ZEIT: World-Class Static Hosting

Course CS455: Software Engineering

Software Requirements Specification

July 8, 2020

Author	*Student ID*
Hitesh Kumar	160298

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to pool requirements to enable the reader to build a Cloud Platform for deployment of **Static Sites** and **Server-less Functions** with minimal efforts from the side of the user. It will illustrate the purpose and complete declaration for the development of system. It will also explain system constraints, interface and interactions with other external applications.

1.2 Document Conventions

Term	Definition
Deployment	A collection of files that were uploaded to the platform.
Preview Deployments	Preview Deployments are the default type of deployment. They are made available by pushing to a git branch through a Git integration, or by using the now command from Now CLI.
Production Deployments	Production Deployments are a promoted type of Deployment, made available by merging into a default branch through a Git integration, or by using the now –prod command from Now CLI
Build Step	The Build Step is the process, before a deploymment is made publically available, in which a deployment's source files are constructed to be production-ready output.
Run Time	Run Time is the process in the deployment lifecycle that the deployment's code is being executed, after successfully being built through during the Build Step.
Custom Domains	A Custom Domain is a domain that has been added to a user or team for use with a project. Custom Domains are automatically updated when the project they are assigned to gains a new Production Deployment.
Source Files	These are the files contained within your project before it is uploaded with ZEIT Now.
Serverless Function	A serverless code execution handler that can be invoked over the web.
Static File	A file that is not executed, but instead served as-is to the program sending the request.
Project	A file that is not executed, but instead served as-is to the program sending the request.
Apex Domain	An Apex Domain is a custom domain without an included subdomain. For example, ”my-zeit-domain.com” is an Apex Domain, but ”docs.my-zeit-domain.com” is not. Apex Domains are sometimes referred to as ”naked”, ”bare”, ”base”, ”root”, or ”zone apex” domains.

1.3 Intended Audience and Reading Suggestions

This Document provides first hand knowledge of the software to the developers,project managers,marketing staff,users,testers.

This Document starts with an overall description of the product and then goes on to explain various functional requirements.Then the document lists various Quality attributes that the product requires and ends with it's business applications.
All the users of this product are suggested to read the section on **"User Interfaces"** to know about various supported features.Marketing staff might find the section on **"Other NonFunctional Requirements"** more useful whereas the Project Managers and developers are encouraged to read this document in the same flow as it is written in.

1.4 Product Scope

The product will help our users to host JAMstack websites and web services that deploy instantly, scale automatically and require no supervision. All with no configuration. The platform also provides easy to use Command-Line Interface (CLI) that can be used to manage deployments and manage integrated access (IAM) to user's account. The application is free to use but limited to one user per account that includes unlimited projects, HTTPS-enabled custom domains, continuous deployments, high performance Content Delivery Network (CDN) and serverless functions. We also provide flexible pricing for teams of all sizes with more features and 24/7/365 chat support. Flexible pricing is offered for Pro, Business and Enterprise teams. These features are described in detail under section 4.

Since the processes in the backend are fully automated, there is no need for human intervention except in case of failover and exponential backoff**.

1.5 References

- IEEE System Requirements Specification Template

1.6 Overview

The remainder of this document includes three chapters and appendixes. The second one provides an overview of the system functionality and system interaction with other systems. This chapter also introduces different types of stakeholders and their interaction with the system. Further, the chapter also mentions the system constraints and assumptions about the product.
The third chapter provides the requirements specification in detailed terms and a description of the different system interfaces. Different specification techniques are used in order to specify the requirements more precisely for different audiences.
The fourth chapter deals with the system features.**
The fifth chapter deals with various non-functional requirements.**
The last section at the end of this document includes all the future plans of this document and our vision about the scope of this product.

2 Overall Description

This section will give an overview of the whole system. The system will be explained in its context to show how the system interacts with other systems and introduce the basic functionality of it. It will also describe what type of stakeholders that will use the system and what functionality is available for each type. At last, the constraints and assumptions for the system will be presented.

2.1 Product Perspective

The product will consist of two parts: Command-line Interface (CLI) and an online dashboard. The CLI will be used to serve as a quick and easy way to deploy your static site along with managing domains, SSL certificates, secrets, instances of your deployment, view logs and manage Integrated Access (IAM) to your project. The complete list of functionalities with details are stated in section 4. The online dashboard will be used to manage the same using GUI which is more intuitive.

