	Designate keyed properties of source objects to the destination object.
isValidDate	Date string, date format, isStrict-Mode (boolean)	Validate whether a date string is a valid date.
isDateInMonth	Number/date string	Validate whether a date/number is a valid month date.
initWeeksDate		Return an array of weeks days
populateWeeksTable	Month (in date string format, e.g '2017-06')	Create and return an array including a 2-dimensional table where there are weeks days as table header and other dates in that month.
getDaysOfMonth	Date/month (in date string format, e.g '2017-06')	Return an array of dates in that month.
getPreviousMonthOverlappingDays	Date/month (in date string format, e.g '2017-06')	Return an array of overlapping dates from previous month to current month.
getNextMonthOverlappingDays	Date/month (in date string format, e.g '2017-06')	Return an array of overlapping dates from next month to current month.

Table 3: moment calendar core's API

4.5.4 Implementation using TDD

A general idea of how the component is designed and structured has been introduced in the previous section. That would serve as API reference when needed. However, in reality, it would be more appropriate and convenient if design, implementations and unit tests are carried out all at the same time. In this section, it is time to take a few steps back and examine how a module in moment-calendar-2 should be implemented in reality.

Take api module for example. Since it plays a role as a factory which produces instances that expose the public interface of the application, there should be a way, a method whose name

is 'getInstance', to produce such an instance object. For that reason, some unit tests should be written like so:

```
var monthView;
beforeEach(function () {
  monthView = month.getInstance();
});

describe('getInstance', function () {
  it('should return a monthCalendar object', function () {
    var monthCalendar = month.getInstance();
    expect(monthCalendar.getCurrentDate()).toBeDefined();
  });

  it('should return a monthCalendar object where its current date is set by default', function () {
    var monthCalendar = month.getInstance();
    expect(monthCalendar.getCurrentDate().format('YYYY-MM-DD')).toEqual(moment().format('YYYY-MM-DD'));
  });

  it('should return a monthCalendar object where its current date is set manually', function () {
    var monthCalendar = month.getInstance('2019-09-19');
    expect(monthCalendar.getCurrentDate().format('YYYY-MM-DD')).toEqual(moment('2019-09-19').format('YYYY-MM-DD'));
  });
});
```

Figure 25: Implement unit test cases for getInstance method

The code inside beforeEach means that these lines of code will be executed in each test case. A test suite is named after a function in interested, in this case, getInstance. Each test suite following it will specify what the function does. In this case there are three features that developer needs to focus on first: getInstance should return a monthCalendar object where its current date is either set by default as today or set manually by developer. To verify whether the spawn object has its current date set properly, it should have a function named getCurrentDate where on execution should return a value of its internal current date. Current date can be today or a date set by developer. Because the mentioned component is built upon MomentJs, any date value will be encapsulated by a MomentJs wrapper, therefore, it should be first formatted/converted to a format of 'YYYY-MM-DD' for ease of comparison.

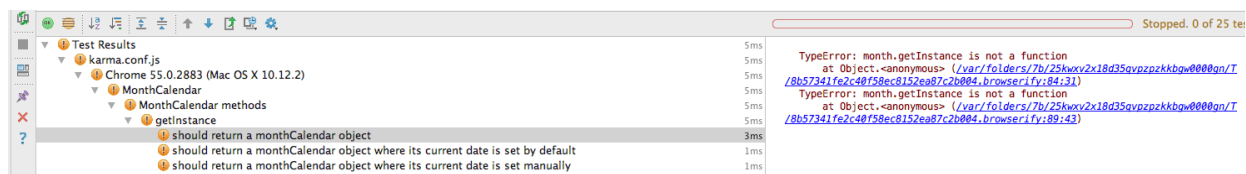


Figure 26: Implement unit test cases for getInstance method

The first time the test is run it will fail since there is no logic implementation at all. Base on the first error message that is logged on the developer tool, it clearly points out that the problem is getInstance is not a function. Switch back to the script the developer start writing the very first function like so:

```
module.exports = {
  getInstance: function(){
  }
};
```

Figure 27: getInstance method

Run the test again we see now that all the tests are still failing:

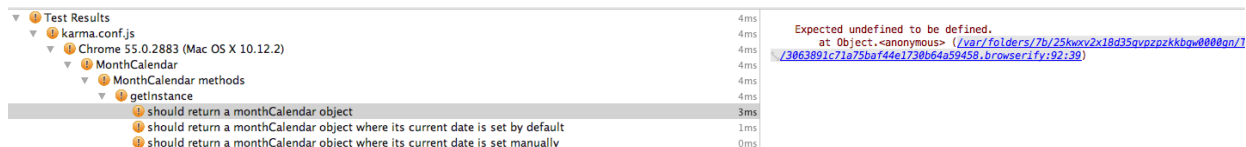


Figure 28: See the tests fail

The tests are failing but the error message is different hinting that the object returned by `getInstance` is undefined instead of defined (“Expected undefined to be defined”). To resolve that developer needs to rewrite the code so that `getInstance` actually return an object:

```
module.exports = {
  getInstance: function(){
    return {};
  }
};
```

Figure 29: getInstance return an object

Run the tests again and at this point it is observed that the error has been gone, instead another error is focused on at this time:

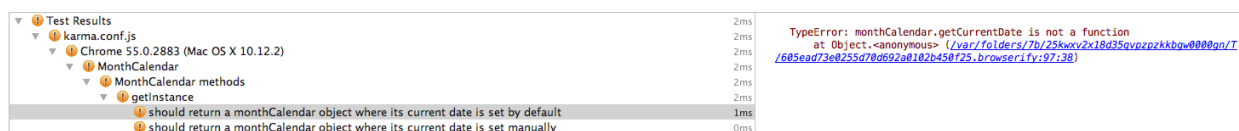


Figure 30: See tests fail again

Now moving forward, the developer focus on how to make `getCurrentDate` a function. If one still remember, this is exactly the same error message that he gets originally when first run the test suite. With that experience in mind, the developer keeps carrying on the implementation of the `getCurrentDate` in order to pass the specific unit test and continue forward to the rest of the whole tests suite. At the end of the day, here is the final version (of the time of this writing) of the `getInstance` function:

```
// Public
function getInstance(date) {
  var monthCalendarInstance = Object.create({
    setCurrentDate: setCurrentDate,
    getCurrentDate: getCurrentDate,
    updateDate: updateDate,
    getDate: getDate,
    getWeeksTable: getWeeksTable,
    getDatePosition: getDatePosition
  });
  monthCalendarInstance.setCurrentDate(date);
  return monthCalendarInstance;
}

module.exports = {
  getInstance: getInstance
};
```

Figure 31: Completed getInstance function

At this stage, all the tests should be passed:

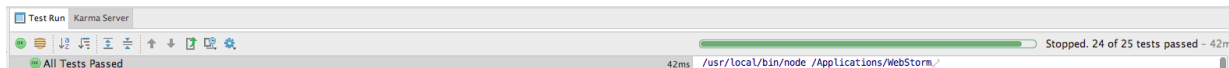


Figure 32: See all tests passed

In this section, an effort of showing how functions are written, developed and tested is provided in a step by step manner. Though only part of the `getInstance` function is examined as an example. The same little steps should be carried on throughout the whole development process until the component is fully developed. At the end not only the code is implemented, the functionalities and features are done, but also tests are completed. Tests are specifications that help the developer in further development and application maintenance. “Writing code is cheap, maintaining code is expensive” - It can not be stressed hard enough that it is not about the code that is run as expected but it should be maintainable. Without tests, the developer will find it way harder later on in the development process. On the hand, the developer will have confidence in further development and especially when it comes to refactoring code since he knows whether or not his changes will break parts of the app or not.

