Beta Test Plan

1. Introduction

Hestia is a mobile app that requires beta-testing to ensure that it's functionalities corresponds to it's user base and their needs.

2. Selection of core functionalities

Core functionalities that will be available in the beta version. These are based on the Action Plan and refined based on project evolution, feedback, and new requirements.

2.1 Core functionalities

- **Account creation**: The ability to create a Hestia user account by registering with a google account.
- **Account configuration**: Creating or joining an existing collocation, setting personal information.
- **Wall page**: Publish posts for other users in the same colocation to see, see other users posts, remove posts.
- **Tasks page**: Set tasks and their deadlines, set the people assigned to said task, set tasks to `done` so they get deleted automaticaly.
- **Settings page**: Change the language of the app, the notification prefferences, send a bug report or a feature request/modification.
- **Budget page**: Create categories, input expenses and redistribute them to a set of users. See the owed money of each users and confirm refunds to set owed money to 0.
- **Groceries page**: Set grocereies lists and their contents, visible to all other people in the same group. Check each elements of each shopping list.

2.2 Adaptations and refinements

- Removal of task priority selection button, not usefull enough and confusing for users.

---

3. Definition of beta testing scenarios

Structured test scenarios to assess the functionality of key features. Each scenario include the user role, feature being tested, expected outcome, and steps to execute the test.

3.1 Test scenario 1 ✔

- **End-user**
- **Connection**
- **Precondition**:
  1. The user needs to launch the application
- **User was able to connect**
- **Steps to execute**:
  1. User touches google connection button
  2. User connects to their google account through google's own login page
  3. User confirms the connection of the account

3.2 Test scenario 2 ✔

- **End-user**
- **Add a post**
- **Precondition**:
  1. The user is logged in
- **Post is added and is readable by other accounts in the same collocation**:
- **Steps to execute**:
  1. User open the app
  2. User goes to the post page by touching the fridge icon
      3. User touches the “+” button
      4. An input form appears, allowing the user to fill in the post content and color
      5. User fill the blank and click on “post”
      6. the post is added to the page

3.3 Test scenario 3 ✔

- **End-user**
- **Delete a post**
- **Precondition**:
  1. The user is logged in.
- **Post is deleted and the page refreshes**:
- **Steps to execute**:
  1. User opens the app.
  2. User goes to the post page by touching the fridge icon
  3. User touches the “x” button next to the post they want to delete
  4. The post is deleted, and the page refreshes

3.4 Test scenario 4 ✔

- **End-user**
- **Go to settings page**
- **Precondition**:
    1. The user is logged in
- **User is redirected to the settings page**:
- **Steps to execute**:
    1. User opens the app
    2. User touches the cog icon
    3. User is redirected to the settings page

3.5 Test scenario 5 ✔

- **End-user**
- **Go to money page**
- **Precondition**:
    1. The user is logged in
- **User is redirected to the money page**:
- **Steps to execute**:
    1. User opens the app
    2. User touches the coin icon
    3. User is redirected to the money page

3.6 Test scenario 6 ✔

- **End-user**
- **Go to task page**
- **Precondition**:
    1. The user is logged in
- **User is redirected to the task page**:
- **Steps to execute**:
    1. User opens the app
    2. User touches the task icon
    3. User is redirected to the task page

3.7 Test scenario 7 ✔

- **End-user**
- **Go to groceries page**
- **Precondition**:
    1. The user is logged in
- **User is redirected to the task page**:
- **Steps to execute**:
    1. User opens the app
    2. User touches the basket icon
    3. User is redirected to the groceries page

3.8 Test scenario 8 ✔

- **End-user**
- **Switch language**
- **Precondition**:
    1. The user is logged in
    2. The user is on the settings page
- **Language is switched to the selected language**:
- **Steps to execute**:
    1. User opens the app and navigates to the settings page
    2. User clicks on the language button
    3. User selects another language from the list
    4. The application language is updated to the selected language

