BeakPeek

Millennium
University Of Pretoria

Algorithmic Innovation Award

In Partnership With
AgileBridge

Group:
22

Mentor:
Ayaaz

2024/10/18

Contents

1 Introduction

We as Millennium have from the start of the capstone project placed a high level of importance on DevOps which was managed by and maintained by Ivan Horak who has been our DevOps engineer.

Seeing that the scale of this project is so large we feel it is important that we give a summary of what BeakPeek is and how we have structured the repository and all of the different parts of how BeakPeek functions.

1.1 What is BeakPeek

BeakPeek is a bird watching app that intends to making birding as easy as fly.

Our core functional requirements are:

- Data ingress and transformation
- Usage of the data from SABAP (South African Bird Atlas Project) for
  - Heat map generation by pentad (A 5 minute by 5 minute area)
  - Bird population estimation
  - Bird quiz generation
- User management
- Bird Achievements
- Lifelists (list of birds that a user has seen)

1.2 Project/Repository structure

Our project has been split into a couple common directories. Those being:

- doc: For documentation
- res: For resources
- scripts: For bash and SQL scripts
- beakpeek: For our flutter frontend
- dotnet: For our two .Net backends
  - BirdApi: For the bird API
  - UserApi: For our user portal and management API

1.3 What we use?

The tools that we have used are:

- Azure to host the bird API and User API
- .Net entity framework core for both API's
- .Net identity for the User portal and API
- Flutter for mobile app development
- MSSQL for both databases also on Azure

1.4 Most Important things to note for DevOps

- We use Azure to host our 2 .Net backends/API's
- We use MSSQL for our database
- We have an Android release on Google Play
- We use GitHub actions for our CI/CD Workflows
- We get our data from SABAP

2 Non-Functional Requirements

We as Millennium set out for ourselve 3 main Non-Functional requirements that we thought would be the most important and applicable to BeakPeek and would best show our wide range of skills and show our abilities as software engineers.

2.1 Performance

One of the most important skills as software engineers is the ability to make performant and scalable solutions.

2.1.1 Load Testing

We use Azure Load Testing to simulate 50 users constantly signing up, logging in, getting their profile, updating their profile and then finally deleting their profile

To show that we can make highly performant and highly scalable solution we have used a load test which is hosted on azure in conjunction with our api's.

To ensure that our users are never unable to login and manage their profiles we used a load test to ensure that even if and when we have high volumes of users we never drop a request. This is done by simulating 50 users simultaneously and constantly signing up, logging in, getting their profile, updating their profile and finally deleting their account. This is done over 10 minutes which gives us approximately 2.49 thousand requests with an error percentage of 0.36%. This shows that our User API can handle a high qauntity of users easily.

2.1.2 Statistics

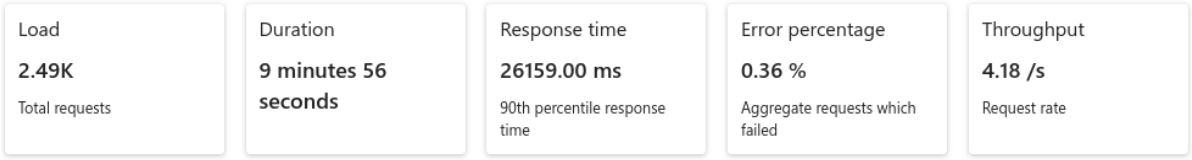

Figure 1: Load test metrics.

