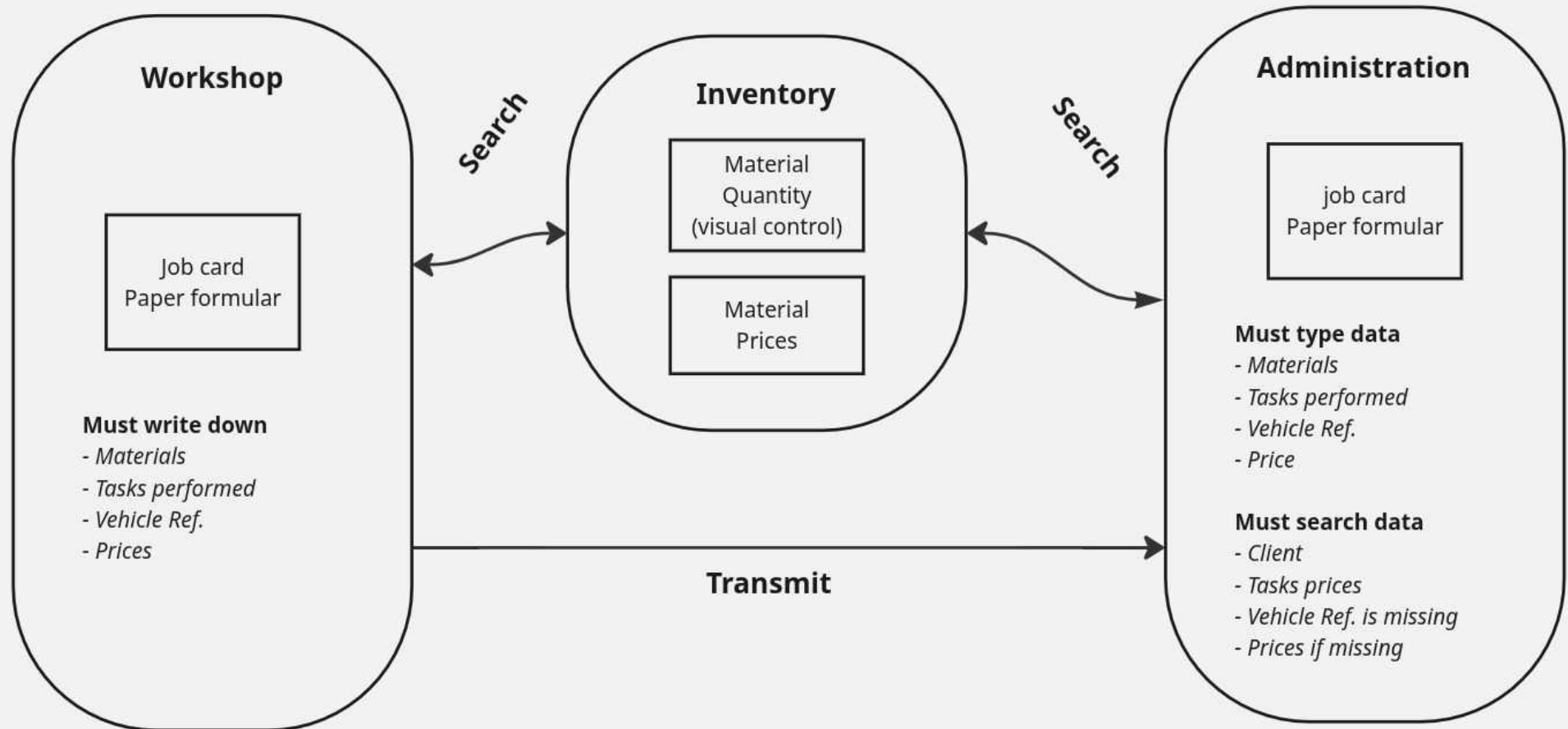


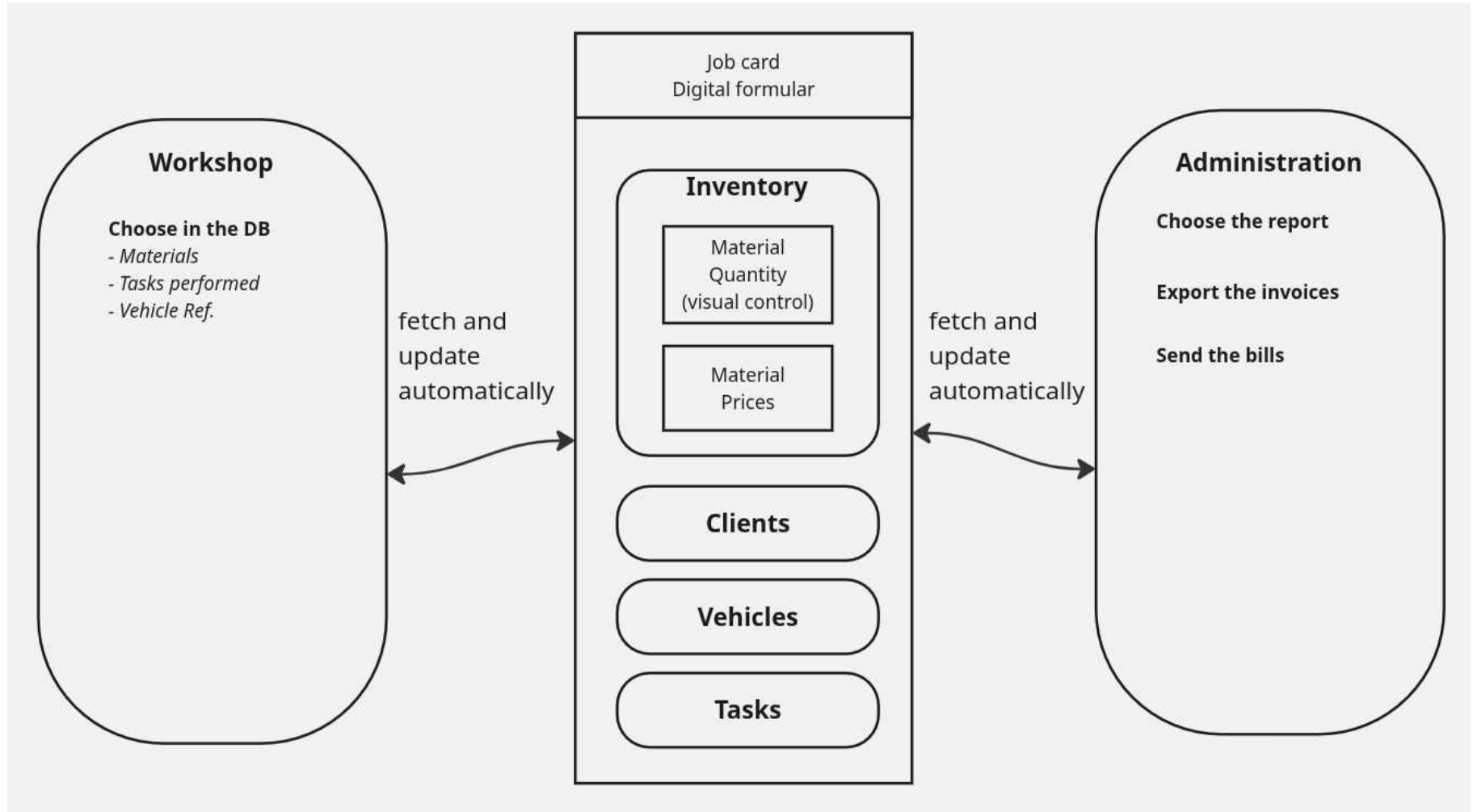
# Garage job card manager

*Digitale Verwaltungsplattform für Fahrzeugreparaturen  
mit Django, React und Docker*