4.5.5 Code coverage

So far tests have been written however it is hard to say whether the tests are robust enough. Will there be any scenario/case that is missing which may potentially expose issue in the program? Code coverage is the answer. It is an important concept when it comes to testing in particular and in software development in general. The developer can write a lot of tests, those tests can be passed when run, but that does not mean that their code is secure and robust. There needed to be a metric that helps them know how much their code has been tested and that is code coverage. Code coverage is a way to measure and describe how extensive the program source code is when it is executed through test suites. Code coverage is calculated in percentage. The higher the code coverage does not mean the better, but coverage more than 70% can be a good indicator that the program is tested thoroughly and has a lower chance of software defects compared to one with lower code coverage.

Code coverage is generated by issuing “`npm test`” on the terminal/command line window.


```

moment-calendar-2 — bash — 152x54
Your branch is up-to-date with 'origin/master'.
LBL:moment-calendar-2 lvn$ npm test

> moment-calendar-2@0.4.4 test /Users/lvn/Dev/github_repo/moment-calendar-2
> karma start karma.conf.js

17 03 2017 23:06:59.534:INFO [framework.browserify]: bundle built
17 03 2017 23:06:59.544:INFO [karma]: Karma v1.5.0 server started at http://0.0.0.0:9876/
17 03 2017 23:06:59.544:INFO [launcher]: Launching browser Chrome with unlimited concurrency
17 03 2017 23:06:59.552:INFO [launcher]: Starting browser Chrome
17 03 2017 23:07:00.462:INFO [Chrome 56.0.2924 (Mac OS X 10.12.3)]: Connected on socket __EYIToT2ShBK1c3AAAA with id 93824924
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 1 of 25 SUCCESS (0 secs / 0.002 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 2 of 25 SUCCESS (0 secs / 0.008 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 3 of 25 SUCCESS (0 secs / 0.01 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 4 of 25 SUCCESS (0 secs / 0.018 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 5 of 25 SUCCESS (0 secs / 0.025 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 6 of 25 SUCCESS (0 secs / 0.028 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 7 of 25 SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 8 of 25 SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 9 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 10 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 11 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 12 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 13 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 14 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 15 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 16 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 17 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 18 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 19 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 20 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 21 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 22 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 23 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 24 of 25 (skipped 1) SUCCESS (0 secs / 0.031 secs)
Chrome 56.0.2924 (Mac OS X 10.12.3): Executed 24 of 25 (skipped 1) SUCCESS (0.083 secs / 0.053 secs)

-----
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Lines |
-----
src/
api.js    | 96.28   | 87.5     | 96.92   | 96.28   |                  |
api.spec.js | 92.59   | 80      | 100     | 92.59   | 10,24           |
api.spec.js | 92.19   | 100     | 91.3    | 92.19   | 10,71,72,73,74 |
core.js   | 98.41   | 92.86   | 100     | 98.41   | 58              |
core.spec.js | 100     | 100     | 100     | 100     |                  |
-----
All files | 96.28   | 87.5     | 96.92   | 96.28   |                  |

```

Figure 33: moment-calendar-2's test coverage in command line window

The command will first execute all the test cases and generate test coverage later. The plugin (browserify-istanbul) also output a nice coverage table on the terminal. If configured in karma's config file, it is also possible to generate coverage file displayed in HTML:

all files src/

96.28% Statements 287/215 87.5% Branches 21/24 96.92% Functions 63/65 96.28% Lines 287/215

File	Statements	Branches	Functions	Lines
api.js	92.59%	25/27	80%	7/7
api.spec.js	92.19%	59/64	0/0	21/23
core.js	98.41%	62/63	13/14	12/12
core.spec.js	100%	61/61	0/0	23/23

Figure 34: moment-calendar-2's test coverage in HTML format

The figure above shows how much code coverage has been done for our moment-calendar-2 component. The number is more than 90%, which exceeds the standard level of 70%. What important to mentioned is the coverage criteria that the report showed us. There are a few coverage criteria. The ones that are covered by karma-coverage, also the mains one, are:

- Statements: this help answer the questions of whether each statement in the program had been executed.
- Branches: this can be easily explained. For example, given an if-statement, this criterion checks whether both true and false branches have been executed.

- Functions: this criteria shows if each function in the program been called
- Lines: as its name, tracking how much lines of source code has been executed after the tests run.

By clicking on each file, we can also have a better view on which lines have been executed, which have not. Take api.js for example, this will be shown where the green parts indicate that those lines have been executed and red parts mean that these have been left out from the spec/test files.

all files / src/ api.js

92.59% Statements 25/27 80% Branches 8/10 100% Functions 7/7 92.59% Lines 25/27

```

1  /*
2   * Created by lvn on 28/08/16.
3   */
4
5  1x var core = require("./core.js");
6
7  // find date position (row, col) on weeks table
8  1x function getDatePosition(date) {
9  12x   if(!core.isDateInMonth(date) ){
10     return undefined;
11   }
12  12x   date = isNaN(date) ? moment(date).get('date') : date;
13  12x   return this.datePositionMap[date];
14 }
15
16 1x function getDate(date) {
17  9x   if(!core.isDateInMonth(date) ){
18  1x   return undefined;
19   }
20   try{
21  8x   var datePosition = this.getDatePosition(date);
22  8x   return this.momentWeeksTable[datePosition.row][datePosition.col];
23   }catch(err){
24     return undefined;
25   }
26 }
27
28 1x function setCurrentDate(date, format, isStrictMode) {
29  20x   date = core.isValidDate(date) ? date : moment();
30  20x   core.extend(this, core.populateWeeksTable(date));
31 }
32
33 1x function updateDate(date, prop) {
34  2x   return core.extend(this.getDate(date), prop);
35 }
36

```

Figure 35: code coverage analysis

In short, even though coverage, at first sight, seems not to be a vital part of the whole process, but it does serve as a guide for developers to see how thoroughly they have been writing their tests.

4.5.6 Documentations and releases

Documentation is a critical part of software engineering where the illustration that accompanies the computer software is written in the text document to explain how the program operates and can be used (Software documentation, Wikipedia). It is fortunate that Github provides an efficient and simple way to publish software documentations. The documentation, or

often called, README, can be written in a Markdown file format, which then can be rendered as a clean documentation page as the one in this project:

moment-calendar-2

npm package 0.0.4

A super lightweight, easy-to-use yet powerful Node module to create month calendar where each date is a momentJS date. You can leverage further with date operations as well as configuring localization through MomentJS.

You can use this component with other frameworks and libraries with ease. This [Demo](#) demonstrates how to use moment-calendar-2 together with angular to build a date picker component.