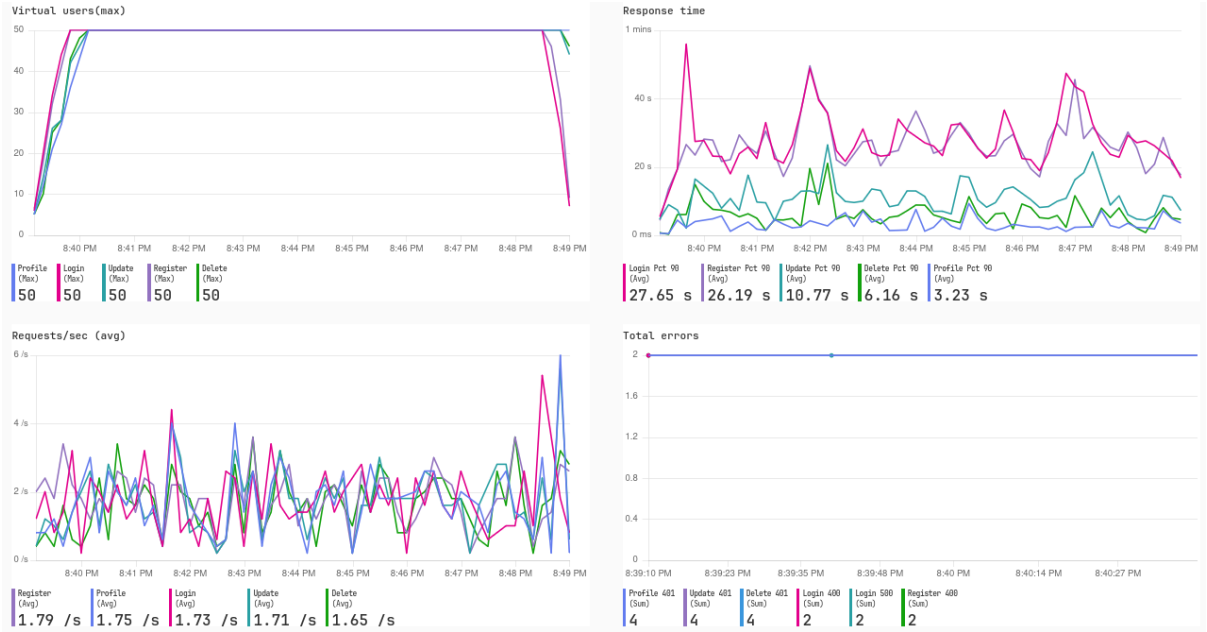

Figure 2: Load test graphs.

2.2 Security

To ensure our users security and ensure that their data stays secure and that only users who are authenticated are allowed to login and edit their data we have used Json Web Tokens which are an easy to transport and easy to use authentication token as it ensures that only users who have been issued a token from our user Api as anyone could decrypt a JWT but only JWTs issued by our user api are valid as they are validated against a secret key on our user API.

2.2.1 User Portal/Api

We used the .Net Identity framework for our user management because it is a widely used and trusted tool made by Microsoft which meant that it worked smoothly with the tools built into Azure and enabled a much more comprehensive DevOps experience.
Using Identity framework also allowed for easy integration of other login and web auth providers such as Google and also allowed easy role based authorization which was important as the user portal/api also serves as an admin portal allowing us to easily give manage our user, roles and Achievements which were also integrated into the user api along with the users life list and other user claims.

2.2.2 Usability

Usability Test Report

3 CI/CD tools and pipelines

3.1 Tools

To ensure the best possible developer experience we used a multitude of tools and integrations in the CI/CD pipelines.

Some of the most important tools were:

- Fastlane for easy builds and releases to Google play for our mobile app
- JMeter was used for load testing our Api
- Codecov for unit testing tracking
- Github actions for automated workflows and tasks

3.1.1 Fastlane

Fastlane is a tool made especially for the building and releasing of mobile apps to the Google Play store and to Apples App store. It is configured in ruby and has plenty of integrations, such as for deploying using firebase and of course building different flutter environments.

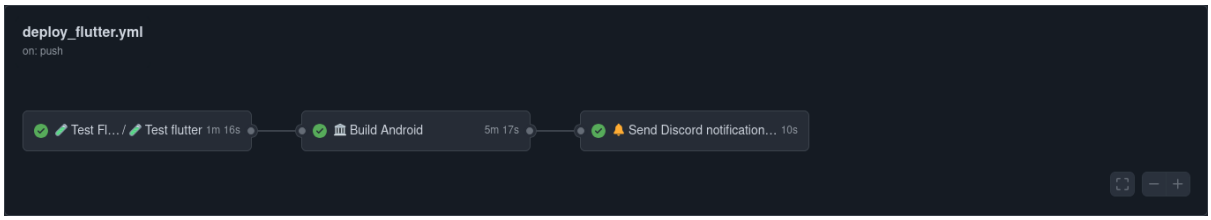

Figure 3: Android Deploy.

3.1.2 JMeter

JMeter was used primarily for our perfomance requirement as JMeter allows for easy configuration using different inputs such as CSV files which as useful for ensuring that every user that was created in the load test was a unique user and was consistent across different runs.