2.2 Product Functions

The product's USP is to be able to host static sites over SmartCDN (See section 1.2). This functionality can be broken down into the following sub-functionalities:

2.2.1 User Authentication

The platform allows only authenticated users to perform any action. There are four ways a user will be able to authenticate: GitHub, GitLab, BitBucket and E-mail with password. The user will be also allowed to enable Two-Factor Authentication for more security.

2.2.2 Hosting

A user will be able to host static website automatically served via edge locations. Using pipelines, user will be able to commit and deploy code right away. The platform will be able to support builds via 16 mostly used web frameworks out of the box.

2.2.3 Pipeline Management

A user will be able to connect the repositories on GitHub, GitLab and Bitbucket in order to setup pipeline for continuous integration and deployment. More on that in section 4.

2.2.4 Buy Domains

A user will be able to buy custom domains of his/her choice on our platform and use it for any of the active deployments on ZEIT. All the payments will be handled via third-party payment gateway.

2.2.5 Online Dashboard

The online dashboard will provide functionality to manage the build and deployment information, both current and previous. It will be used to document platform changes and API references.

2.2.6 Command-Line Interface and API support

A command-line interface will be made available to the user separately which will serve as a quick alternative to online dashboard. User will also be able to access platform's functionalities using APIs exposed via HTTP/1.1 and HTTP/2. More on that in section 3.1.

2.3 User Classes and Characteristics

There will be three types of users who will be interacting with the platform either via Online Dashboard, CLI or API: Administrative users, Paid users, Free users. Each of these classes are different in a way every class has its own allowances and features enabled.

The users in the administrative role will be able to monitor users, deployments, service uptimes and downtimes for issue related on customer end report them to the concerning teams.

Free users are those who will be able to use the platform but some features such as custom SSL certificates etc. and usage allowances will be much more restricted.
Paid user are pro/business and enterprise users who have higher needs than the average free customers. The usage allowances will be much higher and they will be provided instant support via 24/7 by submitting tickets.

See section 2.6 for details on allowances.

2.4 Operating Environment

The platform will run on top of an abstraction over multi-cloud platforms (Google Cloud, Azure and AWS). The platform will auto-detect which framework your code is built upon and automatically prepare an environment for it. Serverless functions support is brought in via AWS Lambda and Firebase Cloud Functions. The continuous deployment feature will be built upon a third-party provider namely, Semaphore.CI.

2.5 User Documentation

The online dashboard will have a **Docs** section which will provide documentation for each and every feature available to the user. Users can also use CLI help command to view all the actions that they can perform via CLI. A number of tutorials will be published on the website to ease the process and make users aware of the platform usage.

2.6 Product Contraints

This section highlights two areas relating to supported frameworks, pricing plans, Usage Allowances, and Plan Limits.

2.6.1 Optimized Frameworks

A variety of frameworks will be optimized for the platform. The following list contains those frameworks that will be optimized right out of the box and what has been done to aid their performance. Any getting started method will immediately be ready to deploy with Now CLI's command from the terminal, or to be deployed with a Git integration:

Framework	CLI command	Optimized
Next.js	npm init next-app my-next-project	Yes
Create React App	npm init react-app my-cra-project	Yes
Vue.js	npx @vue/cli create my-vue-project	Yes
Gatsby	npx gatsby-cli new my-gatsby-project	Yes
Ember.js	npx ember-cli new my-ember-project	Yes
Svelte	npx degit sveltejs/template my-svelte-project	Yes
Stencil	npm init stencil	Yes
Preact	npx preact-cli create default my-preact-project	Yes
Angular	npx @angular/cli new my-angular-project	Yes
Polymer	npx polymer-cli init polymer-3-starter-kit	Yes
Gridsome	npx @gridsome/cli create my-gridsome-project	Yes
UmiJS	npm init umi my-umi-project	Yes
Docusaurus	npx docusaurus-init	Yes
Saber	npm init site my-saber-project	Yes
11ty/Eleventy	npx degit 11ty/eleventy-base-blog my-11ty-project	Yes
Hexo	npx hexo-cli init my-hexo-project	Yes

2.6.2 Usage Allowances

Usage Allowances are the maximum limits provided by each pricing plan. They specify a limit for the amount of resources available for use under that plan before deployments are blocked (Free plan) or additional costs are incurred.