Getting Started

NPM install moment-calendar 2

```
npm install moment-calendar-2
```

Notes that this is a stand-alone version. moment-calendar depends on momentJS, so make sure you have also include that in your app.

Include the script in your html:

```
<script src="node_modules/moment-calendar-2/dist/moment-calendar-2.js"></script>
```

You can also include 'moment-calendar-2' by using 'require':

```
var MomentCalendarFactory = require('./node_modules/moment-calendar-2/dist/moment-calendar-2');
```

In your JS, use 'moment-calendar' like so:

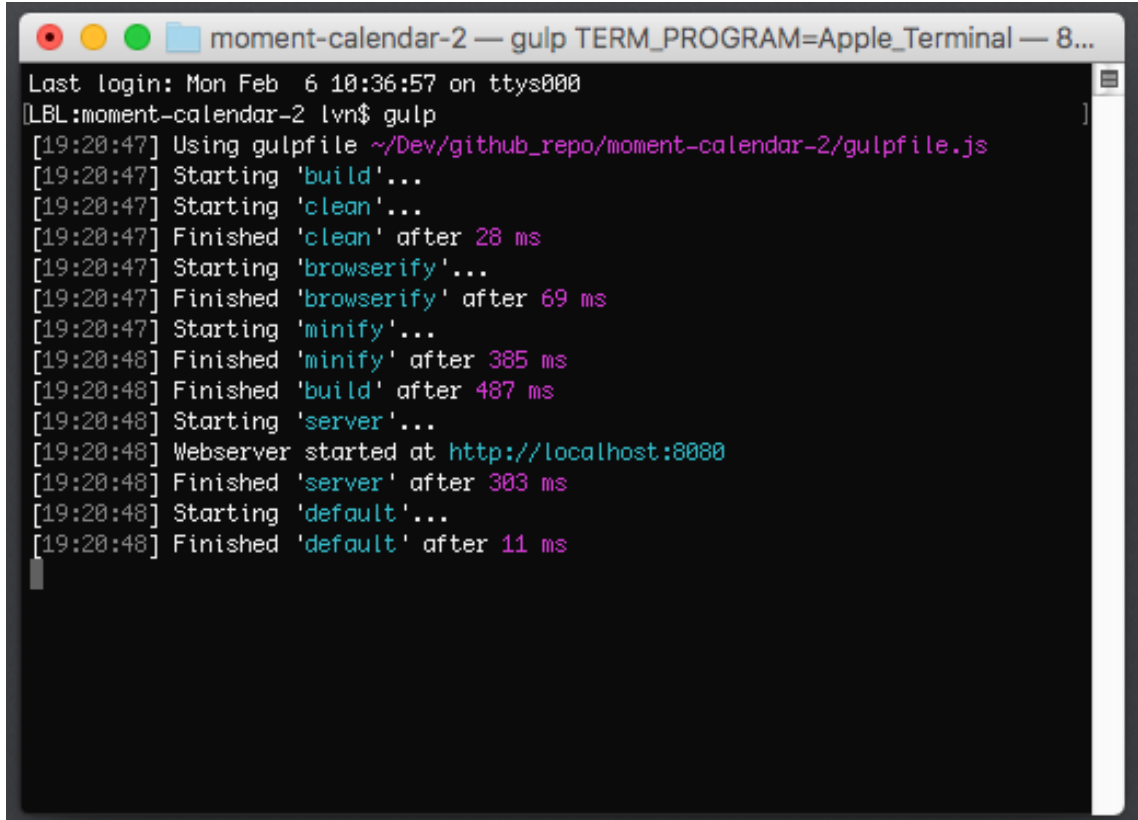
```
var calendar = MomentCalendarFactory.getInstance();  
calendar.getWeeksTable();
```

Figure 36: A screenshot of moment-calendar-2's documentation

A full documentation of this component is available here in appendix 6.

On documenting the application, a good practice is to develop also the demo application where it serves as an example of how the component is used in reality. Not only does it prove that this component is functional and useful but also it provides developer ways to discover and learn more about the component itself, how it works, and how it can be used. The more detail the demo is, with step-by-step instructional text as well as interactive application, the easier it is for the developer to follow and get their hands on experience by playing around and understand how the component can be used at its best.

Within the moment-calendar-2 project, the developer can run "gulp" on the terminal/command line window to utilize the demo application:



```

moment-calendar-2 — gulp TERM_PROGRAM=Apple_Terminal — 8...
Last login: Mon Feb  6 10:36:57 on ttys000
[LBL:moment-calendar-2 lvn$ gulp
[19:20:47] Using gulpfile ~/Dev/github_repo/moment-calendar-2/gulpfile.js
[19:20:47] Starting 'build'...
[19:20:47] Starting 'clean'...
[19:20:47] Finished 'clean' after 28 ms
[19:20:47] Starting 'browserify'...
[19:20:47] Finished 'browserify' after 69 ms
[19:20:47] Starting 'minify'...
[19:20:48] Finished 'minify' after 385 ms
[19:20:48] Finished 'build' after 487 ms
[19:20:48] Starting 'server'...
[19:20:48] Webserver started at http://localhost:8080
[19:20:48] Finished 'server' after 303 ms
[19:20:48] Starting 'default'...
[19:20:48] Finished 'default' after 11 ms

```

Figure 37: Initialize the demo application by terminal

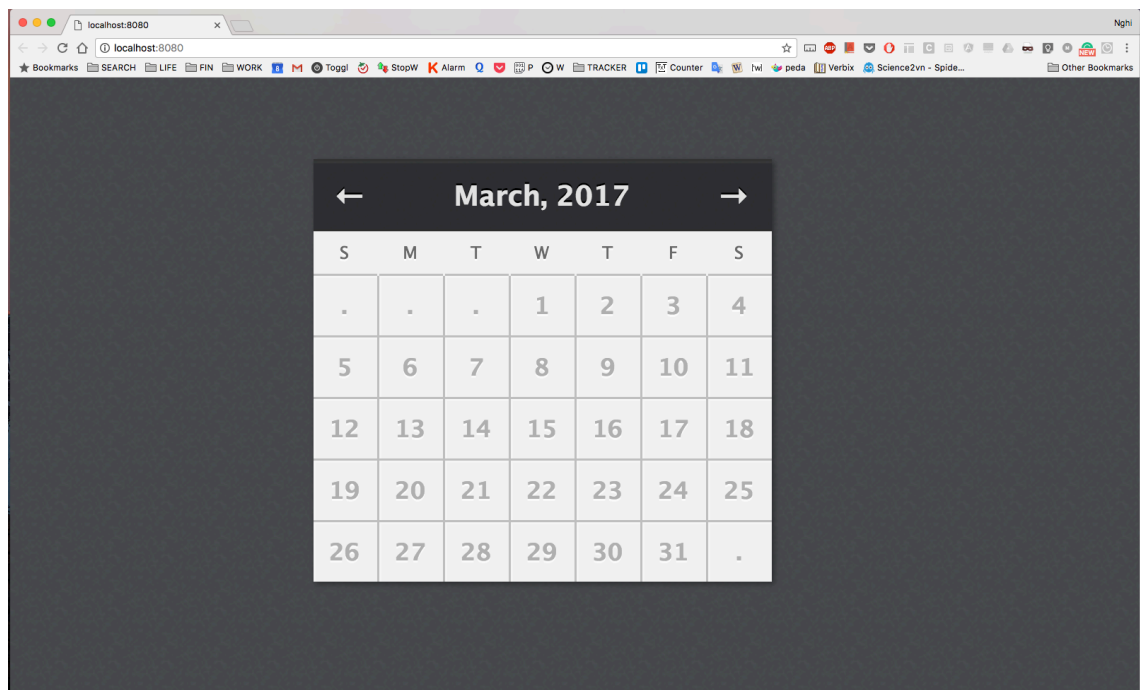


Figure 38: Demo application runs at localhost:8080, illustrating how moment-calendar-2 can be used to create a Date picker component

More information about the demo application and its source code can be found at Appendix 3.