3.1.3 Codecov

To ensure that our the anything we make is up to the proper standards and that there are no unintended consequences of any new features we add we have employed the use of Codecov to be able to track our code coverage across our project and across the different sections of the project.

We have split up our code coverage tracking into two primary areas, the flutter frontend and the .Net backend. This allowed us to easily track and monitor changes in the environment and ensure that any time we made a release or deployment we could be certain that there would be minimal issues in production.

To ensure that our code is properly tested and up to industry standards we have aimed for atleast a 75% line coverage for each different scope of our project.
In this regard we have managed to maintain a 91% code coverage for the BirdApi.

3.2 Workflows

Probably the most important and useful tool at our disposal was GitHub actions as it allowed for easily configurable and integrated automations in the form of workflows for automating tedious or otherwise impossible tasks directly from GitHub.

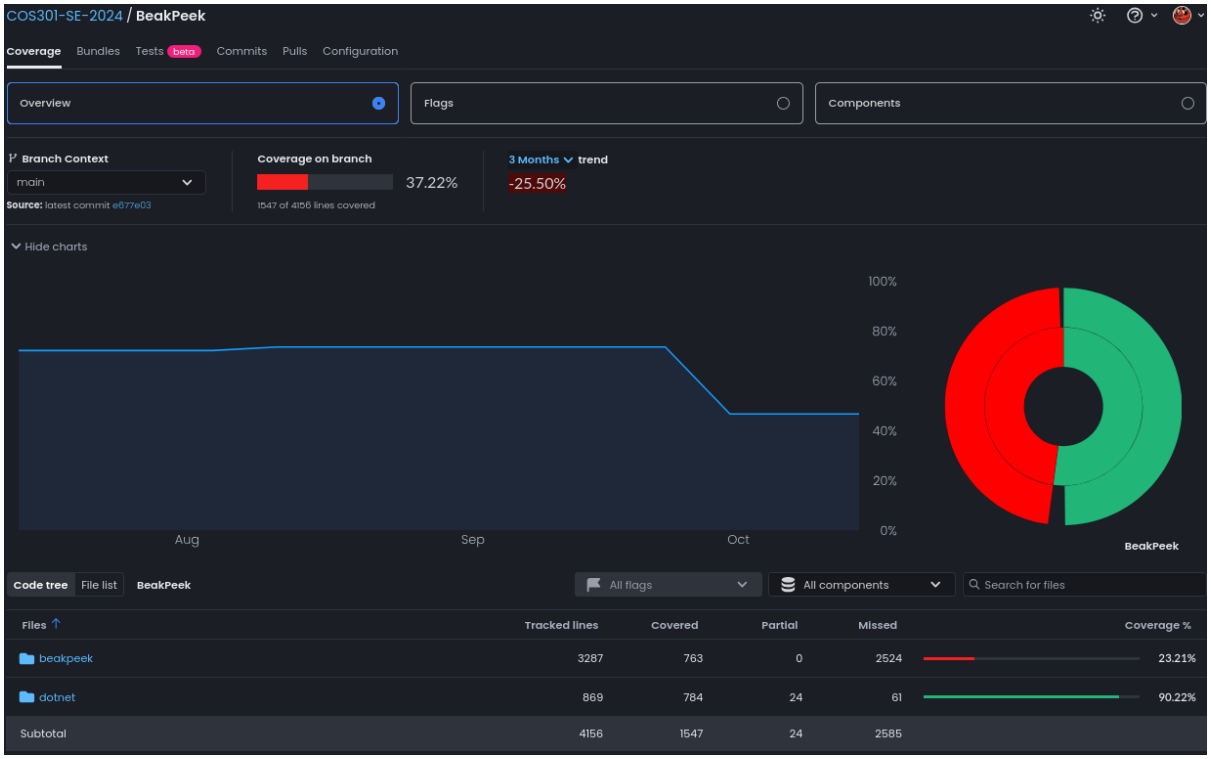

Figure 4: Codecov.

All of the workflows were written by our DevOps engineer (Ivan Horak), who ensured that our developer experience was always as easy as it possibly could be.

Ivan wrote a number of workflows and full pipelines to automate most of the tasks that are often done manually. Bellow is a list of some of the workflows and pipelines.

3.2.1 Discord Notify

The discord notification workflow is a reusable workflow used to send embed messages to the Millennium discord server which allows for easy observability of what is happening in the project and if there are any failing workflows, releases or deployments.