Limit	Free	Pro	Business	Enterprise
Bandwidth (GB)	20	100	1000	Custom
Build Time (Hours)	2	10	100	Custom
Execution Time (Hours)	10	100	1000	Custom
Concurrent Builds	1	3	6	Custom

2.6.3 Plan Limits

Plan Limits are limitations that apply to specific plans that cannot be increased, the only way to do so is to upgrade to a higher tier pricing plan.

Limit	Free	Pro	Business	Enterprise
Deployments (per day)	100	3000	3000	Custom
Deployment History	10	-	-	-
Exec. Duration (Seconds)	10	60	600	900

2.7 Assumptions and Dependencies

As the product spans multi-cloud, we are assuming a decent uptime for the servers provided by Google Cloud, Azure and AWS for hosting and serverless functions and Semaphore.CI for continuous deployments.So our product is indirectly dependendent on these third party softwares to provide good user experience.

It will be assumed that users will be using the platform only for static websites.Any dynamic content hosting on part of user would result in unexpected results being shown to the user as only the first fetch on the part of the server would be rendered without continuous updates.

Also we assume the developers at ZEIT would write the code in a modular manner following good coding standards that would make the process of changes and updates easy.

3 External Interface Requirements

3.1 User Interfaces

To start using the various static hosting features of ZEIT, the user should first signup on the ZEIT Now website. The user should be presented with a signup window as shown in figure 1. The signup window should provide multiple signup options. This should include signup using github, gitlab, bitbucket. Other than these options, the user should also be able to sign up using just an email. Choosing signup through email should redirect the user to a new signup page as shown in figure 2 and 3. Github, gitlab and bitbucket signup option should redirect the user to the respective official website for logging in there.

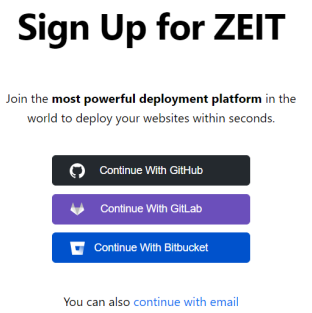

Figure 1: Stage 1 for Sign up

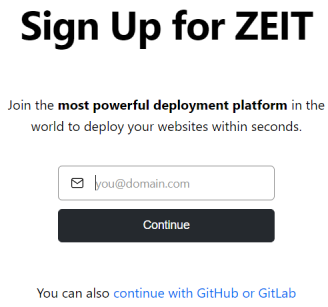

Figure 2: Stage 2 for Sign to up

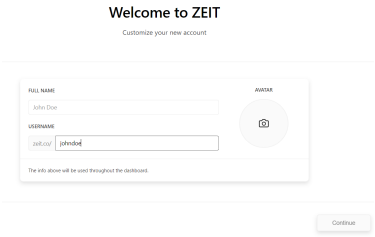

Figure 3: Stage 3 for Sign hi up

Once the signup is completed the user should be shown dashboard. The dashboard page should display all current projects along with a option to create new project as shown in figure 4.

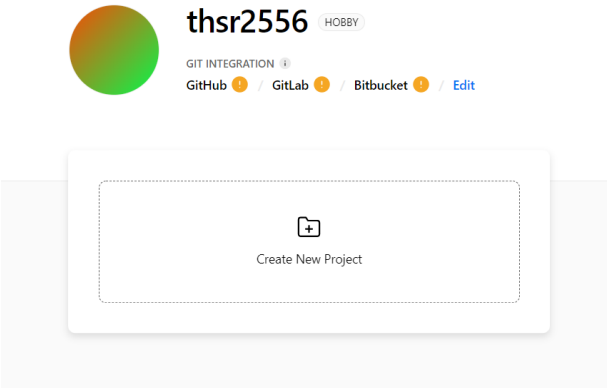

Figure 4: Dashboard for the software

If user plans to create a new project, he/she should be presented a new page for creating projects as in figure 5 and figure 6. The user should again have multiple options to create project using github, gitlab, bitbucket or command line. If the user wants to create project using gitlab but has not integrated gitlab to zeit, user should be provided one click integration button. Once the user has integrated, New Project from Gitlab option should be displayed in place of previous integration button.