3.9 Test Scenario 9 ✔

- **End-user**
- **Add an item to the groceries list**
- **Precondition**:
    1. The user is logged in.
    2. The user is on the Groceries page.
- **Expected Outcome**:
    - The item is added to the groceries list and visible to all users in the same collocation.
- **Steps to execute**:
    1. User opens the app and navigates to the Groceries page.
    2. User clicks any existing grocery list.
    3. User clicks the “+” button to add a new item.
    4. A form appears, prompting the user to enter the item name (e.g., "Milk").
    5. User fills in the item name and clicks "post".
    6. The item appears in the groceries list.

3.10 Test Scenario 10 ✔

- **End-user**
- **Mark a grocery item as "purchased"**
- **Precondition**:
    1. The user is logged in.
    2. There is at least one item in a groceries list.
- **Expected Outcome**:
    - The item is marked as "purchased" (the checkbox asside the item is checked).
- **Steps to execute**:
    1. User opens the app and navigates to the Groceries page.
    2. User click a grocery list containing at least an item.
    3. User clicks the checkbox next to an item (e.g., "Milk").
    4. The item is visually marked as completed (checkbox is checked).

3.11 Test Scenario 11 ▯

- **End-user**
- **Delete an item from the groceries list**
- **Precondition**:
    1. The user is logged in.
    2. There is at least one item in the groceries list.
- **Expected Outcome**:
    - The item is removed from the list.
- **Steps to execute**:
    1. User opens the app and navigates to the Groceries page.
    2. User clicks on an existing list of groceries.
    3. User clicks the "▯" (delete) button next to an item.
    4. A confirmation dialog appears.
    5. User confirms deletion.
    6. The item is removed from the list.

3.12 Test Scenario 12 ✔

- **End-user**
- **Add an expense in the Budget page**
- **Precondition**:
    1. The user is logged in.
    2. The user is on the Budget page.
- **Expected Outcome**:
    - The expense is logged, and the owed amounts are recalculated for all users.
- **Steps to execute**:
    1. User opens the app and navigates to the Budget page.
    2. User clicks on any expense category.
    3. User clicks the "+" button to add a new expense.
    4. User fills in:
        - Name (e.g., food)
        - User who payed (e.g., "John")
        - Selects which users share the expense
        - Selects how to split the expense (e.g., percentage, raw ammount)
    5. User clicks "post".
    6. The expense is added, and the owed amounts update accordingly.

3.13 Test Scenario 13 ✔

- **End-user**
- **Confirm a refund in the Budget page**
- **Precondition**:
    1. The user is logged in.
    2. There is at least one pending owed amount between users.
- **Expected Outcome**:
    - The owed amount is reset to 0 after confirmation.
- **Steps to execute**:
    1. User opens the app and navigates to the Budget page.
    2. User clicks the `balance` button.
    3. User clicks the `repay` button.
    4. User sees that another user owes them money (e.g., "Alice owes you €10").
    5. User clicks the `repay` button.
    6. A confirmation dialog appears.
    7. User confirms.
    8. The owed amount is reset to 0.

3.14 Test Scenario 14 ▯ (No error message, just ignores post request)

- **End-user**
- **Attempt to add an empty post**
- **Precondition**:
    1. The user is logged in.
    2. The user is on the Wall page.
- **Expected Outcome**:
    - The app prevents posting and shows an error message (e.g., "Post cannot be empty").
- **Steps to execute**:
    1. User opens the app and navigates to the Wall page.
    2. User clicks the "+" button to create a post.

3. User leaves the input field blank.
4. User clicks "Post."
5. An error message appears, and the post is not published.

3.15 Test Scenario 15 ✔

- **End-user**
- **Create a new task**
- **Precondition**:
    1. The user is logged in.
    2. The user is on the Tasks page.
- **Expected Outcome**:
    - The task is created and appears in the task list with the correct details and colour.
- **Steps to execute**:
    1. User opens the app and navigates to the Tasks page.
    2. User clicks the “+” button to add a new task.
    3. User fills in:
        - Task name (e.g., "Clean the kitchen")
        - Description (optional, e.g., "Wipe counters, mop floor")
        - Deadline (e.g., "June 30, 2025")
        - Assignee (selects a roommate from the list)
    4. User clicks "post".
    5. The task appears in the task list.