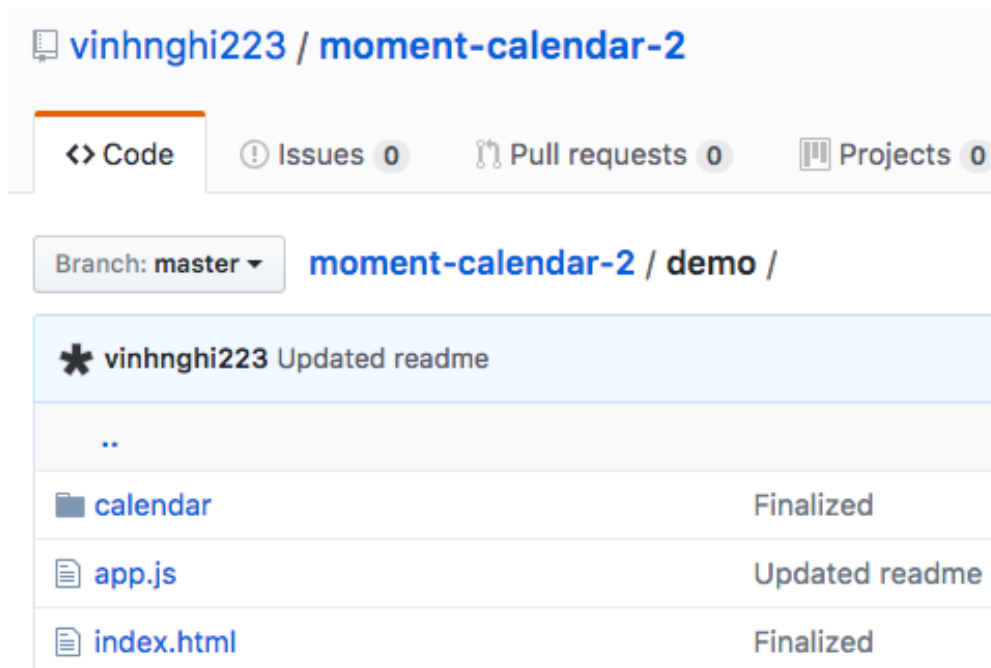


Figure 39: Demo application source code folder

The demo application source code can be a very good resource for other developers to see how moment-calendar-2 is implemented practically since definitions and examples written in documentations sometimes may not be intuitive enough for other developers.

Releasing is often the final stage, or at least, in the very final stage of the whole software release cycle. It is the sum of the stages of development and maturity for a piece of computer software: ranging from its original development to its final release, and including the release's updated version to help improve software quality (Software release life cycle, Wikipedia). To release a software component like moment-calendar-2, if one only use git and NPM, the process can be tiring. Below is a typical steps of that software developer often has to do to release a software component and publish it to NPM:

1. Making sure that the right branch is being used for releasing (often referred as release branch) and all the updates from remote branch have been pulled and resolved if there is conflicts.
2. Bump the version of package.json.
3. Add and commit the changes.
4. Tag the version to git.
5. Push to remote (released) branch
6. Publish to NPM.

Fortunately, there are a few ways to simplify the whole process. In gulp, there are a few gulp plugins that can help the developer to speedup the development by just simply issuing one command. Gulp-release-easy is one of them. It is also a gulp plugin that created by the author with the hope to help others ease their releasing process and publishing to NPM. In this project, gulp-release-easy is used.

According to that, to release a new version and publish it to NPM, instead of going through around 6 steps as described above, one can just simply issuing 'gulp-release' and the whole process will be taken care automatically:

A good part about NPM is whenever a new version of the package is released, it will also be fetched and cloned to the component's NPM page. In this project, the content of moment-

calendar-2 README.md is identical to the description in its NPM page. The two will be synchronized whenever there is a new package being released and published to NPM.

The screenshot shows the NPM page for the package `moment-calendar-2`. The page header includes the NPM logo and a search bar. The package name is `moment-calendar-2` with a `public` label and version `0.0.4`. The description states it is a super lightweight, easy-to-use yet powerful Node module to create month calendar where each date is a momentJS date. The 'Getting Started' section provides instructions on how to install the package using `npm install moment-calendar-2` and includes code snippets for including the script in HTML and using it in JavaScript. The sidebar on the right contains a 'Manage permissions for the whole team' section, the installation command `npm i moment-calendar-2`, the publisher `vinhngchi223` (published 7 months ago), the latest version `0.0.4`, the GitHub repository link, the MIT license, a collaborators list, and download statistics: 0 downloads in the last day, 6 downloads in the last week, and 45 downloads in the last month. There are no open issues on GitHub.

Figure 40: moment-calendar-2 on NPM page

It is also a good practice to have a release notes or CHANGELOG to indicate what have been changed in a specific version. Below is an example of a CHANGELOG:

[x.y.z] - On going

Fixes

- `getCurrentDate` return undefined when `getInstance` is called without parameters.

[0.0.4] - 2016-09-04

Changed

- Documentations updated
- More tests added

Figure 41: CHANGELOG file

5.1 Evaluations of TDD in development

Using TDD has brings many benefits. As a developer who practices TDD almost daily, following are what I observe:

The code quality is improved with development in TDD compared to non-TDD one. Creating tests first ensures that 100% test coverage is achieved without extra cost.

TDD can be applied anywhere in the whole software development project where there are requirements ranging from database testing to interfaces to other technologies.

Doing this way, TDD, therefore, handles requirement changes much more efficiently than other traditional approaches. Due to Agile philosophy on changes and small incremental development, requirements are not something carved in stone but expected to be changed over the time. The requirements can be more and more involved and specific. There is no argument against the fact that implementation changes may turn out to be costly. However, with TDD, the design of the application is incrementally, and further optimized through time, making it both robust and flexible at the same time. Without TDD, changes made especially after production can turn into a maintenance nightmare and can lead to longer time for releasing, higher risks, and more cost for business as might be expected.

5.2 Evaluations of developing common software component using TDD

During the process of developing moment-calendar-2, using TDD, the development went through all the necessary stages: Tests are first written, then are run to prove that they fail. Next real code is implemented to make the tests passed. Code refactoring was then done when needed to make the program more lean and reusable. As a result, we have a code coverage, created by browserify-istanbul, of 94.25% (which is an average of 96.28% statements, 87.5% branches, 96.92% functions, and 96.28% lines), which is close to the goal of TDD of 100% code coverage.