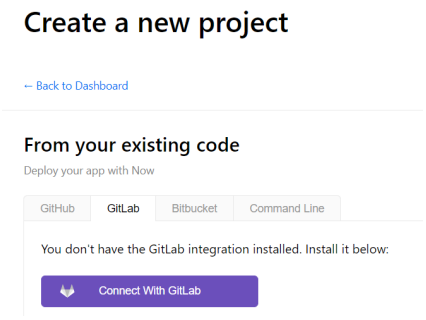

Figure 5: Project Creation -Stage 1

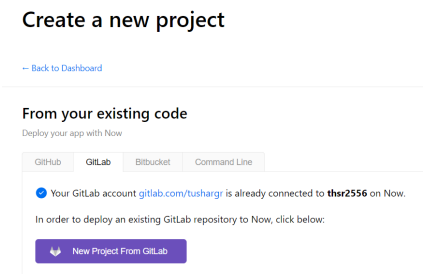

Figure 6: Project Creation- Stage 2

On choosing New project from Gitlab, a new page showing gitlab repositories should be displayed so that user can choose the repo that should be used for deployment. After the user has selected a repo, user should be presented with various deployment command as shown in figure 7 and 8. After providing minimal build instruction, the user can deploy the app.

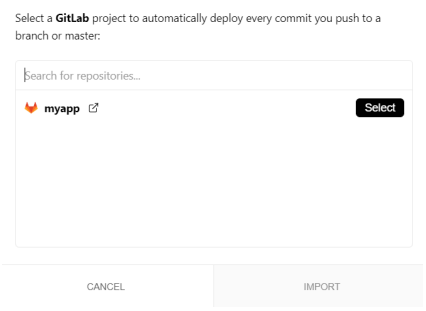

Figure 7: Selecting external repo for deployment

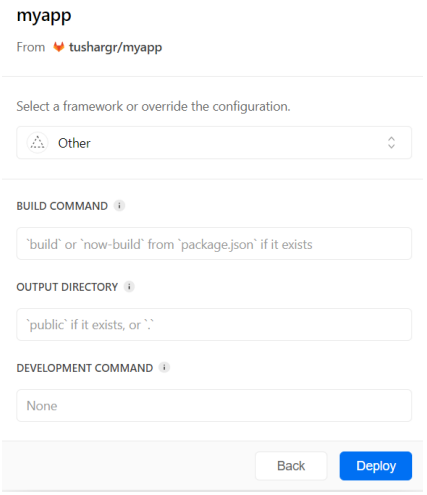

Figure 8: Build commands for deployment

This deployed app should now be available on dashboard showing its status. Similar procedure should be followed for deploying using github and bitbucket also.

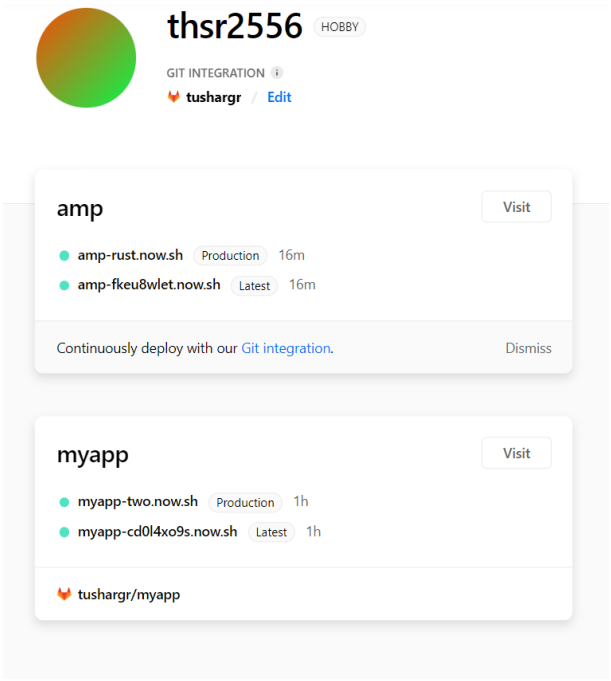

Figure 9: Dashboard showing status of deployed applications

Other than the web portal, ZEIT Now should also provide a Command Line Interface to the user to deploy and manage project.

For deploy and manage using CLI, the user should first login into ZEIT account from the command line using the command shown in figure 10.

```
user@DESKTOP-4LO3737 MINGW64 ~/test/amp
$ now login thsr2556@gmail.com
- Sending you an email
> We sent an email to thsr2556@gmail.com. Please follow the steps provided
  inside it and make sure the security code matches Snowy Manta Ray.
- Waiting for your confirmation
√ Email confirmed
> Congratulations! You are now logged in. In order to deploy something, run `now`.
```

Figure 10: Loggin in using CLI

After login, the user can either use example project provided by ZEIT or can create his/her own project.

