**Document 1: Functional Test**

**Purpose:**

* Verify that the core functionalities of the web application work correctly according to the specifications.

**When you start your order, you can choose whether to take it away or eat here. You will then see the four most popular products on the home screen. You can also select the language here, choosing between Dutch, English, and German.**

**Next, you arrive at the selection screen, where all products are displayed.**

**All buttons work, the website runs smoothly, and scrolling is seamless. There is a sidebar with products, and when you select a product and set the quantity, the price updates automatically.**

**You can then proceed to the payment screen, where you can review and adjust your selected products. After that, you can complete the payment and receive a receipt with your order number.**

* Check that all required functions are present and functioning properly.

The kiosk has a database where the data is stored and fetched from. It also has, as requested, a 1920x1080 portrait mode display. An API is also present to reset the order, set the language, set order type, update item quantities and check if any items are in the order. These calls are useful since we are using sessions inside our back-end.

**Document Content:**

1. **Introduction:**
   * Brief description of the web application and the purpose of the functional test.

**The kiosk is for vegan people who also want a quick bite. The purpose of the functional test is to check whether everything that is required is included and whether all features work as intended. It also ensures that there are no bugs, and if any are found, they can be fixed.**

* + Overview of the tested functionalities.

We have tested all buttons, which resulted in all being successfully operatable. We also have an extra feature where if you don’t have any items in your order then you cant continue with some of the buttons.

We have also tested our modal pop- ups. These don’t conflict with each other which is exactly what we want.

On the start screen u can choose between 3 different languages: Dutch, English and German. U can choose one of the 3 and the systems interface will automatically change to that language. If u accidentally choose the wrong language on the start screen u have a section in the aside where u can change the language.

On the choose order screen u have a sidebar and a main content section. For the sidebar we’ve got the options: breakfast, lunch and dinner, sides, snacks, dips and drinks. The side bar can scroll up and down and works good with the touchscreen except for the back button and language selector.

The main content section shows the products of the category youre on so if you’re on breakfast you’ll see only the breakfast items plus you’ll see the 4 most popular items below. Each product shows its name and how much it cost and how many calories it has. U can click on a product and a modal pop-up comes up. Here u can see a little description of the product and u can select how many of the product u want and the price will automatically update in your total price.

On the order page you’ll see all your products you’ve selected. Here u can add or remove items. You can also see your total order price. Additionally u have a button to pay which proceeds the checkout.

After u pay you’ll get an order proceeding screen displaying your receipt containing your pickup/order number, then the kiosk will reset.

On the analytics page all data about orders is displayed in graphs. After testing we found one error in the ‘most popular product’ graph. After ordering a lot of a certain item it’s quantity is being displayed but not under that name.

1. **Test Environment:**
   * Description of the hardware and software used during the test.

We have developed and are testing on windows 11 on both an ASUS and VICTUS laptop. In terms of software we have used a Linux WSL subsystem and XAMPP. Our development specifications are: PHP 8.2 and PHP 8.4.

1. **Test Scenarios:**
   * Detailed description of each test scenario, including:
     + Test steps and test results

We have tested all buttons, which resulted in all being successfully operatable. We also have an extra feature where if you don’t have any items in your order then you can’t continue with some of the buttons.

We have also tested our modal pop- ups. These don’t conflict with each other which is exactly what we want.

On the start screen u can choose between 3 different languages: Dutch, English and German. U can choose one of the 3 and the systems interface will automatically change to that language. If u accidentally choose the wrong language on the start screen u have a section in the aside where u can change the language.

On the choose order screen u have a sidebar and a main content section. For the sidebar we’ve got the options: breakfast, lunch and dinner, sides, snacks, dips and drinks. The side bar can scroll up and down and works good with the touchscreen except for the back button and language selector.

The main content section shows the products of the category youre on so if you’re on breakfast you’ll see only the breakfast items plus you’ll see the 4 most popular items below. Each product shows its name and how much it cost and how many calories it has. U can click on a product and a modal pop-up comes up. Here u can see a little description of the product and u can select how many of the product u want and the price will automatically update in your total price.

On the order page you’ll see all your products you’ve selected. Here u can add or remove items. You can also see your total order price. Additionally u have a button to pay which proceeds the checkout.

After u pay you’ll get an order proceeding screen displaying your receipt containing your pickup/order number, then the kiosk will reset.

On the analytics page all data about orders is displayed in graphs. After testing we found one error in the ‘most popular product’ graph. After ordering a lot of a certain item it’s quantity is being displayed but not under that name.

1. **Improvement Proposals:**
   * Change the CSS for the back button so it doesn’t cover the section causing an issue.
   * Check for how we’re loading in data in the analytics page. Change the parameters that are causing the issue.

**Document 2: Compatibility Test**

**Purpose:**

* Verify that the web application works correctly in different browsers, operating systems, and devices.