3.2.2 Deploy BirdApi, UserApi

There are two main workflows for deploying the BirdApi and UserApi to Azure, these are the essential for the easy management of the two backends as it gave us a lot of control over how and when the APIs would be deployed.

These workflows also use best practices in that they cache any dependencies for future workflows which significantly reduces deployment times.

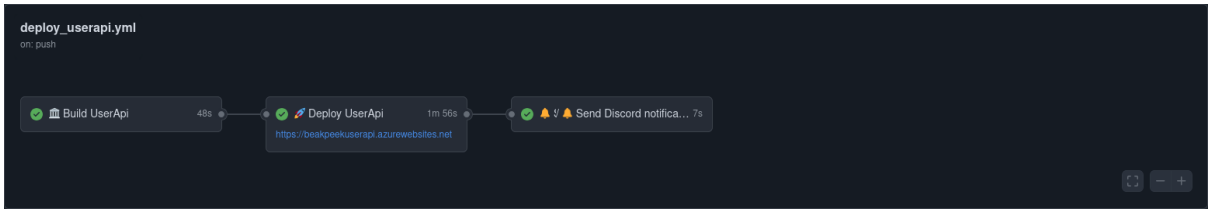

Figure 5: Deploy User.

3.2.3 Update Bird data

The update bird data workflow is a custom workflow that uses in house scripts to download all 9 provinces data as CSV's along with a CSV for all the birds in South Africa which it then uses to import all the data into the Bird database using the Bird API which has a couple endpoints that accept CSV files as input and transform the data in the files to match our Bird and Province models. These CSV files have approximately 1.5 million rows of data that would be difficult for anyone to manage efficiently. For our use case some of the birds that would get fetched from SABAP would have invalid data such as a low report rate which don't work with the population estimates that we make using the data that is imported through this workflow.

Figure 6: Get Bird.

3.2.4 Label Pull Requests

The label pull requests workflow is a simple yet useful workflow that is used to add the appropriate labels to any pull requests based on the branch name and contents which are configured in the labeler.yml file making for easy extensibility and ensuring that every pull request is always appropriately labeled.

3.3 The coverage workflows

The coverage workflows are run for the flutter frontend and the .Net backend. These are usually run together as Codecov does not track any directories that are not included in the commit. This proves problematic as more often than not the frontend and backend workflows are not run together, as changes should very rarely effect both sources in the same commit. To get around this problem we have made use of artifacts that are used to store coverage reports in between workflows for later useage, Ivan has leveraged artifacts by first checking if there is an existing artifact with the same name as the hash of the different sources package lock files, this allows us to only run the tests when a change that would effect the test runs output. We have used this to in a sense short circuit the coverage uploads as if there is an artifact for either of the reports that will be used alongside any new reports that are generated by these workflows.

These workflows can still run independently if needed allowing for rapid iteration. We have also used a seperate workflow as a sort of manager for both of these workflows that if they need to be run without changing the actual source code, the workflows can still be run either manually or automatically.

3.3.1 Publish Android release

The publish Android release is a workflow that uses the Fastlane tool set to build, test and release the flutter frontend as an Android aab file which is a requirement of Google for more secure applications. This workflow also uses dependency caching to allow for faster runs of the workflows.

3.4 Pipelines

To connect all of the seperate workflows that have been mentioned above there is a full CICD pipeline that is used to connect all of the other workflows in a way that ensures that all of the industry best practices are used. This is done by chaining all of the workflows together to ensure that if any step in the build, test or deployment stages fails nothing permanent is done to either of the production databases and the Google play release along with the deployed APIs.