Finally the user can build and deploy using a single command as shown below.

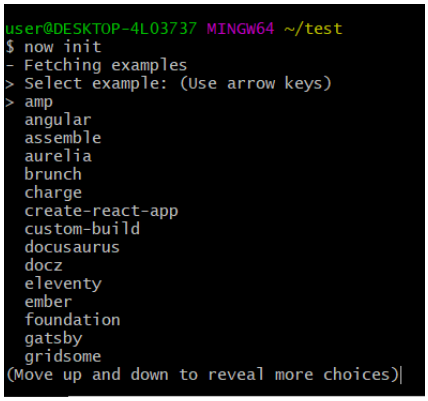

Figure 11: Initializing Example Project-1

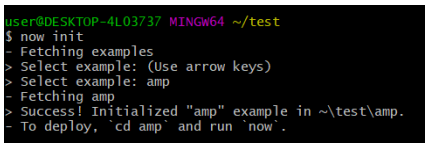

Figure 12: Initializing Example Project-2

Figure 13: Deploying Project

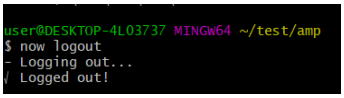

Figure 14: Logging out

ZEIT should also provide APIs to the user to simplify the hosting procedure. The APIs that should be provided by the software are listed below:

- **User Login**: This API should be used by the user to login for accessing various other APIs. This should use two-factor authentication technique to ensure secure login.
- **User Information**: This should retrieves information about the authenticated user. The information should include uid, email, name, billing details, hosted application along with the urls.
- **Upload file**: This api should be used to upload file to server that will be required later for deployment. The api request should also contain hash of the file content to ensure itegrity of file on the server side. After successful receiving of file, server should send ok status back to the client.
- **Deploy**: This should start the deployment of the user application. The files required for deployment should be sent using Upload file api before using this api call. User should be able to send various project settings through the request.
- **List Deployment**: This should simply return user a list of all deployment that are active for the authenticated user. The list should include hosted url for each deployment.
- **Delete a Deployment**: User should be able to delete a deployment using this api by just providing the deployment id or hosted url of the deployment.
- **Get Build Logs**: This api should provide user with the last build logs of the deployment by taking a deployment id from the user.
- **List all domains**: This API should retrieves a list of domains registered for the authenticating user.
- **Add a new domain**:This API should allow user to register a new domain name with ZEIT for the authenticating user. The user can add a domain name provided by ZEIT or an external domain. The server should verify the domain and should return verified status back to the user after making successful entry for the domain.

- **Remove a domain**: This should allow user to delete a previously registered domain name from ZEIT. This will also automatically remove any alias associated with the domain.
- **Check a domain availability**: This API should allow user to check if a domain name may be available to buy or not. The server should response with a true or false.
- **Check price of domain**: This should allow user to check the price of domain that the user wishes to buy. Server should respond with price as well period for which the domain can be purchased.
- **Purchase a domain**:This should allow user to purchase the specified domain. If the domain name is not available, then an error should be returned. In case of a successful purchase, server should returns with success code.

3.2 Hardware Interfaces

The application serves the purpose of allowing and facilitating **Static-Web Hosting** to any user.

Since neither the web-application interface nor the command-line tool have any designated hardware, it does not have any direct hardware interfaces. The physical instances for hosting are managed by third party hosts such as *AWS* or *GCP*, and the network connections between different components of the production stack are managed by the underlying application's software on the server side, see section 4.4.

For the purposes of creation of a new static website, and/or maintenance of the said website which might include updating, inspecting or just accessing the website's files, the user interface or certain software interfaces can be used, see section 3.1.

3.3 Software Interfaces

The web application or the command-line tool communicates with the *ZEIT* server to allow user to perform a number of functions from hosting a new project to deleting a previously hosted project.

For the purpose of hosting a new project, user can provide its online *Github*, *Gitlab* or *Bitbucket* repository. The user can even launch a local project using the CLI. The maintenance of the website or inspection of the logs is managed by the same user-interface or CLI.

The application server communicates with the third party organisations i.e. *AWS* or *GCP* for management of compute or physical instances via their specified APIs.

The communication between the database and the application server consists of operation concerning both reading and modifying the data, this takes place via TCP/IP methods.

3.4 Communications Interfaces

The communication between the user and the application server which is accomplished via the user-interface or the CLI should make use of a universally accepted protocol which is secure, fast and easily available, for this purpose we will use *HTTPS* and *SFTP*.