In addition to that, a new NPM package was released and be a part of the open source community. The component was released with good documentations which are helpful for other developers to learn how to use it. This is especially true when it comes to a common component where it can be reused by different parties in different situations and contexts. In addition to that, a demo application is implemented to facilitate developers to have a chance to do some experiment with the component. Though a plain old textual guidance is good but it is extremely helpful for the developers to have the ability to try out the component themselves, help them learn how to use the component not just by reading but also by “hacking” around the component.

Though the current project set up on GitHub is well-organized, it would be better if it has continuous integration to automatically run the tests whenever there are commits. In addition to that, the demo application also needs more refinement and textual guidance instead of just displaying a single Date picker component. This is, in fact, a location of convenience where developers can both do research on how moment-calendar-2 is used and at the same time learn about its API as well as its limitations. Being aware of moment-calendar-2's shortcomings is important since it motivates developers to further develop and enhance the component. Being an open-source project, as time goes by, with the help of the community, I am positive that the component will gain more attentions and contributions from other developers, including the ones who have more experience than me, which make it becomes more mature, advanced and useful in the future.

With its popularity, other developers can easily find it by simply typing in Google Search keywords “moment-calendar-2”.

Google moment calendar 2

All Images Videos News Maps More Settings Tools

About 13 100 000 results (0,47 seconds)

moment-calendar-2 - npm
<https://www.npmjs.com/package/moment-calendar-2>
 You can use this component with other frameworks and libraries with ease. This Demo demonstrates how to use moment-calendar-2 together with angular to ...

moment-calendar - npm
<https://www.npmjs.com/package/moment-calendar>
 provides queryable calendar data structure. ... Install. \$ npm install moment-calendar. Usage. var Calendar = require('moment-calendar'); Dependencies (2).

Moment.js | Home
<https://momentjs.com/>
 bower install moment --save # bower npm install moment --save # npm ... moment().subtract(10, 'days').calendar(); // 03/15/2017 moment().subtract(6, ...

Moment.js | Docs
<https://momentjs.com/docs/>
 2013-02-08 09 # A calendar date part and hour time part 2013-W06-5 09 # A ... YYYY from version 2.10.5 supports 2 digit years, and converts them to a year ...

javascript - moment.calendar() without the time - Stack Overflow
stackoverflow.com/questions/32420447/moment-calendar-without-the-time
 Sep 6, 2015 - I would like to use the moment.calendar() option without the time... .. moment().calendar() supports custom formatted strings and ... 13k23337 ...

Figure 42: moment-calendar-2 on google

Selecting the first result as in the screenshot leads us to moment-calendar-2 's NPM page. Here, the Stats section in the lower right corner show how active this component has been employed by others for the past days, weeks and months. Here is what is the statistics of moment-calendar-2 at the time of this writing:

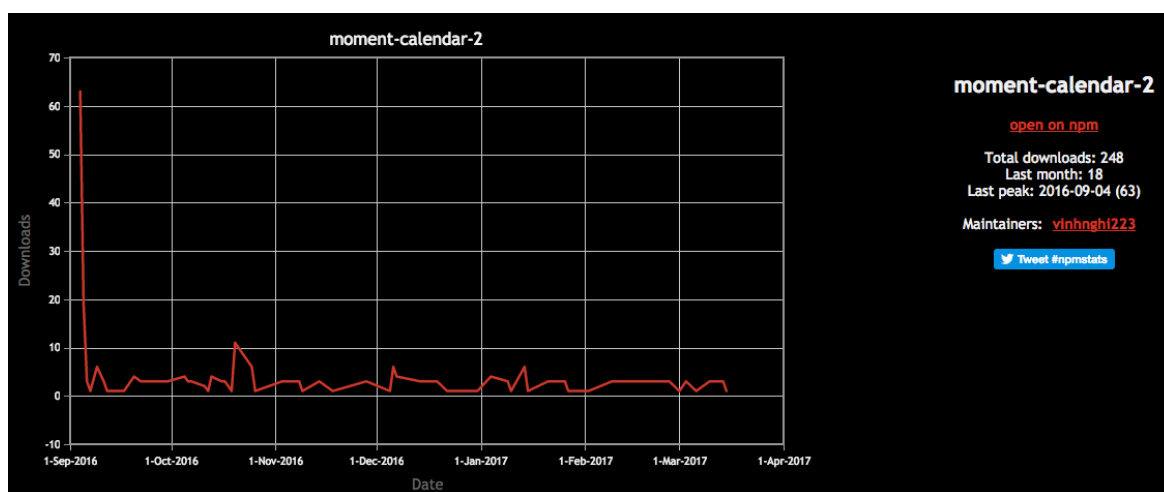


Figure 43: moment-calendar-2 downloads statistics

The number of downloads is high at the first time when the package is published. This is due to NPM's mechanism to verify whether this package is compliant with NPM regulations or not. However, after that, the number will be much lower (or higher) depends on the demand or popularity of this package in the market.

6 Conclusion

As stated in the goals and objectives, the common component which is then be focused in the implementation part of this thesis is a common Node module that is only limited to use on the web platform. This component has no UI at all. It plays a role as a purely functional common component whose main goals is to facilitate the development of Date picker component. The component, though originally, intended to be developed for Nordea, it can be used anywhere else just like any open-source project.

At the time of this writing, the digital solution that Nordea is building gets initial success using the strategy of building common components using TDD. The software has been released and is used by a few pilot users currently. Unit tests and UI tests are parts of the development process. With the help of Continuous Integration, code coverage metrics is calculated frequently. It leverages their benefits making development and maintenance process much smoother, guaranteeing a high-quality software product ready for releasing and delivering. These metrics serves as an indicator showing which areas are more vulnerable to changes and therefore needs more focus on testing.

Also even though the thesis promotes strongly the use of TDD throughout the entire software development process, and a good strategy in Agile software development, it does not mean that this technique has no drawback. TDD can be dangerous if applied in areas that are sensitive to changes. That is why the role of software architecture, especially in a big project, is extremely important where identifying risks and planning up-front concerns which are very costly to mutate later.

Another situation where implement TDD can become really clumsy is when a software program requires cooperation from many other software programs and systems. The only way to get around this issue is to mock other systems and third-party software and only focusing on writing tests against the software program in focus. If the process of mocking other third party software and systems sometimes is more challenging and consumes time as much as time spent on implementing the real test, one should consider avoiding doing TDD strictly. As a rule of thumb, whenever something can be very expensive in terms of time, money to changed later, TDD may not be a good idea to follow strictly.

TDD alone, however, does not make software great. This thesis also shows why developing common components, from system design to real code implementation, is a crucial part of the development process. Its benefits are profound. Not only does it bring consistency, maintainability, and scalability for the software project which implements it, but also doing the right way, with good and generous intention, it can help producing a piece of software that can be shared and contributed to the open source community. This development strategy is especially important for a large financial application project, where hundreds of people are involved. It allows bank to serve their customers better, keep pace with market changes, and develop with high quality, low time to market, and with a greater return on their IT investment, which, in turn, helps them stay competitive with their peers as well as disrupting FinTech startups.