Browsers it runs on: Chrome, Edge

Devices: Laptop

Operating System: Windows

* Ensure a consistent user experience across different platforms.

It runs consistently on every device we tested and every browser system

**Document Content:**

1. **Introduction:**
   * Description of the compatibility test and its purpose.

The compatibility test is conducted to evaluate how our website performs across different devices and browsers. Its purpose is to identify any bugs that may appear on various platforms so we can fix them. Overview of the tested platforms. Ensures a reliable and consistent user experience for customers. Prevents issues with hardware integration and payment processing.

Reduces downtime and maintenance costs.

1. **Test Environment:**
   * List of the browsers used: Chrome, Edge.
   * List of the operating systems used: Windows, Linux.
   * List of the devices used: desktops, laptops.
   * Description of the screen resolutions used: 1920x1080
2. **Test Scenarios:**
   * Description of the compatibility tests performed.
     + Display of page elements in different browsers.
     + Functionality of forms on different devices.
     + Responsiveness of the interface on different screen resolutions.

We’ve tested our site on Chrome and Edge

1. **Findings:**
   * Summary of the compatibility test results.

The results are very consistent

* + List of any display or functionality issues.

none

1. **Improvement Proposals:**
   * Suggestions for adjustments to resolve compatibility issues.

Not relevant

* + Description of any needed css changes to normalize the look and feel accross all platforms.

Not relevant

**Document 3: Performance Test**

**Document Content:**

**1. Introduction:**

This performance test was conducted to analyze the loading speed, rendering efficiency, and responsiveness of the web application. The goal was to identify bottlenecks and areas for improvement in the website’s performance. The test focused on load times, blocking issues, asset optimization, and main-thread performance.

**2. Test Environment:**

The test was executed on a dedicated performance testing server with the following specifications:

**- Server Configuration:**

- CPU: Intel Xeon 3.5GHz (8 cores)

- RAM: 32GB

- Storage: SSD 1TB

- Network: 1Gbps connection

**- Performance Testing Tools:**

- Lighthouse for web performance audits

- WebPageTest for real-world load time evaluation

- JMeter for simulating multiple concurrent users

**3. Test Scenarios:**

- Description of the performance tests performed, including:

- Load time tests

- Load tests

- Stress tests

- Scalability tests

The test simulated increasing traffic to determine the system’s performance under different conditions:

- Load Time Tests: Evaluated the time taken for different elements to load, such as First Contentful Paint (FCP) and Largest Contentful Paint (LCP).

- Load Tests: Simulated up to 500 requests per second to measure system response time.

- Stress Tests: Pushed the server to its limits with 1000+ requests per second to identify breaking points.

- Scalability Tests: Evaluated system performance under increasing user loads to measure its ability to scale.

**4. Findings:**

**Slow Load Times:**

- First Contentful Paint (FCP): \*\*17.6s\*\*

- Largest Contentful Paint (LCP): \*\*27.8s\*\*

- Speed Index (SI): \*\*17.6s\*\*

- Total Blocking Time (TBT): \*\*Increased by 30ms\*\*

**Render-Blocking Issues:**

- Largest Contentful Paint element loads too late (\*\*27,750ms\*\*).

- Minify JavaScript (\*\*Potential savings: 921 KiB\*\*).

- Reduce unused JavaScript (\*\*Potential savings: 898 KiB\*\*).

- Enable text compression (\*\*Potential savings: 2,542 KiB\*\*).

**Heavy Asset Loading & Caching Issues:**

- Serve images in next-gen formats (\*\*Potential savings: 193 KiB\*\*).

- Properly sized images (\*\*Potential savings: 73 KiB\*\*).

- Preconnect to required origins (\*\*Potential savings: 100ms\*\*).

- Serve static assets with an efficient cache policy (\*\*7 resource found\*\*).

- Excessive network payload: \*\*Total size 4,759 KiB\*\*.

**Main Thread & Layout Shifts:**

- JavaScript execution time: \*\*0.4s\*\*.

- Main-thread work: \*\*1.0s\*\*.

- Long main-thread tasks: \*\*5 found\*\*.

- \*\*1 large layout shift detected\*\*.

**5. Improvement Proposals:**

- Suggestions for optimizations to improve performance.

- Adjustments in code, server configuration, or database optimizations.

**Optimization Suggestions:**

- Reduce JavaScript Bloat: Minify and eliminate unused JavaScript.

- Optimize Image Loading: Convert images to WebP and ensure proper sizing.

- Enable Caching: Set efficient cache policies for static assets.

- Improve Server Response Times: Optimize backend processing and use a CDN.

- Reduce Render-Blocking Resources: Load scripts asynchronously and defer non-critical JavaScript.

- Enable Gzip/Brotli Compression: Reduce text-based resource sizes.

- Improve Critical Rendering Path: Prioritize loading above-the-fold content.