All the communication from the user-interface (i.e. web-application) will be via HTTPS, hence the user's browser must support HTTPS v1.1 or higher.

To support functionality of file transfer via upload or from an online repository for hosting the project or making changes to it, we will be using the SFTP protocol.

All the communication amongst the internal services and databases will be via TCP/IP and the choice of this protocol does not affect the user so it can be arbitrary based on the requirements, see section 4 for more details.

4 System Features

4.1 User Authentication

There are identity forms available on ZEIT, called ”scopes”: Users and teams. When you signed up for our service the first time, you have created a user account. With this type of identity, you can enjoy all the features ZEIT Now has to offer and access to the data created by those features is limited to just your user account.

4.1.1 Signup

Use will be able to signup via Email-Password transferred via *HTTPS* only. Alternatively, user will be able to signup with Social Authentication which will include GitHub, BitBucket and GitLab.

4.1.2 Secure Login

Use will be able to login via Email-Password transferred via *HTTPS* only. Alternatively, user will be able to login with Social Authentication which will include GitHub, BitBucket and GitLab.

4.1.3 Logout

User will be able to logout of the account both on CLI and web.

4.1.4 Change User Password

User will be able to change the account's password. An email will be send on associated account's email which will contain password reset link. User will be prompted to create a new password for the account.

4.2 Pipeline Management

This feature allows user to automatically test, build and deploy code with ease. For this feature, users need to connect their existing platform account with at least one of GitHub, GitLab, BitBucket. This feature consists of the following sub-features:

4.2.1 Source Code Management

Based on pipeline structure defined in now configuration, users will directly be able to deploy the changes pushed in the hooked repository. This feature will directly pull code from the repository over SFTP and cooked for deployment.

4.2.2 Continuous Integration

The platform will be able to automatically determine the framework of the code and prepare its build environment to run tests, build tasks, generate artifacts and prepare deployment. The process will be initiated via webhooks provided by one of GitHub, GitLab or BitBucket.

The artifacts of the deployments' of Free users will be available publicly.

4.2.3 Continuous Deployment

After Continuous Integration process, the generated artifacts will be deployed in a virtualised filesystem (see section 4.4) pushing files inside a designated environment. The platform will randomly assign a domain, generate SSL certificates for HTTPS and store them for the user.

4.3 Users and Teams

There are identity forms available on ZEIT Now, called ”scopes”: Users and teams. When you signed up for our service the first time, you have created a user account. With this type of identity, you can enjoy all the features ZEIT Now has to offer and access to the data created by those features is limited to just your user account.

4.3.1 Create a Team

Once they have created their own user account, you can create a team by clicking on the scope selector on the top left, and then choosing ”Create a Team”

4.3.2 Invite Link

In order to invite another person, click the ”INVITE” button on the right and enter the email address of the user account of the person you would like to invite.

4.3.3 Switching between Users and Teams

If you are part of one or multiple teams, you can choose which to show project information for or manage from the ZEIT Dashboard.

4.3.4 Changing the User or Team Settings

Just like different scopes have different resources contained within them (access is limited to people that are members of the scope), they also allow for different configurations.

4.3.5 Selecting a Scope for API Requests

When communicating with our API, all interactions only consider the resources contained within your own user account by default. If you want to access a team’s resources, you first need to find out the ID of the team.

4.4 Virtualisation

Virtualisation is implemented at two levels to simulate isolated environment for multiple projects. The information i.e. files in one project should not be accessible by other projects by any way.

One key point to make note of here is that we are making an application for *Static-Web Hosting*, so there is no need of 2 different machines or containers for hosting 2 different projects.

Hosting multiple projects on the same machine makes it more efficient and cost-effective to manage.

Although, due to functionality of application which allows user to view any files by specifying their route, it becomes necessary to implement isolation of files between 2 projects, see section 4.4.1.

Another important point to take care of if 2 projects are hosted on the same machine is correctly forwarding incoming connections to desired projects without any overlaps, see section 4.4.2.

4.4.1 File-System

The files associated with one project should not be accessible by any other project by any route possible, to implement this, we will use the concept of *Chroot Jailing*.

A chroot operation changes the apparent root directory for a running process (in our case project’s process) and its children. It allows us to run a program (process) with a root directory other than /. The program cannot see or access files outside the designated directory tree, allowing complete isolation from the rest of the file-system.

This artificial root directory is called a *chroot jail*.