Nordea can use the moment-calendar-2 component in the future not only for the Date picker component, but any component that requires date/time operations. The component is developed with scalability in mind. In fact, it is so scalable that it can be used elsewhere in another project, in a totally different business context, even maybe with different technology stack other than front end framework, without limited to only the banking and financial services area where it was originally built for.

During the thesis project, I have learned and had a deeper understanding of the philosophy behind Atomic design and web common component in general. At the same time I had the opportunity and freedom to cultivate my programming skills using TDD, which, helps me enhance my coding skills as a developer and, hence, be able to write better, contribute well-maintained source codes for my company project. I was also satisfied and proud of moment-calendar-2 being published and valuable not only to my project at hand but also to other de-

velopers, not only in Nordea, but also in the open-source community, who wants to develop their date/time-related software features.

In "No Silver Bullet - Essence and Accident in Software Engineering", Turing Award winner Fred Brooks wrote that "there is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement within a decade in productivity, in reliability, in simplicity." Though he advocates that software should "grow" organically through incremental development and points out devising, implementing the main and subprograms should be done right at the beginning of the development project. This thesis shares Brooks's belief as well as aligns with his vision of subprograms development through the development of common component in addition to using TDD methodology during the development process in order to enhance the software design and functionality little by little. Developing software in this fashion keeps engineers motivated by being able to learn new things and improve themselves, provides a working system at every stage of the development process and, in the end, helps build more scalable, more maintainable, better quality software products.

References

Print sources

Brooks F, 1995. The Mythical Man-Month: Essays on Software Engineering . Addison-Wesley

Overton J, Strimpel J. 2015. Developing Web Components: IO from JQuery to Polymer. California: O'Reilly Media, Inc.

Online sources

Agile Adoption by the Financial Services Industry. 2012. Accessed 5 March 2017
<https://www.cprime.com/2012/09/agile-adoption-financial-services-industry/>

Asynchrony. 2017. Accessed 10 April 2017
<https://en.wikipedia.org/wiki/Asynchrony>

Chronometer. 2017. Accessed 10 March 2017
<https://en.wikipedia.org/wiki/Chronometer>

Code refactoring. 2015. Accessed 31 December 2016
https://en.wikipedia.org/wiki/Code_refactoring

Code refactoring. 2017. Accessed 10 April 2017
https://en.wikipedia.org/wiki/Code_refactoring

Continuous integration. 2017. Accessed 5 March 2017
https://en.wikipedia.org/wiki/Continuous_integration

Dynamic programming language. 2017. Accessed 10 March 2017
https://en.wikipedia.org/wiki/Dynamic_programming_language

Factory method pattern. 2016. Accessed 31 November 2016
https://en.wikipedia.org/wiki/Factory_method_pattern

Gherkin. Accessed 6 January 2017
<https://github.com/cucumber/cucumber/wiki/Gherkin>

Goerner, A. 2016. How better faster cheaper happens. Powerpoint slides. Accessed 11 March 2017
http://www.alani-consulting.com/uploads/8/5/4/6/85469078/how_better_faster_cheaper_happens_-_alani_version_v2.1.pdf

High-level programming language. 2017. Accessed 10 March 2017
https://en.wikipedia.org/wiki/High-level_programming_language

Internet Impact on Economy. 2009. Accessed 29 December 2016.
<http://news.softpedia.com/news/Internet-Impact-on-Economy-113952.shtml>

ISO 8601. 2017. Accessed 5 January 2017
https://en.wikipedia.org/wiki/ISO_8601#Dates

Jasmine - introduction.js. Accessed 5 March 2017
<https://jasmine.github.io/2.0/introduction.html>

JavaScript Frameworks: The Best 10 for Modern Web Apps. 2016. Accessed 31 March 2016.

<http://noeticforce.com/best-JavaScript-frameworks-for-single-page-modern-web-applications>

JavaScript. 2016. Accessed 30 December 2016
<https://en.wikipedia.org/wiki/JavaScript>

JavaScript. 2017. Accessed 10 April 2017
<https://en.wikipedia.org/wiki/JavaScript>

Mocks, fakes, and stubs. 2017. Accessed 11 April 2017
https://en.wikipedia.org/wiki/Mock_object#Mocks.2C_fakes.2C_and_stubs

moment.js. 2017. Accessed 10 April 2017
<https://momentjs.com>

Node.js. 2017. Accessed 10 April 2017
<https://en.wikipedia.org/wiki/Node.js>

Nondeterministic algorithm. 2017. Accessed 10 April 2017
https://en.wikipedia.org/wiki/Nondeterministic_algorithm

Nordea. 2016. Accessed 11 February 2017
<https://en.wikipedia.org/wiki/Nordea>

Paper Date picker. 2016. Accessed 11 March 2017
<https://github.com/bendavis78/paper-date-picker>

Software design pattern. 2016. Accessed 31 December 2016
https://en.wikipedia.org/wiki/Software_design_pattern

Software design pattern. 2017. Accessed 10 April 2017
https://en.wikipedia.org/wiki/Software_design_pattern

Software documentation. 2016. Accessed 30 December 2016.
https://en.wikipedia.org/wiki/Software_documentation

Software release life cycle. 2016. Accessed 30 March 2016.
https://en.wikipedia.org/wiki/Software_release_life_cycle

Test harness. 2017. Accessed 11 April 2017
https://en.wikipedia.org/wiki/Test_harness

Testing the controller. Accessed 7 March 2017
<http://gavd.github.io/step-by-step-web-apps-with-lithium-php/testing-the-controller.html>

The history and legacy of agile development. Accessed 30 January 2017.
<https://techbeacon.com/agility-beyond-history%E2%80%94legacy%E2%80%94agile-development>

Thinking in components. 2015. Accessed 5 March 2017
http://www.hedleysmith.com/posts/thinking_in_components.html

Understanding Web Components. 2016. Accessed 14 March 2017
<https://medium.com/the-ui-files/understanding-web-components-d051baa66019#.iuk0m0ghl>

Unit testing. 2017. Accessed 10 April 2017
https://en.wikipedia.org/wiki/Unit_testing

Unit testings. 2017. Accessed 5 March 2017

https://en.wikipedia.org/wiki/Unit_testing

Use UI Automation To Test Your Code. 2017. Accessed 5 March 2017
<https://msdn.microsoft.com/en-us/library/dd286726.aspx>

Version control. 2016. Accessed 31 November 2016.
https://en.wikipedia.org/wiki/Version_control

What is Agile. Accessed 7 March 2017
<http://www.agilenutshell.com/>

What Is Agile? (10 Key Principles of Agile). 2007. Accessed 7 March 2017
<http://www.allaboutagile.com/what-is-agile-10-key-principles/>