3.16 Test Scenario 16 ✔

- **End-user**
- **View task details**
- **Precondition**:
    1. The user is logged in.
    2. There is at least one task in the list.
- **Expected Outcome**:
    - The task details (name, description, assignee, deadline) are displayed correctly.
- **Steps to execute**:
    1. User opens the app and navigates to the Tasks page.
    2. User clicks on a task in the list.
    3. A details overlay opens, showing:
        - Task name
        - Description (if provided)
        - Assigned roommate
        - Deadline
    4. User clicks outside of the task overlay to make the details overlay dissapear.

3.17 Test Scenario 17 ✔

- **End-user**
- **Mark a task as "done"**
- **Precondition**:
    1. The user is logged in.
    2. There is at least one active (uncompleted) task.
- **Expected Outcome**:
    - The task is marked as completed and disappears from the active task list after a day.
- **Steps to execute**:
    1. User opens the app and navigates to the Tasks page.
    2. User clicks on a task to open the detail overlay
    3. User clicks the checkbox (or "Mark as Done" button) in the task overlay.
    4. The task is removed from the active list (after a day so other users can see completion of the task).

3.18 Test Scenario 18 ✔

- **End-user**
- **Enroll in an unassigned task**
- **Precondition**:
    1. The user is logged in.
    2. There is at least one task with no assignee.
- **Expected Outcome**:
    - The task is now assigned to the enrolling user.
- **Steps to execute**:
    1. User opens the app and navigates to the Tasks page.
    2. User clicks on an unassigned task (marked as "Unassigned" or similar).
    3. User clicks "Enroll !".
    4. The task is now assigned to the current user.

3.19 Test Scenario 19 ⌧

- **End-user**
- **Edit a task’s deadline**
- **Precondition**:
    1. The user is logged in.
    2. There is at least one task in the list.
- **Expected Outcome**:
    - The task’s deadline is updated and reflected in the list.

- **Steps to execute**:
  1. User opens the app and navigates to the Tasks page.
  2. User clicks on a task to open details.
  3. User clicks "Edit" (or equivalent).
  4. User changes the deadline (e.g., from "June 30" to "July 2").
  5. User clicks "Save".
  6. The updated deadline is visible in the task list.

3.20 Test Scenario 20 □ (No error, just ignore post request)

- **End-user**
- **Attempt to create a task with no name**
- **Precondition**:
  1. The user is logged in.
  2. The user is on the Tasks page.
- **Expected Outcome**:
  - The app prevents task creation and shows an error (e.g., "Task name is required").
- **Steps to execute**:
  1. User opens the app and navigates to the Tasks page.
  2. User clicks the “+” button to add a new task.
  3. User leaves the "Task name" field blank but fills in other details.
  4. User clicks "Save."
  5. An error message appears, and the task is not created.

3.21 Test Scenario 21 ✔

- **End-user**
- **Attempt to register an account with no group**
- **Precondition**:
  1. The user is on the register page.
- **Expected Outcome**:
  - The app opens the group creation/join page.
- **Steps to execute**:
  1. User opens the app and navigates to the register page.
  2. User does not fill a collocation ID and registers with their google account.
  3. The register succeeds.

3.22 Test Scenario 22 ✔

- **End-user**
- **Attempt to log out of an account**
- **Precondition**:
  1. The user is on the settings page.
- **Expected Outcome**:
  - The user is logged out and brought to the login/register page.
- **Steps to execute**:
  1. User scrolls down to the logout button on the settings page.
  2. User clicks the logout button.
  3. User is successfully logged out.