4.4.2 Network

When multiple projects are hosted on the same machine or container, the incoming connections should be routed correctly to the corresponding project, i.e. each incoming connection should be identified and sent to the designated project only, to implement this, we will use the *Network Address and Port Translation (NAPT/NAT)* method.

All the incoming connections are identified by the server name and information they were trying to access, this information can be domain name, route. After identifying the connection, it is sent for further processing to the process corresponding to its designated project, which is listening on a specific port on the said machine or container.

This translation process is called *NAPT*.

4.5 Domain and Ingress Management

Domains are identifiers which map to IP addresses over the internet and when hosting a website, you need to have a domain associated with it for others to access.

Our application gives users functionality for *Static-Web Hosting*, and for this purpose, we need to provide users with domains on which their websites are going to be hosted, this task will be fulfilled in 2 steps.

4.5.1 Automatic Domain Generation

Each time a new project is deployed or an already existing project is updated to a new one, a new domain will be generated automatically for that project and user can use this domain to access the website.

4.5.2 Domain Selection (Premium Feature)

If a certain user wants a specific domain name, then he/she can purchase one if it is not already in use, and associate their project with this domain name. This allows other people to access the said website with the purchased domain name.

4.5.3 Ingress Management

A user can associate any number of domains to a project, in all of such cases, the other domains will work as an alias. This will allow users to buy domains from any merchant and use those domains to host their project using our web application.

4.6 Certificates and Secrets Management

Certificates and Secrets are parts of the project which need special protection as they are entities which make the website secure in multiple ways.

4.6.1 Certificates

Each website has a certificate associated with it which is used to identify whether a website is legitimate, i.e. the user won't be a victim of *phishing* and it also ensures that all the communication between the user and the website is encrypted and no attacker in the middle can decipher the information.

Our web-application will automatically issues and renews certificates for each project and they will be stored in our own internal database. Users won't need to create or manage certificates for their projects.

4.6.2 Secrets

Secrets are parts of website which cannot be directly added to the code, these can be passwords, keys or maybe some other privileged information. Our application allows users to add these secrets via the CLI directly into the project environment or they can be added in the form of a secret file supplied with the project.

4.7 Project Maintenance

Once the project has been deployed, the users would need to access information regarding the deployments, logs or just make updates to the project settings, for these purposes, we will provide diverse features to the users via CLI and dashboard.

4.7.1 Deployments Information

The user must be able to retrieve any information regarding his/her projects, major ones are as listed below:

- Listing all of the projects deployed by a user himself/herself.
- Listing all the deployments associated with his/her project.
- Retrieving information for an associated deployment/project.
- Viewing the entire deployment/project history.

4.7.2 Removing Deployments

The user should be able to delete a deployment corresponding to a project by using the CLI or dashboard.

Even after deletion, it should be possible to view its history and logs for a certain period which the user can define as per his/her preference for the project. Also Removing a deployment should also make its current domain free for use.

4.7.3 Logging

Logging is a very important feature for any kind of application which exists out there.

The user should be able to view the HTTP Access logs, error logs and output logs (if any) at anytime.

Although, logs can get quite large, so each user will be allocated a size limit for the logs after which the older logs will not be preserved, this limit can be increased for premium users.

4.7.4 Scaling

Hosting a website will certainly take care of dealing with a lot of traffic, each website a user hosts can have a large number of visitors and to facilitate good browsing experience for the end users, the website loading time should not be high.

The user can scale the number of deployments hosting a website according to his/her needs to facilitate good browsing experience.

4.7.5 Freeze/Unfreeze Deployments

If a user does not need a deployment for certain period of time then instead of deleting the deployment and re-creating it afterwards, we will provide the functionality of freezing the deployment and unfreezing it later.

This is like saving the snapshot of the deployment and loading it back from the snapshot without the need of going through deletion and creation processes, also the deployment will pick up things from whence it left off.

4.8 Payment Gateway

A third-party secure payment-gateway, CCAvenue will be used to process payments for users buying new domains via the platform.
Payment can be made through credit cards, debit cards, net banking in 27 major foreign currencies. There will be no option for EMI. After a successful purchase of domain, the money will be transferred to ZEIT's merchant account.