Figures

Figure 1: Architecture design of a general banking application where moment-calendar-2 is used	9
Figure 2: Date picker UI example	10
Figure 3: Five levels in atomic design	12
Figure 4: Atom level in atomic design	13
Figure 5: Molecules level in atomic design	14
Figure 6: Organisms level in atomic design - Date picker with input field	14
Figure 7: Template level in atomic design.....	15
Figure 8: Page level in atomic design - Nordea front page.....	16
Figure 9: Test pyramid model.....	18
Figure 10: Webstorm UI.....	22
Figure 11: Github UI.....	23
Figure 12: NPM UI.....	24
Figure 13: A typical UI of Date picker	27
Figure 14: Date picker with input field.....	28
Figure 15: Date range picker.....	28
Figure 16: Date picker with different localizations	29
Figure 17: Date picker design with atomic design.....	30
Figure 18: Create a new repository	32
Figure 19: Basic gulp tasks	33
Figure 20: Karma configuration file.....	34
Figure 21: moment-calendar-2 sub modules.....	35
Figure 22: How another module is included	35
Figure 23: getInstance factory method.....	35
Figure 24: moment-calendar-2's interface	37
Figure 25: Implement unit test cases for getInstance method	38
Figure 26: Implement unit test cases for getInstance method	38
Figure 27: getInstance method	38
Figure 28: See the tests fail.....	39
Figure 29: getInstance return an object	39
Figure 30: See the tests fail again	39
Figure 31: Completed getInstance function	39
Figure 32: See all tests passed.....	40
Figure 33: moment-calendar-2's test coverage in command line window.....	41
Figure 34: moment-calendar-2's test coverage in HTML format	41
Figure 35: code coverage analysis	42
Figure 36: A screenshot of moment-calendar-2's documentation	43
Figure 37: Initialize the demo application by terminal.....	43
Figure 38: Demo application runs at localhost:8080, illustrating how moment-calendar-2 can be used to create a Date picker component	44
Figure 39: Demo application source code folder.....	45
Figure 40: moment-calendar-2 on NPM page	46
Figure 41: CHANGELOG file	46
Figure 42: moment-calendar-2 on google	48
Figure 43: moment-calendar-2 downloads statistics	48

Tables

Table 1: gulp tasks definitions	33
Table 2: monthCalendar instance's API	36
Table 3: moment calendar core's API	37

Appendixes

Appendix 1: Moment-calendar-2's core.js module	57
Appendix 2: Moment-calendar-2's api.js module	68
Appendix 3: Demo application	59
Appendix 4: Gulp file.....	60
Appendix 5: Documentation of Moment-calendar-2	61

Appendix 1: Moment-calendar-2's core.js module

```

/**
 * Created by lvn on 28/08/16.
 */

var cache = {};
var currentMonthDate;

function getDaysOfMonth(date) {
  var numberOfDays = moment(date).daysInMonth();
  var daysArray = [];
  for (var i = 1; i <= numberOfDays; i++) {
    daysArray.push(i);
  }
  return daysArray;
}

// Get number of days from previous month overlapping current month
function getPreviousMonthOverlappingDays(date) {
  var firstWeekDay = moment(date).startOf('month').weekday();
  var prevMonthDays = [];
  for (var i = 1; i <= firstWeekDay; i++) {
    prevMonthDays.push(i);
  }
  return prevMonthDays;
}

// Get number of days from previous month overlapping current month
function getNextMonthOverlappingDays(date) {
  var lastWeekDay = moment(date).endOf('month').weekday();
  var nextMonthDays = [];
  var howManyNextDays = 6 - lastWeekDay;
  for (var i = 1; i <= howManyNextDays; i++) {
    nextMonthDays.push(i);
  }
  return nextMonthDays;
}

function initWeeksDay() {
  return [0, 1, 2, 3, 4, 5, 6].map(function (el, index) {
    return moment().weekday(index).format('ddd').charAt(0);
  });
}

function isDateInMonth(date) {
  try {
    if (!isNaN(date)) {
      if (isValidDate(date)) {
        var cacheKey = moment(date).year() + '-' + moment(date).month();
        return !!cache[cacheKey];
      } else {
        return false;
      }
    } else {
      return (date >= currentMonthDate.startOf('month').get('date')) && (date <= currentMonthDate.endOf('month').get('date'));
    }
  } catch (err) {
    return false;
  }
}

// Use memoization here or another service
function populateWeeksTable(date) {
  var cacheKey = moment(date).year() + '-' + moment(date).month();
  currentMonthDate = moment(date);
  //console.log('pop currentMonthDate: ', currentMonthDate);

  if (!cache[cacheKey]) {
    var weeksTable = [];
    var momentWeeksTable = [];
    var datePositionMap = [];

    var monthDays = getPreviousMonthOverlappingDays(date).concat(getDaysOfMonth(date)).concat(getNextMonthOverlappingDays(date));
    var weekDays = initWeeksDay();

    weeksTable.push(weekDays);
    momentWeeksTable.push(weekDays);

    for (var j = 0, rowId = 1; j < monthDays.length; j += 7) {
      var week = monthDays.slice(j, j + 7);
      week.forEach(function (day, colId) {
        datePositionMap[day] = [rowId, colId];
      });
      weeksTable.push(week);
      momentWeeksTable.push(week.map(function (d) {
        return d === '?' ? d : moment(date).set('date', d);
      }));
      rowId++;
    }

    cache[cacheKey] = {
      datePositionMap: datePositionMap,
      weeksTable: weeksTable,
      momentWeeksTable: momentWeeksTable
    };
  }

  return cache[cacheKey]
}

function isValidDate(date, format, isStrictMode) {
  return format ? moment(date, format, isStrictMode).isValid() : moment(date, isStrictMode).isValid();
}

function extend(src, dest) {
  for (var i in dest) {
    if (dest.hasOwnProperty(i)) {
      src[i] = dest[i];
    }
  }
}

function getCurrentMonthDate() {
  return currentMonthDate;
}

module.exports = {
  currentMonthDate: currentMonthDate,
  extend: extend,
  isValidDate: isValidDate,
  isDateInMonth: isDateInMonth,
  initWeeksDay: initWeeksDay,
  populateWeeksTable: populateWeeksTable,
  getDaysOfMonth: getDaysOfMonth,
  getCurrentMonthDate: getCurrentMonthDate,
  getPreviousMonthOverlappingDays: getPreviousMonthOverlappingDays,
  getNextMonthOverlappingDays: getNextMonthOverlappingDays
};