# Current Process




# New Process



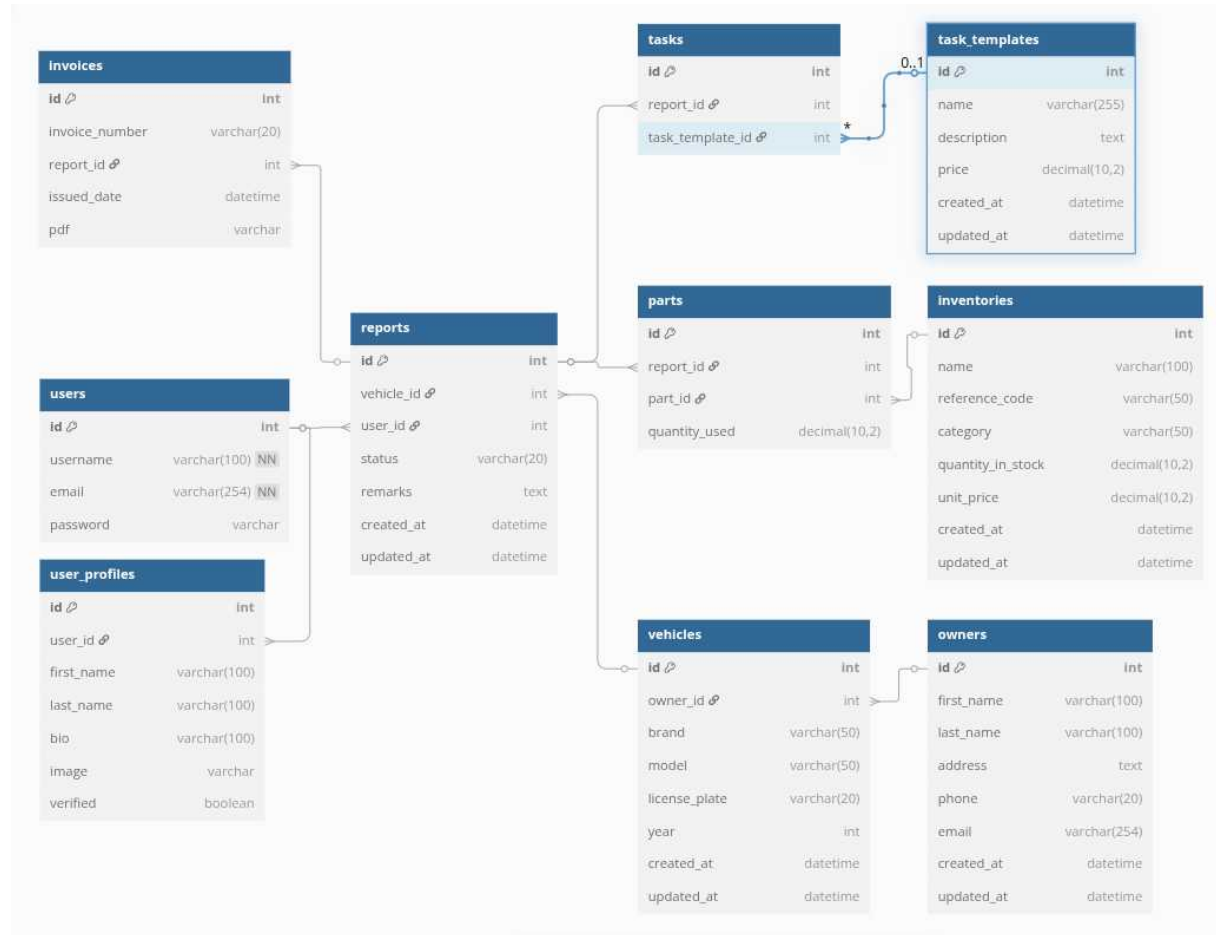
# Project structure and code snippets

- ERD Diagram
- Flow Chart
- Architecture
- Code snippets