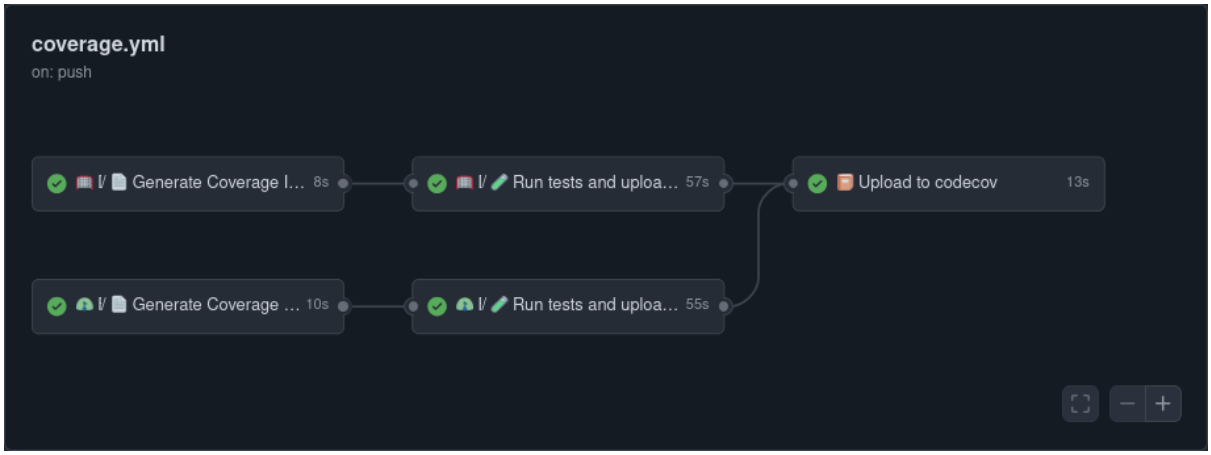

Figure 7: Generate Coverage.

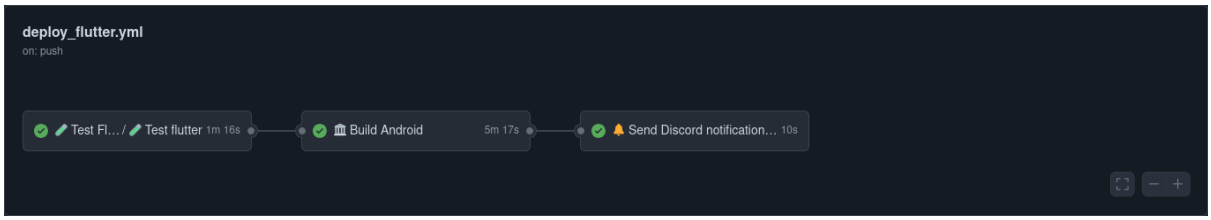

Figure 8: Generate Coverage.

The full pipeline is made up of all of the other workflows in such a way that discord notifications are sent at each stage of the pipeline with links to any build artifacts or errors from any of the steps and or jobs. The final steps in the pipeline is releasing the app and also updating the bird data.

Figure 9: Full Pipeline

4 Observability Principles

To ensure that we follow all of the best observability principles Ivan has put together a couple different tools for monitoring the APIs and database as well as the traffic to the APIs. He has also added in other methods which are used to monitor workflows and builds as well as the projects repository itself.

4.1 Dashboards

One of the primary tools that we have used to monitor our APIs is through the use of different Dashboards for all of the different tools that we use.