---

4. Coverage of key user journeys

Ensure the test scenarios reflect real-world use cases and cover all major interactions, including edge cases or failure points.

4.1 Key user journeys

- **Journey 1**: **New user installing the app** ✔
  - A new user downloads the app, creates an account using their Google account, they join a collocation, they set up their profile. They explore the wall page, add a post, and check the tasks page.
- **Journey 2**: **Daily usage** □ (Missing notification preferences)
  - A registered user logs in, checks the wall page for updates, adds a new post, assigns a task to a roommate on the tasks page, and updates their notification preferences in the settings page.
- **Journey 3**: **Language and settings management** □ (Missing notification preferences)
  - A user goes to the settings page, switches the app language, adjusts their notification preferences to customize their experience.
- **Journey 4**: **Setting cleaning duties** ✔
  - A user goes to the tasks page, sets multiple tasks and assigns them to multiple roommates in the house.
- **Journey 5**: **Shopping for two** ✔
  - User 1 creates a groceries list on the groceries page filled with multiple items, sets a task to go buy said groceries in the task page.
  - User 2 sees task and enrolls, goes out to get groceries, check each item in the list.
  - User 2 comes home, mark the task as `done`, goes to the money page and set a nex expense for the groceries.
  - User 1 sees the completed task and the new expense, reimburses User 2, then confirm reimbursement.
- [illegible]

- **Journey 6**: **Creating a new group** ✔
    - User 1 who allready has an account creates a new group for their colocation.
    - User 2 creates an account and joins the new group by specifying the collocation ID provided to user 1 during group creation.

4.2 Edge cases and failure points

- **Edge Case 1**: **Failed google account connection** ▯ (No displayed error if past google auth, fails conncetion)
    - **Description**: The user attempts to connect their Google account, they encounters an error (network issues)
    - **Testing**: - Simulate a failed connection by disabling internet access. Verify that the app displays an appropriate error message and allows the user to retry.
- **Edge Case 2**: **Post deletion by unauthorized user** ✔
    - **Description**: A user attempts to delete a post while not having propper credentials.
    - **Testing**: - Verify that the posts and their “x” button is only visible to authorized users. Ensure unauthorized users cannot delete or see posts.
- **Edge Case 3**: **Task assignment to non-existing user** ▯
    - **Description**: A user tries to assign a task to a user who is no longer part of the collocation.
    - **Testing**: - Simulate assigning a task to a deleted user. Verify that the app does not display the deleted user or displays an error message and prevents the assignment (eg. user existed when the app was getting the collocation users).
- **Edge Case 4**: **App crash during post creation** ▯
    - **Description**: The app crashes while the user is creating a post.
    - **Testing**: - Simulate an app crash during post creation. Verify that the app recovers and does not lose the user's input.
- **Edge Case 5**: **Task deadline in the past** ▯ (Task created with specified date, no error displayed)
- **Description**: A user sets a task deadline to a date that has already passed.
- **Testing**:
    1. Attempt to create/edit a task with a past date (e.g., "January 1, 2020").
    2. Verify the app either:
        - Rejects the input with an error (e.g., "Deadline must be in the future").
        - Automatically adjusts to the current date.
- **Edge Case 6**: **Expense with negative or zero value** ▯ (Field won't accept negative values, 0 amount is still created but no error)
- **Description**: A user tries to log an expense with a negative amount (e.g., -€10) or €0.
- **Testing**:
    1. On the Budget page, attempt to add an expense with:
        - A negative value (e.g., "-10").
        - A zero value (e.g., "0").
    2. Verify the app rejects the input with an error (e.g., "Amount must be greater than 0").
- **Edge Case 7**: **Grocery item with empty name** ▯ (Grocery item not created, no error displayed)
- **Description**: A user tries to add a grocery item with no name (empty field).
- **Testing**:
    1. On the Groceries page, in a groceries list, click the "+" button and leave the name field blank.
    2. Verify the app displays an error (e.g., "Item name cannot be empty") and prevents submission.
- **Edge Case 8**: **Network loss during task completion** ▯
- **Description**: The user marks a task as "done," but the app loses internet connectivity.
- **Testing**:
    1. Simulate network disconnection (e.g., enable airplane mode).
    2. Attempt to mark a task as completed.
    3. Verify:
        - The app shows a "No connection" error.
        - The task reverts to "incomplete" or syncs once connectivity is restored.
- **Edge Case 9**: **Assigning a task to a user who leaves mid-assignment** ▯
- **Description**: A user is assigned a task, but they leave the collocation before completing it.
- **Testing**:
    1. Assign a task to User A.
    2. Simulate User A leaving the collocation (remove them from the group).
    3. Verify:
        - The task is automatically unassigned or reassigned to another user.
- **Edge Case 10**: **Duplicate grocery items** ▯ (Not comparing text imput)
- **Description**: A user adds the same grocery item twice (e.g., "Milk" and "milk").
- **Testing**:
    1. Add "Milk" to the groceries list.
    2. Attempt to add "milk" (case variation) or "Milk" again.
    3. Verify the app either:

        - Merges duplicates (e.g., increments quantity).
        - Displays a warning (e.g., "Item already exists").

- **Edge Case 11**: **Currency mismatch in Budget page** ▯ (only euro support as of now)
- **Description**: Users in the same collocation have different device currencies (e.g., € vs. $).
- **Testing**:
    1. Set User A's device to EUR and User B's to USD.
    2. Have User A add an expense (e.g., €10).
    3. Verify User B sees the correct converted amount (or a warning about currency differences).
- **Edge Case 12**: **App crash during refund confirmation** ▯
- **Description**: The app crashes while confirming a refund in the Budget page.
- **Testing**:
    1. Initiate a refund confirmation.
    2. Simulate a crash mid-process (e.g., force-close the app).
    3. Reopen the app and verify:
        - The refund is either fully processed or reverted (no partial state).
        - Owed amounts are accurate.
- **Edge Case 13**: **Task with extremely long name/description** ▯ (Task name limited to set ammount of char)
- **Description**: A user creates a task with a 500-character name or description.
- **Testing**:
    1. Attempt to create a task with:
        - A 500-character name (e.g., lorem ipsum).
        - A 1000-character description.
    2. Verify the app:
        - Truncates the text with ellipses.
        - Or displays it correctly (no UI breakage).
- **Edge Case 14**: **Simultaneous edits to groceries list** ✔ (Handled with order operation, can not edit a deleted item, ignores the changes)
- **Description**: Two users try to edit the same grocery item at the same time.
- **Testing**:
    1. Have User A and User B open the groceries list.
    2. User A edits "Milk" to "2L Milk" while User B deletes "Milk".
    3. Verify:
        - Syncs changes correctly for both users.
- **Edge Case 15**: **Owed amount exceeds system limits** ▯
- **Description**: A user logs an expense with a value exceeding app limits (e.g., €1,000,000).
- **Testing**:
    1. Attempt to add an expense of €1,000,000.
    2. Verify the app:
        - Rejects the input (e.g., "Amount too high").
        - Or handles it correctly (no calculation errors).
- **Edge Case 16**: **Group with no users** ✔
- **Description**: All user accounts from a group have left a group or deleted their account.
- **Testing**:
    1. Create then leave a group.
    2. Verify the server:
        - Group and contents are deleted after a week.

---

5. Clear evaluation criteria

Define success metrics and baseline performance or usability expectations to determine whether a feature is ready for release.

5.1 Success metrics

- **Feature success**: Features succeed more then 98% of the time (ex: adding a post on the wall page succeeds, assigning user to a task succeeds). To get metric, send message from app to server upon success/failure of feature.
- **Responsiveness**: Server requests response time stay lower than 500ms for small request (not for big data transfers like images)

5.2 Baseline expectations

- **Usability Baseline**: 95% of testers rate the interface as "easy to use"