5 Other Nonfunctional Requirements

5.1 Performance Requirements

- **Load Balancing**: There should be Content Delivery Network (CDN) in addition to the hosting servers. CDN should be used for the quick transfer of assets needed for loading Internet content. There should be segmentation of the users based on their geographical location and the requests should be sent to nearest edge server.
- **Multi-Cloud Platform**: Many cloud services should be used including AWS, Azure and Google Cloud to leverage specialized features offered by these services. For user application deployment, ZEIT should choose from these three cloud services based on the location and should also consider which cloud service matches the application requirement the most.
- **Compression**: The network layer should implement compression methods- Gzip and Brotli. These compression methods will minimize the network bandwidth used and thus should increase the performance of the system. Any client that is making requests for deployment should define Accept-Encoding header to leverage the benefits of these compression methods.
- **Caching**: Zeit should cache the content so that it can serve data to users fast. There should be automatic static caching for all the deployments. This implies that the user should not be required to make any changes in the header of the request. Zeit should support caching broadly at 3 levels-
  1. **L1** - small cache for each process.
  2. **L2** - shared cache for all processes on a single node.
  3. **L3** - regional cache which should be shared with all nodes in a region.

5.2 Security Requirements

- **File System Virtualization**: Jailing should be used for virtualizing the file system to allow many applications run on the same deployment machine. This File system virtualization will prevent various scenario of malicious access and corruption of other application source code.
- **HTTPS support**: All the deployments that are done on ZEIT should be served over HTTPS connection. ZEIT should automatically generate the SSL certificates for the deployed application urls without any charge. Wildcard certificates should be produced for each deployment and should be automatically updated. Any HTTP request coming for an application should be automatically forwarded to HTTPs using 308 status code.
- **Two-Factor Authentication**: Two-factor authentication could be used whenever the user access their account on the web portal on basis of user discretion. Other than this, this should again be used at the time of pushing new code for deployment. There shouldn't be two factor authentication while pulling deployed code to make it convenient for multiple developers accessing the code.
- **STFP support**: SFPT should be used in place of usual ftp for uploading and downloading source code to and from the deployment machine. This should make ZEIT free from various network channel attacks.

5.3 Software Quality Attributes

- **Reliability**: The Content Delivery Network and the Multi-Cloud Platform should ensure fault tolerance. The hosted applications should be properly accessible and all its feature should work correctly even when some of the hosting servers or cloud service are down.
- **Availability**: Client should have an active internet connection. The load balancing architecture should ensure that the hosted applications are available all the time for development purpose as well as for end user accessing.

- **Maintainability**: The code written for ZEIT should follow good coding standards. There should be a separate file listing various standards and the code should follow these standards. Coding standards may include rules for indentation, rules for naming variable. Version Control System should be used to have access to various working version of the software.ALso the code should be written in a modular manner with proper mapping between deifferent components and modules.
- **Portability**: Portability should be ensured by using version control system. The design should not be monolithic rather the whole system should be broken into micro-services. The software should be platform independent. The software may not be portable to android for now.
- **Reusability**:The code for the software should be reusable. The code should be easily understandable and should be modular. There should not be any redundancy in the software code. Each function should have a comment describing what it does.
- **Testability**:Code writing should be done in a functional way. And for each function in the software, there should be many unit tests checking basic features provided by the software.

6 Appendix

6.1 Future Releases

There are a few functionalities that we plan to provide to the user of **ZEIT Now** in the future that can really help smooth-en the end user's experience.Some of them are listed Below.

- **Deploy Button:** It can be used to create a ZEIT project for any GitHub repository or GitLab project, including branches and subdirectories, and optionally create a copy of the original repository in user's Git account. The **Deploy Now** button could be rendered with both Markdown as well as HTML file.
- **Deploy Hooks:**Deploy Hooks would allow the user to create URLs that accept HTTP POST requests in order to trigger deployments, to re-run the Build Step, from outside ZEIT. These URLs would be uniquely linked to user's project, repository, and branch, so there is no need to use any authentication mechanism or provide any payload to the POST request.
- **Constant Domain Deployment:**Currently with every build of a project,a new temporary domain is being assigned to the user where his site is hosted.We would provide a domain that remains same for all deployments of a project.
- **Integrations Marketplace:** We plan to provide a marketplace that would allow the user to automate and extend his workflow by using integrations for his favourite tools.Some integrations that we plan on to provide soon are listed below.
  - **Slack:** Integration with Slack would let the user receive ZEIT events in his/her slack account.
  - **Google Cloud:** Would Provide support for Cloud scheduler,Cloud SQL,Firestore etc.
  - **Lighthouse:** Would automatically run audits for our deployments