```

Appendix 2: Moment-calendar-2's api.js module

```
/**
 * Created by ivn on 28/08/16.
 */

var core = require("./core.js");

// find date position (row, col) on weeks table
function getDatePosition(date) {
  if(!core.isDateInMonth(date)){
    return undefined;
  }
  date = isNaN(date) ? moment(date).get('date') : date;
  return this.datePositionMap[date];
}

function getDate(date) {
  if(!core.isDateInMonth(date)){
    return undefined;
  }
  try{
    var datePosition = this.getDatePosition(date);
    return this.momentWeeksTable[datePosition.row][datePosition.col];
  }catch(err){
    return undefined;
  }
}

function setCurrentDate(date, format, isStrictMode) {
  date = core.isValidDate(date) ? date : moment();
  core.extend(this, core.populateWeeksTable(date));
}

function updateDate(date, prop) {
  return core.extend(this.getDate(date), prop);
}

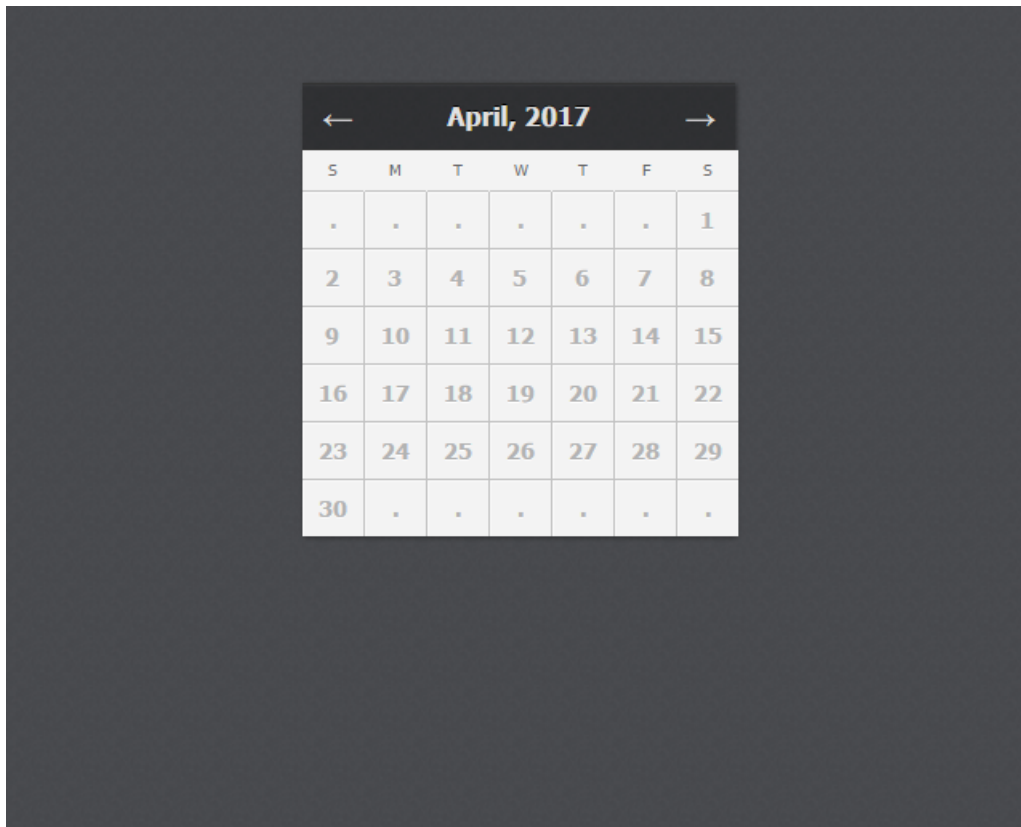
function getWeeksTable(isPlainWeeksTable) {
  return isPlainWeeksTable ? this.weeksTable : this.momentWeeksTable;
}

function getCurrentDate() {
  return core.getCurrentMonthDate();
}

// Public
function getInstance(date) {
  var monthCalendarInstance = Object.create({
    setCurrentDate: setCurrentDate,
    getCurrentDate: getCurrentDate,
    updateDate: updateDate,
    getDate: getDate,
    getWeeksTable: getWeeksTable,
    getDatePosition: getDatePosition
  });
  monthCalendarInstance.setCurrentDate(date);
  return monthCalendarInstance;
}

module.exports = {
  getInstance: getInstance
};
```

Appendix 3: Demo application



- Demo application can be accessed at <https://vinhnghi223.github.io/moment-calendar-2/>
- Demo application source code can be accessed at: <https://github.com/vinhnghi223/moment-calendar-2/tree/master/demo>

Appendix 4: Gulp file

```
var gulp = require('gulp');
var $ = require('gulp-load-plugins')();
require('gulp-release-easy')(gulp, {releaseBranch: 'master'});
var sass = require('gulp-sass');
var gulpSequence = require('gulp-sequence');
var del = require('del');
var browserify = require('browserify');
var source = require('vinyl-source-stream');

gulp.task('clean', function () {
  return del(['dist/*.*']);
});

gulp.task('minify', function () {
  return gulp.src('dist/moment-calendar-2.js')
    .pipe($.uglify())
    .pipe($.rename('moment-calendar-2.min.js'))
    .pipe(gulp.dest('dist'));
});

gulp.task('browserify', function () {
  return browserify('./src/api.js',{
    standalone: 'Moment-calendar-factory'
  })
    .bundle()
    .pipe(source('moment-calendar-2.js'))
    .pipe(gulp.dest('./dist'));
});

gulp.task('build', function (cb) {
  return gulpSequence('clean', 'browserify', 'minify', cb);
});

gulp.task('server', ['build'], function () {
  return gulp.src(['demo', 'node_modules', 'dist'])
    .pipe($.webserver({
      port: 8080,
      livereload: true,
      open: true
    }));
});

gulp.task('default', ['server'], function () {
  gulp.watch('src/**/*.js', ['browserify']);
});
```

Appendix 5: Documentation of moment-calendar-2

A super lightweight, easy-to-use yet powerful Node module to create month calendar where each date is a momentJS date. You can leverage further with date operations as well as configuring localization through MomentJS.

You can use this component with other frameworks and libraries with ease. This [Demo](#) demonstrates how to use moment-calendar-2 together with angular to build a date picker component.

Getting Started

NPM install moment-calendar 2

```
npm install moment-calendar-2
```

Notes that this is a stand-alone version. moment-calendar depends on momentJS, so make sure you have also include that in your app.

Include the script in your html:

```
<script src="node_modules/moment-calendar-2/dist/moment-calendar-2.js"></script>
```

You can also include 'moment-calendar-2' by using 'require':

```
var MomentCalendarFactory = require('./node_modules/moment-calendar-2/dist/moment-calendar-2');
```

In your JS, use 'moment-calendar' like so:

```
var calendar = MomentCalendarFactory.getInstance();
calendar.getWeeksTable();
```

API

Calling MomentCalendarFactory.getInstance(); will return an instance of momentCalendar of which methods are as following:

Methods	Arguments	Description
setCurrentDate	a date (eg. '2017-01-01')	Set the current date for month calendar. If argument is left blank or is an invalid date then current time being will be used
getCurrentMonthDate		Get the current date of month calendar. Return will be a moment object.
getDate	a number or date (eg. '1', '2017-01-01' etc.)	Get a moment object corresponding to the date you want to get from month calendar.
updateDate	(date, obj) (eg. ('2017-01-01', {isHoliday: true}))	Extend a date object of calendar with a object defined by your own. This is good when you need to attached a certain customized properties/methods that are not defined by momentJS. If you need to override those properties, just simply pass another object with same property keys again. Later when you call getDate, you can access those properties from the returned object.
getWeeksTable	boolean (false by default if argument is missing)	Return an array of weeks table, the one that is similar like a month calendar view in date picker (with first element is an array of weekday ('S', 'M', 'T' etc.)). Each date is a moment date. If you want a simplified version of weeks table where each date is an integer (eg. from 1 to 31), pass in true as the argument.
getDatePosition	a number or date (eg. '1', '2017-01-01' etc.)	Get back date position in the weeks table mentioned above. The returned object has 2 properties 'row' and 'column' that indicate where the date is located in the weeks table.