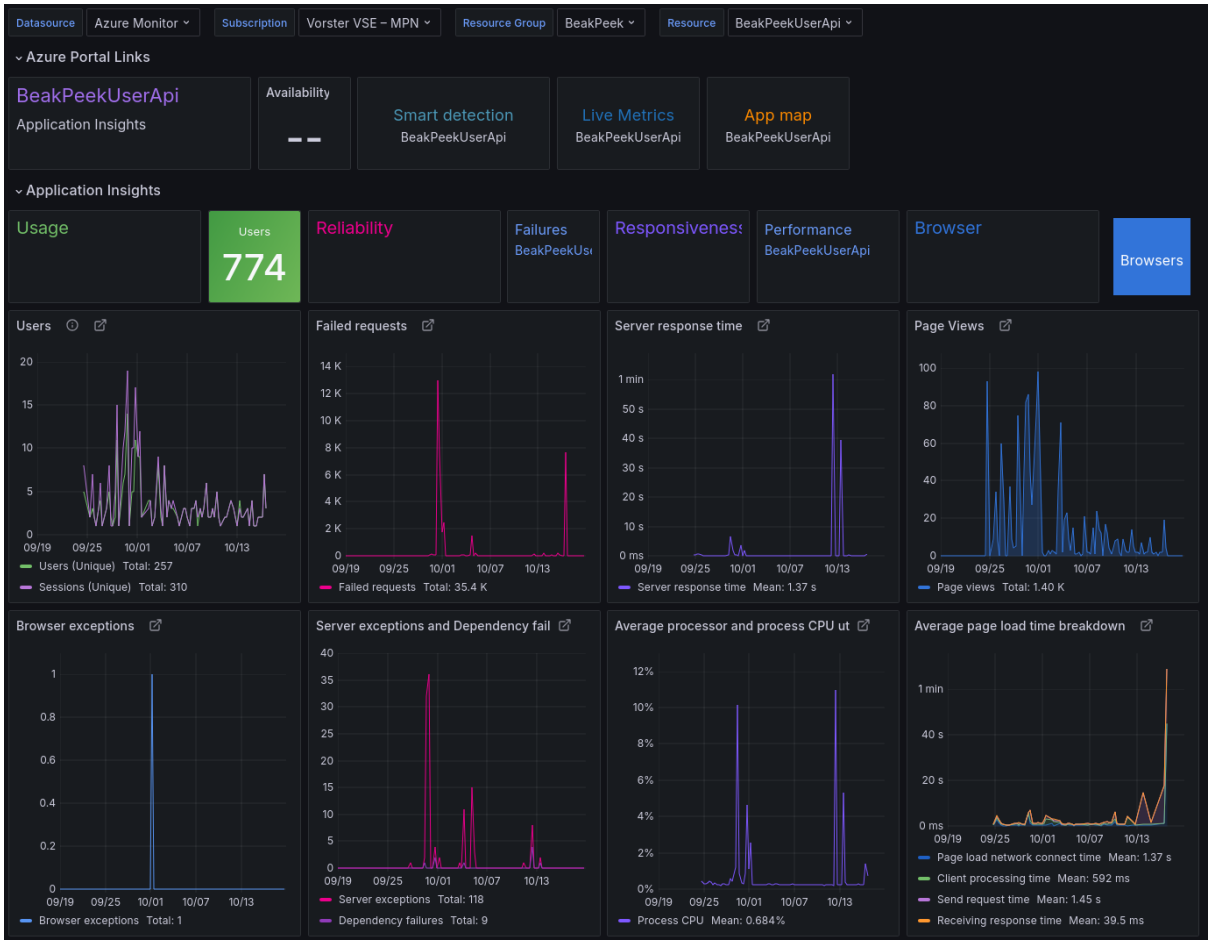

Figure 10: Grafana Dashboard.

4.1.1 Azure Metrics

Azure offers an extensive list of tools that can be used and that we use to make dashboards for monitoring metrics like the response time of the APIs or the usage of the CPU for the different virtual machines that are hosted on Azure.

4.2 Azure Monitoring

Azure also offers a service called application insights which we have used for tracking the hosted APIs on Azure and sending alerts if an error or some exception is raised.

Azure also offers tools for automatic up scaling and downscaling of the resources that we manage and host on Azure. We have in particular used the health check service to ping the server and alert us if any of the containers become unhealthy.

4.3 Azure Logs

We have also used the azure storage account along with the provided logging tools to ensure that if any of our deployments go down or stop working for any reason we can always go back and look at what happened at the time of the incident.

4.4 GitHub Actions

We also use GitHub actions to ensure that we have maximum observability at any given time. As we can't always be at our computers however all of us do have discord on our phones which allows us to get notified easily and quickly any time an incident occurs.

5 Operational Practices

In terms of the operational practices we have used to ensure that we can always recover from a fatal mistake, we have made backups of the data stored on SABAP to ensure that we its unlikely that we could find ourselves in a situation where we cannot recover any lost bird data.

The bird data is also automatically updated once a day through the usage of the update bird date workflow which has a schedule that it runs on as well as any other actions that can cause the workflow to be run, such as the BirdApi being deployed.

5.1 Data Ingress

As it was mentioned above we ensure that any data we use is easily managed and should never require manual transformations or editing. This extends to the bird images which are stored in an Azure storage account and are automatically fetched and updated as bird info is requested.

5.2 Users data is stored locally on device

One of the features that we implemented for BeakPeek was the ability for the mobile app to run without an internet connection. This means that even if a problem were to arise on our side our users should be minimally effected.

5.3 Repository Management

Finally the most important resource we have is the repository and to make sure that nothing effects the operation and usage of the repository we have put Ivan in charge of anything related to it. In this role Ivan reviews all pull requests ensuring that they use proper form and proper descriptions and that there are no other problems.