- UI / UX

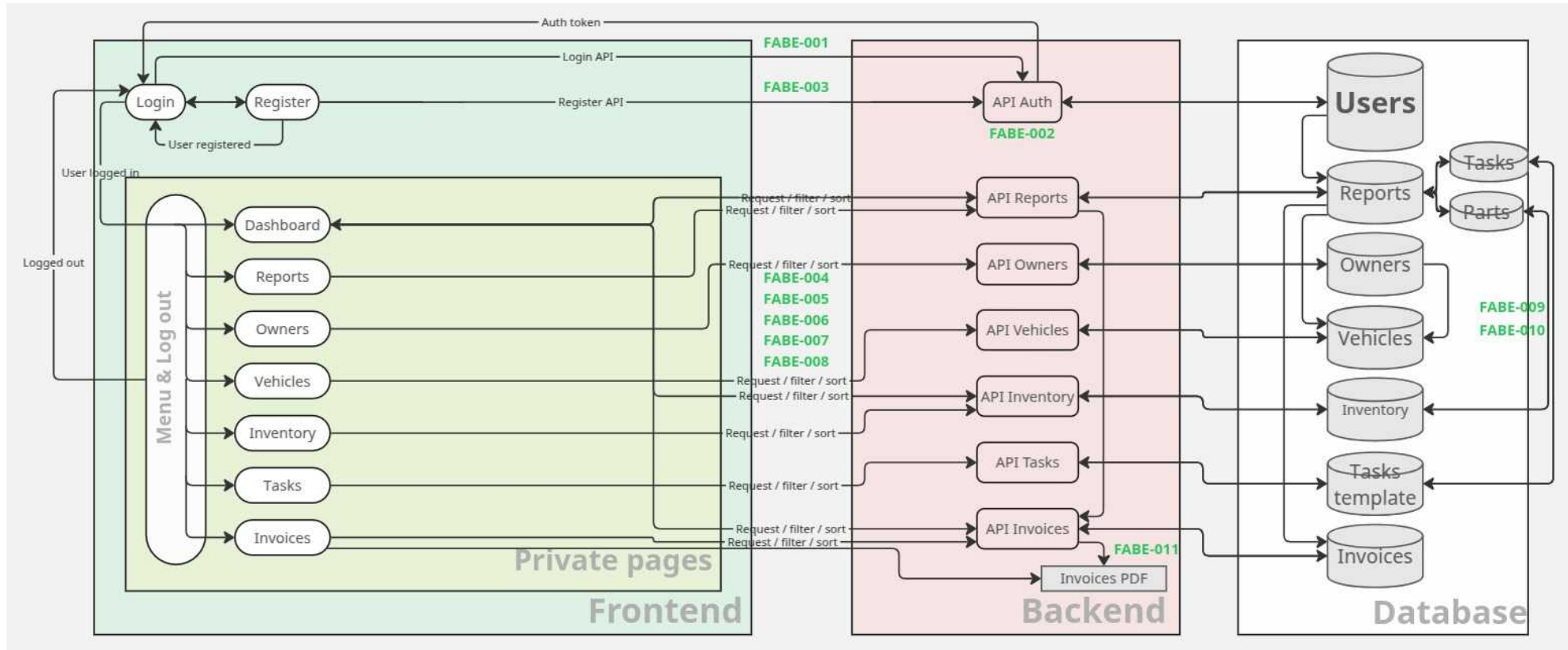


- database population
- report model
- report serializer
- report view
- invoice generation
- Zustand fetcher component

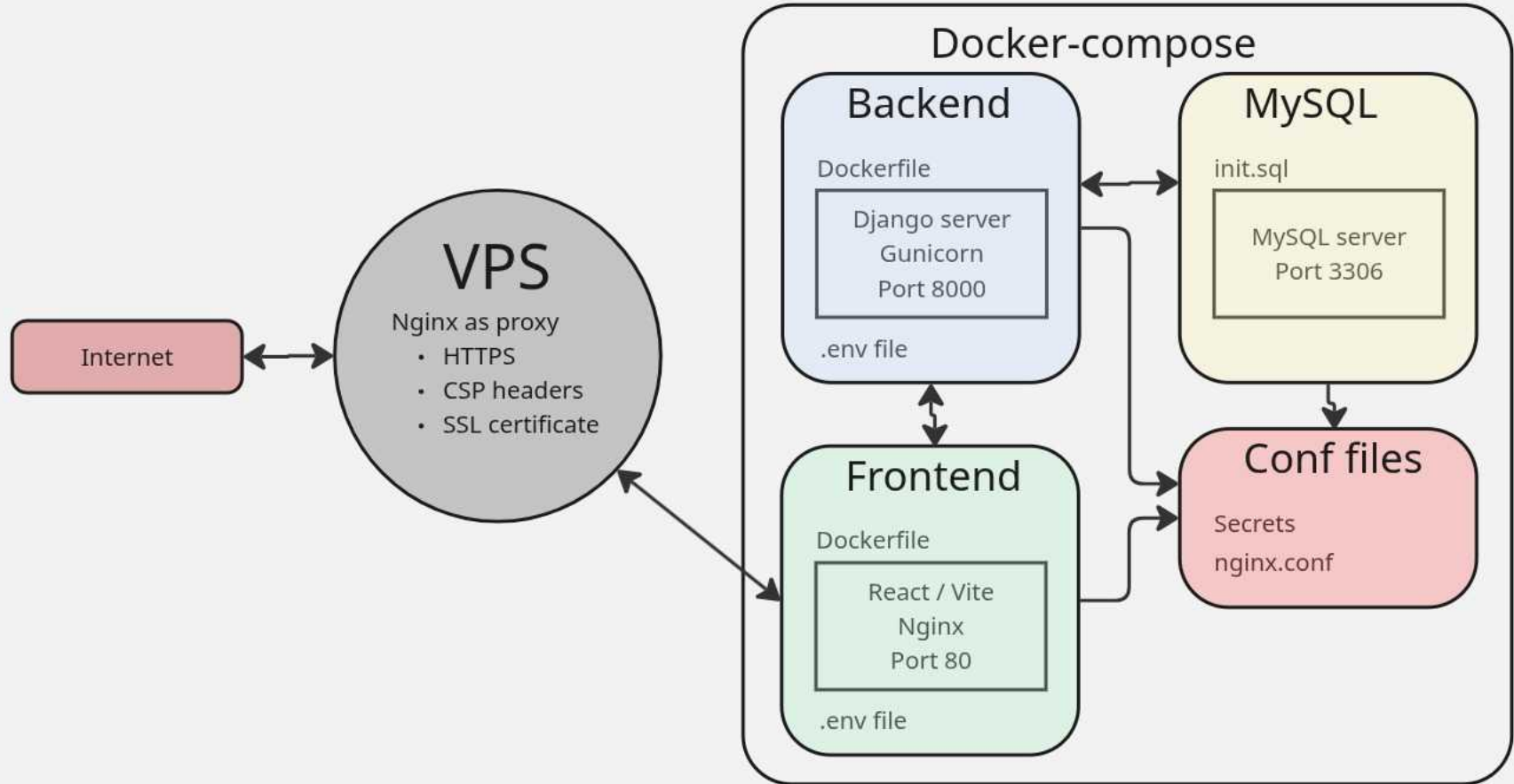
# ERD Diagram




# Flow Chart



# Architecture



# User Interface



**The Garage**

Logged in as :  
Test user

Log Out

Dashboard

Reports


Owners

Vehicles

Invoices

Inventory

Task templates



Vehicle

Owner

All Owner

Users


All Users

Status

All Status

Success

Report updated successfully!



**BMW X3 (795 9RW)**

Edit


Delete

Created Date: March 17, 2025

Owner: Terri Hurst

Status: **In Progress**

Created By: test user



**BMW X5 (4TR T66)**

Edit


Delete

Created Date: March 17, 2025

Owner: Vicki Quinn

Status: **Completed**

Created By: walternelson



**Citroen C4 (M27 0EY)**

Edit


Delete

Created Date: March 17, 2025

Owner: Alexa Day DVM

Status: **Pending**

Created By: walternelson



**Dacia Sandero (362 KS2)**

Edit


Delete

Created Date: March 17, 2025

Owner: Allen Davis

Status: **Pending**

Created By: kcarroll



**Ford Focus (UVB Y64)**

Edit


Delete

Created Date: March 17, 2025

Owner: Dana Phillips

Status: **Pending**

Created By: sryan



**Ford Focus (UVB Y64)**

Edit

Delete

Created Date: March 17, 2025

Owner: Dana Phillips

Status: **Completed**

Created By: nrivera



# Populate DB snippet

```
def populate_reports(users, vehicles):
    if not Inventory.objects.exists():
        populate_inventory()
    if not TaskTemplate.objects.exists():
        populate_task_templates()

    if Report.objects.count() > 8:
        print("Reports already populated.")
        return Report.objects.all()

    for vehicle in vehicles:
        report = Report.objects.create(
            vehicle=vehicle,
            user=random.choice(users),
            remarks=fake.sentence(),
            status=fake.random_element(['pending', 'completed', 'in_progress']),
            created_at=fake.date_time()
        )

        # Add 1 to 3 tasks to the report
        tasks = TaskTemplate.objects.order_by('?')[:random.randint(1, 3)]
        for task_template in tasks:
            Task.objects.create(report=report, task_template=task_template)

        # Add 0 to 3 parts to the report
        parts = Inventory.objects.order_by('?')[:random.randint(0, 3)]
        for part in parts:
            if part.quantity_in_stock > 5:
                Part.objects.create(report=report, part=part, quantity_used=random.randint(1, 5))

    print(f'Created report for vehicle {vehicle.license_plate}')
```

# Model snippet

```
# ----- REPORT & TASKS -----
class Report(models.Model):
    """
    Inspection or service report associated with a vehicle and a user.

    Tracks the status of vehicle inspection and repair tasks.
    """
    STATUS_CHOICES = [
        ('pending', 'Pending'),
        ('in_progress', 'In Progress'),
        ('completed', 'Completed'),
        ('exported', 'Exported')
    ]

    vehicle = models.ForeignKey(Vehicle, on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES)
    remarks = models.TextField(blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):
        return f"Report number {self.id} for {self.vehicle} - {self.status}"

    def get_status_display(self):
        """Returns the user-readable status."""
        return dict(self.STATUS_CHOICES).get(self.status, self.status)

    def delete(self, *args, **kwargs):
        """
        On delete, cascade and restore inventory quantities
        from related parts.
        """
        for part in self.part_set.all():
            part.delete()
        super().delete(*args, **kwargs)
```

# Serializer snippet

```
def create(self, validated_data):
    """
    Creates a report and its associated tasks and parts.
    """
    tasks = validated_data.pop('tasks', [])
    parts = validated_data.pop('parts', [])

    report = Report.objects.create(**validated_data)

    Task.objects.bulk_create([
        Task(report=report, task_template_id=task_id) for task_id in tasks
    ])

    # Create parts one by one to trigger inventory logic
    for part_data in parts:
        Part.objects.create(
            report=report,  # (variable) part_data: Any
            part_id=part_data["part"],
            quantity_used=Decimal(part_data["quantity_used"])
        )

    return report
```

# View snippet

```
class ReportViewSet(viewsets.ModelViewSet):    "viewsets": Unknown word.
    """
    API endpoint for managing maintenance reports.

    Includes logic for pagination, filtering, ordering, and concurrency control.
    Exposes related tasks and parts.
    """

    queryset = Report.objects.select_related('vehicle').all()
    serializer_class = ReportSerializer
    permission_classes = [permissions.IsAuthenticated]

    # disable Pagination
    pagination_class = None

    # To set up filters from the backend side
    filter_backends = [DjangoFilterBackend, filters.OrderingFilter]
    filterset_fields = {    "filterset": Unknown word.
        'status': ['exact', 'in'],
        'vehicle__brand': ['exact'],
        'vehicle__owner': ['exact'],
    }

    ordering_fields = ['vehicle__brand', 'vehicle__model', 'created_at', 'updated_at', 'status']

    def list(self, request, *args, **kwargs):
        limit = request.query_params.get("limit")
        offset = request.query_params.get("offset")
        ordering = request.query_params.get("ordering", "vehicle__brand,vehicle__model")

        queryset = self.filter_queryset(self.get_queryset()).order_by(*ordering.split(","))

        if limit or offset:
            self.pagination_class = CustomPagination
            paginator = self.pagination_class()
            paginated_queryset = paginator.paginate_queryset(queryset, request)
            return paginator.get_paginated_response(self.get_serializer(paginated_queryset, many=True))

        # If no pagination params are set, return all results
        return Response(self.get_serializer(queryset, many=True).data)

    def update(self, request, *args, **kwargs):
        """Allow partial updates while keeping existing values for missing fields."""
        partial = kwargs.pop('partial', False)
        instance = self.get_object()
```



# Helper function snippet

```
def generate_invoice(self, report, request):
    """Create an invoice and generate a PDF for an exported report."""
    invoice_number = f"INV-{report.id:06d}"

    # Create an Invoice entry
    invoice = Invoice.objects.create(
        invoice_number=invoice_number,
        report=report
    )

    # Generate invoice PDF and calculate the total cost with VAT
    html_content = self.generate_invoice_pdf(invoice)

    invoice.save()

    pdf_file = HTML(string=html_content, base_url=request.build_absolute_uri()).write_pdf()

    # Save the PDF to the invoice model
    invoice.pdf.save(f"invoices/invoice_{invoice_number}.pdf", ContentFile(pdf_file), save=True)

    return Response({"message": "Invoice generated successfully", "invoice_id": invoice.id})

def generate_invoice_pdf(self, invoice):
    """Generate and return a PDF file for the invoice."""
    tasks = Task.objects.filter(report=invoice.report)
    parts = Part.objects.filter(report=invoice.report)
    # Join task with task template and part with inventory
    tasks = tasks.select_related('task_template')
    parts = parts.select_related('part')
```

# Store snippet

```
const REPORT_API_URL = import.meta.env.VITE_API_URL + '/reports/'

const useReportStore = create((set) => ({
  reports: [],
  pagination: null,
  loading: false,
  error: null,

  fetchReports: async (params = {}) => {
    set({ loading: true })
    try {
      const cleanParams = Object.fromEntries(Object.entries(params).filter(([_, v]) => v != null))
      const queryParams = new URLSearchParams(cleanParams)
      if (params.ordering) queryParams.append('ordering', params.ordering)

      const response = await axiosInstance.get(`${REPORT_API_URL}?${queryParams}`)

      const isPaginated = 'results' in response.data
      const results = isPaginated ? response.data.results : response.data
      const pagination = isPaginated
        ? { count: response.data.count, next: response.data.next, previous: response.data.previous }
        : null

      set({
        reports: results,
        pagination,
        loading: false,
      })
    } catch (error) {
      set({ error: error.message, loading: false })
    }
  },
})
```